ESCUELA SUPERIOR POLITÉCNICA DE CHIMBORAZO

MODELO DE UNA INTERFAZ DE PROGRAMACIÓN DE APLICACIONES REST UTILIZANDO GO BASADO EN NORMAS Y PRINCIPIOS DE SEGURIDAD DE LA INFORMACIÓN Y APLICACIONES WEB

MAYRA ALEJANDRA LARA MÉNDEZ

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:

MAGÍSTER EN SEGURIDAD TELEMÁTICA

RIOBAMBA - ECUADOR

Junio 2021

CERTIFICACIÓN:

EL TRIBUNAL DEL TRABAJO DE TITULACIÓN CERTIFICA QUE:

El Trabajo de Titulación modalidad Proyectos de Investigación y Desarrollo, denominado: "**Modelo de una interfaz de programación de aplicaciones REST utilizando GO basado en normas y principios de seguridad de la información y aplicaciones web**", de responsabilidad de la señorita Mayra Alejandra Lara Méndez, ha sido minuciosamente revisado y se autoriza su presentación.

Ing. Luis Hidalgo Ph.D.
PRESIDENTE

LUIS EDUARDO HIDALGO ALMEIDA

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

Firmado electrónicamente por: WILIAN XAVIER SANCHEZ LABRE

Dr. Julio Roberto Santillán Castillo, M.Sc.
MIEMBRO DEL TRIBUNAL

Firmado electrónicamente por: JULIO ROBERTO SANTILLAN CASTILLO

Ing. Marco Vinicio Ramos Valencia, M.Sc.
MIEMBRO DEL TRIBUNAL

MARCO VINICIO RAMOS VALENCIA

Riobamba, junio de 2021

DERECHOS INTELECTUALES

Yo, Mayra Alejandra Lara Méndez, declaro que soy responsable de las ideas, doctrinas y resultados expuestos en el **Trabajo de Titulación modalidad Proyectos de Investigación y Desarrollo**, y que el patrimonio intelectual generado por la misma pertenece exclusivamente a la Escuela Superior Politécnica de Chimborazo.

Mayra Alejandra Lara Méndez

No. 0604433771

DECLARACIÓN DE AUTENTICIDAD

Yo, Mayra Alejandra Lara Méndez, 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 autora, asumo la responsabilidad legal y académica de los contenidos de este proyecto de investigación de maestría.

Riobamba, febrero de 2021

Mayra Alejandra Lara Méndez

No. 0604433771

DEDICATORIA

El presente trabajo lo quiero dedicar a Dios y a mis padres y mi hermano, que siempre han estado a mi lado y son el pilar de mi vida.

A Danier, por hacer esta meta de los dos, por ayudarme a crecer profesionalmente y juntos como personas. Porque ha sido mi mejor amigo, mi inspiración, mi pareja, mi soporte y más… Sin duda nada sería igual si no hubiera sostenido mi mano estos años. Gracias por estar a mi lado para celebrar cada logro alcanzado y a pesar de su carga laboral estar para mí, siempre.

AGRADECIMIENTO

Mi profundo agradecimiento a mi familia y a mi esposo. Agradezco a todos los docentes que me han guiado e impartido sus conocimientos a lo largo de mi formación profesional en la ESPOCH.

ÍNDICE GENERAL

ÍNDICE TABLAS

ÍNDICE FIGURAS

ÍNDICE DE ABREVIATURAS

API	Application Programming Interface
REST	Representational State Transfer
ISO	International Organization for Standardization
OWASP	Open Web Application Security Project
JSON	JavaScript Object Notation
SOAP	Simple Object Access Protocol
SGSI	Sistema de Gestión de Seguridad de la Información
JWT	Jason Web Token
ORM	Object Relational Model
JS	Java Script
PII	Información Personal Identificable
SSL	Secure Sockets Layer
CVE	Common Vulnerabilities and Exposures
NVD	National Vulnerability Database

RESUMEN

El presente trabajo de investigación tiene por objetivo proporcionar un modelo seguro y pragmático para API REST con las mejores prácticas, para aplicaciones web modernas, utilizando el lenguaje de programación GO. La investigación muestra un análisis de las principales vulnerabilidades para proteger una Interfaz de Programación de Aplicaciones. Además, un estudio de las normas, estándares y guías de seguridad de la información para construir un modelo que servirá de referencia para desarrollar aplicaciones web más seguras. Para comprobar la validez del modelo, se realizó el escaneo de vulnerabilidades en el PROTOTIPO I (sin modelo) y el PROTOTIPO II (utilizando el modelo). Se observó un 53.33% en el prototipo I y un 2.77% en el prototipo II, mostrando una mejora en la seguridad en el prototipo II de un 50.36% respecto a los resultados obtenidos del prototipo I. Para la comprobación de la hipótesis se utilizó la prueba estadística T de student. El resultado, una evidente mejora en la seguridad de la API. En conclusión, el modelo basado en el uso de normas, estándares y buenas prácticas es un modelo que aporta con directrices para construir Interfaces de Programación de Aplicaciones con mayor seguridad, por lo cual, se recomienda el uso del modelo para la implementación de APIs REST utilizando el lenguaje de programación GO. Es importante saber que la seguridad es un proceso, no un producto, por lo cual, se recomienda usar estándares y/o buenas prácticas durante todo el ciclo de vida del software, monitorearlo y darle mantenimiento durante su vida útil. Finalmente, siempre conocer las reglas del negocio y la relación entre los recursos, tener el software actualizado, mantenerse al tanto sobre vulnerabilidades de la base de datos Common Vulnerabilities and Exposures (CVE), National Vulnerability Database (NVD), temas de seguridad informática y sobre todo nunca parar de investigar.

Palabras Clave: <SEGURIDAD TELEMÁTICA>, <INTERFAZ DE PROGRAMACIÓN DE APLICACIONES (API)>, <SEGURIDAD DE APLICACIONES WEB>, <ARQUITECTURA REST>, < GO (LENGUAJE DE PROGRAMACIÓN) >.

SUMMARY

The present research work aims to provide a safe and pragmatic model for REST API with best practices, for modern web applications, using the GO programming language. The research shows an analysis of the main vulnerabilities to protect an Application Programming Interface. In addition, a study of the norms, standards and information security guides to build a model that will serve as a reference to develop more secure web applications. To check the validity of the model, a vulnerability scan was performed in PROTOTYPE I (without model) and PROTOTYPE II (using the model). A 53.33% was observed in prototype I and 2.77% in prototype II, showing an improvement in safety in prototype II of 50.36% with respect to the results obtained from prototype I. To verify the hypothesis, the Statistical Student's T test. The result, an obvious improvement in API security. In conclusion, the model based on the use of norms, standards and good practices is a model that provides guidelines to build Application Programming Interfaces with greater security, therefore, the use of the model is recommended for the implementation of REST APIs. using the GO programming language. It is important to know that security is a process, not a product, therefore, it is recommended to use standards and / or good practices throughout the life cycle of the software, monitor it and give it maintenance during its useful life. Finally, always know the business rules and the relationship between resources, have the software updated, stay up-to-date on vulnerabilities in the Common Vulnerabilities and Exposures (CVE) database, National Vulnerability Database (NVD), computer security issues and above all never stop investigating.

Keywords: <APPLICATION PROGRAMMING INTERFACE (API)>, <REST APPLICATION PROGRAMMING INTERFACE>, <WEB APPLICATION SECURITY>, <REST ARCHITECTURE>, <SECURE API>, <GO PROGRAMMING LANGUAGE>.

CAPÍTULO I

1 INTRODUCCIÓN

1.1. Problema de investigación

1.1.1. Planteamiento del Problema

El mundo actual, tan interconectado a través de internet, impulsado por la tecnológica evolución e innovación y el creciente número de dispositivos conectados, hace que la web cada vez se vuelve más compleja. Las empresas y organizaciones gestionan casi la totalidad de sus datos mediante computadoras, smartphones, servicios en la nube y aplicaciones web a través de la red. En este entorno, los incidentes de seguridad y ciberataques se incrementan exponencialmente.

La seguridad informática o ciberseguridad es la forma de proteger los activos informáticos de los ciberdelincuentes y es crítica para salvaguardar la información como activo de alto valor de las empresas. Casi todo negocio o servicio está gestionado a través de aplicaciones web, razón por la cual, son el blanco más frecuente de ataques y, es necesario desarrollar aplicaciones robustas. Para conseguirlo, la seguridad en las aplicaciones debe ser parte de las mismas desde su concepción hasta el producto final.

Pero, construir software es complejo, e incluso las empresas que cuentan con departamentos de calidad, fallan. Los desarrolladores se enfrentan a la presión del desarrollo rápido de software y migración a nuevas tecnologías y terminan sacrificando el proceso de aseguramiento del software. Si a la vez, sumamos la complejidad y alcance de las soluciones que a menudo crecen con el tiempo, obtenemos agujeros de seguridad difíciles de detectar entre cientos o miles de líneas de código. En cuanto a lenguajes de programación se refiere, cada lenguaje es diferente, cada uno tiene su paradigma, problemas, sintaxis y marco de trabajo.

El problema de seguridad en el software es tan perjudicial, que las grandes empresas, han adoptado como estrategia, realizar programas de recompensas para quien encuentre vulnerabilidades en sus aplicaciones, para que sean corregidas.

Actualmente, los lenguajes de programación, la arquitectura y tecnologías para el desarrollo de software están innovando la web con nuevos conceptos y paradigmas que buscan satisfacer las necesidades cambiantes de las empresas e internautas. Sin embargo, para construir aplicaciones web funcionales, eficientes y seguras, el uso de un "buen lenguaje de programación" no es suficiente, se necesita buenas prácticas de parte de los programadores. En consecuencia, se crearon proyectos para la creación de estándares internacionales que proporciona principios y buenas prácticas de seguridad como: OWASP, ISO 27001, OSSTMM, CISA, etc.

La aplicación de normas y estándares internacionales, proporcionan guías y principios para construir software más seguro y mitigar vulnerabilidades. Estos principios, son genéricos y serán la base de este estudio para diseñar un modelo de una API REST basado en recomendaciones y buenas prácticas de seguridad.

1.2 Formulación del problema

¿Cómo aporta un modelo de API REST aplicando normas y principios de seguridad de la información y aplicaciones web?

1.3 Sistematización del problema

¿Cuáles son las vulnerabilidades más conocidas y explotadas en aplicaciones web?
¿Cuáles son los aspectos que se deben considerar para proteger una API REST?
¿Qué normas y/o estándares aportan para conseguir un software seguro?
¿Cómo diseñar un modelo de API segura utilizando GO?

1.4 Justificación de la investigación

1.4.1 Justificación Teórica

El desarrollo de software inseguro es un riesgo potencial que afecta a la Confidencialidad, Integridad y Disponibilidad de la información como bien intangible de las empresas cuya pérdida, robo o sabotaje puede tener consecuencias desastrosas.

Hoy en día las Interfaces de Programación de Aplicaciones (APIs) conforman una tecnología importante e innovadora muy utilizada, ya que permite la comunicación entre aplicaciones y que los desarrollos ya construidos sean aprovechados. Las APIs por su naturaleza permiten la exposición de datos para que sean consumidos por otro software de manera rápida y ágil. Sin embargo, el desarrollo de una APIs de manera apresurada y sin un modelo seguro puede provocar productos inseguros. Algunos de los problemas más relevantes que se encuentran en APIs según el OWASP API Security Top 10 son: ausencia de controles, mecanismos incorrectos de autenticación, validaciones, exposición de datos sensibles, componentes internos expuestos en servicios REST públicos, entre otros.

Actualmente, existen trabajos de investigación que proponen metodologías, métodos o guías de buenas prácticas que abordan la problemática desde una perspectiva genérica, excepto el trabajo propone un método basado en técnicas para desarrollo de software seguro, en entonto PHP. Si indagamos en internet existen diversas opiniones acerca de diseños de APIs, pero, no existe un estándar, o modelo sustentado por buenas prácticas que funcione para la mayoría de casos. Por lo tanto, la presente investigación tiene como objetivo aportar un modelo de una API REST utilizando el lenguaje de programación GO aplicando principios y guías de seguridad.

1.4.1 Justificación Práctica

El análisis de vulnerabilidades se realizará en dos escenarios de prueba utilizando dos prototipos. El Prototipo I es una API existente que es usada en una empresa y el Prototipo II es el modelo creado en base a las normas, estándares y guías que serán seleccionadas; posteriormente realizará un análisis con herramientas para la detección de vulnerabilidades y se compararán los resultados con la finalidad de demostrar la mejora de la seguridad.

1.5 Objetivos

1.5.1 Objetivo general

Diseñar un modelo de API REST utilizando GO basado en normas y principios de seguridad de información y aplicaciones web.

1.5.2 Objetivos específicos

- Analizar las vulnerabilidades que se deben considerar para proteger en una API
- Estudiar los principios y buenas prácticas de normas y estándares de seguridad de la información y aplicaciones web.
- Diseñar un modelo de API REST seguro utilizando GO
- Comprobar la mejoría de seguridad aplicando el modelo de API REST basado en GO

1.6 Hipótesis

Un modelo de API REST utilizando GO basado en normas y principios de seguridad de información y aplicaciones web mejorará el nivel de seguridad.

CAPÍTULO II

2 MARCO DE REFERENCIA

2.1 Antecedentes del Problema

Basado en la problemática de la seguridad inherente de la información y sistemas informáticos, se han creado normas y estándares internacionales para concientizar sobre la importancia de construir software seguro y proporcionar guías, para la mitigación de vulnerabilidades que pueden presentar un producto software.

Actualmente, los estudios existentes sobre metodologías, técnicas y buenas prácticas para el desarrollo del software seguro son:

- "Metodologías para el Desarrollo de Software Seguro". (López, 2015). Propone una nueva metodología para el desarrollo de software seguro a partir de las metodologías existentes como: ISO 27001, OSSTMM, CISA, entre otras. Finalmente, propone para trabajos futuros añadir controles específicos para diferentes lenguajes de programación.
- "Guía de buenas prácticas de desarrollo de aplicaciones web seguras aplicado al sistema control de nuevos aspirantes Empresa Grupo LAAR". (Yánez, 2014). Este trabajo presenta una guía de buenas prácticas basada en metodologías de evaluación de riesgos.
- "Desarrollo de una Propuesta Metodológica para determinar la seguridad en una aplicación web". (Ascencio, M. y Moreno, P., 2011). Esta propuesta muestra una metodología para evaluar la seguridad en aplicaciones.
- "Propuesta de un método utilizando técnicas de programación seguras para el desarrollo de aplicaciones web en entorno php para mitigar riesgos potenciales de seguridad". (Monar, J.,2017). La propuesta de esta investigación se enfoca en un método para mitigar vulnerabilidades al desarrollar aplicaciones web utilizando el lenguaje PHP.

- “Análisis de la metodología SDL para su aplicación en el desarrollo de software seguro en instituciones financieras”. (Aleaga, R.,2019). Se trata de la aplicación de una metodología en el proceso de desarrollo del ciclo de vida del software.

Las investigaciones han sido un aporte importante con propuestas de metodologías, técnicas o guías generales para el desarrollo de software seguro, el último trabajo aporta con técnicas seguras y es el único que se enfoca en un entorno en particular, el lenguaje de programación PHP. Pero, la tendencia tecnológica actual, requiere la actualización de conocimientos y nuevas propuestas para las necesidades actuales.

En los últimos años, la arquitectura de las aplicaciones ha cambiado significativamente, donde, las API son una pieza importante en esta nueva arquitectura de microservicios, aplicaciones móviles, IoT, etc. El OWASP API Security Top 10 publicado en el año 2019, es la primera edición y es un aporte necesario para crear conciencia sobre los problemas de seguridad de las API modernas. (OWASP, 2019).

2.2 Bases Teóricas

2.2.1 Aplicaciones Web

Una aplicación web es una aplicación que se encuentra alojada en un servidor web y puede ser accedida mediante un navegador web a través de internet o una intranet.

2.2.2 Riesgos en la Seguridad de las Aplicaciones

Existen diferentes caminos que un atacante puede encontrar en una aplicación web para explotarlos, causando efectos adversos en una organización. Sin embargo, cada efecto de una vulnerabilidad explotada tiene un impacto o gravedad diferente. Además, cada camino tiene diferente nivel de exposición, es decir, se puede descubrir con mayor o menor dificultad.

Para determinar el riesgo para una organización se debe evaluar la amenaza y asociarla con el impacto que tendría en la organización en caso de ser explotada. (OWASP, 2017)

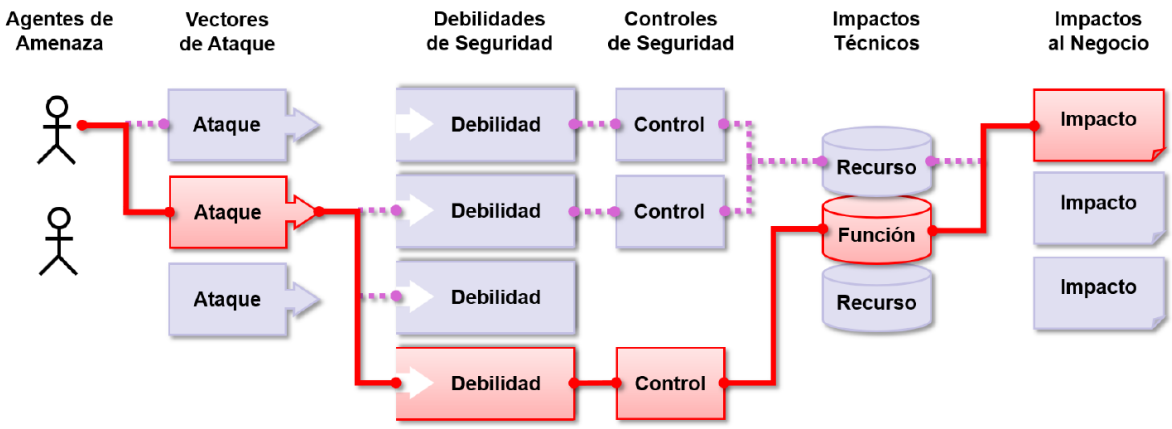


Figura 1-2. Riesgos en seguridad de aplicaciones

Fuente: OWASP 2017

2.2.3 Servicios Web

Los servicios web son una tecnología que integra aplicaciones independientemente de lenguaje de programación en el que estén desarrolladas y de la plataforma en la que se ejecuten. Esta tecnología utiliza protocolos y estándares para establecer la comunicación e intercambio de datos entre aplicaciones, en formato estándar, comúnmente HTTP. Los lenguajes más usados son XML basado en etiquetas y JSON basado en Javascript.

La empresas como Facebook, MySpace, Microsoft, Twitter,etc., utilizan servicios web.

Tipos de servicios web:

- Protocolo Simple de Acceso a Objetos (SOAP)
- Transferencia de Estado Representacional (REST)

Soap

El Protocolo Simple de Acceso a Objetos (SOAP) ha sido utilizado desde la concepción de los servicios web, utiliza un lenguaje de definición de interfaces WSDL y soporta solo formato XML. Los servicios SOAP utilizan generalmente el protocolo HTTP, pero pueden usar otros protocolos como: FTP, POP3, TCP, etc. Es una arquitectura muy potente, pero a su vez, compleja de implementar y gestionar. (Blancarte, 2017)

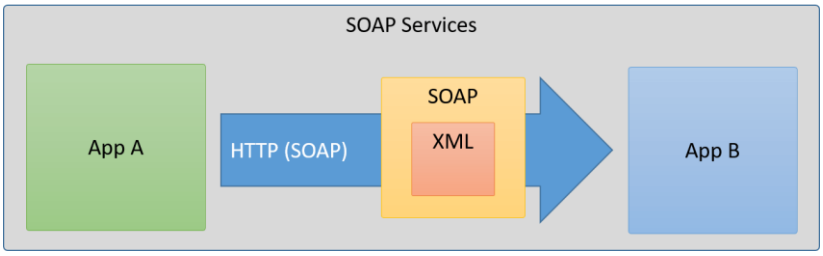


Figura 2-2. Servicio SOAP

Fuente: https://www.oscarblancarteblog.com/2017/03/06/soap-vs-rest-2/

Rest

Este tipo de arquitectura ha enriquecido los servicios, utiliza el protocolo HTTP para la comunicación. El uso de HTTP hace que la API pueda ser utilizada por cualquier lenguaje de programación, garantizando el intercambio de datos en formato estándar. La flexibilidad de REST permite transmitir cualquier tipo de datos, entre ellos, XML, JSON, Binarios, Text, etc. Otra característica de REST es que admite la utilización de los diversos métodos que proporciona HTTP como GET para consulta y lectura, POST para crear, PUT para editar y DELETE para eliminar. (Blancarte, 2017)

REST y JSON hoy en día, se han convertido en la mejor opción y la más utilizada en cuanto a servicios de aplicaciones se refiere, ya que permite el crecimiento en horizontal, por lo tanto, la escalabilidad de proyectos. El formato JSON, es más sencillo, liviano y de rápido procesamiento en referencia a XML, solucionando la complejidad de SOAP. JSON es interpretado por JavaScript, lo que ha hecho que frameworks como Angular y React puedan enviar peticiones directas al servidor por medio de AJAX y obtener los datos de una forma nativa. (Rosa, 2018)

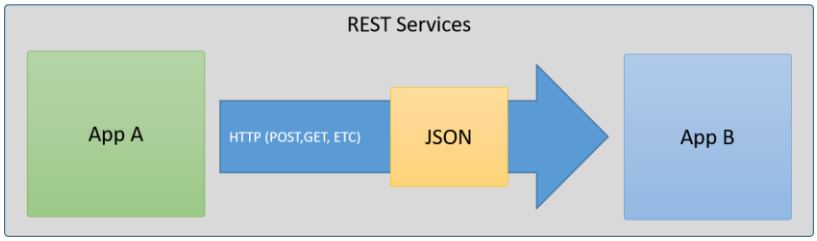


Figura 3-2. Servicio REST

Fuente: https://www.oscarblancarteblog.com/2017/03/06/soap-vs-rest-2/

2.2.4 API

Una Interfaz de Programación de Aplicaciones (API) es un conjunto de funciones, procedimientos o métodos creados para ser consumidos o utilizados por otro software de forma segura. Una API facilita la comunicación entre aplicaciones en formato estándar para el intercambio de datos. Las APIs pueden habilitar el acceso a sus recursos a aplicaciones de terceros de forma controlada y segura. (Redhat, 2018)

Figura 4-2. API

Fuente: https://www.webempresa.com/blog/rest-api-de-wordpress.html

Servicios web y API

Ambos son canales de comunicación, sin embargo, el servicio Web permite la interacción entre dos máquinas a través de una red, mientras que una API es una interfaz de programación de aplicaciones que permite la comunicación entre dos aplicaciones a través de la misma. (GO4IT Solutions, 2018)

Diferencias:

- La API puede definir el modo y el método o métodos que un programa usará para comunicarse con otros.
- No todas las API son un Servicio Web, pero, todos los Servicios Web sí son APIs.
- Un Servicio Web emplea solo tres protocolos, mientras que una API puede hacerlo con cualquiera.
- El servicio Web necesita una web para funcionar siempre, la API no. (GO4IT Solutions, 2018)

2.2.5 API REST

Un API es un conjunto de reglas y especificaciones para permitir que las aplicaciones se comuniquen. Esta interfaz de programación de aplicaciones se apoya en la arquitectura REST.

Entre los sitios web más populares que utilizan REST son: Twitter, Google, Facebook, YouTube, Amazon, Netflix, LinkedIn, etc. Las empresas tienen su lógica de negocio en un API REST y la presentación de datos en el frontend por separado, lo cual permite construir una API más potente y eficiente. (Geeky Theory, 2019)

Las reglas de una arquitectura REST son las siguientes:

- **Cliente-servidor**: El cliente y el servidor se mantienen débilmente acoplados, es decir, el cliente no necesita saber los detalles de la implementación del servidor, y el servidor no se preocupa en proporcionar los datos, mas no en cómo son utilizados.
- **Peticiones sin estado**: cada petición enviada al servidor es independiente, es decir, sin necesidad de mantener sesiones.

- **Cacheable:** compatibilidad con un sistema de almacenamiento en caché de varios niveles. Esto permite recuperar un mismo recurso sin necesidad de establecer nuevamente conexiones entre el cliente y el servidor.
- **Interfaz genérica:** Cada recurso del servicio REST debe tener una única dirección para administrar cada interacción entre cliente y servidor, lo cual simplifica y separa la arquitectura.
- **Arquitectura de capas:** La posibilidad de disponer diferentes capas para la implementación del servidor ayuda a mejorar la escalabilidad, rendimiento y seguridad.
- **Uri:** Los objetos en REST siempre se manipulan a partir de la URI. La URI facilita acceder a la información para realizar operaciones sobre ella. (Geeky Theory, 2019)

2.2.6 Principales vulnerabilidades en APIs

Las API son la tecnología innovadora del intercambio de información de aplicaciones, es así, que el 83% del tráfico de toda la red pasa a través de APIs. Son una parte crítica de las aplicaciones web modernas. La APIs por su naturaleza exponen datos e información de las personas, por lo cual se han hecho un objetivo de los atacantes. (Latam CyberSecurity, 2019)

La importancia de construir APIs seguras ha hecho que el proyecto OWASP publique el top 10 en mayo del 2019, dedicado a las vulnerabilidades en APIs independiente del top 10 para aplicaciones web, a escasos meses antes del desarrollo del presente trabajo.

Las vulnerabilidades presentadas por OWASP API Security Top 10 2019 son:

- Control de Acceso a Nivel de Objeto Roto
- Autenticación Rota
- Filtrado de datos Incorrecto
- Falta de Recursos y Limitación de Velocidad
- Falta de Control de Acceso a Nivel función/recurso
- Asignación Masiva
- Configuración de Seguridad Incorrecta
- Inyección

- Gestión Inadecuada de Activos
- Registro y Monitoreo Insuficientes

2.2.7 Lenguaje de programación GO

Es un lenguaje de programación concurrente, compilado, de tipado estático y no orientado a objetos. Fue desarrollado por Google y publicado bajo licencia de código abierto en el año 2009, es un lenguaje inspirado en la sintaxis de C, sirve para el desarrollo de scripts, backend (API REST) y sockets. Es un lenguaje relativamente nuevo pero robusto, con el cual ya se han desarrollado miles de proyectos y su popularidad está en ascenso por ser: simple, sencillo, eficiente, altamente concurrente y tener alto rendimiento. (KeepCoding, 2017)

2.2.8 Principios de la seguridad

"El único sistema seguro es aquél que está apagado en el interior de un bloque de hormigón protegido en una habitación sellada rodeada por guardias armados. Aun así, yo no apostaría mi vida por él" - Gene Spafford.

Basado en la afirmación de Gene Spafford, que no existe un sistema 100% seguro; se entiende la seguridad como un equilibrio entre los riegos asumibles y rendimiento. Así mismo, debe entenderse la seguridad como un proceso que de forma cíclica debe ir revisando y mejorando las políticas y medidas establecidas. (Hidalgo, 2014)

2.2.9 Seguridad Informática

La seguridad informática comprende la protección de software, hardware, redes de computadoras y la información como parte fundamental y activo de gran valor para una empresa. Define como las medidas y controles que garantizan la confidencialidad, integridad y disponibilidad de los activos del sistema de información, incluyendo hardware, software, firmware, así como de la información que se procesa, almacena y comunica. (Marroquín, 2013)

2.2.10 Seguridad de la Información

Es el conjunto de medidas preventivas y reactivas de las organizaciones que permiten resguardar y proteger la información del acceso, modificación, uso o destrucción no autorizados de la información. (Hidalgo, 2014)

Los principios básicos de la seguridad de la información:

1. Confidencialidad: La información solo debe ser accedida únicamente las personas autorizadas.
2. Integridad: Se debe garantizar que la información no sea modificada, reemplazada o eliminada sin autorización.
3. Disponibilidad: La información deberá estar accesible y disponible para las personas o sistemas autorizados cuando lo requieran.
4. Autenticidad/Autenticación: Que el autor de la información es quien dice ser.
5. No repudio: Evita la posibilidad por parte del autor de rechazar la autoría de la información.

Amenaza: Es todo elemento (un atacante externo, un usuario interno, una inestabilidad del sistema, etc.), que puede aprovechar una vulnerabilidad para comprometer un sistema informático.

Riesgo: Es la probabilidad de la materialización de una amenaza por medio de una vulnerabilidad.

Vulnerabilidad: Es una debilidad o ausencia de control que pone en riesgo la seguridad de un sistema.

2.2.11 Normas y principios de seguridad de información y aplicaciones

CISA

La sigla CISA corresponden a Auditor de Sistemas de Información. Esta certificación presenta ciertas similitudes con OSSTMM en el alcance, cubriendo toda la seguridad de la información de una organización. Se diferencia al tener en cuenta la visión de negocio de una organización y los diferentes

niveles de seguridad que pueden ser necesarios dependiendo de la información que debe ser salvaguardada. Es una certificación para auditores respaldada por la Asociación de Control y Auditoría de Sistemas de Información (ISACA). (López, 2015)

Es un reconocimiento para profesionales que respaldan sus conocimientos y aptitudes en áreas específicas:

- Auditoría en sistemas de información
- Gobierno y mantenimiento de tecnología de información
- Adquisición, desarrollo e implementación de sistemas de información
- Operaciones, mantenimiento y soporte de sistemas de información
- Protección de activos de información

La certificación CISA avala la experiencia, habilidades, conocimientos de un profesional en auditoría.

ISO 27001

La norma ISO 27001 proporciona un marco de trabajo para implementar y mantener un Sistema de Gestión de la Seguridad de la Información. El enfoque central de esta norma es proteger la Confidencialidad, Integridad y Disponibilidad de la información, lo cual permite a las organizaciones identificar los riesgos, evaluarlos e implementar controles para mitigarlos. (López, 2015)

La norma ISO 27001 a diferencia de OSSTMM y de OWASP no presenta casos concretos sobre los que trabajar en cambio proporciona un marco de trabajo con el que se facilita la gestión de la seguridad de la información de una organización. (López, 2015)

La ISO 27001 establece los siguientes documentos obligatorios:

- Alcance del SGSI
- Objetivos y política de seguridad de la información
- Metodología de evaluación y tratamiento de riesgos
- Declaración de aplicabilidad

- Plan de tratamiento de riesgos
- Informe de evaluación de riesgos
- Definición de roles y responsabilidades de seguridad
- Inventario de activos
- Uso aceptable de los activos
- Política de control de acceso
- Procedimientos operativos para gestión de TI
- Principios de ingeniería para sistema seguro
- Política de seguridad para proveedores
- Procedimiento para gestión de incidentes
- Procedimientos para continuidad del negocio
- Requisitos legales, normativos y contractuales

OSSTMM

El Manual de Metodología Abierta de Testeo de Seguridad (OSSTMM) es un estándar de los más completos y profesionales enfocado en la auditoria de seguridad de un sistema informático. Es un proyecto de código abierto avalado por más de 150 expertos, el mismo incluye un marco de trabajo que describe las fases de ejecución para una auditoria. (López, 2015)

Los módulos que componen a OSSTMM están representados en la Figura II- 4.

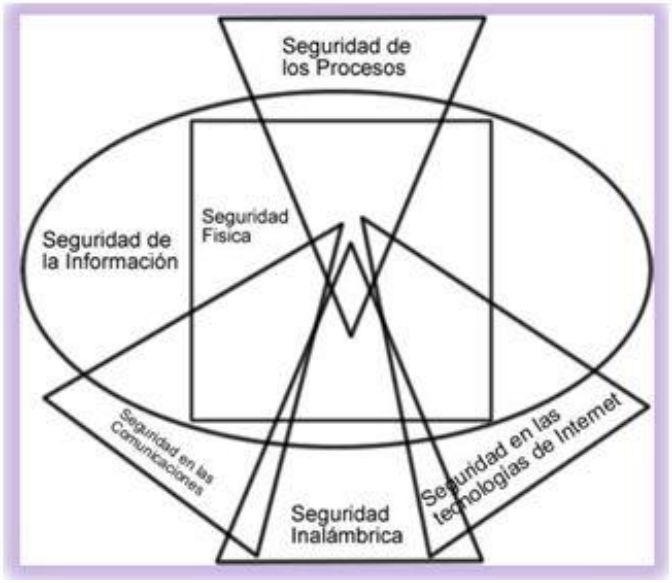


Figura 5-2. Mapa de seguridad de OSSTMM

Fuente: http://upcommons.upc.edu/handle/2099.1/24902

La metodología OSSTMM está compuesta por seis secciones:

- Seguridad de la información
- Seguridad de procesos
- Seguridad física
- Seguridad en las comunicaciones
- Seguridad inalámbrica
- Seguridad en las tecnologías de internet

OWASP

Es un proyecto abierto de seguridad de aplicaciones web, es una organización mundial sin fines de lucro que lidera varios proyectos para mejorar la seguridad del software y en aplicaciones web en particular. OWASP está conformado por empresas, organizaciones educativas y particulares de todo mundo, constituyen una comunidad de seguridad informática que trabaja para crear artículos, metodologías, documentación, herramientas y tecnologías que se proporcionan de forma gratuita al mundo. (OWASP, 2017)

OWASP Testing Guide

La guía de desarrollo de OWASP es una pauta para programadores y arquitectos de software para construir software y servicios más seguros. Esta contiene principios generales, es decir, no específicos para un lenguaje o marco de trabajo. Se enfoca en aplicar los principios básicos de la ingeniería de sistemas seguros desde las contramedidas y debilidades hasta la ingeniería de software segura a la seguridad de aplicaciones. (OWASP, 2016)

Esta guía contiene una metodología para todas las áreas que supongan vectores de ataque para una aplicación web. Es una guía indispensable para los desarrolladores de software y especialistas en seguridad de la información la cual se enfoca en la seguridad de las aplicaciones web y se complementa con las guías Devoloper y Code Review. (OWASP, 2016)

Dentro de las pruebas de seguridad de aplicaciones web tenemos:

Recopilación de información: A través de pruebas, obtienen información de la aplicación web para posteriormente atacarla con fines maliciosos.

Pruebas de gestión de configuración e implementación: Controles de seguridad en la configuración del servidor de la aplicación, la red, conexiones a la web, etc.

Pruebas de gestión de identidad: Controles de seguridad en la definición de roles, usuarios, numeración de cuentas y políticas de acceso en la aplicación/web.

Pruebas de autenticación: Controles de seguridad relacionado con el acceso a la aplicación web como las contraseñas.

Pruebas de autorización: Controles de seguridad para probar las posibles escaladas de privilegios o evasión de métodos de autorización en el portal.

Pruebas de gestión de sesiones: Controles de seguridad relacionados con las cookies y tiempos de sesión.

Prueba de validación de entrada: Controles de seguridad en todas las entradas de información que disponga la aplicación/web (formularios de contacto, búsquedas, registros de información).

Prueba de manejo de errores: Seguridad y correcto funcionamiento de los mensajes de error que muestra la aplicación.

Prueba de criptografía débil: Controles para probar la seguridad y robustez de los mecanismos de cifrado que implemente la aplicación/web.

Pruebas de lógica empresarial: Controles de comportamiento de la aplicación frente a usos inesperados que pueden ocasionar brechas de seguridad.

Pruebas del lado del cliente: Todas las pruebas de seguridad realizadas desde el punto de vista del usuario.

OWASP Development

La Guía de prácticas de desarrollo proporciona una orientación utilizando ejemplos de código J2EE, ASP.NET y PHP. La Guía de desarrollo está dirigida a una amplia gama de problemas de seguridad de aplicaciones web y servicios, como inyección SQL, phishing, manejo de sesiones, tarjetas de crédito, fijación de sesiones, falsificaciones de solicitudes entre sitios, cumplimiento y problemas de privacidad. (OWASP Development, 2016)

OWASP Code Review

La Guía de revisión de código de OWASP es un libro técnico escrito para los responsables de las revisiones de código (administración, desarrolladores, profesionales de seguridad). El enfoque principal de este libro se ha dividido en dos secciones principales. La sección uno es por qué y cómo hacer las revisiones de código y la sección dos está dedicada a las vulnerabilidades que deben buscarse durante una revisión manual de código. Si bien los escáneres de seguridad están mejorando cada día, la necesidad de revisiones manuales del código de seguridad aún debe ocupar un lugar destacado en

las organizaciones SDLC (ciclo de vida de desarrollo seguro) que desea un buen código seguro en la producción. (OWASP Code Review, 2017)

La segunda sección trata sobre vulnerabilidades. Se basa en el top 10 de OWASP 2013. Aquí se encuentra la mayoría de los ejemplos de código para saber qué no hacer y qué hacer, con una orientación práctica utilizando ejemplos de J2EE, ASP.NET, PHP. (OWASP Code Review, 2017)

OWASP Top 10 - 2017

Es un documento en el que constan los riesgos más críticos de seguridad que afectan a las aplicaciones web. Este proyecto tenía como objetivo inicialmente concientizar a desarrolladores y usuarios, sin embargo, se ha convertido en un estándar de uso imperativo. (OWASP, 2017)

Los diez riesgos más importantes que amenazan la seguridad de las aplicaciones web:

A1:2017 - Inyección

Los problemas de inyección LDAP, SQL, NoSQL y OS suceden cuando datos corruptos con código malicioso son enviados como comando o consulta a una aplicación. El objetivo de este tipo de ataque es burlar al intérprete con el fin de que ejecute comandos involuntarios y así acceder a datos sin autorización. (OWASP, 2017)

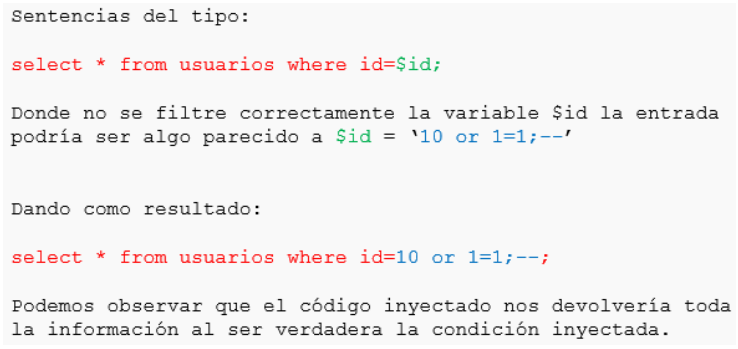


Figura 6-2. Descripción Inyección SQL

Fuente: https://www.seguridad.unam.mx/historico/documento/index.html-id=17

Como prevenir

Para prevenir ataques de inyección, es necesario separar los datos de los comandos y las consultas.

- Implementar validaciones de las entradas de datos en el lado del servidor, haciendo uso listas blancas. Existen muchas aplicaciones que requieren el uso de caracteres especiales, como en APIs o aplicaciones móviles, por lo cual, esto no es una defensa perfecta.
- El hacer uso de una API segura, para evitar el uso de intérpretes y proveer una interfaz parametrizada. Migrar y utilizar una herramienta de Mapeo Relacional de Objetos (ORMs).
- Nota: A pesar de parametrizar, los procedimientos almacenados pueden insertar una inyección SQL cuando el procedimiento T-SQL o PL/SQL concatena consultas o se ejecutan parámetros con EXECUTE IMMEDIATE o exec().
- Para prevenir la fuga masiva de registros en el caso de un ataque de inyección SQL, utilizar LIMIT y controles SQL.
- Escapar caracteres especiales utilizando la sintaxis de caracteres específica, para consultas dinámicas residuales.

A2:2017 - Pérdida de Autenticación

Cuando las funciones de una aplicación de autenticación y gestión de sesiones son implementadas incorrectamente, permiten al atacante comprometer credenciales, token de sesiones o explotar cualquier otra falla de implementación existente para apoderarse de la identidad de usuarios de forma temporal o permanente. (OWASP, 2017)

Como prevenir

- Implementar controles para prevenir el uso de contraseñas débiles
- Para evitar ataques automatizados, reutilización de credenciales robadas o ataques de fuerza bruta, implementar autenticación multi-factor
- No utilizar credenciales por defecto, especialmente en usuarios con privilegios elevados como administradores

- Implementar una política de longitud, complejidad y rotación de contraseñas con las recomendaciones para Secretos Memorizados de la Guía NIST 800-63 B's u otras de contraseñas actuales.
- Implementar mensajes de salida genéricos, verificar que el registro, recuperación de credenciales y el uso de APIs, no permitan ataques de enumeración de usuarios.
- Limitar o incrementar el tiempo de respuesta de cada error de inicio de sesión, registrar todos los errores y notificar ataques de fuerza bruta a los administradores.
- Utilizar un gestor de sesión en el servidor, integrado, seguro y que genere un nuevo ID de sesión aleatorio con alta entropía después del inicio de sesión. El Session-ID no debe ser incluido en la URL, este debe ser almacenado de forma segura e invalidado después del cierre de sesión o de un tiempo de inactividad.

A3:2017 - Exposición de Datos Sensibles

Una gran cantidad de aplicaciones web y APIs no protegen apropiadamente los datos sensibles que almacenan, como información financiera o Información Personalmente Identificable (PII), etc. Los datos sensibles requieren métodos de protección adicionales, como el cifrado en almacenamiento y tránsito. Los atacantes pueden robar o alterar datos para cometer delitos. (OWASP, 2017)

Como prevenir

Como mínimo, se recomienda implementar lo siguiente:

- Clasificar los datos procesados, almacenados o transmitidos por el sistema e identificar la información sensible de acuerdo a las leyes o requisitos del negocio y del país.
- Emplear los debidos controles para cada clasificación.
- No almacenar datos sensibles innecesariamente. Descartarlos tan pronto como sea posible o utilizar un sistema de tokenización que cumpla con PCI DSS. Los datos que no son almacenados no pueden ser robados.
- Cifrar los datos sensibles cuando sean almacenados.

- Cifrar los datos en tránsito empleando protocolos seguros como TLS con cifradores que utilicen Perfect Forward Secrecy (PFS), priorizando los algoritmos en el servidor. Aplicar el cifrado utilizando directivas como HTTP Strict Transport Security (HSTS).
- Implementar una gestión adecuada de contraseñas utilizando exclusivamente algoritmos y protocolos estándares fuertes.
- Comprobar de forma independiente la efectividad de configuraciones y parámetros.
- Utilizar funciones de hashing adaptables con un factor de trabajo además de SALT, como scrypt, bcrypt, Argon2 para almacenar contraseñas.
- Deshabilitar el almacenamiento en cache de todos los datos sensibles.

A4:2017 - Entidades Externas XML (XXE)

Muchos procesadores XML anticuados permiten la especificación de una entidad externa en documentos XML. Las entidades externas pueden ser utilizadas para encontrar archivos internos mediante la URI o en servidores no actualizados, escanear puertos de la red de área local, ejecutar código arbitrario de forma remota y ejecutar ataques de denegación de servicio (DoS). (OWASP, 2017)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
  <!DOCTYPE foo [
  <!ELEMENT foo ANY>
  <!ENTITY xxe SYSTEM "file:///etc/passwd">]>
<foo>&xxe;</foo>
```

Figura 7-2. Descripción de entidad XXE maliciosa

Fuente: https://www.seguridad.unam.mx/historico/documento/index.html-id=17

Como prevenir

Para identificar y mitigar defectos de XXE es importante la experiencia del programador es esencial y además de los puntos siguientes:

- Evitar la serialización de datos confidenciales y usar formatos de datos menos complejos como JSON.
- Actualizar procesadores y bibliotecas XML que utilice la aplicación o el sistema subyacente. Utilizar validadores de dependencias y mantener actualizado SOAP.
- Deshabilitar las entidades externas de XML y procesamiento DTD en todos los analizadores sintácticos XML en su aplicación, según la hoja de trucos para prevención de XXE de OWASP.
- Las herramientas SAST pueden ayudar a revelar XXE en el código fuente, sin embargo, la mejor opción es la revisión manual de código para aplicaciones grandes y complejas.
- Implementar validación de entrada positiva en el servidor, filtrado y sanitización para evitar el ingreso de datos perjudiciales en cabeceras, documentos y nodos XML.
- Verificar que la funcionalidad de carga de archivos XML o XSL validar el XML entrante, usando validación XSD o similar.
- Si estos controles no son posibles, es recomendable emplear parcheo virtual, gateways de seguridad de API, o Firewalls de Aplicaciones Web (WAFs) para detectar, monitorear y bloquear ataques de tipo XXE.

A5:2017 - Pérdida de Control de Acceso

Las restricciones no son aplicadas adecuadamente sobre las funcionalidades que les corresponden a los usuarios autorizados. Los atacantes pueden encontrar y aprovechar estas falencias para acceder, de sin la autorización, a funcionalidades y/o datos, otras cuentas de usuarios, acceder a archivos restringidos, alterar datos, cambiar autorizaciones de acceso, etc. (OWASP, 2017)

Como prevenir

El control de acceso sólo es efectivo si es aplicado del lado del servidor o en Server-less API, donde el atacante no puede modificar la verificación de control de acceso o los metadatos.

- De forma predeterminada a excepción de los recursos públicos, la política debe ser denegar el acceso a los recursos.
- Los tokens JWT deben ser invalidados luego de la finalización de cada sesión o un tiempo determinado.
- Utilizar los mecanismos de control de acceso en toda la aplicación, incluyendo minimizar el control de acceso HTTP (CORS).
- Los controles de acceso al modelo deben obligar la propiedad (propietario) de los registros, en lugar de admitir que el usuario pueda crear, leer, actualizar o eliminar cualquier registro.
- Los modelos de dominio deben hacer cumplir los requisitos exclusivos de los límites de negocio de las aplicaciones.
- Verificar que los metadatos/fuentes de archivos y copia de seguridad no se encuentren en las carpetas públicas y deshabilitar el listado de directorios del servidor web.
- Limitar la tasa de acceso a las APIs para minimizar el daño de ataques con herramientas automatizadas.
- El personal de calidad y programación deben contener pruebas de control de acceso en las pruebas unitarias y de integración.

A6:2017 - Configuración de Seguridad Incorrecta

El problema de configuración de seguridad incorrecta es muy común y en parte se debe a la configuración manual, ad hoc o por omisión o por falta de configuración. Ejemplo: cabeceras HTTP mal configuradas, mensajes de error con contenido sensible, falta de actualizaciones en componentes software, frameworks, parches, etc. (OWASP, 2017)

Como prevenir

Adoptar procesos seguros de instalación, tales como:

- Un proceso de fortalecimiento reproducible que facilite la implementación de otro entorno asegurado. Los entornos de desarrollo, control de calidad y producción deben configurarse de manera idéntica per con diferentes credenciales para cada uno.
- Eliminar o no instalar frameworks y funcionalidades innecesarias, mantener una plataforma minimalista.
- Remitir a los clientes directivas de seguridad como cabeceras de seguridad.
- Seguir un proceso de gestión de parches, adoptar un proceso para revisar y actualizar las configuraciones adecuadas de acuerdo a advertencias de seguridad. Revisar especialmente los permisos de almacenamiento en la nube.
- Tener una arquitectura segmentada que proporcione una separación segura y efectiva entre componentes y acceso a terceros, contenedores o grupos de seguridad en la nube (ACLs).
- Emplear un proceso automatizado para verificar la efectividad de las configuraciones en los ambientes existentes.

A7:2017 - Cross-Site Scripting (XSS)

Los ataques de XSS suceden cuando una aplicación envía datos no confiables al navegador web, sin validaciones y codificación adecuada o actualiza una página web existente con datos proveídos por medio de una API que ejecuta JavaScript en el navegador. Esto permiten que un atacante ejecute comandos en el navegador de la víctima para secuestrar una sesión, modificar sitios web, o redireccionar al usuario hacia un sitio malicioso. (OWASP, 2017)

Tipo	Nombre	Descripción
Tipo 0	Ataque basado en el DOM o local	Si un código de JavaScript accede a una URL como un parámetro de una petición al servidor y utiliza esta información para escribir HTML en la misma página sin ser codificada empleando entidades HTML
Tipo 1	Ataque no persistente o reflejado	Si los datos no validados por el usuario son incluidos en la página resultante sin codificación HTML, se le permite al cliente inyectar código en la página dinámica.
Tipo 2	Ataque persistente o almacenado	La información proporcionada por el usuario es almacenada en la base de datos, en el sistema de archivos o algún otro lugar; después es mostrada a otros usuarios que visiten la página

Figura 8-2. Definición general de los tipos de XSS

Fuente: https://www.seguridad.unam.mx/historico/documento/index.html-id=17

Como prevenir

Mantener los datos no confiables separados del contenido del navegador para prevenir ataques XSS.

- Implementar frameworks seguros y confiables, existen frameworks que por diseño, automáticamente codifican el contenido para prevenir XSS, como React JS.
- Codificar los datos de requerimientos HTTP no confiables en los campos de salida HTML (cuerpo, atributos, JavaScript, CSS, o URL) resuelve los XSS Reflejado y XSS Almacenado.
- Emplear codificación sensitiva al contexto, cuando se modifica el documento en el navegador, ayuda a prevenir DOM XSS. Cuando esta técnica no se puede aplicar, se pueden usar otras técnicas similares de codificación, para prevenir XSS DOM como explica la hoja de trucos OWASP.
- Aplicar una Política de Seguridad para la mitigación de vulnerabilidades XSS, asumiendo que no existen otras vulnerabilidades que permitan introducir código malicioso vía inclusión de archivos locales, bibliotecas vulnerables en fuentes conocidas almacenadas de forma local o en Redes de Distribución de Contenidos.

A8:2017 - Deserialización Insegura

Estas fallas ocurren cuando una aplicación recibe objetos serializados dañinos y estos objetos pueden ser manipulados o borrados por un atacante para llevar a cabo ataques de repetición, inyecciones o incluso escalar privilegios de ejecución. En el peor escenario, la deserialización insegura puede llevar a la ejecución remota de código en el servidor. (OWASP, 2017)

Como prevenir

El único patrón seguro de arquitectura es no aceptar objetos serializados de fuentes no confiables o utilizar medios de serialización que únicamente permitan tipos de datos primitivos. Si esto no es posible, considerar alguno de los siguientes puntos:

- Aplicar verificaciones de integridad como firmas digitales en cualquier objeto serializado, para detectar modificaciones que no hayan sido autorizadas.
- Exigir el cumplimiento de verificaciones de tipo de dato, antes y durante la deserialización y antes de la creación del objeto. Se puede pasar por alto esta técnica, por lo que no es se recomienda confiar sólo en ella.
- Aislar el código para la deserialización, para que se ejecute en un entorno con los mínimos privilegios posibles.
- Monitorear los procesos de deserialización, alertando la deserialización constante de un cliente.
- Registrar las excepciones y errores en la deserialización, como cuando el tipo recibido no es el esperado.
- Restringir y monitorear las conexiones de entrada y salida de red desde contenedores o servidores que utilizan funcionalidades de deserialización.

A9:2017 - Uso de Componentes con Vulnerabilidades Conocidas

Si un componente vulnerable es explotado, el ataque puede provocar una pérdida de datos e incluso tomar el control del servidor. Los componentes como bibliotecas, frameworks y otros módulos se ejecutan con los mismos privilegios que la aplicación. Las aplicaciones y API con componentes con

vulnerabilidades conocidas pueden debilitar las defensas de las aplicaciones y permitir varios ataques e impactos. (OWASP, 2017)

Como prevenir

Remover dependencias, componentes, archivos y documentación innecesaria.

- Monitorear de forma permanente fuentes tales como CVE y NVD para detectar vulnerabilidades en los componentes utilizados en la aplicación. Para facilitar la búsqueda se puede usar herramientas de análisis de vulnerabilidades.
- Mediante una herramienta tener un inventario de versiones de componentes como tanto del cliente como del servidor.
- Los componentes únicamente adquirirlos de fuentes oficiales y a través de canales seguros. Con el fin de reducir la probabilidad del uso de versiones manipuladas, utilizar paquetes firmados.
- Supervisar bibliotecas y componentes que no poseen mantenimiento o no liberan parches de seguridad para sus versiones obsoletas o sin soporte. Si no es posible el parcheo, considerar desplegar un parche virtual para monitorizar, detectar o protegerse contra la debilidad detectada.

Cada organización debe poseer un plan para monitorizar, evaluar y aplicar actualizaciones o cambios de configuraciones durante el ciclo de vida de las aplicaciones. (OWASP, 2017)

A10:2017 - Registro y Monitoreo Insuficientes

El registro, monitoreo insuficiente y la falta de respuesta ante incidentes permiten a los atacantes mantener el ataque en el tiempo, además, pivotear a otros sistemas y manipular, extraer o eliminar datos. El tiempo de detección de una brecha de seguridad es mayor a 200 días, siendo frecuentemente detectado por terceros. (OWASP, 2017)

Como prevenir

En dependencia del riesgo de los datos almacenados o procesados:

- Toda transacción de alto valor debe tener una traza de auditoría con controles de integridad con el fin de detectar alguna modificación o la eliminación.
- Las transacciones de alto impacto deben tener una pista de auditoría con controles de integridad para prevenir alteraciones o eliminaciones.
- Establecer un plan de respuesta y/o recuperación de incidentes.
- Implementar una monitorización y alerta efectivos de tal manera que las actividades sospechosas sean detectadas y respondidas dentro de períodos de tiempo aceptables.
- Comprobar el registro de todos los errores de inicio de sesión, control de acceso y validación de entradas de datos del lado del servidor para poder identificar cuentas sospechosas y mantener los registros durante el tiempo suficiente para permitir un análisis forense.

Existen frameworks de protección de aplicaciones comerciales y de código abierto como OWASP AppSensor, firewalls de aplicaciones web como ModSecurity utilizando el Core Rule Set de OWASP, y software de correlación de registros con paneles personalizados y alertas. (OWASP, 2017)

OWASP API Security Top 10

"Un elemento fundamental de la innovación en el mundo actual basado en aplicaciones es la API. Desde bancos, comercio minorista y transporte hasta IoT, vehículos autónomos y ciudades inteligentes, las API son una parte fundamental de las aplicaciones móviles, SaaS y web modernas y se pueden encontrar en aplicaciones internas orientadas al cliente, orientadas a los socios. Por naturaleza, las API exponen la lógica de la aplicación y los datos confidenciales, como la información de identificación personal (PII) y, debido a esto, se han convertido cada vez más en un objetivo para los atacantes. Sin API seguras, la innovación rápida sería imposible." (OWASP, 2019).

API1:2019 - Control de Acceso a Nivel de Objeto Roto

Las API tienden a exponer puntos finales que manejan identificadores de objetos, creando un problema de control de acceso de nivel de superficie de ataque amplio. Las comprobaciones de autorización a nivel de objeto deben considerarse en cada función que accede a una fuente de datos utilizando la entrada del usuario. (OWASP, 2019)

Figura 9-2. Control de acceso de objeto roto

Fuente: https://www.slideshare.net/QAFest/qa-fest-2019-api

Como prevenir

- Implementar un mecanismo de autorización adecuado que se base en las políticas y la jerarquía del usuario.
- De preferencia no usar un ID que haya sido enviada desde el cliente, sino un ID que esté almacenado en el objeto de sesión cuando acceda a un registro de base de datos mediante el ID de registro.
- Utilizar un mecanismo de autorización para verificar si el usuario conectado tiene acceso para realizar la acción solicitada en el registro en cada función que utiliza una entrada del cliente para acceder a un registro en el base de datos.
- De preferencia utilizar valores aleatorios e impredecibles como GUID para las ID de los registros.
- Escribir pruebas para evaluar el mecanismo de autorización. No implementar cambios vulnerables que rompan las pruebas.

API2:2019 - Autenticación Rota

Los mecanismos de autenticación a menudo se implementan incorrectamente, lo que permite a los atacantes comprometer los tokens de autenticación o explotar fallas de implementación para asumir las identidades de otros usuarios de temporal o permanentemente. La capacidad comprometida del sistema para identificar al cliente / usuario compromete la seguridad general de la API. (OWASP, 2019)

Figura 10-2. Perdida de autenticación

Fuente: https://www.slideshare.net/QAFest/qa-fest-2019-api

Como prevenir

- Conocer todos los flujos posibles para autenticarse en la API (enlaces móviles / web / profundos que implementan la autenticación con un clic / etc.)
- Preguntar a los ingenieros qué flujos perdió.
- Leer sobre los mecanismos de autenticación. Entender qué y cómo se usan. OAuth no es autenticación, ni tampoco claves API.
- No reinventar la rueda en autenticación, generación de tokens, almacenamiento de contraseña.
- Los puntos finales de recuperación de credenciales / olvidar contraseña deben tratarse como puntos finales de inicio de sesión en términos de fuerza bruta, limitación de velocidad y protecciones de bloqueo.
- Usar la hoja de referencia de autenticación OWASP
- Donde sea posible, implementar la autenticación multifactor.
- Implementar mecanismos para mitigar el relleno de credenciales, ataques de diccionario y de fuerza bruta en sus puntos finales de autenticación.
- Implementar un mecanismo de bloqueo / captura de cuenta para evitar la fuerza bruta contra usuarios específicos.

- Implementar verificaciones de contraseña débil.
- Las claves API no deben usarse para la autenticación de usuarios, sino para la autenticación de la aplicación / proyecto del cliente.

API3:2019 - Filtrado de datos incorrecto

Al esperar implementaciones genéricas, los desarrolladores tienden a exponer todas las propiedades de los objetos sin tener en cuenta su sensibilidad individual, confiando en que los clientes realicen el filtrado de datos antes de mostrarlo al usuario. Sin controlar el estado del cliente, los servidores reciben cada vez más filtros que se pueden abusar para obtener acceso a datos confidenciales. (OWASP, 2019)

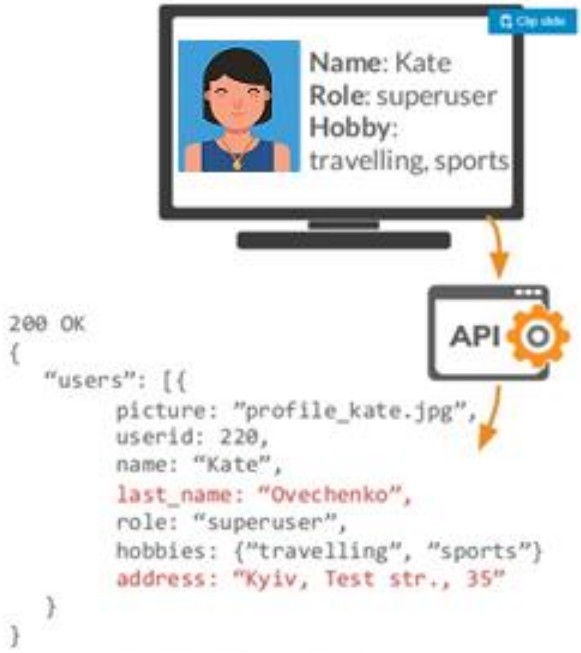


Figura 11-2. Exposición de datos

Fuente: https://www.slideshare.net/QAFest/qa-fest-2019-api

Como prevenir

- Nunca confiar en el lado del cliente para realizar el filtrado de datos confidenciales.
- Revisar las respuestas de la API para asegurarse de que solo contengan datos legítimos.
- Definir y aplicar explícitamente los datos devueltos por todos los métodos API, incluidos los errores. Siempre que sea posible: usar esquemas para respuestas, patrones para todas las cadenas y nombres de campo claros.
- Definir toda la información sensible y de identificación personal (PII) que la aplicación almacena y trabaja y revisar todas las llamadas a la API que devuelven dicha información para ver si estas respuestas pueden ser un problema de seguridad.

API4:2019 - Falta de Recursos y Limitación de Velocidad

Muy a menudo, las API no imponen ninguna restricción sobre el tamaño o la cantidad de recursos que puede solicitar el cliente / usuario. Esto no solo puede afectar el rendimiento del servidor API, lo que lleva a la denegación de servicio (DoS), sino que también deja la puerta abierta a fallas de autenticación como la fuerza bruta. (OWASP, 2019)

Como prevenir

- Docker facilita la limitación de memoria, CPU, número de reinicios, descriptores de archivos y procesos.
- Implementar un límite sobre la frecuencia con la que un cliente puede llamar a la API dentro de un período de tiempo definido.
- Para operaciones confidenciales como inicio de sesión o restablecimiento de contraseña, considerar los límites de velocidad por método API (por ejemplo, autenticación), cliente (por ejemplo, dirección IP), propiedad (por ejemplo, nombre de usuario).
- Notificar al cliente cuando se excede el límite proporcionando el número de límite y la hora a la que se restablecerá el límite.

- Agregar la validación adecuada del lado del servidor para la cadena de consulta y los parámetros del cuerpo de la solicitud, específicamente el que controla el número de registros que se devolverán en la respuesta.
- Definir y aplicar el tamaño máximo de datos en todos los parámetros entrantes y cargas útiles, como la longitud máxima de las cadenas y el número máximo de elementos en las matrices.
- Si un API acepta archivos comprimidos, verificar las relaciones de compresión antes de expandir los archivos para protegerse contra las "bombas zip".

API5:2019 - Falta de Control de Acceso a Nivel Función/Recurso

Las políticas complejas de control de acceso con diferentes jerarquías, grupos y roles, y una separación poco clara entre las funciones administrativas y regulares, tienden a generar fallas de autorización. Al explotar estos problemas, los atacantes obtienen acceso a los recursos y / o funciones administrativas de otros usuarios. (OWASP, 2019)

Figura 12-2. Control de acceso sobre recursos

Fuente: https://www.slideshare.net/QAFest/qa-fest-2019-api

Como prevenir

Una aplicación debe tener un módulo de autorización consistente y fácil de analizar que se invoque desde todas sus funciones comerciales. Con frecuencia, dicha protección es proporcionada por uno o más componentes externos al código de la aplicación.

- El (los) mecanismo (s) de aplicación debería denegar todo el acceso por defecto, requiriendo concesiones explícitas a roles específicos para acceder a cada función.
- Revisar los puntos finales de API en relación con fallas de autorización de nivel de función, teniendo en cuenta la lógica empresarial de la aplicación y la jerarquía de grupos.
- Verificar que todos sus controladores administrativos hereden de un controlador abstracto administrativo que implemente comprobaciones de autorización basadas en el grupo / rol del usuario.
- Verificar que las funciones administrativas dentro de un controlador regular implementen comprobaciones de autorización basadas en el grupo y el rol del usuario. (OWASP, 2019)

API6:2019 - Asignación Masiva

La vinculación de datos proporcionados por el cliente (ejemplo, JSON) a modelos de datos, sin un filtrado de propiedades adecuado basado en una lista blanca, generalmente conduce a la asignación masiva. Adivinar las propiedades de los objetos, explorar otros puntos finales de la API, leer la documentación o proporcionar propiedades de objetos adicionales en las cargas útiles de solicitud, permite a los atacantes modificar las propiedades de los objetos que no deberían hacerlo. (OWASP, 2019)

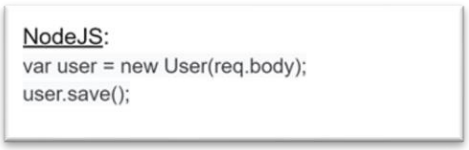


Figura 13-2. Asignación masiva

Fuente: https://www.slideshare.net/QAFest/qa-fest-2019-api

Como prevenir

Si es posible, evitar utilizar funciones que enlacen automáticamente la entrada de un cliente en variables de código u objetos internos.

- Incluir en la lista blanca solo las propiedades que el cliente debe actualizar.
- Usar las funciones integradas para incluir en la lista negra las propiedades a las que los clientes no deberían acceder.
- Si corresponde, definir explícitamente y aplicar esquemas para las cargas útiles de datos de entrada. (OWASP, 2019)

API7:2019 – Configuración de Seguridad Incorrecta

La configuración incorrecta de seguridad suele ser el resultado de configuraciones predeterminadas inseguras, configuraciones incompletas o ad-hoc, almacenamiento en la nube abierta, encabezados HTTP mal configurados, métodos HTTP innecesarios, uso compartido de recursos de origen cruzado (CORS) permisivo y mensajes de error detallados que contienen información confidencial. (OWASP, 2019)

Como prevenir

El ciclo de vida de la API debe incluir:

- Un proceso de endurecimiento repetible que conduce a la implementación rápida y fácil de un entorno bloqueado adecuadamente.
- Una tarea para revisar y actualizar configuraciones en toda la pila de API. La revisión debe incluir: archivos de orquestación, componentes de API y servicios en la nube (por ejemplo, permisos de depósito S3).
- Un canal de comunicación seguro para todas las interacciones API acceso a activos estáticos (por ejemplo, imágenes).

- Un proceso automatizado para evaluar continuamente la efectividad de la configuración y la configuración en todos los entornos.
- Para evitar que se envíen rastros de excepción y otra información valiosa a los atacantes, si corresponde, definir y aplicar todos los esquemas de carga útil de respuesta API, incluidas las respuestas de error.

API8:2019 – Inyección

Los defectos de inyección, como SQL, NoSQL, Inyección de comandos, etc., ocurren cuando se envían datos no confiables a un intérprete como parte de un comando o consulta. Los datos maliciosos del atacante pueden engañar al intérprete para que ejecute comandos no deseados o acceda a los datos sin la autorización adecuada. (OWASP, 2019)

Figura 14-2. Inyección

Fuente: https://www.slideshare.net/QAFest/qa-fest-2019-api

Como prevenir

La prevención de la inyección requiere mantener los datos separados de los comandos y consultas.

- Realizar la validación de datos utilizando una biblioteca única, confiable y mantenida activamente.
- Validar, filtrar y desinfectar todos los datos proporcionados por el cliente u otros datos procedentes de sistemas integrados.

- Los caracteres especiales se deben escapar utilizando la sintaxis específica para el intérprete de destino.
- Elegir una API segura que proporcione una interfaz parametrizada.
- Siempre limitar el número de registros devueltos para evitar la divulgación masiva en caso de inyección.
- Validar los datos entrantes utilizando filtros suficientes para permitir solo valores válidos para cada parámetro de entrada.
- Para evitar fugas de datos, definir y aplicar esquemas para todas las respuestas API.
- Definir tipos de datos y patrones estrictos para todos los parámetros de cadena.

API9:2019 - Gestión Inadecuada de Activos

Las API tienden a exponer más puntos finales que las aplicaciones web tradicionales, lo que hace que la documentación adecuada y actualizada sea muy importante. Los hosts adecuados y el inventario de versiones de API implementadas también juegan un papel importante para mitigar problemas como versiones de API obsoletas y puntos finales de depuración expuestos. (OWASP, 2019)

Como prevenir

- Tener inventario de todos los hosts API y documentar los aspectos importantes de cada uno de ellos, enfocándose en el entorno API (por ejemplo, producción, prueba, desarrollo), que debe tener acceso de red al host (p. ej., público, interno, socios) y la versión API.
- Tener un inventario de los servicios integrados y documentar aspectos importantes como su papel en el sistema, qué datos se intercambian (flujo de datos) y su sensibilidad.
- Documentar todos los aspectos de su API, tales como autenticación, errores, redireccionamientos, limitación de velocidad, política de intercambio de recursos de origen cruzado (CORS) y puntos finales, incluidos sus parámetros, solicitudes y respuestas.
- Generar documentación automáticamente mediante la adopción de estándares abiertos. Incluir la compilación de documentación en su canalización de CI / CD.
- Hacer que la documentación de la API esté disponible para aquellos autorizados a usar la API.

- Usar medidas de protección externas como los firewalls de seguridad de API para todas las versiones expuestas de las API, no solo para la versión de producción.
- Evitar usar datos de producción con implementaciones de API que no sean de producción. Si esto es inevitable, estos puntos finales deberían recibir el mismo tratamiento de seguridad que los de producción.
- Cuando las versiones más recientes de las API incluyen mejoras de seguridad, realizar un análisis de riesgos para tomar la decisión de las acciones de mitigación requeridas para la versión anterior: por ejemplo, si es posible
- Respaldar las mejoras sin romper la compatibilidad de la API o sacar la versión anterior rápidamente y obligar a todos los clientes a pasar a la última versión.

API10:2019 - Registro y Monitoreo Insuficientes

El registro y la supervisión insuficientes, junto con la integración faltante o ineficaz con la respuesta a incidentes, permite a los atacantes atacar aún más los sistemas, mantener la persistencia, pasar a más sistemas para manipular, extraer o destruir datos. La mayoría de los estudios de incumplimiento demuestran que el tiempo para detectar un incumplimiento es superior a 200 días, generalmente detectado por partes externas en lugar de procesos internos o monitoreo. (OWASP, 2019)

Como prevenir

- Registrar todos los intentos fallidos de autenticación, acceso denegado y errores de validación de entrada.
- Los registros deben escribirse usando un formato adecuado para ser consumido por una solución de administración de registros, y deben incluir suficientes detalles para identificar al actor malicioso.
- Los registros deben manejarse como datos confidenciales, y su integridad debe garantizarse en reposo y tránsito.
- Configurar un sistema de monitoreo para monitorear continuamente la infraestructura, la red y el funcionamiento de la API.

- Utilizar un sistema de información de seguridad y gestión de eventos (SIEM) para agregar y administrar registros de todos los componentes de la pila API y los hosts.
- Configurar paneles personalizados y alertas, permitiendo detectar actividades sospechosas y responderlas antes.

2.2.12 Estudio de principios, normas y estándares

En la Tabla 1-2 se encuentra la descripción de las normas y proyectos de seguridad para comparar y seleccionar las normas que van a ser utilizadas como herramienta de apoyo para el presente trabajo de investigación.

Tabla 1-2. Descripción de normas, estándares y proyectos de seguridad

Norma	Descripción	Dirigido a
CISA	Es un reconocimiento para profesionales que respaldan sus conocimientos y aptitudes en áreas específicas	Auditores
ISO 27001	Proporcionar un marco de trabajo para implementar y mantener un Sistema de Gestión de la Seguridad de la Información	Seguridad de la información
OSSTMM	Es un estándar de los más completos y profesionales enfocado en la auditoria de seguridad de un sistema informático	Sistemas Informáticos
OWASP Testing Guide	Metodología para que contiene pruebas para todas las áreas que supongan vectores de ataque para una aplicación web	Aplicaciones web
OWASP Top 10	Es documento en el que constan los riesgos más críticos de seguridad que afectan a las aplicaciones web	Aplicaciones web
OWASP API Security Top 10	Proporciona estrategias y soluciones para comprender y mitigar las vulnerabilidades y los riesgos de seguridad de las interfaces de programación de aplicaciones	API

Realizado por: Alejandra Lara

Según el análisis de las normas y/o estándares para la seguridad relacionadas al presente proyecto se han seleccionado los proyectos de forma total o parcial con el objetivo de que aporten en la construcción o validación de la Interfaz de Programación de Aplicaciones, en este caso: OWASP Testing Guide, OWASP Top 10 y OWASP API Security Top 10, serán la pauta para la incorporación de mecanismos y herramientas para la mitigación de las vulnerabilidades críticas especificadas.

2.2.13 Consideraciones para construir software seguro

Es importante ser consciente de que no todo proyecto es igual, ya sea, por el modelo del negocio, la información que almacena, regulaciones y políticas del país, entre otros aspectos.

El presente trabajo ha sido construido en base a proyectos y normas internacionales el cual, a su vez, se basan en la colección de conocimiento de especialistas en seguridad y desarrollo de software, pero al existir negocios diferentes, con necesidades diferentes, se listan algunas consideraciones adicionales a las de desarrollo y otras que deben ser analizadas por cada proyecto individual para ser aplicadas de acuerdo a las necesidades y realidad de cada proyecto.

SDLC

EL OWASP Testing Guide recomienda incorporar procesos de Seguridad en cada etapa del Ciclo de Vida de Desarrollo de Software (SDLC), esto ayudará a alcanzar el producto deseado. El detectar errores dentro del SDLC y corregirlos de manera temprana puede hacer una gran diferencia de costos en la solución.

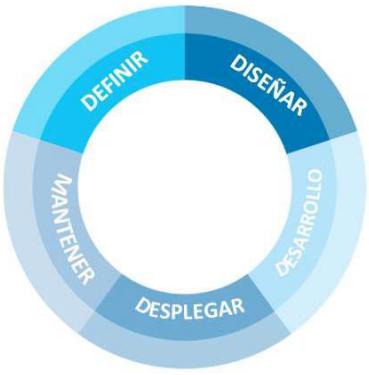


Figura 15-2. Modelo SDLC genérico

Fuente: https://www.dragonjar.org/owasp-testing-guide-4-0-en-espanol.xhtml

Definir el ámbito

Se debe definir el nivel de seguridad que requiere el proyecto, comprender la relevancia de los activos para definir el ámbito de los mismos.

En el caso de aplicaciones para instituciones financieras requiere que implementen aplicaciones que mitiguen los riesgos de autenticación débil con autenticación de múltiples etapas y control de seguridad multifactorial. Si, la aplicación trata con información personal identificable (PII) y datos sensibles, se debe cifrar la información de dichos datos tanto en tránsito como en almacenamiento.

Establecer requisitos funcionales como requisitos de seguridad, ejemplo:

a) Las contraseñas deben contener al menos 8 caracteres compuestas: números, letras y caracteres especiales.
b) Las cuentas deben bloquearse al *n* intento fallido.
c) Los mensajes de error deben ser genéricos

Estos requisitos es recomendable hacerlos parte de las políticas de seguridad del desarrollo de software de la organización.

Pruebas

Las pruebas de software deben ir más allá de las pruebas de funcionalidad, en las que solo se prueba el comportamiento deseado de la aplicación. Lo ideal, es que la aplicación sea sometida a pruebas de con situaciones no esperadas y maliciosas, pensar como un atacante intentaría saltar la seguridad. Sin embargo, la seguridad es más compleja que probar cuestiones técnicas. La guía de pruebas de OWASP menciona que un programa efectivo de pruebas debe tener componentes que prueban:

Personas. – para asegurar que existe una educación adecuada y consciente;
Proceso. – para asegurar que hay criterios y políticas adecuadas y que las personas sepan cómo seguir estas políticas;
Tecnología. – para garantizar que el proceso ha sido eficaz en su implementación

Por lo tanto, probar realmente un sistema requiere una evaluación de seguridad más completa.

En la construcción de software se debe incluir casos de uso y de uso correcto y de uso indebido, es decir; escenarios de uso no deseado y malicioso. Desarrollar escenarios realistas de amenazas para determinar las vulnerabilidades para clasificarlas y así, tomar la decisión de mitigarlas, aceptarlas o transferir a terceros como una empresa de seguros.

Es importante mencionar que las pruebas deben estar presente durante el desarrollo, despliegue y mantenimiento. Es esencial utilizar un método para probar todas las vulnerabilidades conocidas y documentar todas las actividades de pruebas de seguridad.

Herramientas de seguridad

Existen fuentes tanto de start-ups como de proveedores que trabajan constantemente en herramientas para aportar a la gestión de seguridad, detección de vulnerabilidades y productividad de APIs. Es importante investigar y probar las nuevas herramientas antes de incorporarlas en los proyectos.

Arquitectura de la aplicación

Es importante cuando se trabaja en proyectos muy grandes, realizar un mapa de la arquitectura de la aplicación para determinar los componentes que son parte de la aplicación web, abordar problemas de seguridad conocidos, revisar la configuración del servidor alojada en la aplicación web.

Limpieza de la aplicación

Para testear una aplicación se generan cuentas de prueba. Se debe eliminar dichas cuentas del sistema antes del despliegue en producción. Si, la aplicación en el registro cuenta con contraseñas por defecto, la misma debe obligar a cambiar la contraseña en la primera sesión.

Agregar comentarios y escribir código autodescriptivo es una práctica muy recomendada ya que hace al código mantenible en el tiempo, pero se debe tener especial cuidado al revisar comentarios con el fin de determinar información que pueda fugarse a través de archivos expuestos o comentarios en las páginas HTML.

Mecanismo de bloqueo

Contar con un buen mecanismo de bloqueo ayuda a mitigar los ataques por fuerza bruta o que el atacante adivine la contraseña. Las cuentas deben ser bloqueadas a los tres o cinco intentos de autenticación con contraseñas fallidas y desbloqueadas de acuerdo a la necesidad del negocio.

Existen este tipo de mecanismos según la guía de pruebas de OWASP:

- Bloqueo y desbloqueo temporizado.
- Desbloqueo con autoservicio (desbloqueo que envía un correo electrónico a la dirección de correo electrónico registrada).
- Desbloqueo manual por un administrador.
- Desbloqueo manual por un administrador con identificación de usuario positiva.

Usar TLS

Usar certificados SSL para que todas las comunicaciones sean cifradas y https. El aumento de latencia por el uso de certificados SSL es mínimo con respecto a la seguridad conseguida en la información que es transmitida entre el servidor y el cliente.

Usar Docker

El uso de contenedores para liberar la aplicación facilita la limitación de memoria, CPU, número de reinicios y procesos.

Securizar la API

Según lo amerite, incluir en la arquitectura herramientas como: gateways, WAF, establecer número de procesos por usuario, control de tráfico, plataformas de gestión de APIs.

CAPÍTULO III

3 DISEÑO DE LA INVESTIGACIÓN

3.1 Tipo de investigación

La presente investigación se clasifica en tres tipos: científica, aplicativa y experimental.

- **Método Científico**: El método principal que se utilizado en la presente investigación. Las etapas del método científico son:
  - Planteamiento del problema
  - Formulación de Hipótesis
  - Levantamiento de la información
  - Análisis de resultados comprobación de Hipótesis
  - Presentación de resultados
- **Método Analítico**: Este método se aplica durante la etapa de análisis donde se recopiló la información, para proyectar lo que se va a realizar y como se va realizar la investigación.
- **Experimental**: porque se basa en pruebas realizadas en escenarios de laboratorio, en las que se observa los elementos más importantes del objeto de estudio que se investiga para obtener una captación de los fenómenos a primera vista.

3.2 Diseño de la investigación

El diseño de la presente investigación se enmarca dentro del tipo de estudio cuasi-experimental al establecer una metodología para realizar una serie de ataques a una infraestructura informática, en base a la ingeniería del caos para descubrir fallas en producción.

3.3 Métodos y Técnicas

3.3.1 Métodos

La presente investigación se basa en el método científico ya que se refiere a la serie de etapas para obtener un conocimiento válido desde el punto de vista científico, el cual consta de las siguientes etapas:

- Planteamiento del problema
- Formulación de la hipótesis
- Levantamiento de la información
- Análisis e interpretación de resultados
- Comprobación de la hipótesis
- Difusión de resultados

El método hipotético – inductivo

Se emplea para la presente investigación puesto que, a partir de una serie de observaciones particulares en experimentos, se formulan las correspondientes hipótesis, que permiten la producción de conclusiones generales.

3.3.2 Técnicas

Las técnicas que serán utilizadas en la presente investigación son:

- Recopilación de información.
- Planteamiento del problema
- Formulación de la hipótesis
- Levantamiento de la información
- Análisis e interpretación de resultados
- Comprobación de la hipótesis
- Difusión de resultados

3.4 Instrumentos

Las herramientas más populares del mercado para la detección de vulnerabilidades en aplicaciones web se describen a continuación:

Acunetix

Es un escáner de análisis de seguridad de aplicaciones web y detección de vulnerabilidades, a través de un conjunto de pruebas configurables por el usuario. Identifica las vulnerabilidades y proporciona un diagnóstico con el nivel de riesgo y recomendaciones para mitigarla dichas vulnerabilidades. Es una herramienta muy potente, amigable y fácil de usar. (Acunetix, 2019)
Entre las principales características de la herramienta:

- Pruebas de inyección SQL y XSS
- Pruebas se seguridad de Ajax, XML, JSON y aplicaciones web 2.0
- Comprueba la seguridad de aplicaciones web: editor de HTTP, HTTP Sniffer y HTTP Fuzzer
- Ataques personalizados
- Genera una variedad de informes de seguridad y personalizados.
- Analizador multihilo que permite analizar cientos de páginas web a la vez.
- Analiza sitios web con contenido como HTML5, Flash, SOAP y AJAX.
- Escanea y comprueba los puertos de red y servidores web en busca de vulnerabilidades.

Burp Suite

Es una herramienta para encontrar vulnerabilidades de seguridad de aplicaciones web. Utiliza una url para análisis como otros escáneres, pero este modelo tiene limitaciones significativas. Burp emplea un paradigma diferente, enfocado en darle al usuario el control sobre cada solicitud que se escanea y comentarios directos sobre los resultados, esto permite evitar desafíos técnicos que enfrentan otras herramientas convencionales. Burp proporciona un gran marco de pruebas que permite adaptarse a técnicas y metodologías para realizar pruebas manuales y semiautomáticas de penetración. (PortSwigge, 2019)

Características de la herramienta:

- Target: Genera un sitemap de las webs que ha pasado por el proxy.
- Proxy: Intercepta las peticiones entre el navegador y la aplicación.
- Spider: Recopila los recursos de la aplicación de manera automatizada.
- Scanner: Detecta diferentes tipos de vulnerabilidades tanto de forma pasiva como activa.
- Intruder: Automatiza tareas: (fuzzing, fuerza bruta, enumeración, etc).
- Repeater: Permite repetir y manipular las peticiones que pasan por el proxy.
- Secuencer: Analiza la aleatoriedad de los tokens de sesión o cadenas.
- Decoder: Codificador y decodificado de strings(URL, Base64, Hex, hashes, etc).
- Comparer: Compara distintas peticiones y respuestas.
- Extender: Nos permite agregar funcionalidades extras para Burp (plugins).

OWASP Zed Attack Proxy (ZAP)

Zed Attack Proxy (ZAP) es una herramienta de código abierto que proviene del proyecto OWASP (Open Web Application Security Project), cuya finalidad es monitorizar la seguridad de redes y aplicaciones web en busca de cualquier posible fallo de seguridad, mala configuración o alguna vulnerabilidad desconocida que pueda suponer un problema para la red. (ZAP Dev Team, 2019)

Las principales características:

- Herramienta totalmente gratuita y de código abierto.
- Herramienta multiplataforma, compatible incluso con Raspberry Pi.
- Posibilidad de comprobar todas las peticiones y respuestas entre cliente y servidor.
- Posibilidad de localizar recursos en un servidor.
- Servidor proxy de interceptación.
- Rastreadores web tradicionales y por AJAX.
- Escáner automatizado.
- Escáner pasivo.
- Navegación forzada.

- Fuzzer
- Soporte para WebSocket.
- Soporte para lenguajes de scripting y compatibilidad con Plug-n-Hack.
- Capacidad para utilizar certificados SSL dinámicos.
- Análisis de sistemas de autenticación.
- Extensiones (plugins) con las que añadir más funcionalidades a la herramienta.

Postman

Es una herramienta que permite a los desarrolladores crear peticiones HTTP de forma sencilla a APIs, se puede definir el tipo de petición (GET, POST, PUT,HEAD, DELETE,), tokens de autenticación, cabeceras asociadas a la petición, etc. Es muy utilizada para probar, monitorear y depurar API REST. (Postman, 2019)

Características de la herramienta:

- Crea peticiones, las almacena para reutilizarlas posteriormente
- Gestionar documentación de la API basado en colecciones
- Provee entorno Colaborativo
- Genera código de invocación, para diferentes lenguajes de programación: C, C#, Go, Java, JavaScript, NodeJS, Objective-C, PHP, Python, Ruby, Shell, Swift, etc.
- Permite establecer variables
- Soporta Ciclo Vida API management.

ReadyAPI

Es un escáner de seguridad para APIs que permite crear y administrar scripts de pruebas y cuenta con cuatro herramientas en una sola plataforma: prueba de funcionamiento, prueba de rendimiento, prueba de seguridad y API virtualización web. (Smartbear, 2019)

- **SoapUI Pro**: prueba para verificar si la API funciona según lo previsto. Puede crear pruebas ad-hoc y automatizadas para validar el comportamiento esperado de la API, puede registrar el tráfico HTTP/S y filtrar los resultados para extraer las transacciones de API de interés.
- **Secure Pro**: API incluye pruebas de seguridad básicas para que pueda validar que ningún ataque estándar funcionará contra el API. Permite crear pruebas de seguridad desde cero desde inyección SQL, secuencias de comandos entre sitios, escaneos de límites, etc. Tiene integrado un generador de prueba para construir automáticamente un conjunto de pruebas de seguridad que intentan romper su API usando una variedad de técnicas.
- **LoadUI**: Ayuda a verificar el rendimiento de su API bajo carga, aplica las opciones que dependen del API para saber que el servicio puede soportar los volúmenes que espera, puede ejecutar pruebas de carga simultáneamente, utilizando diferentes estrategias y volúmenes, para ver qué tan diferente es el tráfico.
- **ServiceV Pro**: La virtualización del servicio API permite ejecutar todas las virtudes que necesite localmente para que pueda continuar construyendo y probando. Puede simular varias condiciones de servidor y red y establecer respuestas de error para validar su comportamiento.

3.5 Validación de instrumentos

Las pruebas de software es el proceso de someter metódicamente a evaluación, la ejecución de métodos, procedimientos o funciones a través de aplicaciones de software externas que reproduzcan un conjunto de técnicas para simular el uso de APIs con variantes en los parámetros de las mismas, con el objetivo de evidenciar errores.

Existen una gran variedad de herramienta para evaluar automáticamente o de manera personalizada vulnerabilidades en aplicaciones. Se han seleccionado las aplicaciones más populares existentes en el mercado, especializadas en detectar vulnerabilidades, indistintamente del tipo de licencia.

Tabla 1-3. Análisis de herramientas de escaneo de vulnerabilidades para APIs

Vulnerabilidades	Acunetix	Postman	Owasp ZAP	ReadyAPI	BurpSuite
1. Control de Acceso a Nivel de Objeto Roto	✗	✓	✓	✓	✓
2. Autenticación Rota	✓	✓	✓	✓	✓
3. Filtrado de datos Incorrecto	✓	✓	✗	✓	✗
4. Falta de Recursos y Limitación de Velocidad	✗	✓	✗	✓	✓
5. Falta de Control de Acceso a Nivel función/recurso	✗	✓	✓	✓	✓
6. Asignación Masiva	✗	✗	✗	✗	✗
7. Configuración de Seguridad Incorrecta	✗	✓	✗	✓	✗
8. Inyección	✓	✓	✓	✓	✓
9. Cross-Site Scripting (XSS)	✗	✓	✗	✓	✓
10. Uso de Componentes con Vulnerabilidades Conocidas	✓	✗	✗	✗	✗

Realizado por: Alejandra Lara

Como se observa en la Tabla III- 1, ninguna herramienta cubre todas las pruebas que se van a evaluar, por lo cual, se utilizará ReadyApi y Postman por ser las que cubren la mayor parte de vulnerabilidades y Acunetix para cubrir las vulnerabilidades complementarias. Postman permitirá crear pruebas manuales. En el caso de las herramientas de pago para el análisis se utilizará las versiones de prueba.

3.6 Fuentes de información

Las fuentes que serán utilizadas en el presente estudio de investigación son las siguientes:

- Trabajos de investigación.
- Artículos científicos.
- Libros electrónicos.
- Sitios web oficiales y blogs de especialistas.
- Observación.
- Análisis.
- Pruebas

3.7 Recursos

a) Recursos de oficina

- Útiles de Oficina
- Dispositivos de Almacenamiento
- Internet

b) Recursos técnicos

Tabla 2-3. Recursos técnicos

CONCEPTO	CANTIDAD	DETALLE	TOTAL
Servidor	1	Servidor	15000
Laptop	1	Intel ® Core™ i7-6500 CPU 2.5GHz Memoria RAM 16 Gb Windows 10 64 bits, x64	1350.00
Costos Operativos	1	Rubros como transporte y mantenimiento hardware	400.00
Otros costos	1	Costos incurridos en el proceso de investigación, así como en la generación del material documental requerido para su entrega.	1200.00
TOTAL			17.950.00

3.8 Planteamiento de la hipótesis

Un modelo de Interfaz de Programación de aplicaciones REST utilizando GO basado normas y principios de seguridad de información y aplicaciones web mejorará el nivel de seguridad.

3.8.1 Determinación de las variables

La variable dependiente: Nivel de seguridad del software.

La variable independiente: El modelo de API REST utilizando GO

3.9 Operacionalización conceptual de variables

Tabla 1-3. Operacionalización de variables

Hipótesis	Variables	Indicadores
Un modelo de API REST utilizando GO basado en normas y principios de seguridad de información y aplicaciones web mejorará el nivel de seguridad.	**Independiente** El modelo de API REST utilizando GO **Dependiente** Nivel de seguridad del software	Normas Principios Estándares Vulnerabilidades

Realizado por: Alejandra Lara

3.10 Matriz consistencia

Tabla 2-3. Matriz de consistencia

Formulación del problema	**Objetivo general**	**Hipótesis**	**Variables**	**Indicadores**	**Técnica**	**Instrumento**
¿Cómo aporta un modelo de API REST aplicando normas y principios de seguridad de la información y aplicaciones web?	Diseñar un modelo de API REST utilizando GO basado en normas y principios de seguridad de información y aplicaciones web	Un modelo de API REST utilizando GO basado en normas y principios de seguridad de información y aplicaciones web mejorará el nivel de seguridad.	**Independiente** El modelo de API REST utilizando GO **Dependiente** Nivel de seguridad del software	Normas Principios Controles Vulnerabilidades	Observación	Ambiente de Pruebas

Realizado por: Alejandra Lara

3.11 Población y muestra

Análisis de las vulnerabilidades que se deben considerar para proteger en una API. Para determinar la población de vulnerabilidades para validar el modelo, se seleccionarán las vulnerabilidades especificadas tanto en el Top 10:2017 de OWASP para aplicaciones web como en el Top 10:2019 para APIs en referencia a la programación como se muestra a continuación.

Tabla 3-3. Top 10 Web App

TOP 10 WEB APP
1. Inyección
2. Pérdida de Autenticación
3. Exposición de Datos Sensibles
4. Entidades Externas XML (XXE)
5. Pérdida de Control de Acceso
6. Configuración de Seguridad Incorrecta
7. Cross-Site Scripting (XSS)
8. Deserialización Insegura
9. Uso de Componentes con Vulnerabilidades Conocidas
10. Registro y Monitoreo Insuficientes

Tabla 4-3. Top 10 API

TOP 10 API
1. Control de Acceso a Nivel de Objeto Roto
2. Autenticación Rota
3. Filtrado de datos Incorrecto
4. Falta de Recursos y Limitación de Velocidad
5. Falta de Control de Acceso a Nivel función/recurso
6. Asignación Masiva
7. Configuración de Seguridad Incorrecta
8. Inyección
9. Gestión Inadecuada de Activos
10. Registro y Monitoreo Insuficientes

Se puede observar que el TOP 10 de vulnerabilidades en aplicaciones web y APIs, tiene ítems que tienen relación o son similares. Entonces, se toma como base el OWASP API Security Top 10 para obtener las vulnerabilidades que afectan al desarrollo de una API y se agrega Cross-Site Scripting (XSS) y uso de Componentes con Vulnerabilidades Conocidas; que afectan a las aplicaciones web, para conformar la muestra de vulnerabilidades que se va a evaluar sobre los prototipos.

Tabla 5-3. Vulnerabilidades como muestra

Vulnerabilidades
1. Control de Acceso a Nivel de Objeto Roto
2. Autenticación Rota
3. Filtrado de datos Incorrecto
4. Falta de Recursos y Limitación de Velocidad
5. Falta de Control de Acceso a Nivel función/recurso
6. Asignación Masiva
7. Configuración de Seguridad Incorrecta
8. Inyección
9. Cross-Site Scripting (XSS)
10. Uso de Componentes con Vulnerabilidades Conocidas

Finalmente, del análisis de las vulnerabilidades de aplicaciones web y APIs, se obtuvo un conjunto de 10 vulnerabilidades que serán probadas en los prototipos para evidenciar el nivel de seguridad de cada uno.

CAPITULO IV

4 DISCUSIÓN Y RESULTADOS

4.1 Desarrollo de pruebas

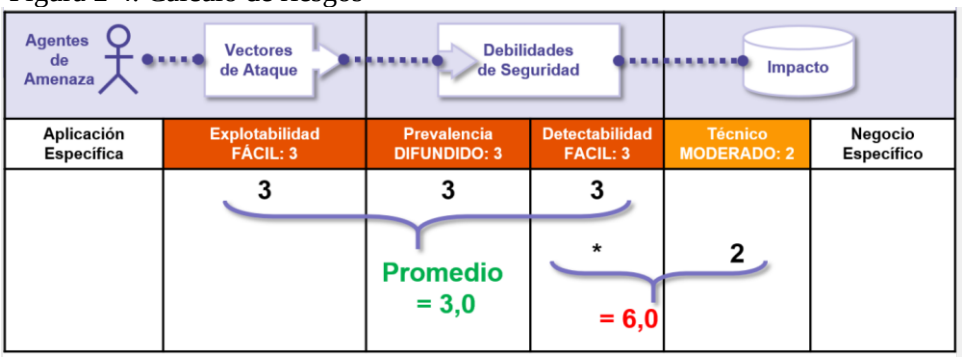
Para la evaluación del modelo propuesto se realizó un escaneo de vulnerabilidades en el Prototipo I (aplicación existente) y el Prototipo II (modelo API REST creado en base a las normas, estándares y guías seleccionadas). Los datos serán obtenidos del análisis comparativo de la valoración de vulnerabilidades de dos prototipos, utilizando el sistema de evaluación de riesgos de OWASP. En la metodología incluye facilidad de explotación, prevalencia, posibilidad de detección y factor de impacto técnico sobre el negocio. La escala de riesgos para cada factor utiliza el rango de 1 (bajo) a 3 (alto).

Figura 1-4. Evaluación de amenazas de OWASP

Agente de Amenaza	Explotabilidad	Prevalencia de Vulnerabilidad	Detección de Vulnerabilidad	Impacto Técnico	Impacto de Negocio
Específico de la Aplicación	Fácil 3	Difundido 3	Fácil 3	Severo 3	Específico del Negocio
	Promedio 2	Común 2	Promedio 2	Moderado 2	
	Difícil 1	Poco Común 1	Difícil 1	Mínimo 1	

Fuente: OWASP

Figura 2-4. Calculo de riesgos



Fuente: OWASP

Para realizar la clasificación de los resultados se utilizará la escala de Likert. El nivel de gravedad de acuerdo a la puntuación de los riesgos será ponderado según la Tabla IV- 1.

Tabla 3-4. Ponderación del nivel de gravedad

Nivel de gravedad	Ponderación
Bajo	0 a 2
Medio	>2 a 5
Alto	>5 a 9

Realizado por: Alejandra Lara

Prototipo I

Esta aplicación web está desarrollada en el lenguaje de programación PHP, no contiene cifrado en las comunicaciones y en los datos almacenados. Este software está disponible solamente en la intranet de la empresa, sirve para registrar los cobros diarios de impuesto al uso de la vía púbica. Los datos son ingresados por una aplicación móvil fuera de línea y posteriormente, sincronizados y almacenados mediante dicha aplicación.

Tabla 4-4. Resultado de vulnerabilidades Prototipo I.

Vulnerabilidad	Gravedad	Explotabilidad	Prevalencia	Detección	Impacto	Puntuación
1. Control de Acceso a Nivel de Objeto Roto	Alta	3	3	2	3	8
2. Autenticación Rota	Alta	3	3	2	3	8
3. Filtrado de datos Incorrecto	Baja	-	-	-	-	0
4. Falta de Recursos y Limitación de Velocidad	Baja	2	1	2	1	1.67
5. Falta de Control de Acceso a Nivel función/recurso	Alta	3	3	1	3	7
6. Asignación Masiva	Baja	2	2	2	1	2
7. Configuración de Seguridad Incorrecta	Media	2	3	1	2	4
8. Inyección	Alta	3	3	2	3	8
9. Cross-Site Scripting (XSS)	Baja	1	2	1	1	1.33
10. Uso de Componentes con Vulnerabilidades Conocidas	Alta	3	3	2	3	8

Realizado por: Alejandra Lara

Prototipo II

La aplicación web es el resultado de la implementación del modelo de software basado en las recomendaciones de los estándares seleccionados. La construcción del back-end para el API se basa en el lenguaje Go y para el front-end se utilizó React JS, ambas partes integradas en una aplicación web.

Tabla 5-4. Resultado de vulnerabilidades Prototipo II.

Vulnerabilidad	Gravedad	Explotabilidad	Prevalencia	Detección	Impacto	Puntuación
1. Control de Acceso a Nivel de Objeto Roto	Baja	0	0	0	0	0
2. Autenticación Rota	Baja	0	0	0	0	0
3. Filtrado de datos Incorrecto	Baja	0	0	0	0	0
4. Falta de Recursos y Limitación de Velocidad	Baja	0	0	0	0	0
5. Falta de Control de Acceso a Nivel función/recurso	Baja	0	0	0	0	0
6. Asignación Masiva	Baja	0	0	0	0	0
7. Configuración de Seguridad Incorrecta	Baja	0	0	0	0	0
8. Inyección	Baja	0	0	0	0	0
9. Cross-Site Scripting (XSS)	Baja	0	0	0	0	0
10. Uso de Componentes con Vulnerabilidades Conocidas	Media	2	3	3	1	2.67

Realizado por: Alejandra Lara

Se calculan los datos en porcentaje de criticidad detectada en cada prototipo. El porcentaje se obtiene de la puntuación del riesgo obtenido por 100, dividido para 9, el cual, es el valor máximo que puede tomar como valor la criticidad de un riesgo.

Tabla 6-4. Resultados de vulnerabilidades en porcentajes

Vulnerabilidad	Prototipo I	Prototipo II
1. Control de Acceso a Nivel de Objeto Roto	88.89 %	0 %
2. Autenticación Rota	88.89 %	0 %
3. Filtrado de datos Incorrecto	0 %	0 %
4. Falta de Recursos y Limitación de Velocidad	18.56 %	0 %
5. Falta de Control de Acceso a Nivel función/recurso	77.78 %	0 %
6. Asignación Masiva	22.22 %	0 %
7. Configuración de Seguridad Incorrecta	44.44 %	0 %
8. Inyección	88.89 %	0 %
9. Cross-Site Scripting (XSS)	14.78 %	0 %
10. Uso de Componentes con Vulnerabilidades Conocidas	88.89 %	29.67 %
Promedio	**53.33%**	**2.97 %**

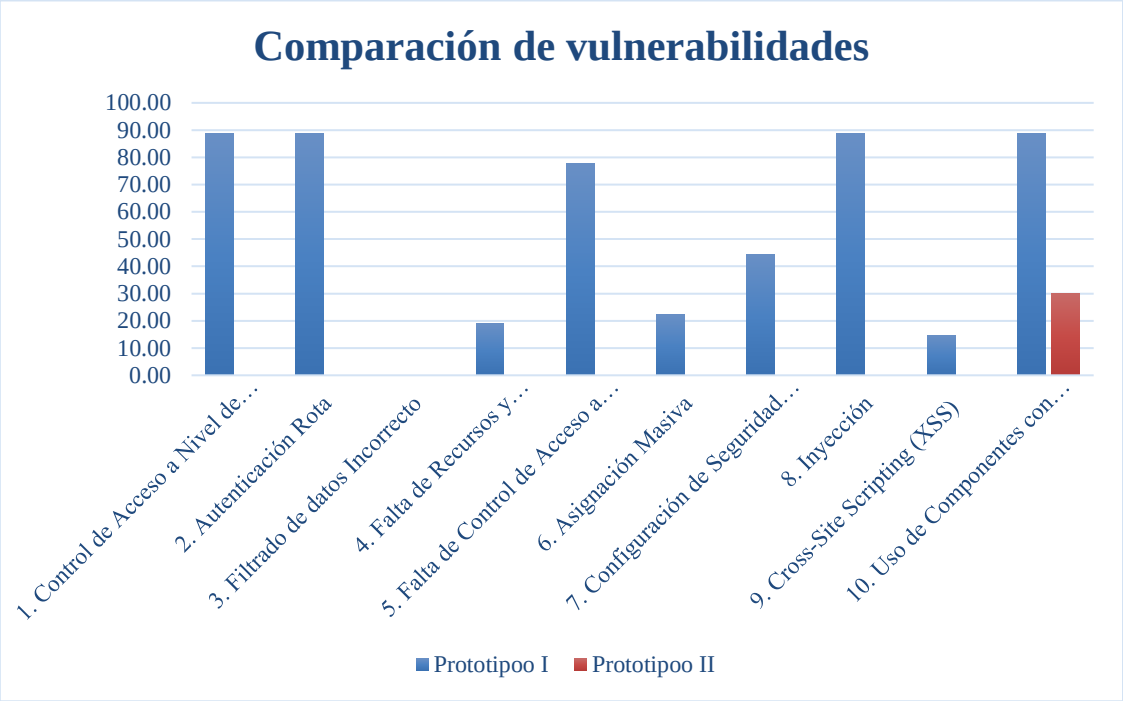
Realizado por: Alejandra Lara

En el prototipo II en el desarrollo del proyecto se utilizaron frameworks y librerías seguras. Sin embargo, en la fase final de pruebas existieron nuevas vulnerabilidades de categoría alta encontradas en las dependencias del proyecto.

4.2 Comprobación de hipótesis

A través de la evaluación de vulnerabilidades encontradas mediante herramientas software se observa un 53.33% en el prototipo I y un 2.77% en el prototipo II, evidenciando una mejora en la seguridad en el prototipo II de un 50.36% respecto a los resultados obtenidos del prototipo I.

Figura 3-3. Comparación de vulnerabilidades



Realizado por: Alejandra Lara

La hipótesis científica se someterá a prueba para aceptar o rechazar dicha hipótesis.

Planteamiento de la hipótesis

H0: Un modelo de API REST utilizando GO basado en normas y principios de seguridad de información y aplicaciones web mejorará el nivel de seguridad.

H1: Un modelo de API REST utilizando GO basado en normas y principios de seguridad de información y aplicaciones web no mejorará el nivel de seguridad.

H0 = Xa > Xb

H1 = Xa <= Xb

Para la investigación se establece un nivel de significancia α = 0.05

En la Tabla IV- 6, se muestran los resultados obtenidos de las pruebas de vulnerabilidades del modelo API REST en el API sin aplicar normas y estándares de seguridad (prototipo I) y modelo API aplicando las normas y estándares de seguridad (prototipo II).

Tabla 7-4. Tabla comparativa de vulnerabilidades

Vulnerabilidad	Prototipo I	Prototipo II
1	8	0
2	8	0
3	0	0
4	1.67	0
5	7	0
6	2	0
7	4	0
8	8	0
9	1.33	0
10	8	2.67
Media	**4,8**	**0,27**
Desv. Est.	**3.15**	**0.80**
Varianza	**9.92**	**0.64**

Realizado por: Alejandra Lara

En este caso, se utilizará la prueba T de Student por ser la muestra menor a 30.

$$t = \frac{\bar{x}_1 - \bar{x}_2}{\sqrt{\frac{n_1 S_1^2 + n_2 S_2^2}{n_1 + n_2 - 2}\left(\frac{1}{n_1} + \frac{1}{n_2}\right)}}$$

Donde;

$\bar{x}$ = media

n= tamaño de la muestra

S^2 = varianza

Especificación de las regiones de aceptación de la hipótesis

Grados de libertad para dos muestras

GL = n1 + n2 - 2

GL = 10 + 10 – 2 = 18

De acuerdo los grados de libertad GL=18 y nivel de significancia $\alpha = 0.05$, según la tabla T de Student el valor de Z=1,73

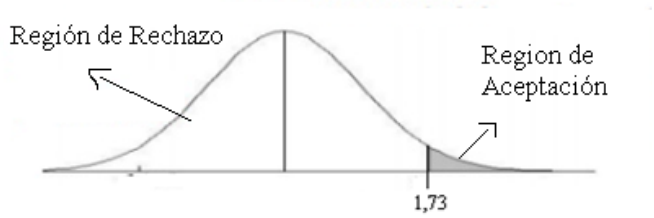


Figura 4-4. Verificación de hipótesis

Realizado por: Alejandra Lara

Si t > z se acepta la hipótesis H0; caso contrario se rechaza H0 y se acepta H1

Calculo de t

$$t = \frac{\bar{x}_1 - \bar{x}_2}{\sqrt{\frac{n_1 S_1^2 + n_2 S_2^2}{n_1 + n_2 - 2}\left(\frac{1}{n_1} + \frac{1}{n_2}\right)}}$$

$$t = \frac{(4{,}8 - 0{,}27)}{\sqrt{\frac{10(9{,}92)+10(0{,}64)}{18}}\sqrt{\frac{1}{10}+\frac{1}{10}}} = \quad 4{,}18$$

De acuerdo a los resultados obtenidos, t > z; se encuentra en la región de aceptación de la hipótesis H0. Entonces, se acepta la hipótesis H0: el modelo construido basado en normas y estándares de seguridad para API REST mejora el nivel de seguridad.

Se ha comprobado la hipótesis: Un modelo de API REST utilizando GO basado en normas y principios de seguridad de información y aplicaciones web mejorará el nivel de seguridad.

CAPITULO V

5 MODELO

5.1 Modelo de API REST con GO

El presente modelo se basa en un conjunto de normas y estándares para la seguridad de aplicaciones web y seguridad de la información. Para la obtención del mismo se ha investigado las mejores opciones funciona para la construcción de una API como modelo que funcione para la mayor parte de desarrollo.

5.1.1 Estructura del proyecto

El proyecto está distribuido de la siguiente forma:

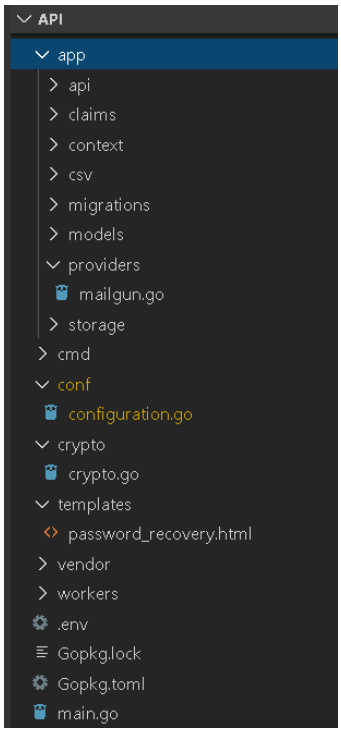


Figura 1-5. Estructura del api

Fuente: Alejandra Lara

Api: Contiene la lógica del negocio y los end points
Claims: estructura que se cifrada y se retorna en el login como un token de seguridad
Migrations: El código que ejecuta la creación o migración de las tablas de la base de datos
Models: Contiene el modelo de las entidades
Providers: recuperación de contraseña
Conf: contiene la estructura de configuración del API
Crypto: contiene el cifrado del token y la clave
Templates: plantilla de recuperación de contraseña
Vendor: las dependencias de la api
Workers: utilitarios de la api
.env: archivo de configuración

5.1.2 ORM

Gorm

Es una biblioteca de mapeo de objeto-relacional (ORM) de Golang. Es un ORM de propósito general, el cual puede ser usado por la mayor parte de base de datos, es decir, de aquellas que tengan previamente implementadas los dialectos.

api
└— app
└— **models**
└— connection.go

```go
import (
    "fmt"
    "log"

    "api/conf"

    "github.com/jinzhu/gorm"
)

// Connect handles the connection to the database and returns it
func Connect(config *conf.Config) (*gorm.DB, error) {

    dbURI := fmt.Sprintf("%s:%s@tcp(%s:%s)/%s?charset=%s&parseTime=True",
        config.DB.Username,
        config.DB.Password,
        config.DB.Host,
        config.DB.Port,
        config.DB.Name,
        config.DB.Charset,
    )
    db, err := gorm.Open(config.DB.Dialect, dbURI)

    if err != nil {
        log.Fatalln("error db conect", err)
    }
    return db, nil
}
```

Figura 2-5. Conexión a base de datos MySql con gorm

Fuente: Alejandra Lara

Características de Gorm:

- Características completas ORM (casi)
- Asociaciones (Tiene uno, Tiene algunos, pertenece a, muchos a muchos, polimorfismo)
- Ganchos (antes/después Crear/guardar/actualizar/eliminar/buscar)
- Precarga (carga impaciente)
- Transacciones
- Clave primaria compuesta

- Constructor SQL
- Migraciones automáticas
- Registrador
- Extensible, escribir Plugins basado en callbacks GORM

Previene	**Descripción**
A1: 2017 Inyección	• Utilizar una API segura, se utiliza una herramienta de Mapeo Relacional de Objetos (ORMs), en este caso Gorm.

Fuente: Alejandra Lara

5.1.3 Cifrado

Toda información sensible como: números de cuenta, datos tarjeta de credito, teléfono, email, tokens, password, etc. Deben ser cifrados.

Bcrypt

Es una librería para guardar contraseñas, genera un hash basado en el algoritmo Blowfish de una sola vía. Esto es bueno, ya que el algoritmo matemático no producirá el problema de colisiones que tienen otros algoritmos como MD5, SHA1 entre otros.

api

└— app

└— **models**

└— user.go

```
import (
    "time"

    "github.com/ale/api/crypto"

    "golang.org/x/crypto/bcrypt"
    "gopkg.in/guregu/null.v3"
)

// EncryptPassword sets the encrypted password of the user from a plain string
func (u *User) EncryptPassword(password string) error {
    pw, err := bcrypt.GenerateFromPassword([]byte(password), bcrypt.DefaultCost)
    if err != nil {
        return err
    }

    u.Password = string(pw)
    return nil
}
```

Figura 3-5. Cifrado de contraseña con bcrypt

Fuente: Alejandra Lara

Previene		**Descripción**
A3: 2017 Exposición de datos sensibles	API3: 2019 Filtrado de datos incorrecto	• Las claves son información sensible por lo cual se debe cifrar para evitar la exposición cuando sean procesadas, almacenadas o transmitidos por el sistema. • Se utiliza algoritmos y protocolos estándares y fuertes e implemente una gestión adecuada de claves, en este caso funciones de hashing recomendadas como bcrypt.

Fuente: Alejandra Lara

5.1.4 Autenticación

Es imprescindible aplicar una autenticación fuerte y con soluciones probadas.

JWT

Este método utiliza token web JSON (JWT) para la autenticación. La autenticación basada en sesiones o cookies necesitaba mantener una sesión o pasar repetidamente credenciales de inicio de sesión al servidor. La autenticación JWT a diferencia de la tradicional permite a una aplicación verificar que la persona que envía una solicitud es realmente quien dice ser, sin almacenar información sobre el usuario en el sistema.

El formato JWT consta de tres componentes: un encabezado, la carga útil y una firma. Estas tres partes se codifican en base64, luego se concatenan con puntos como separadores. El encabezado es más o menos fijo, y el objeto JSON de la carga útil se forma al configurar el ID de usuario y el tiempo de vencimiento en milisegundos unix.

La aplicación que emite el token tendrá una clave, que es un valor secreto, y solo conocida por kdicha aplicación. Las representaciones de base64 del encabezado y la carga útil se combinan con la clave secreta y luego se pasan a través de un algoritmo hash.

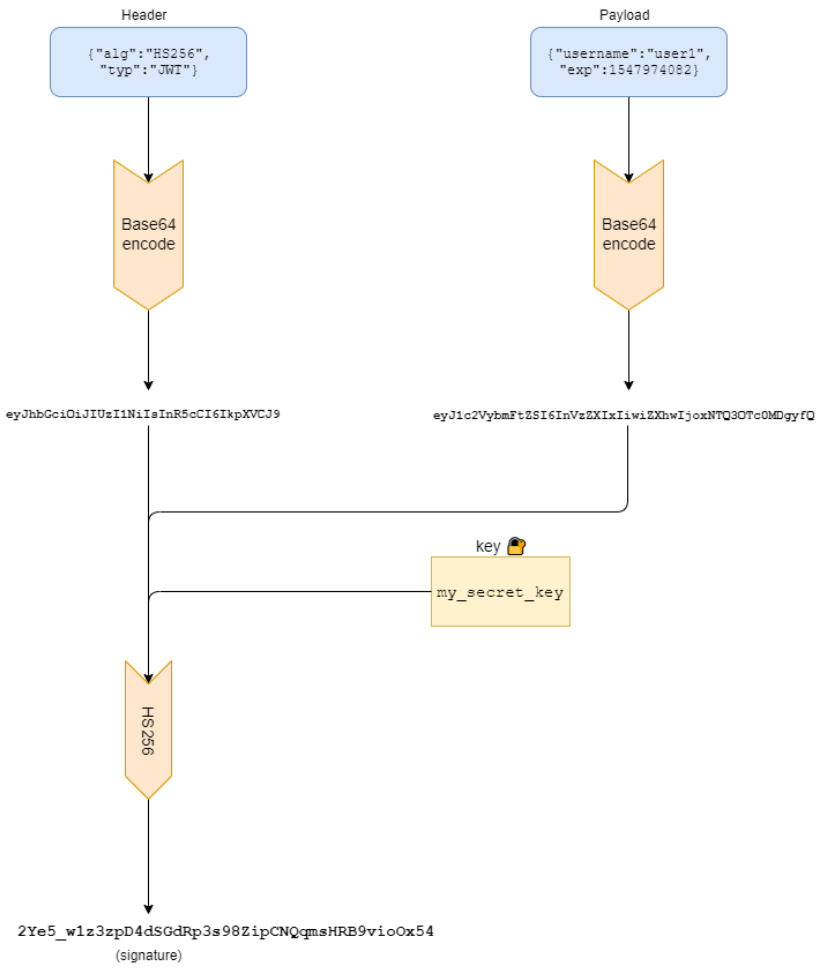


Figura 4-5. Autenticación JWT

Fuente: https://www.sohamkamani.com/blog/golang/2019-01-01-jwt-authentication/

Sin conocer la clave, no hay forma de generar una firma válida. Si se intenta manipular la carga útil existente de un JWT válido, las firmas ya no coincidirán. Es lo que hace a este método seguro.

api

└— app

└— **models**

└— user.go

```go
// Authenticate check if the password entered corresponds to the user
func (u *User) Authenticate(pwd string) bool {
    err := bcrypt.CompareHashAndPassword([]byte(u.Password), []byte(pwd))
    return err == nil
}

// RefreshToken model
type RefreshToken struct {
    ID        int `json:"id" gorm:"PRIMARY_KEY; AUTO_INCREMENT;size:11" `
    Token     string
    User      *User
    UserID    int
    Revoked   bool
    CreatedAt time.Time
}

// NewRefreshToken creates a new refresh token
func NewRefreshToken(u *User) *RefreshToken {
    return &RefreshToken{
        Token:   crypto.SecureToken(),
        User:    u,
        UserID:  u.ID,
        Revoked: false,
    }
}
```

Figura 5-5. Autenticación basdo en JWT

Fuente: Alejandra Lara

api

└— app

└— **api**

└— user.go

```go
// LoginUser login user
func (a *App) LoginUser(c *gin.Context) {
    params := &loginParams{}
    if err := c.ShouldBindJSON(params); err != nil {
        c.JSON(http.StatusBadRequest, gin.H{"error": err.Error()})
        return
    }

    if params.Email == "" {
        c.JSON(http.StatusBadRequest, gin.H{"error": "The email is required"})
        return
    }

    if params.Password == "" {
        c.JSON(http.StatusBadRequest, gin.H{"error":"The password is required"})
        return
    }

    user := &models.User{}
    query := userQuery(a)

    if result := query.Where("email=?", params.Email).First(&user);
    result.Error != nil {
        if result.RecordNotFound() {
        c.JSON(http.StatusNotFound, gin.H{"error": "Credenciales incorrectas"})
            return
        }
        c.Error(result.Error)
        return
    }

    if user.Active == 5 {
        c.JSON(http.StatusNotFound, gin.H{"error": "Usuario bloqueado"})
        return
    }
```

```go
    if !user.Authenticate(params.Password) {
        user.Active = user.Active + 1
        a.DB.Save(&user)
        c.JSON(http.StatusUnprocessableEntity, gin.H{"error": "Credenciales inco
rrectas"})
        return
    }
```

```
    if result := a.DB.Save(&refreshToken); result.Error != nil {
        c.Error(result.Error)
        return
    }

    token, err := generateAccessToken(user, time.Second*time.Duration(a.Config.J
WT.Exp), a.Config.JWT.Secret)
    if err != nil {
        c.Error(err)
        return
    }
    user.Active = 1
    a.DB.Save(&user)

    c.JSON(http.StatusOK, &accessTokenResponse{
        Token:        token,
        TokenType:    "bearer",
        ExpiresIn:    a.Config.JWT.Exp,
        RefreshToken: refreshToken.Token,
    })
}

// LogoutUser will logout the user by removing the refresh token from database
func (a *App) LogoutUser(c *gin.Context) {
    claims := mcontext.GetUserClaims(c)

    if claims == nil {
        c.AbortWithStatusJSON(http.StatusUnauthorized, gin.H{"error": "You need
to be authenticated to access this resource"})
        return
    }
    userID := claims.ID

    if result := a.DB.Where("user_id=?",userID).Delete(&models.RefreshToken{});
result.Error != nil {
        c.Error(result.Error)
        return
    }
    c.JSON(http.StatusOK, gin.H{"status": "logout"})
}
```

Figura 6-5. Login y Logout

Fuente: Alejandra Lara

api

└── app

└── **claims**

└── claims.go

```go
import (
    "github.com/ale/api/app/models"

    "github.com/dgrijalva/jwt-go"
)

// UserClaims model
type UserClaims struct {
    jwt.StandardClaims
    ID        int          `json:"id"`
    FirstName string       `json:"firstName"`
    LastName  string       `json:"lastName"`
    Email     string       `json:"email"`
    Role      *models.Role `json:"role"`
}
```

Figura 7-5. Estructura del objeto que se cifra y se retorna como token de seguridad.

Fuente: Alejandra Lara

api

└── app

└── **context**

└── context.go

```
import (
    "github.com/ale/api/app/claims"

    "github.com/gin-gonic/gin"
-)

// GetUserClaims get user claims
func GetUserClaims(c *gin.Context) *claims.UserClaims {
    token, _ := c.Get("claims")
    if token == nil {
        return nil
    }
    return token.(*claims.UserClaims)
}
```

Figura 8-5. Codifica el token de seguridad para obtener la informacion de la estructura Claim

Fuente: Alejandra Lara

api

└── **conf**

└── configuration.go

```
import (
    "log"
    "os"
    "strconv"
    _ "github.com/jinzhu/gorm/dialects/mysql"
    "github.com/joho/godotenv"
    "github.com/kelseyhightower/envconfig"
)

// DBConfig database configuration
type DBConfig struct {
    Dialect  string `default:"mysql"`
    Username string
    Password string
    Host     string `default:"localhost"`
    Port     string `default:"3306"`
    Name     string
    Charset  string `default:"utf8"`
}
```

```go
// AWSConfiguration holds all the Amazon Web Services configuration
type AWSConfiguration struct {
    Region          string
    Bucket          string
    AccessKeyID     string `split_words:"true"`
    SecretAccessKey string `split_words:"true"`
    Token           string
    TempPath        string `split_words:"true"`
    AssetPath       string `split_words:"true"`
}

// JWTConfiguration holds all the jwt configuration
type JWTConfiguration struct {
    Secret string
    Exp    int
}

// P2PConfiguration holds all the p2p configuration
type P2PConfiguration struct {
    Login      string
    Secret     string
    Locale     string
    Expiration int
    RerutnURL  string
    URLPayment string
}
```

```go
// MailgunConfiguration holds all the mailgun smtp configurations
type MailgunConfiguration struct {
    Domain    string
    Sender    string
    APIKey    string
    PublicKey string
}

// Config struct
type Config struct {
    Port    int `default:"9000"`
    Front   string
    DB      *DBConfig
    AWS     *AWSConfiguration
    JWT     *JWTConfiguration
    MAILGUN *MailgunConfiguration
    P2P     *P2PConfiguration
}
```

```go
func loadEnvironment(filename string) error {
    var err error

    if filename != "" {
        err = godotenv.Load(filename)
    } else {
        err = godotenv.Load()
        if os.IsNotExist(err) {
            return nil
        }
    }
    return err
}
```

```go
// GetConfig gloval configuration
func GetConfig(filename string) *Config {
    err := godotenv.Load()
    if err != nil {
        log.Fatal("Error loading .env file")
    }
    USERNAME := os.Getenv("MYSQL_USERNAME")
    PASSWORD := os.Getenv("MYSQL_PASSWORD")
    DATABASE := os.Getenv("MYSQL_DATABASE")
    MYSQLPORT := os.Getenv("MYSQL_PORT")
    CHARSET := os.Getenv("CHARSET")
    DBHOST := os.Getenv("MYSQL_HOST")
    secret := os.Getenv("JWT_SECRET")
    exp := os.Getenv("JWT_EXPIRATION")
    front := os.Getenv("FRONTURL")
    expiration, err := strconv.Atoi(exp)

    return &Config{
        Front: front,
        MAILGUN: &MailgunConfiguration{
            Domain:    "mg.stagingppm.com",
            APIKey:    "key-45845fjhfksdf86j5678iuhghfr45678",
            PublicKey: "pubkey-xb4445544bjfgfdlkdf97ñllffd2f5d80ed",
            Sender:    "PPM <no-reply@mg.stagingppm.com>",
        },
        DB: &DBConfig{
            Dialect:  "mysql",
            Host:     DBHOST,
            Username: USERNAME,
            Password: PASSWORD,
```

```
            Name:     DATABASE,
            Port:     MYSQLPORT,
            Charset:  CHARSET,
        },
        JWT: &JWTConfiguration{
            Secret: secret,
            Exp:    expiration,
        },
    }

}

// LoadConfig loads the configuration from a file
func LoadConfig(filename string) (*Config, error) {

    config := GetConfig(filename)
    // log.Fatalf("conf: %+v")
    if err := envconfig.Process("RTP_API", config); err != nil {
        return nil, err
    }
    return config, nil
}
```

Figura 9-5. Archivo de configuración

Fuente: Alejandra Lara

api

└── .env

```
.env
1   MYSQL_USERNAME=root
2   MYSQL_PASSWORD=12345
3   MYSQL_DATABASE=api_v1
4   MYSQL_HOST=localhost
5   MYSQL_MAX_OPEN_CONNS=3
6   CHARSET=utf8
7   PARSETIME=true
8   MYSQL_PORT=3306
9   DBCONNECTION_TTL=1
10  TEST=development
11  JWT_SECRET:alejandra
12  JWT_EXPIRATION: 280000
13
```

Figura 9-5. Configuración de variables

Fuente: Alejandra Lara

Previene		Descripción
A1:2017 Inyección	API8:2019 Inyección	• Se utiliza controles para evitar la fuga masiva de registros en caso de inyección SQL, FIRST.
A2:2017 Perdida de autenticación	API2: 2019 Autenticación Rota	• Se previene ataques de enumeración de usuarios mediante mensajes genéricos iguales en todas las salidas. • El gestor de sesión en el servidor, genera un nuevo ID de sesión aleatorio por medio de una clave con el inicio de cada sesión. El ID de sesión se almacena de forma segura y es invalidado después del cierre de sesión y de un tiempo.
A3:2017 Exposición de Datos Sensibles	API3: 2019 Filtrado de datos incorrecto	• No se almacena datos sensibles innecesarios. Se utiliza un sistema de tokenización para el manejo de sesiones.
A5:2017 Pérdida de Control de Acceso	API5:2019 Falta de control de acceso a nivel función/recurso	• Los tokens JWT son invalidados después de la finalización de una sesión por parte del usuario y en intervalos de tiempo.
A6:2017 Configuración de Seguridad Incorrecta	API7:2019 Configuración de Seguridad Incorrecta	• Se maneja los errores sin dar información sensible, evita taques de enumeración de usuarios.

Fuente: Alejandra Lara

5.1.5 Bloqueo y recuperación de contraseña

Es preciso contar con un buen mecanismo de bloqueo y recuperación de contraseñas.

Mailgun

Es una librería que permite enviar mensajes a través de Mailgun utilizando Go, permite interactuar fácilmente con la API con una cantidad de código reducido.

api

└— app

└— **models**

└— user.go

```go
// RestorePassword model
type RestorePassword struct {
    ID        int       `json:"id" gorm:"PRIMARY_KEY; AUTO_INCREMENT;size:11" `
    Token     string    `json:"token"`
    User      *User     `json:"user"`
    UserID    int       `json:"userID"`
    Revoked   bool      `json:"revoked"`
    CreatedAt time.Time `json:"create"`
    UpdatedAt time.Time `json:"updatedAt"`
}

// NewRestorePassword creates a new refresh token
func NewRestorePassword(token string, userID int) *RestorePassword {
    return &RestorePassword{
        Token:   token,
        UserID:  userID,
        Revoked: false,
    }
}

//Email sending
type Email struct {
    Attachments []Attachment `json:"attachments"`
    To          []string     `json:"to"`
    CC          []string     `json:"cc"`
    BCC         []string     `json:"bcc"`
    From        string       `json:"from"`
    Subject     string       `json:"subject"`
    Message     string       `json:"message"`
    HTMLSource  string       `json:"html_source"`
}
type Attachment struct {
    Name      string `json:"name"`
    Extension string `json:"extension"`
    Content   string `json:"content"`
}
```

Figura 10-5. Modelo de recuperación de contraseña

Fuente: Alejandra Lara

api

└— app

└— **api**

└— user.go

```go
//RestorePassword restored password metodo
func (a *App) RestorePassword(c *gin.Context) {
    emailParams := c.Param("email")

    if emailParams == "" {
        c.JSON(http.StatusBadRequest, gin.H{"error": "Email es requerido"})
        return
    }

    var user models.User
    query := userQuery(a)
    if result := query.Where("email = ?", emailParams).First(&user); result.Erro
r != nil {
        if result.RecordNotFound() {
            c.JSON(http.StatusNotFound, gin.H{"error": "Usuario no encontrado"})
            return
        }
        c.Error(result.Error)
        return
    }

    token := createRandString(24)
    restore := models.NewRestorePassword(token, user.ID)

    if result := a.DB.Create(restore); result.Error != nil {
        c.Error(result.Error)
        return
    }

    data := resetPasswordData{
        Name:  user.FirstName,
        Email: user.Email,
        Token: restore.Token,
    }
```

```
  a.sendResetPasswordTemplate(&data)

    c.JSON(http.StatusOK, "Correo enviado")

}

// letterBytes all letters
const letterBytes = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567
890"

func createRandString(lengt int) string {

    randString := make([]byte, lengt)
    for i := range randString {
        randString[i] = letterBytes[rand.Int63()%int64(len(letterBytes))]
    }
    return string(randString)
}
```

Figura 11-5. Recuperación de contraseña

Fuente: Alejandra Lara

api

└— app

└— **providers**

└— mailgun.go

```
import (
    "bytes"
    "encoding/base64"

    "api/app/models"

    mailgun "gopkg.in/mailgun/mailgun-go.v1"
)

type Mailer interface {
    BuildSender(*models.Email) error
}
```

```go
type mailer struct {
    Provider  mailgun.Mailgun
    Domain    string
    APIKey    string
    PublicKey string
}

//NewMailer build with config
func NewMailer(Domain string, APIKey string, PublicKey string) Mailer {
    mg := mailgun.NewMailgun(Domain, APIKey, PublicKey)
    return &mailer{
        Provider:  mg,
        Domain:    Domain,
        APIKey:    APIKey,
        PublicKey: PublicKey,
    }
}

func (m *mailer) BuildSender(email *models.Email) error {
    message := m.Provider.NewMessage(
        email.From,
        email.Subject,
        "",
    )

    for _, to := range email.To {
        message.AddRecipient(to)
    }

    for _, cc := range email.CC {
        message.AddCC(cc)
    }

    for _, bcc := range email.BCC {
        message.AddBCC(bcc)
    }

    for _, attachment := range email.Attachments {
        dec, err := base64.StdEncoding.DecodeString(attachment.Content)
        if err != nil {
            panic(err)
        }
       message.AddBufferAttachment(attachment.Name+"."+attachment.Extension,dec)
    }
```

```
    if email.HTMLSource == "" {

        var html bytes.Buffer
        html.WriteString(email.Message)
        message.SetHtml(html.String())
    }

    _, _, err := m.Provider.Send(message)
    if err != nil {

        return err
    }

    return nil
}
```

Figura 12-5. Envía correo de recuperación

Fuente: Alejandra Lara

api

└— app

└— **templates**

└— password_recovery.html

```
<!DOCTYPE html
  PUBLIC "-
//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">

<head>
  <title>Tu pagina web</title>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
  <style type="text/css">
    html, body, div, form, fieldset,
    legend, label, img, tr {
      margin: 0;
      padding: 0;
    }
```

```
    table {
      border-collapse: collapse;
      border-spacing: 0;
    }

    th, td {
      text-align: left;
    }

    h1, h2, h3, h4, h5, h6, th, td, caption {
      font-weight: normal;
    }

    img {
      border: 0;
      display: block;
      padding: 0;
      margin: 0;
    }
  </style>
</head>

<body bgcolor="#ffffff">
  <table width="100%">
    <tr>
      <td>
        <center>
          <table style="display: inline-
table;" border="0" cellpadding="0" cellspacing="0" width="650">
            <tr>
              <td width="650" height="58">
                <div style="display:block; margin-left: 8%; font-
size: 16px;font-weight: 600;">
                  Hola: <span style="color: #00549e;">{{.name}}</span>
                </div>
              </td>
            </tr>
            <tr>
              <td><img style="display:block" name="restorepass_r8_c1"
                  src="https://chaide-y-chaide.s3.amazonaws.com/correos-
img/restorepass_r8_c1.jpg" width="650"
                  height="219" id="restorepass_r8_c1" alt="" /></td>
            </tr>
            <tr>
```

```
            <td><img style="display:block" name="restorepass_r10_c1"
                src="https://chaide-y-chaide.s3.amazonaws.com/correos-
img/restorepass_r10_c1.jpg" width="650"
                height="90" id="restorepass_r10_c1" alt="" /></td>
          </tr>
          <tr>
            <td>
              <table style="display: inline-
table;" align="left" border="0" cellpadding="0" cellspacing="0"
                width="650">
                <tr>
                  <td width="650" height="18" style="font-size: 14px; font-
weight: 300;text-align: center;">
                    {{.email}}
                  </td>
                </tr>
              </table>
            </td>
          </tr>
          <tr>
            <td><img style="display:block" name="restorepass_r15_c1"
                src="https://chaide-y-chaide.s3.amazonaws.com/correos-
img/restorepass_r15_c1.jpg" width="650"
                height="61" id="restorepass_r15_c1" alt="" /></td>
          </tr>
          <tr>
            <td>
              <table style="display: inline-
table;" align="left" border="0" cellpadding="0" cellspacing="0"
                width="650">
                <tr>
                  <td><img style="display:block" name="restorepass_r16_c1"
                      src="https://chaide-y-chaide.s3.amazonaws.com/correos-
img/restorepass_r16_c1.jpg" width="195"
                      height="47" id="restorepass_r16_c1" alt="" /></td>
                  <td><a href="{{.url}}" target="_blank"><img style="display:b
lock"
                       name="restorepass_r16_c3"
                       src="https://chaide-y-chaide.s3.amazonaws.com/correos-
img/restorepass_r16_c3.jpg" width="256"
                       height="47" id="restorepass_r16_c3" alt="" /></a></td>
                  <td><img style="display:block" name="restorepass_r16_c7"
                      src="https://chaide-y-chaide.s3.amazonaws.com/correos-
img/restorepass_r16_c7.jpg" width="199"
                      height="47" id="restorepass_r16_c7" alt="" /></td>
```

```
                </tr>
              </table>
            </td>
          </tr>
          <tr>
            <td><img style="display:block" name="restorepass_r19_c1"
                src="https://chaide-y-chaide.s3.amazonaws.com/correos-
img/restorepass_r19_c1.jpg" width="650"
                height="106" id="restorepass_r19_c1" alt="" /></td>
          </tr>
          <tr>
            <td>
              <table style="display: inline-table; background-
color: #505050; color: #FFFFFF; font-
size: 12px;"  align="left" border="0" cellpadding="0" cellspacing="0"
                width="650">
                <tr>
                  <td><img style="display:block" name="restorepass_r22_c1"
                      src="https://chaide-y-chaide.s3.amazonaws.com/correos-
img/restorepass_r22_c1.jpg" width="38"
                      height="19" id="restorepass_r22_c1" alt="" /></td>
                  <td>
                    ®{{.year}} Pie del correo.
                  </td>
                </tr>
              </table>
            </td>
          </tr>
          <tr>
            <td><img style="display:block" name="restorepass_r24_c1"
                src="https://chaide-y-chaide.s3.amazonaws.com/correos-
img/restorepass_r24_c1.jpg" width="650"
                height="18" id="restorepass_r24_c1" alt="" /></td>
          </tr>
        </table>
      </center>
    </td>
  </tr>
 </table>
</body>
</html>
```

Figura 13-5. Codigo html de correo de recuperación

Fuente: Alejandra Lara

Figura 14-5. Correo electrónico de recuperación

Fuente: Alejandra Lara

Previene		Descripción
A2: 2017 Pérdida de Autenticación	API2: 2019 Autenticación rota	Se utiliza un mecanismo de bloqueo y recuperación de contraseña por correo electrónico.

Fuente: Alejandra Lara

5.1.6 Autorización

Se define los objetos y recursos a los que pueden acceder

Rol	Permiso	Objeto	Restricción
Administrador	Todo	Gestión de usuarios, Roles, Géneros, Propietarios, Predios, Títulos	N/A
Supervisor	Todo	Géneros Propietarios, Predios, Títulos	
Visualizador	Lectura	Predios, Títulos	Solo sus recursos

Fuente: Alejandra Lara

api

└── app

└── **api**

└── api.go

```
// NewAPI starts the App
func NewAPI(db *gorm.DB, config *conf.Config) *App {
    api := &App{
        DB:     db,
        Config: config,
    }

    r := gin.Default()
    r.Use(errorHandler())

    r.Use(cors.New(cors.Config{
        AllowMethods:     []string{"GET", "POST", "PUT", "HEAD", "DELETE"},
        AllowHeaders:     []string{"Origin", "Content-Length", "Content-
Type", "Authorization"},
        AllowCredentials: false,
        AllowAllOrigins:  true,
        MaxAge:           12 * time.Hour,
    }))
    r.POST("/auth/login", api.LoginUser)
    r.POST("/auth/token", api.RefreshToken)

    auth := r.Group("/")
    auth.Use(api.withToken())
    {
        auth.Use(requireAuth())
        auth.Use(api.withUser())
        auth.POST("/auth/logout", api.LogoutUser)

        // Gestion de usuarios
        auth.GET("/users", api.ensureRoleAccess("administrator"), api.ListUsers)
        auth.POST("/users", api.ensureRoleAccess("administrator"), api.CreateUse
r)
        auth.GET("/users/:userID", api.ensureRoleAccess("administrator"), api.Ge
tUser)
        auth.PUT("/users/:userID", api.ensureRoleAccess("administrator"), api.Up
dateUser)
```

```
        auth.DELETE("/users/:userID", api.ensureRoleAccess("administrator"), api
.DeleteUser)

        //Roles
        auth.GET("/rols", api.ensureRoleAccess("administrator"), api.ListRoles)

        //Generos
        auth.GET("/gender", api.ensureRoleAccess("assistant"), api.ListGender)
        auth.POST("/gender", api.ensureRoleAccess("assistant"), api.CreateGender
)
        auth.PUT("/gender/:genderID", api.ensureRoleAccess("assistant"), api.Upd
ateGender)
        auth.GET("/gender/:genderID", api.ensureRoleAccess("assistant"), api.Get
Gender)
        auth.DELETE("/gender/:genderID", api.ensureRoleAccess("assistant"), api.
DeleteGender)

        //Propietarios
        auth.GET("/tcatpropietario/:propietarioID", api.ensureRoleAccess("assist
ant"), api.GetPropietario)

        //Predios
        auth.GET("/tcatpredio/:predioID", api.ensureRoleAccess("visualizer"),
api.GetPredio)

        //Titulos
        auth.GET("/tcattitulo/:predioID", api.ensureRoleAccess("visualizer"),
api.GetTitulos)
    }

    api.handler = r
    return api
}
```

Figura 15-5. Autorización a los recursos del api

Fuente: Alejandra Lara

api

└── app

└── **api**

└── auth.go

```
import (
    mcontext "api/app/context"
    "api/app/models"
    "api/app/claims"
    "errors"
    "net/http"
    "regexp"

    jwt "github.com/dgrijalva/jwt-go"
    "github.com/gin-gonic/gin"
)

var bearerRegexp = regexp.MustCompile(`^(?:B|b)earer (\S+$)`)

// withToken  is a middleware that adds and parses the token present in the head
er to the context of the request
func (a *App) withToken() gin.HandlerFunc {
    return func(c *gin.Context) {
        token, err := extractBearerToken(c)
        if err != nil {
            c.AbortWithStatusJSON(http.StatusUnauthorized, gin.H{"error": err.Er
ror()})
            return
        }

        if token == "" {
            c.Next()
            return
        }

        claims, err := parseJWTClaims(a.Config.JWT.Secret, token)
        if err != nil {
            c.AbortWithStatusJSON(http.StatusUnauthorized, gin.H{"error": err.Er
ror()})
            return
        }

        c.Set("claims", claims)
        c.Next()
    }
}

func extractBearerToken(c *gin.Context) (string, error) {
    authHeader := c.GetHeader("Authorization")
```

```
    if authHeader == "" {
        return "", nil
    }

    matches := bearerRegexp.FindStringSubmatch(authHeader)
    if len(matches) != 2 {
        return "", errors.New("Invalid Authorization Header")
    }

    return matches[1], nil
}

func parseJWTClaims(secret, bearer string) (*claims.UserClaims, error) {
    token, err := jwt.ParseWithClaims(bearer, &claims.UserClaims{}, func(token *
jwt.Token) (interface{}, error) {
        return []byte(secret), nil
    })

    if token == nil {
        return nil, errors.New("Invalid token")
    }

    if claims, ok := token.Claims.(*claims.UserClaims); ok && token.Valid {
        return claims, nil
    }

    return nil, err
}

func requireAuth() gin.HandlerFunc {
    return func(c *gin.Context) {
        claims := mcontext.GetUserClaims(c)
        if claims == nil {
            c.AbortWithStatusJSON(http.StatusUnauthorized, gin.H{"error": "You n
eed to be authenticated to access this resource"})
        }

        c.Next()
    }
}

func (a *App) ensureRoleAccess(requiredRole string) gin.HandlerFunc {
    return func(c *gin.Context) {

        var role models.Role
```

```
        if result := a.DB.Where("name = ?", requiredRole).First(&role);
       result.Error != nil {
            c.JSON(http.StatusBadRequest, gin.H{"error": "Role not found"})
            c.Abort()
        }
        claims := mcontext.GetUserClaims(c)

        if claims == nil {
            c.AbortWithStatusJSON(http.StatusUnauthorized, gin.H{"error": "You n
eed to be authenticated to access this resource"})
        }

        if role.AccessLevel < claims.Role.AccessLevel {
            c.AbortWithStatusJSON(http.StatusUnauthorized, gin.H{"error": "You n
eed to have authorization to access this content"})
        }
        c.Next()

    }
}
```

Figura 16-5. Validacion de autorización a recursos del sistema

Fuente: Alejandra Lara

Previene	**Descripción**
API1:2019 Control de acceso a nivel de objeto roto	• Uso de mecanismo de autorización basado en políticas y jerarquía del usuario. • Verificación de acceso del usuario conectado para realizar la acción solicitada.
API5:2019 Falta de control de acceso a nivel función/recurso	• El mecanismo de la aplicación deniega el acceso por defecto, requiriendo permisos explícitos a roles específicos para acceder a cada recurso o función. • Se ejecuta comprobaciones de autorización basadas en el rol del usuario.

Fuente: Alejandra Lara

5.1.6 Control de sesiones

Para control de sesiones se utiliza tokens que son invalidados cada cierre de sesión y caducan en un lapso de tiempo.

api

└── app

    └── models

        └── user.go

```go
// RefreshToken model
type RefreshToken struct {
    ID        int `json:"id" gorm:"PRIMARY_KEY; AUTO_INCREMENT;size:11" `
    Token     string
    User      *User
    UserID    int
    Revoked   bool
    CreatedAt time.Time
}

// NewRefreshToken creates a new refresh token
func NewRefreshToken(u *User) *RefreshToken {
    return &RefreshToken{
        Token:   crypto.SecureToken(),
        User:    u,
        UserID:  u.ID,
        Revoked: false,
    }
}
```

Figura 17-5. Token model

Fuente: Alejandra Lara

api

└── app

    └── api

        └── user.go

```go
// RefreshToken refresh token
func (a *App) RefreshToken(c *gin.Context) {

    params := &refreshTokenParams{}
    if err := c.ShouldBindJSON(params); err != nil {
        c.JSON(http.StatusBadRequest, gin.H{"error": err.Error()})
        return
    }

    if params.RefreshToken == "" {
        c.JSON(http.StatusBadRequest, gin.H{"error": "Required refreshToken"})
        return
    }

    var refreshToken models.RefreshToken
    var user models.User

    if result := a.DB.Where("token = ?", params.RefreshToken).First(&refreshToke
n); result.Error != nil {
        c.JSON(http.StatusBadRequest, gin.H{"error": "Invalid refreshToken"})
        return
    }

    if result := a.DB.Where("id = ?", refreshToken.UserID).First(&user); result.
Error != nil {
        c.JSON(http.StatusBadRequest, gin.H{"error": "Invalid refreshToken"})
        return
    }
```

```
    if refreshToken.Revoked {
        log.Println("Posible abuse attempt")
        c.JSON(http.StatusBadRequest, gin.H{"error": "Invalid refreshToken"})
        return
    }

    tx := a.DB.Begin()

    // Remove old Refresh Token and create a new one
    if result := tx.Where("token = ?", params.RefreshToken).Delete(&models.Refre
shToken{}); result.Error != nil {
        tx.Rollback()
        c.Error(result.Error)
        return
    }

    newRefreshToken := models.NewRefreshToken(&user)
    if result := tx.Create(newRefreshToken); result.Error != nil {
        tx.Rollback()
        c.Error(result.Error)
        return
    }

    token, err := generateAccessToken(&user, time.Second*time.Duration(a.Config.
JWT.Exp), a.Config.JWT.Secret)
    if err != nil {
        tx.Rollback()
        c.Error(err)
        return
    }

    if result := tx.Commit(); result.Error != nil {
        tx.Rollback()
        c.Error(result.Error)
        return
    }

    c.JSON(http.StatusOK, &accessTokenResponse{
        Token:        token,
        TokenType:    "bearer",
        ExpiresIn:    a.Config.JWT.Exp,
        RefreshToken: newRefreshToken.Token,
    })
}
```

```go
func generateAccessToken(u *models.User, expiresIn time.Duration, secret string)
 (string, error) {
    claims := &claims.UserClaims{
        StandardClaims: jwt.StandardClaims{
            ExpiresAt: time.Now().Add(expiresIn).Unix(),
        },
        ID:        u.ID,
        FirstName: u.FirstName,
        LastName:  u.LastName,
        Email:     u.Email,
        Role:      u.Role,
    }

    token := jwt.NewWithClaims(jwt.SigningMethodHS256, claims)
    return token.SignedString([]byte(secret))
}
func (a *App) withUser() gin.HandlerFunc {
    return func(c *gin.Context) {
        claims := mcontext.GetUserClaims(c)

        var user models.User

        if result := a.DB.Where("id = ?", claims.ID).First(&user); result.Error
!= nil {
            c.JSON(http.StatusBadRequest, gin.H{"error": "User not found"})
            return
        }

        var refreshToken models.RefreshToken
        if result := a.DB.Where("user_id =?", claims.ID).First(&refreshToken); r
esult.Error != nil {
            c.AbortWithStatusJSON(http.StatusUnauthorized, gin.H{"error": "Auten
ticación necesaria para acceder a este servicio"})
            return
        }

        c.Set("user", user)
        c.Next()
    }
}
```

Figura 18-5. Manejo de sesion con token

Fuente: Alejandra Lara

Previene		Descripción
A2:2017 Perdida de autenticación	API2: 2019 Autenticación Rota	• Se utiliza tokens para sesiones, se invalida al cierre de sesión y según el lapso de tiempo configurado.

Fuente: Alejandra Lara

5.1.7 Validación de entradas

Una técnica eficaz para prevenir ataques de inyección, es la validación de datos entrantes usando patrones específicos o listas blancas.

Regexp

Es un paquete que implementa una secuencia de caracteres que forma un patrón de búsqueda, utilizada para la validación de patrones de cadenas de caracteres.

api

└— app
    └— **api**
        └— auth.go

```go
var bearerRegexp = regexp.MustCompile(`^(?:B|b)earer (\S+$)`)

func extractBearerToken(c *gin.Context) (string, error) {
    authHeader := c.GetHeader("Authorization")
    if authHeader == "" {
        return "", nil
    }

    matches := bearerRegexp.FindStringSubmatch(authHeader)
    if len(matches) != 2 {
        return "", errors.New("Invalid Authorization Header")
    }

    return matches[1], nil
}
```

Figura 19-5. Validación de datos de entrada

Fuente: Alejandra Lara

Previene		Descripción
A1:2017 Inyección	API8:2019 Inyección	• Validación de datos entrantes en el servidor a través de patrones estrictos, regexp.

Fuente: Alejandra Lara

5.1.8 Frameworks y librerías

Gin-gonic

Es un framework que ejecuta middleware con un excelente rendimiento y muy completo.

React JS

Es una biblioteca Javascript de código abierto que facilita la creación de aplicaciones de una sola página. React ayuda a crear componentes interactivos de manera, por lo cual, se utilizará en el presente proyecto para el desarrollo de la parte front-end.

- ***npm audit***

Es un comando de auditoria genera un informe de vulnerabilidades conocidas con el nombre del paquete afectado, gravedad, descripción de la vulnerabilidad, la ruta, etc. Y, si está disponible, comandos para aplicar parches para estas vulnerabilidades. Se ejecuta automáticamente cuando instala un paquete con *npm install*.

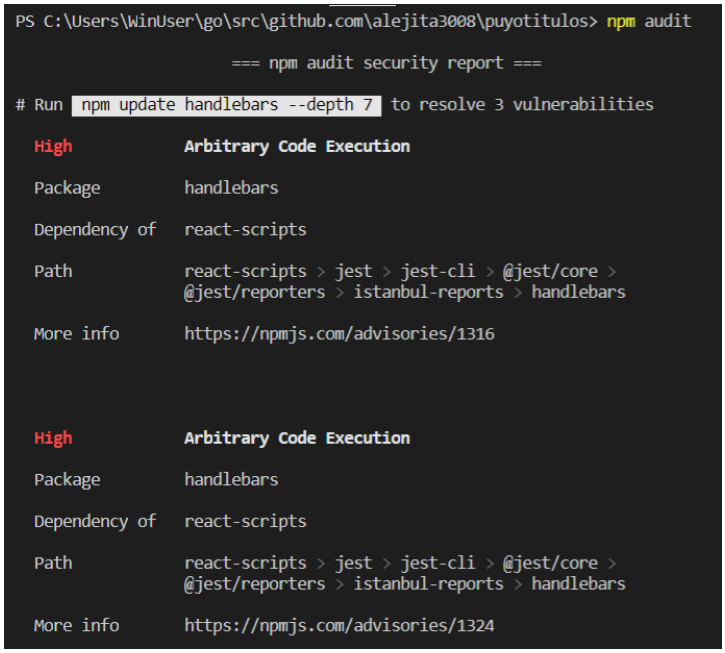


Figura 20-5. Ejecución de comando npm audit

Fuente: Alejandra Lara

- ***npm audit fix***

Este subcomando instala automáticamente actualizaciones para dependencias vulnerables.

```
PS C:\Users\WinUser\go\src\github.com\alejita3008\puyotitulos> npm audit fix
npm WARN bootstrap@4.3.1 requires a peer of jquery@1.9.1 - 3 but none is installed. You must install peer dependencies yourself.
npm WARN ts-pnp@1.1.4 requires a peer of typescript@* but none is installed. You must install peer dependencies yourself.
npm WARN tsutils@3.17.1 requires a peer of typescript@>=2.8.0 || >= 3.2.0-dev || >= 3.3.0-dev || >= 3.4.0-dev || >= 3.5.0-dev ||
.0-dev || >= 3.7.0-beta but none is installed. You must install peer dependencies yourself.

updated 1 package in 6.601s
fixed 3 of 3 vulnerabilities in 905814 scanned packages
```

Figura 21-5. Ejecución del comando npm audit fix

Fuente: Alejandra Lara

- ***Router y Switch***

React Router ayuda a configurar rutas dinámicas en una aplicación, es decir, elegir que renderizar según la locación actual.

- ***Switch***

Este componente, hace que solo se renderice el primer hijo Route o Redirect que coincida con la URL o locación actual.

Es recomendable configurar una página Page 404 cuando un recurso no se ha encontrado.

```
function App() {

  return (
    <Router>
      <Switch>
      <Route exact path='/' component={Consulta} />
      <Route exact path='/titulos' component={Titulos} />
      <Route exact path='/titulosp' component={Titulosp} />
      <Route exact path='/titulospat' component={Titulospat} />
      <Route exact path='/titulospata' component={Titulospata} />
      <Route component={Page404} />
      </Switch>
    </Router>);
}

export default App;
```

Figura 22-5. Configuración de rutas de la aplicación

Fuente: Alejandra Lara

Previene	Descripción
A7:2017 Cross-Site Scripting (XSS)	• React JS por su diseño automáticamente codifican el contenido para prevenir XSS.

Fuente: Alejandra Lara

reCaptcha

El reCaptcha es una herramienta que permite a los sitios web comprobar si un usuario es un humano o un robot. La versión 3 de esta herramienta trabaja de manera oculta mientras el usuario navega por la red, analizará su comportamiento, recogerá señales como el movimiento del ratón o los tiempos entre clics entre varias páginas web, y, si la herramienta de Google sospecha que no es humano, pedirá resolver algún problema para permitir el acceso a una página web.

El sistema de seguridad de Google ayuda a proteger los formularios de la aplicación, previene que malos usuario, bots o similares ingresen información basura, participen en encuestas, proteger información, etc.

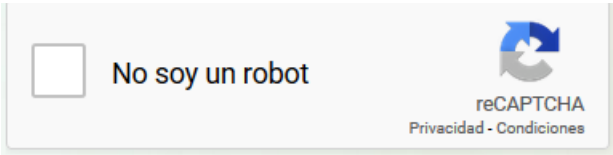


Figura 23-5. Recaptcha de Google

Fuente: Alejandra Lara

Previene	Descripción
API4:2019 Falta de recursos y limitación de proporción	• Limita la frecuencia con la que un cliente puede llamar a la API dentro de un período de tiempo.

Fuente: Alejandra Lara

CONCLUSIONES

- Con el modelo propuesto (prototipo II) se logró reducir los riesgos a un promedio de 2.97%, alcanzando una mejora de un 50.36% respecto al prototipo I, demostrando, así, que existe un aporte con directrices para construir APIs con mayor seguridad.
- Las vulnerabilidades que se deben considerar para el desarrollo de una API más segura son: Control de Acceso a Nivel de Objeto Roto, Autenticación Rota, Filtrado de datos Incorrecto, Falta de Recursos y Limitación de Velocidad, Falta de Control de Acceso a Nivel función/recurso, Asignación Masiva, Configuración de Seguridad Incorrecta e Inyección.
- Muchas empresas tienen poco presupuesto o carecen del mismo para invertir en pruebas de seguridad de software. En estos casos, el uso consciente las guías OWASP Testing Guide, OWASP Top 10 y OWASP API Security Top 10, aportan al desarrollo de código seguro ayudando a prevenir problemas de vulnerabilidades de aplicaciones.
- La aplicación fue construida con paquetes y librerías seguras al momento del desarrollo, pero, al finalizar el proyecto se detectaron en dos paquetes nuevas vulnerabilidades que fueron corregidas al actualizarlos, evidenciando la importancia de las actualizaciones y monitoreo.
- Entre tantos elementos atrás de una aplicación web, hay un sin número de causas y factores por las que una aplicación puede fallar, por esto es importante las actualizaciones y el monitoreo permanente.

RECOMENDACIONES

- Se recomienda el uso del modelo para la implementación de APIs REST utilizando el lenguaje de programación GO, ya que utilizando este modelo se ha comprobado una importante reducción en los riesgos a un promedio de 2.97%.
- Existen proyectos con lógicas de negocios muy variadas, la adopción de herramientas y funciones de seguridad va a depender de las necesidades de cada caso. Es importante definir el ámbito, requisitos de seguridad, la información que almacena, regulaciones y políticas del país, entre otros aspectos, con el fin de adoptar las mejorares herramientas y soluciones de seguridad para el software. Entender el flujo de datos y la relación entre los recursos de la API es crucial para crear una colección de pruebas adecuadas.
- Mantener actualizado todo el software que hace parte de la aplicación, instalar parches y no utilizar software que ya no tenga soporte o esté descontinuado.
- Se debe tener claro que la seguridad es un proceso, no un producto. Se debería usar estándares y/o buenas prácticas durante todo el ciclo de vida del software, monitorear y dar mantenimiento para revisar y mejorar las políticas establecidas.
- Las plataformas de versionamiento, permiten que otros profesionales aporten al mismo. Además, herramientas como Github, detecta las versiones del software del proyecto que contienen CVE y las notifica. Finalmente, es importante actualizarse constantemente en contenido especializado de seguridad, indagar sobre vulnerabilidades en las bases de datos como CVE y NVD y sobre todo nunca parar de investigar.

BIBLIOGRAFÍA

- **[1] ARMENTANO, L.** (2017). Buenas prácticas para el Diseño de una API RESTful Pragmática. https://elbauldelprogramador.com/buenas-practicas-para-el-diseno-de-una-api-restful-pragmatica/. 2019-08-09.

- **[2] Azaustre, C. (2015)**. ¿Qué es la autenticación basada en Token? Recuperado de https://carlosazaustre.es/que-es-la-autenticacion-con-token. 2019-09-19.

- **[3] Blancarte, O. (2017)**. SOAP vs REST ¿cuál es mejor? Recuperado de https://www.oscarblancarteblog.com/2017/03/06/soap-vs-rest-2/. 2019-10-02.

- **[4] B-SECURE**. (2019). Herramientas y principios para la seguridad de aplicaciones ¿Qué funciona y que...no tanto? Recuperado de https://www.b-secure.co/blog/herramientas-y-principios-para-la-seguridad-de-aplicaciones. 2019-08-09.

- **[5] Geeky Theory.** (2019). Qué es una API REST y para qué se utiliza. Recuperado de https://geekytheory.com/que-es-una-api-rest-y-para-que-se-utiliza. 2019-08-09.

- **[6] HERZOG, P.** (2017). Open Source Security Testing Methodology Manual (OSSTMM). Recuperado de http://www.isecom.org/research/ 2019-08-09.

- **[7] Hildago, L**. (2014). Seguridad en aplicaciones web: una visión práctica. Recuperado de http://e-archivo.uc3m.es/handle/10016/20146. 2020-01-04.

- **[8] Jinzhu (2019)** The fantastic ORM library for Golang, aims to be developer friendly. (n.d.). Recuperado de http://gorm.io/. 2019-11-16.

- **[9] KeepCoding. (2017).** Lenguaje de programación Go y sus características. Recuperado de https://keepcoding.io/es/blog/caracteristicas-lenguaje-de-programacion-go/. 2019-08-09.

- **[10] López Provencio, F. (2015).** Metodologías para el desarrollo de software seguro. Recuperado de http://upcommons.upc.edu/handle/2099.1/24902. 2019-08-09.

- **[11] MANICO**, J., & others. (2016). Owasp Development Guide. Recuperado de https://www.owasp.org/images/b/b2/OWASP_Development_Guide_2.0.1_Spanish.pdf. 2019-08-09.

- **[12] MONAR, J. S.** (2017). Propuesta de un método utilizando técnicas de programación seguras para el desarrollo de aplicaciones web en entorno PHP para mitigar riesgos potenciales de seguridad. http://dspace.espoch.edu.ec/handle/123456789/6749. 2019-08-09.
- **[13] Muñoz, O. (2018)**. Prueba de seguridad en API Rest - Backtrack Academy. Recuperado de https://backtrackacademy.com/articulo/prueba-de-seguridad-en-api-rest. 2019-10-31
- **[14] OWASP**. (2015). OWASP Testing Project. https://www.owasp.org/index.php/OWASP_Testing_Project 2019-08-09.
- **[15] OWASP**. (2017). OWASP Code Review Project. https://www.owasp.org/index.php/Category:OWASP_Code_Review_Project 2019-08-09.
- **[16] OWASP**. (2016). OWASP Testing Guide. https://www.owasp.org/index.php/OWASP_Testing_Guide_v4_Table_of_Contents. 2019-08-09.
- **[17] OWASP.** (2019). OWASP API Security Project. https://www.owasp.org/index.php/OWASP_API_Security_Project. 2019-08-29.
- **[18] REDHAT.** (2018). ¿Qué es una API? https://www.redhat.com/es/topics/api/what-are-application-programming-interfaces. 2019-08-09.
- **[19] Solutions, G.** (2018). Diferencias entre API y servicio Web. Recuperado de https://go4it.solutions/es/blog/diferencias-entre-api-y-servicio-web. 2019-08-09.
- **[20] Ruben, R. (2017)**. Mejores prácticas para la seguridad en APIs. Recuperado de https://ciberseguridad.blog/mejores-practicas-para-la-seguridad-en-apis/. 2020-01-20.
- **[19] PINILLA, J**. Buenas Prácticas para el Desarrollo de Código Seguro. 1–6. 2019-08-09.
- **[20] Postman. (2019)**. Postman. Recuperado de https://www.postman.com/. 2019-08-10.
- **[21] Rosa, J. (2018)**. ¿Qué es REST? Conoce su potencia. Recuperado de https://openwebinars.net/blog/que-es-rest-conoce-su-potencia/. 2019-08-25.
- **[22] Soham Kamani. (2019)**. Implementing JWT based authentication in Golang ? - Soham's blog. Retrieved November 8, 2019, from https://www.sohamkamani.com/blog/golang/2019-01-01-jwt-authentication/. 2019-11-08

- **[23] Satinfo. (2019)**. Acunetix. Recuperado de https://satinfo.es/web-wp/wp-content/downloads/acunetix/Acunetix_7.PDF. 2019-12-19.
- **[24] Smartbear**. (2019). ReadyAPI. Recuperado de https://smartbear.com/product/ready-api/overview/.2019-10-01
- **[25] Team, Z. D. (2019)**. ZAP. Recuperado de https://www.zaproxy.org/getting-started/. 2019-12-26.