UNIVERSIDAD CATÓLICA SANTO TORIBIO DE MOGROVEJO

ESCUELA DE POSGRADO

Metodología de seguridad de software para mejorar la seguridad del software como servicio en empresas de desarrollo

TESIS PARA OPTAR EL GRADO ACADÉMICO DE MAESTRO EN INGENIERÍA DE SISTEMAS Y COMPUTACIÓN CON MENCIÓN EN DIRECCIÓN ESTRATÉGICA DE TECNOLOGÍAS DE INFORMACIÓN

AUTOR

Jorge Sanchez Barrueto

ASESOR

Gregorio Manuel León Tenorio

https://orcid.org/0000-0002-9650-4427

Chiclayo, 2025

Metodología de seguridad de software para mejorar la seguridad del software como servicio en empresas de desarrollo

PRESENTADA POR

Jorge Sanchez Barrueto

A la Escuela de Posgrado de la
Universidad Católica Santo Toribio de Mogrovejo
para optar el grado académico de

MAESTRO EN INGENIERÍA DE SISTEMAS Y COMPUTACIÓN CON MENCIÓN EN DIRECCIÓN ESTRATÉGICA DE TECNOLOGÍAS DE INFORMACIÓN

APROBADA POR

María Ysabel Arangurí García
PRESIDENTE

Ricardo David Iman Espinoza
SECRETARIO

Gregorio Manuel León Tenorio
VOCAL

Dedicatoria

A mi familia, que siempre ha estado a mi lado impulsándome a seguir adelante.

A mí, por cada esfuerzo y sacrificio realizado, por cada pequeña victoria alcanzada, son el reflejo de un compromiso profundo conmigo mismo al forjar un carácter que se sostiene en la voluntad de superación.

Agradecimientos

Al asesor Mgtr. Gregorio León Tenorio por trasmitir sus conocimientos y experiencia profesional para lograr una solución propuesta eficiente, a todos los colegas que me apoyaron en el presente estudio y a los expertos por su predisposición y buena voluntad para evaluar y enriquecer el presente trabajo de investigación.

Metodología de seguridad de software para mejorar la seguridad del software como servicio en empresas de desarrollo

INFORME DE ORIGINALIDAD

18%	16%	3%	6%
INDICE DE SIMILITUD	FUENTES DE INTERNET	PUBLICACIONES	TRABAJOS DEL ESTUDIANTE

FUENTES PRIMARIAS

1	repositorio.ug.edu.ec Fuente de Internet	3%
2	www.laravel-enlightn.com Fuente de Internet	2%
3	tesis.usat.edu.pe Fuente de Internet	2%
4	www.cpnnetsecurity.com Fuente de Internet	1%
5	repositorio.ucenfotec.ac.cr Fuente de Internet	1%
6	Submitted to Universidad Internacional de la Rioja Trabajo del estudiante	1%
7	www.fundacionsadosky.org.ar Fuente de Internet	1%
8	repositorio.utp.edu.pe Fuente de Internet	1%

Índice

Resumen

Esta investigación se realizó con el propósito principal de crear una metodología de seguridad de software que permitió mejorar la seguridad del mismo en un proyecto de software terminado de una empresa de desarrollo que ofrece su software como servicio y desarrolla para su propio uso, lo cual se realizó a través de un caso de estudio en el que se obtuvo una reducción del 98,5 % de las vulnerabilidades identificadas. La metodología se basó en los modelos de madurez BSIMM y SAMM. Se extrajeron las actividades relacionadas a la seguridad de software de ambos modelos y se integraron las más idóneas para dar solución al problema utilizando el método de Pardo. Posteriormente, expertos evaluaron los elementos de la metodología en base a criterios como la coherencia, suficiencia, relevancia y claridad, logrando demostrar su validez tras medir la confiabilidad usando Alfa de Cronbach, así como la concordancia entre los expertos mediante W de Kendall. Por último, se demostró la utilidad de la metodología resultante al ser evaluada por quienes estuvieron en contacto con la misma.

Palabras clave: Seguridad de software, ciclo de vida de desarrollo de software (SDLC)

Abstract

This research was carried out with the main purpose of creating a software security methodology that allowed improving its security in a finished software project of a development company that offers its software as a service and develops for its own use, which was done through a case study in which a 98,5 % reduction of the identified vulnerabilities was obtained. The methodology was based on the BSIMM and SAMM maturity models. The activities related to software security from both models were extracted and the most suitable ones were integrated to solve the problem using the Pardo method. Subsequently, experts evaluated the elements of the methodology based on criteria such as coherence, sufficiency, relevance and clarity, demonstrating its validity by measuring reliability using Cronbach's alpha and inter-expert agreement using Kendall's W test. Finally, the usefulness of the resulting methodology was demonstrated when evaluated by those who had used it.

Keywords: Software security, software security life cycle (SDLC)

1. Introducción

La velocidad por encima de la seguridad parece ser la consigna de muchas empresas que producen *software*, y es que el 85 % de sus aplicaciones tienen alguna vulnerabilidad. Así pues, el 13 % de las aplicaciones tienen al menos una falla de muy alta severidad, lo que significa que representan un grave riesgo para el negocio si se aprovechan. Incluso, como una especie de plaga, se observa año tras año que aparecen las mismas vulnerabilidades en el código, como las fallas altamente explotables de *Cross-Site Scripting* en casi el 49 % de ellas, y la inyección de SQL (lenguaje de consulta estructurado) en casi el 28 % del *software* probado. Por consiguiente, este problema sucede por igual en todos los sectores y en empresas de todos los tamaños a nivel global [1]. Por su parte, "los errores y las debilidades en el *software* son comunes, es decir, el 84 % de las brechas de *software* explotan vulnerabilidades en la capa de aplicación donde entre un 70 % a un 80 % de los ataques exitosos suceden por vulnerabilidades conocidas" [2].

La seguridad no debe verse como una ocurrencia tardía o el paso final a la entrega del producto *software*; en su lugar debe integrase en todo el proceso del desarrollo de *software*. Para entregar este de manera segura, las prácticas de seguridad deben evolucionar más rápido que las técnicas utilizadas por los agentes maliciosos, como, por ejemplo, los piratas informáticos, que cada vez buscan comprometer los sistemas de manera más sofisticada y en cantidad. Debido a que el *software* es uno de sus medios para acceder a la infraestructura de las empresas, en el peor de los casos afecta a miles de clientes, como sucedió con SolarWinds y Codecov a inicios de 2020, en donde comprometieron el sistema de compilación y la secuencia de comandos *bash uploader*, respectivamente [3]. Ese mismo año, se expusieron más de 22 000 millones de registros de información personal confidencial o datos comerciales, sin mencionar el daño incalculable a la reputación y a la confianza [4]. Es por ello que la industria debe pasar de un enfoque preventivo a uno de diagnóstico, donde los equipos de desarrollo de *software* deben asumir que sus sistemas ya se encuentran comprometidos e incorporar la seguridad en su cadena de suministro [3].

Según [5], el sector del *Software* y Servicios de Informática a nivel nacional lo conforman 400 empresas aproximadamente, de las cuales la mayor parte son micro y pequeñas empresas, es decir, el 63 % y 27 % del total, respectivamente. No obstante, existen organizaciones de mayor tamaño como las multinacionales SAP, Oracle, IBM, entre otras. Esta industria se concentra mayormente en la capital del Perú (Lima), aunque también se encuentran empresas en Piura, Lambayeque, Huánuco, Arequipa, Junín, Cusco y Tacna

(incluida la zona franca). Este sector abarca tanto el desarrollo a medida como la prestación de servicios. El primero ofrece programas y soluciones informáticas bajo demanda y el segundo especializadas. Ahora bien, los principales tipos de *software* ofrecen soluciones estandarizadas para procesos específicos, especializadas para industrias determinadas, *software* embebido en un soporte físico y desarrollo a medida de las necesidades del cliente.

Al respecto, PROMPERÚ [5] señala que entre las soluciones más destacadas se encuentran, por ejemplo, soluciones transaccionales, plataformas de comercio electrónico y aprendizaje en línea, *software* para gestión (control de proveedores, manejo de activos), automatización de procesos y sistemas de control, sistemas de gestión relacionados con la salud (hospitalaria, farmacéutica, laboratorios, consultorios), seguridad informática (tecnología de cadena de bloques, manejo de contraseñas), *software* de realidad aumentada, *software* para tecnología vestible (*wearables*), generación y optimización de rutas de distribución, sistemas de autoconsulta, inteligencia artificial para asistencia y comunicación con los clientes, integración de sistemas, robots y asistentes virtuales. Todas ellas tienen como clientes principales a empresas locales de minería, servicios financieros, telecomunicaciones, salud, *retail*, turismo y administración pública.

Según el análisis realizado a las tres empresas que conforman el caso de estudio (véase ANEXO 2), se identificó que estas construyen principalmente aplicaciones web que están alojadas en la nube, de las que manifiestan haber experimentado problemas de seguridad en cuanto a la manipulación de la lógica de negocio de sus aplicaciones, así como sobrecargas de tráfico de fuerza bruta o que incluso aún no lo saben porque no lo han descubierto. Esto se debe a que en su mayoría por desconocimiento, no cuentan con una seguridad perimetral como cortafuegos de aplicaciones web o sistemas de detección de intrusos o de prevención de estos, sumado a que no aplican estándares de codificación segura ni realizan un análisis de seguridad profundo al código fuente del *software* porque presentan como principal desafío la velocidad o agilidad para implementar mecanismos o procesos de seguridad o la integración de herramientas que faciliten esta tarea en las diferentes etapas del SDLC (ciclo de vida del desarrollo de *software*).

Luego de haber presentado los problemas relacionados con la seguridad del *software*, se ha planteado el siguiente problema de investigación: ¿Cómo mejorar la seguridad en el *software* como servicio que producen las empresas de desarrollo? Asimismo, se planteó la siguiente hipótesis: La implementación de una metodología de seguridad de *software* mejorará significativamente la seguridad del *software* como servicio en empresas de desarrollo que la apliquen. Por consiguiente, el objetivo general de la tesis es el de crear una

metodología para mejorar la seguridad del *software* como servicio. Para conseguir ello, los objetivos específicos fueron los siguientes:

- Armonizar modelos que sirven como base para la presente investigación.
- Realizar un piloto de la aplicación de la metodología mediante caso de estudio.
- Validar la metodología propuesta para mejorar la seguridad del *software* en función de las fases.
- Validar la utilidad de la metodología de seguridad del *software*.

La presente tesis tiene justificación tecnológica, porque esta metodología contribuirá a identificar y escoger las herramientas necesarias para ayudar en la detección de vulnerabilidades del *software* que brindan como servicio. Desde el punto de vista económico, porque, por un lado, trae mayor valor al *software* entregado, reputación y, con ello, beneficios significativos para las empresas en términos de costos y velocidad, puesto que remediar problemas de seguridad en etapas anteriores es menos costoso que cuando se está en la etapa de implementación dado que evita retrasos. Por otro lado, porque aparecen nuevos problemas cuando se implementan controles de seguridad después de que los sistemas se encuentran funcionando y madurando. En cuanto a lo científico, este estudio sirve como antecedente para investigaciones posteriores, relacionadas con la problemática de estudio y la aplicabilidad de la metodología.

2. Revisión de literatura

2.1. *Antecedentes*

Correa *et al.* [6] elaboran una metodología para evaluar y prevenir problemas de vulnerabilidades de seguridad para aplicaciones web. El proceso de análisis se basa en el uso de técnicas y herramientas que permiten realizar evaluaciones de seguridad de caja blanca y caja negra para validar la seguridad de la aplicación de forma ágil y precisa. Mediante caso de estudio, toma como unidad de análisis al *software* CMS Made Simple. Así pues, el objetivo general es proporcionar *software* eficiente y seguro en términos de recursos y tiempo. El tipo de investigación es experimental, aplicada y el método es el caso de estudio al *software* CMS Made Simple.

Concluyen que la metodología es repetible y flexible, pues, permite su adaptación a cualquier lenguaje, o tecnología web. Primero, se puede utilizar como parte de otras metodologías más generales que no cubren cómo utilizar herramientas de análisis estático y

dinámico en las fases de implementación y prueba del SDLC. Segundo, en base a la implementación de la metodología de evaluación, se encontró que el grado de efectividad fue del 88,9 % basado en la identificación de vulnerabilidades clasificadas como verdaderos positivos y falsos positivos. Tercero, el análisis estático encontró muchas vulnerabilidades de seguridad con pocos falsos positivos 10,69 % identificados y confirmados por la auditoría posterior de cada vulnerabilidad reportada. Cuarto, para auditar los resultados se utiliza el análisis dinámico con comprobación manual, se comprobó el 24,6 % de las vulnerabilidades de seguridad reportadas por el análisis estático y permitió estudiar qué vulnerabilidades se pueden explotar directamente de forma externa. Finalmente, el uso de ambas herramientas permitió que el proceso de pruebas de seguridad sea más productivo y efectivo. Se relaciona con la presente investigación al considerar el mismo método en el análisis del código fuente para la identificación de vulnerabilidades.

Sancho *et al.* [7] presentan una revisión de los modelos de *software* seguro más populares y proponen un modelo de *software* seguro organizado en cuatro áreas de desarrollo (políticas, metodología de desarrollo seguro, supervisión y observatorio) y catorce actividades de seguridad. Se realiza un experimento en una empresa de desarrollo de *software*, mediante caso de estudio, en un proyecto de *software* real. Los resultados se presentan y comparan en dos escenarios de desarrollo: uno clásico con enfoque de seguridad reactiva y otro emergente con enfoque de seguridad preventivo, que lo aplica por defecto en todas las fases del SDLC. En el caso de estudio, la cantidad total de vulnerabilidades se redujo en un 68,42 %, así pues, disminuye su criticidad y el impacto temporal de sus resoluciones y demuestra que el nuevo enfoque emergente proporciona un *software* más seguro.

Mediante caso de estudio, toma como unidad de análisis a dos módulos (M1 y M2) del mismo proyecto de *software* desarrollado por el mismo equipo. En el experimento se consideraron dos escenarios: clásico para el desarrollo del módulo M1 y emergente para el módulo M2. El objetivo general es proponer un nuevo modelo de desarrollo de *software* seguro emergente con enfoque de seguridad preventivo. El caso de estudio se realizó en una fábrica de *software* llamada Viewnext que cuenta con oficinas en España y Portugal, y está compuesta por más de 4 500 profesionales. El proyecto experimental se desarrolló dentro del sector de la industria eléctrica. Las fases de implementación fueron ocho, las tres primeras en un escenario clásico para el M1 y las demás en uno emergente para el M2: desarrollo, evaluación y auditoría, corrección de vulnerabilidades, entrenamiento en seguridad, ecosistema seguro, modelo de integración, evaluación y auditoría; y corrección de vulnerabilidades.

Este artículo presentó el modelo Viewnext-UEx, un enfoque nuevo, preventivo y flexible para desarrollar *software* seguro. Se estudiaron y compararon los modelos más conocidos en desarrollo de *software* seguro, así pues, se identificaron sus mejores prácticas y algunas de sus deficiencias. El modelo incluye las mejores actividades de seguridad de estos modelos conocidos, además de otras tareas de seguridad, corrige las debilidades de los modelos propuestos y sigue un enfoque preventivo. Se prueba con datos reales y demuestra a través de un caso de estudio que el número de vulnerabilidades detectadas se redujo en un 66 %. Asimismo, la criticidad de las vulnerabilidades también se redujo significativamente. Todo ello produce una evidente reducción de costes y tiempos, mucho más acentuada en las fases finales de desarrollo. En líneas generales, se incrementó la seguridad y calidad del *software*, así como la productividad del desarrollo. Finalmente, el artículo permitió identificar indicadores relacionados con la seguridad del *software*, cuyo nivel de seguridad estuvo determinado por el número, tipo y criticidad de las vulnerabilidades detectadas. Además del ecosistema de herramientas para el desarrollo seguro de *software*.

En [8], la garantía de seguridad (SA) es una técnica que ayuda a las organizaciones a evaluar la confianza en que un sistema puede operar de manera correcta y segura. Para fomentar una SA eficaz, debe haber técnicas sistemáticas que reflejen el hecho de que el sistema cumple con sus requisitos de seguridad y, al mismo tiempo, es resistente a las vulnerabilidades y fallas de seguridad. La evaluación cuantitativa de SA aplica técnicas computacionales y matemáticas para obtener un conjunto de métricas de SA para expresar el nivel de seguridad que alcanza un sistema. Dichas métricas están destinadas a cuantificar las fortalezas y debilidades del sistema que se pueden usar para respaldar iniciativas mejoradas de toma de decisiones y planificación estratégica. El uso de métricas para capturar y evaluar la postura de seguridad de un sistema ha ganado atención en los últimos años. Sin embargo, escasos trabajos han descrito cómo combinar la evaluación de SA a la vez que tiene en cuenta el modelado de métricas de SA y el análisis de métricas de SA.

El objetivo de este artículo es complementar el vacío de investigación al desarrollar un enfoque novedoso para el modelado, cálculo y análisis de métricas de garantía o aseguramiento de seguridad, con los siguientes componentes: (1) Un metamodelo de SA cuantitativo para describir la estructura del cálculo de métricas; (2) Un conjunto completo de métricas y los algoritmos de cálculo correspondientes; y (3) La analítica de SA ilustrada para presentar e interpretar métricas. El enfoque metodológico para la evaluación cuantitativa de SA consta de dos partes: (i) Un enfoque de modelado para la evaluación cuantitativa de SA y (ii) Métricas propuestas, así como las reglas de cálculo correspondientes basadas en el modelo

de evaluación de SA. Concluye que el modelo está diseñado de una manera suficientemente genérica que se puede aplicar a cualquier dominio de aplicación, independientemente del tema de la evaluación. El enfoque considera la adaptabilidad y la precisión de las métricas de SA con respecto al dominio de la aplicación y el contexto organizacional. Finalmente, el artículo sirvió como base para tener en cuenta ciertas métricas para la evaluación de garantía de seguridad en la fase de gobierno para mejorar la toma de decisiones, priorizar iniciativas y recursos.

Khan *et al.* [9] realizaron una revisión sistemática de la literatura (SLR) para clasificar los estudios importantes y aprender sobre los riesgos y las prácticas de seguridad del *software* para que los métodos de desarrollo de *software* seguro puedan diseñarse mejor. Así pues, identificaron 145 riesgos de seguridad y 424 mejores prácticas que ayudan a las organizaciones de desarrollo de *software* a administrar la seguridad en cada fase del SDLC. Para lograr un SDLC seguro, este estudio prescribió diferentes actividades de seguridad, que deben seguirse en cada una de las seis fases del SDLC. La integración exitosa de estas actividades minimiza el esfuerzo, el tiempo y el presupuesto al mismo tiempo que ofrece aplicaciones de *software* seguras. Los hallazgos de este estudio ayudan a las organizaciones de desarrollo de *software* a mejorar el nivel de seguridad de sus productos de *software* y también a mejorar su eficiencia de seguridad. Esto también aumenta la conciencia del desarrollador sobre las prácticas de desarrollo seguras.

Toma como unidad de análisis a los artículos científicos relacionados con la ingeniería de *software* seguro, que proporcionen al menos un riesgo o práctica relevante para las especificaciones de seguridad del proceso de desarrollo de *software*, el diseño, el código, las pruebas y seguridad del mantenimiento; de las bases de datos Sciencedirect, IEEE Xplore, ACM *Digital Library*, Wiley *Online Library,* Springer Link y el motor de búsqueda Google Scholar, que son de uso frecuente en estudios de SLR y de mapeo sistemático en la disciplina de ingeniería de *software*. El objetivo general es aprender sobre los riesgos y las prácticas de seguridad del *software* para que los métodos de desarrollo de *software* seguro puedan diseñarse mejor. Para lo cual, consideraron estudios publicados en inglés desde 2000 hasta 2020. En base a criterios de inclusión, exclusión y evaluación de la calidad, se seleccionaron un total de 121 estudios. El método de investigación del mapeo sistemático sigue las pautas para realizar las SLR en ingeniería de *software* de Kitchenham y Charters. Las fases principales para la realización de este fueron tres: planificación de la revisión, realización de la revisión e informe de la revisión.

Concluyen que asegurar los sistemas de *software* en las fases posteriores al desarrollo es insuficiente y se requieren con urgencia mejores formas y medios para asegurar los sistemas de *software* en las primeras etapas. Para incorporar la seguridad en el SDLC general, realizan una revisión detallada de la literatura e identifican 145 riesgos de seguridad y 424 mejores prácticas que ayudan a las organizaciones de desarrollo de *software* a administrar la seguridad en cada fase del SDLC. En este estudio se especificaron las actividades importantes a seguir durante el SDLC para crear *software* seguro. Las acciones especificadas se incorporan con éxito en cada fase del SDLC, lo que reduce el esfuerzo, el tiempo y el presupuesto, al mismo tiempo que ofrece un *software* seguro. Este esfuerzo debería ayudar a las empresas de desarrollo de *software* a aumentar el nivel de seguridad de sus productos y a mejorar su rendimiento de seguridad. Ello también aumentará la comprensión del desarrollador sobre los métodos de desarrollo seguro. Finalmente, el artículo permitió conocer más riesgos y prácticas existentes en la literatura, las cuales sirvieron para la presente investigación.

Moyo y Mnkandla [10] proponen una metodología ágil de desarrollo de *software* seguro que promueve la calidad y la seguridad en los productos de *software* de los desarrolladores individuales o independientes. Integran prácticas de calidad con prácticas de seguridad ligeras para producir prácticas de desarrollo de *software* seguras y ágiles. Asimismo, extraen prácticas de calidad de un marco de desarrollo de *software* individual diseñado en un estudio anterior propio del autor, mientras que las prácticas de seguridad se extrajeron de metodologías ligeras existentes. Adaptaron el algoritmo de Keramati y Mirian-Hosseinabadi para integrar los dos conjuntos de prácticas y cuidaron de mantener un grado óptimo de agilidad en la metodología objetivo. Por último, evalúan la utilidad de la metodología resultante a través de un caso de estudio y demuestran que la metodología propuesta se puede utilizar para crear productos de *software* seguros y de calidad sin comprometer la agilidad de la metodología.

Mediante un caso de estudio, toman como unidad de análisis a estudiantes participantes de un entorno académico y evalúan sus percepciones después de usar la metodología propuesta, que las recopilan a través de discusiones de grupos focales y análisis de documentos. Estos datos recopilados se analizaron cualitativamente. Así pues, el objetivo general fue diseñar una metodología de desarrollo de *software* (SDM) que promueva la calidad del *software* en un entorno individual, para lo cual abordaron la deficiencia de los métodos ágiles en este tipo de entornos. La metodología utilizada para diseñar el Secure-SSDM propuesto fue *Design Science Research* (DSR), ya que buscaron diseñar una solución

satisfactoria para integrar prácticas de seguridad y calidad con la finalidad de integrar la calidad en el proceso de desarrollo de *software*. Este ciclo DSR consta de (1) Identificación del problema; (2) Definición de los objetivos de la solución; (3) Diseño y desarrollo; (4) Demostración de la solución; (5) Evaluación de la solución; y (6) Comunicación de resultados.

Concluyen que su principal contribución al entorno de desarrollo de *software* en solitario o independiente es la introducción de prácticas de promoción de la seguridad a las prácticas de calidad existentes. Contribuyen a la teoría con la adición de prácticas de seguridad a la base de conocimientos de desarrollo de *software* en solitario. Mediante la adaptación del algoritmo Keramati y Mirian-Hosseinabadi, se centraron en incorporar prácticas ligeras que son ejecutables por un individuo con el mínimo esfuerzo. En lo que respecta a la práctica, proponen la metodología Secure-SSDM, que puede ser utilizada por un individuo en el desarrollo de productos de *software* de tamaño pequeño a mediano. En consecuencia, demuestran exitosamente la utilidad de esta para crear aplicaciones de *software* seguras y de alta calidad en los sectores de educación, salud, gobierno y negocios. Finalmente, el artículo permitió conocer las formas de validación de la utilidad y usabilidad de una metodología a través de un caso de estudio y otros ejemplos que han demostrado la popularidad de estos en la evaluación de nuevos métodos.

Según [11], recientes investigaciones han demostrado que el modelado de amenazas puede proporcionar una base para la creación de *software* sin defectos que puede soportar cualquier ataque potencial. El modelado de amenazas describe una amenaza sujeta a un sistema y el daño que podría surgir en las vulnerabilidades. Las vulnerabilidades surgen como fallas en la especificación o el diseño de los requisitos o dan como resultado una implementación incompleta o arrojan varios errores en la etapa de prueba. Insiste en que los problemas de seguridad no deben considerarse más como requisitos no funcionales y aislados en una sola fase del SDLC. Estos problemas de seguridad pueden identificarse con el uso del modelo de amenazas en casi todas las fases de SDLC. El estudio realizado fue de tipo descriptivo.

Concluye que existen diversos enfoques de ingeniería de *software* que se siguen para desarrollar *software,* como, por ejemplo, el modelado en cascada, el modelo en espiral, el modelo de prototipos, el modelo de desarrollo rápido de aplicaciones y el modelo incremental. Aunque estos son eficientes, en algún lugar los problemas de seguridad se descuidan y por tanto muchos de los productos fallan en el mercado. Los problemas de seguridad se convierten en una preocupación en el ciclo de vida del desarrollo de *software* seguro

(SSDLC). Por tanto, describió la necesidad de implementar el modelado de amenazas como una metodología de análisis de seguridad y destacó su importancia en cada fase del SSDLC. Finalmente, el artículo sirvió como base para la presente investigación para ayudar a conocer y a poder determinar el modelado de amenazas a utilizar.

Según [12], su investigación sirvió para descubrir las mejores prácticas que podrían implementarse para mejorar la seguridad en desarrollo y operaciones (DevOps). Consideraron las soluciones proporcionadas en los artículos de investigación a los que se hizo referencia, así como los resultados de estos artículos de investigación y las revisiones. A partir de los datos recopilados, se realizó una revisión descriptiva de la literatura para desarrollar todos los efectos posibles, las prácticas más ampliamente utilizadas para implementar desarrollo, seguridad y operaciones (DevSecOps) y los desafíos que podrían enfrentarse durante la implementación del mismo. Mediante caso de estudio, toma como unidad de análisis a tres empresas basadas en tecnologías de la información que siguen prácticas DevOps, han implementado DevSecOps en diferentes niveles y siguen varias prácticas para el mismo.

El objetivo general es descubrir las mejores prácticas que se podrían seguir para garantizar la seguridad en DevOps. El método utilizado inició con una revisión de la literatura en las bases de datos de las bibliotecas digitales IEEE Xplore, Elsevier y ACM para comprender las implicaciones y los antecedentes generales de DevSecOps, incluida su importancia, técnicas de implementación y desafíos. Primero se buscaron individualmente las palabras clave DevOps y DevSecOps, y luego se buscaron las siguientes frases clave: 1) DevOps y seguridad; 2) DevOps y limitaciones; 3) DevSecOps e implementación. Dicha investigación se llevó a cabo en dos partes. En la primera, se realizó la recopilación de datos mediante un caso de estudio a tres empresas que han implementado DevSecOps. En la segunda, un análisis de datos descriptivo que se llevó a cabo producto de combinar los resultados de la revisión de la literatura con un caso de estudio.

Concluye que los desafíos más comunes que cualquier organización puede presentar al implementar DevSecOps se clasifican en tres áreas: velocidad, colaboración e integración. En cuanto a las prácticas más comunes y efectivas, afirma que pueden cubrirse al considerar indicadores principales. El primero es el entrenamiento, que puede impulsar el grado de protección de la seguridad. Este debe cubrir la importancia de la seguridad, las contribuciones individuales que se espera que mantengan a esta y la impartición del conocimiento de los sistemas de información que la organización impulsa. Como segundo indicador, la integración de herramientas o sistemas de seguridad, porque, realizada la integración con los sistemas originales, se debe realizar un seguimiento y reparación continuo durante todo el proceso de

desarrollo. El tercer indicador consiste en tomar medidas preventivas, que implican asegurar a todos los componentes como códigos fuente, bases de datos, redes y entorno físico, por ejemplo, a través de controles, codificación segura y filtrado de tráfico para mitigar el riesgo de que la seguridad se vea comprometida. En resumen, estos desafíos pueden prevalecer, pero la seguridad puede preservarse mediante estas mejores prácticas o la combinación de las mismas. Finalmente, el artículo sirvió para conocer la forma en que se desarrolla un caso de estudio, así como el propósito de este y sus consideraciones.

En [13], proporciona una descripción general de las tendencias en el desarrollo de *software* seguro para ayudar a reconocer aquellos temas que han sido estudiados ampliamente y aquellos que aún necesitan serlo. Toma como unidad de análisis a los artículos científicos de desarrollo de *software* seguro de las bases de datos Web of Science, IEEE Xplore, Scopus y ACM *Digital Library* que son de uso frecuente en estudios de mapeo sistemático en la disciplina de ingeniería de *software*. El objetivo general es proporcionar un panorama general y estructurado de las tendencias en el desarrollo de *software* seguro. Para ello, realizan un estudio de mapeo sistemático con estrategias de búsqueda PICo (población, interés y contexto), donde consideran estudios publicados en inglés desde 2014 hasta 2019 y seleccionan 528 artículos válidos para su revisión. El método de investigación del mapeo sistemático sigue las pautas para realizar SLR en ingeniería de *software* de Kitchenham y Charters; y las pautas para estudios de mapeo sistemático en ingeniería de seguridad de Felderer y Carver. Los pasos para la realización de este fueron siete: planificación del estudio, búsqueda de estudios, selección de estudios, evaluación de la calidad de los estudios, extracción de datos y clasificación, análisis e informe.

Concluyen que en la literatura se ha reportado que los enfoques de desarrollo de *software* seguro tienen debilidades o son poco o nada accesibles para su uso en la industria del *software*. En este estudio descubrieron que los temas más reportados son: la seguridad en la construcción de *software* o codificación segura, la investigación de evaluación en seguridad, los estudios de casos, las vulnerabilidades en las aplicaciones web, la necesidad de asistencia y capacitación en seguridad para los desarrolladores y la propuesta de nuevos modelos de seguridad de *software*. Los temas que se mencionan con menos frecuencia son: principios de seguridad, técnicas y métodos ágiles, modelos populares de desarrollo de *software* seguro (OWASP o Microsoft SDL) y técnicas de inteligencia artificial. En cuanto a los principales hallazgos en la etapa de requisitos de seguridad, los más frecuentemente reportados son elicitación y análisis de requisitos de seguridad y la técnica del caso de uso indebido (*misuse case)*. En el diseño de *software* seguro son la seguridad en el desarrollo de *software* basado en

componentes, el modelado de amenazas y los patrones de seguridad. En la construcción de código seguro, el análisis de código estático, la detección de vulnerabilidades y la base de datos nacional de vulnerabilidades (NVD). En la etapa de pruebas de seguridad del *software*, el escaneo de vulnerabilidades y las pruebas de penetración.

Por último, concluyen que existe una diversidad de metodologías, modelos y herramientas con objetivos específicos en cada fase del desarrollo de *software* seguro, por ende, para conocer las características de un método, técnica o herramienta de desarrollo de *software* seguro, es necesario profundizar en el estudio mediante la realización de revisiones sistemáticas. Este estudio además de ayudar a conocer los temas más reportados en el desarrollo de *software* seguro, ayudó a conocer modelos de desarrollo de *software* seguro, los problemas que tienen los desarrolladores respecto al desarrollo de *software* seguro, así como a identificar algunas de las causas principales de las vulnerabilidades en el *software* y la necesidad de un modelo integral que adapte las actividades de seguridad al proceso de desarrollo de *software* el cual comprende la seguridad de los requisitos, el diseño, la construcción y las pruebas del *software*. Por ello se requieren estudios que muestren las tendencias del desarrollo de *software* seguro en metodologías, notaciones, herramientas y técnicas.

Según [14], realiza un estudio sobre ataques inyección, donde presenta la forma segura para desarrollar aplicaciones web y contrarrestar este tipo de ataques. Por un lado, menciona que las empresas en su necesidad de tener la información en el más corto plazo posible, descuidan la seguridad al construir aplicaciones. Asimismo, los controles de seguridad no son implementados o son configurados de forma errónea y no son evaluados en su totalidad. Por ello los sistemas son vulnerables a diversos tipos de ataques. Por otro lado, gran parte de las vulnerabilidades se producen por errores básicos y comunes de programación, que conllevan a problemas de seguridad más grandes que generan impactos negativos en la economía de dichas empresas. El estudio incluye pautas de las mejores prácticas de SANS, OWASP y CVE, componentes del estándar ISO 27001, COBIT, modelos de calidad de *software*, entre otros que aseguran la protección contra ataques de inyección.

Toma como unidad de análisis a un grupo de personas relacionadas al desarrollo de aplicaciones web. El objetivo general es proponer las mejores prácticas de desarrollo contra ataques de inyección a sistemas web para los involucrados en la construcción de soluciones informáticas. Se realiza un estudio de tipo explicativo y diseño experimental, este último de tipo cuasi experimental para realizar la validación de hipótesis. Utiliza dos técnicas: la encuesta mediante un cuestionario estructurado, para analizar, medir y evaluar el

conocimiento sobre las buenas prácticas en el desarrollo de proyectos web que tienen las personas involucradas; y la observación en un laboratorio de *testing* mediante código básico y la herramienta SQLMap, para validar si en la actualidad aún persiste el error de malas prácticas contra ataque de inyección por parte de los involucrados.

Concluye que las entidades que desarrollan *software* deben implementar y cumplir políticas basadas en estándares y modelos de aseguramiento de calidad de *software*, así como hacer cumplir estos al adquirir *software* por terceros. Además, adaptar un modelo de calidad de desarrollo de *software* a los requerimientos y recursos de la organización. Se evidencian los errores comunes que suelen cometer los desarrolladores y sus consecuencias; por ello propone pautas de seguridad que debe cumplir tanto en la programación como en la implementación y la configuración correcta de las herramientas que evitan los ataques de inyección. Esta tesis ayudó a identificar algunas herramientas para detección y explotación de vulnerabilidades ante inyección SQL, contramedidas en la codificación y base de datos; y herramientas de defensa. Además de enunciados de buenas prácticas de codificación de *software*.

2.2. *Bases teóricas*

2.2.1. DESARROLLO DE SOFTWARE

2.2.1.1. CICLO DE VIDA DEL DESARROLLO DE SOFTWARE

El ciclo de vida del desarrollo de *software* es “una metodología formal o informal para diseñar, crear y mantener *software* (incluido el código integrado en el *hardware*)” [15]. También se ha definido como el alcance de las actividades asociadas con un sistema, que abarca la iniciación, el desarrollo y la adquisición, la implementación, la operación y el mantenimiento del sistema y, en última instancia, su eliminación que provoca la iniciación de otro sistema [16]. En otros términos, es un proceso estructurado, eficiente y rentable, que establece una serie de pasos que dividen el proceso de desarrollo en tareas que se pueden asignar, completar y medir; y por lo general incluye fases como el análisis, el diseño, la codificación, las pruebas, el despliegue y las operaciones o mantenimiento. De esta manera, los equipos de desarrollo pueden producir *software* de alta calidad y bajo costo, en el menor tiempo posible. Asimismo, el SDLC tiene como objetivo minimizar los riesgos del proyecto mediante una planificación previa que permite que el *software* cumpla con las expectativas del cliente [17].

Existen diversos modelos de SDLC como cascada, espiral, incremental o iterativo, agile, lean, DevOps y otros. Sin embargo, pocos abordan explícitamente aspectos de

seguridad de *software* en detalle y por lo general es necesario integrar este tipo de prácticas al modelo con el que estemos trabajando por tres razones principales: i) Para reducir la cantidad de vulnerabilidades en el *software* publicado; ii) Para reducir el impacto potencial de la explotación de vulnerabilidades no detectadas o no abordadas; iii) Para abordar la causa raíz de las vulnerabilidades y prevenir recurrencias [15].

2.2.1.1.1. *ANÁLISIS*

Dentro de esta fase de análisis se realiza la toma de requisitos por medio de la elicitación, es decir, la transmisión fluida de la información producto de la interacción con los interesados y usuarios, para poder recopilar y documentar los requerimientos del proyecto. A este proceso también se le denomina ingeniería de requisitos. Estos requisitos pueden ser de dos tipos: funcionales y no funcionales. Los funcionales describen lo que hace el sistema, mientras que los no funcionales están referidos a las propiedades emergentes de este, tales como el rendimiento, la disponibilidad, la seguridad, la capacidad de almacenamiento, entre otros; y algunos de ellos son cruciales para que el sistema cumpla su propósito correctamente y se mantenga utilizable [18].

2.2.1.1.1.1. *Métodos para requisitos de seguridad*

a) Casos de uso indebidos

Este es uno de los métodos más utilizados para los requisitos de seguridad [13]. Es lo inverso al caso de uso, ya que describe cómo se ejecuta una acción maliciosa contra el sistema, dicho de otro modo, describe una acción que no debería suceder, mientras que el caso de uso describe en positivo cualquier acción que puede ser realizada en el sistema.

b) SAFECode

Es una guía que proporciona la forma de crear historias de seguridad o requisitos. Se muestra la historia de seguridad con el desglose de las actividades para implementarla de manera segura.

2.2.1.1.2. *DISEÑO*

2.2.1.1.2.1. *Principios de diseño seguro*

Los principios son declaraciones de dirección que gobiernan las selecciones e implementaciones, es decir, proporcionan una base para la toma de decisiones [19]. Saltzer y Schroeder [20] en un artículo de 1974 propusieron ocho principios de diseño para asegurar los

sistemas informáticos, los cuales establecen un alto nivel para los diseñadores de sistemas seguros. Seguirlos estrictamente puede no ser sencillo, sin embargo, es probable que los diseñadores que sigan sus instrucciones creen sistemas que cumplan con los objetivos de proteger la información y resistir a los ataques. Estos principios son relevantes para el diseño de cualquier sistema, ya sea cliente–servidor, un servicio en la nube o un dispositivo de IoT (internet de las cosas). Los detalles de su aplicación pueden variar, por ejemplo, un servicio en la nube puede requerir varios roles administrativos, cada uno con su propio privilegio mínimo, mientras que un dispositivo de IoT requerirá la necesidad de actualizaciones de seguridad y necesitará fallar de manera segura [21]. Aunque estos principios se propusieron hace décadas, han demostrado ser más importantes para los diseñadores de sistemas modernos y son tan precisos como cuando se redactaron inicialmente, por lo que se han utilizado de forma repetida y exitosa en el diseño e implementación de diversos sistemas [22].

Algunos artículos y libros usan términos diferentes, por ejemplo, caracterizan la separación de privilegios más como un control que como un principio. Otros realizan variantes a algunos principios, por ejemplo, la segregación de funciones y la denegación por defecto no son más que la separación de privilegios y los valores predeterminados a prueba de fallas, respectivamente [23]. Por añadidura, existen otros investigadores que han propuesto otros principios que por experiencia han demostrado ser importantes para la seguridad de los sistemas de *software* como la defensa en profundidad, el fallar de forma segura y el diseño para la actualización [21]; y también para reflejar la tecnología y la terminología actual, como la validación de entrada [24]. La aplicación de algunos principios de seguridad dependerá de factores como los sistemas en construcción y el entorno en el que se implementa. Debido a que los principios de diseño no son reglas obligatorias sino pautas o directrices generales, puede que no sea necesario aplicar cada principio en detalle ya sea por su dificultad de implementación con un tipo de sistema, porque no hay necesidad de tal principio o porque no aplica como con el caso de la mediación completa o el diseño abierto; o aplicarlo hasta cierto punto como en el caso de separación de privilegios y la aceptabilidad psicológica [25].

Tabla 1. Principios de diseño seguro

PRINCIPIO DE DISEÑO SEGURO	DESCRIPCIÓN
Economía del mecanismo o simplicidad (*Economy of mechanism*)	Mantener el diseño del sistema lo más simple y pequeño posible, porque un diseño más simple es más fácil de probar y validar.

Valores predeterminados a prueba de fallas (*Fail-safe defaults*)	Basar las decisiones de acceso en el permiso en lugar de la exclusión, es decir, a un usuario se le permite explícitamente el acceso a un recurso en vez de que a un usuario se le niegue explícitamente el acceso a un recurso.
Mediación completa (*Complete mediation*)	Cada acceso a cada objeto debe ser verificado para su autorización, esto es, comprobar todos los accesos sin excepción.
Diseño abierto (*Open design*)	No debe depender de la seguridad por oscuridad.
Separación de privilegios (*Separation of privileges*)	Usar privilegios separados o autorización de múltiples partes, como por ejemplo, dos llaves para desbloquear.
Mínimo privilegio (*Least privilege*)	Cada programa y cada usuario del sistema debe operar utilizando la menor cantidad de privilegios necesarios para completar el trabajo.
Mecanismos comunes mínimos (*Least common mechanisms*)	Minimiza la cantidad de función o mecanismo común a más de un usuario y del que dependen todos los usuarios.
Aceptabilidad psicológica (*Psychological acceptability*)	Es fundamental que la interfaz humana esté diseñada para facilitar su uso, de manera que los usuarios apliquen los mecanismos de protección de forma rutinaria y automática de forma correcta.
Defensa en profundidad (*Defense in depth*)	Diseñar el sistema para que pueda resistir un ataque incluso si se descubre una única vulnerabilidad de seguridad o se pasa por alto una sola característica de seguridad. Puede implicar la inclusión de múltiples niveles de mecanismos de seguridad o el diseño de un sistema para que falle en lugar de permitir que un atacante obtenga el control total.
Fallar de forma segura (*Fail securely*)	Un contrapunto a la defensa en profundidad es que un sistema debe estar diseñado para permanecer seguro incluso si encuentra un error o falla.
Diseño para la actualización (*Design for updating*)	Debido a que es probable que un sistema permanezca libre de vulnerabilidades de seguridad para siempre, los desarrolladores deben planificar la instalación segura y confiable de actualizaciones de seguridad.
Validación de entrada (*Input validation*)	Un formato de entrada inesperado puede hacer que el sistema se comporte de forma imprevista.

Fuente: Adaptación a partir de los artículos "*The protection of information in computer systems"* [20], "*Fundamental Practices for Secure Software Development"* [21], "*Avoiding the top 10 software security design flaws"* [24]

2.2.1.1.2.2. *Modelado de amenazas*

"El modelado de amenazas es una forma de evaluación de riesgos que modela aspectos de los lados de ataque y defensa de una entidad lógica, como un dato, una aplicación, un host, un sistema o un entorno" [26]. Este implica un enfoque proactivo en la gestión del riesgo, es decir, no esperamos a que las amenazas se materialicen para saber cuáles nos pueden afectar, sino que intentamos anticiparnos a los incidentes, analizando cómo se podrían producir [27]. El modelado de amenazas tiene como propósito: i) Comprender el perfil de amenazas de un sistema; ii) Proporcionar una base para un diseño y una implementación segura; iii) Descubrir vulnerabilidades; iv) Proporcionar comentarios sobre el ciclo de vida de la seguridad de la aplicación [28]. Así pues, el objetivo es identificar los activos críticos del sistema, analizarlos, documentarlos y priorizar los problemas de vulnerabilidad del sistema. Asimismo, ayuda a identificar el punto de entrada por el cual un atacante podría ingresar al sistema para atacar y el punto de salida por el cual abandona el sistema después de atacar. Ambos puntos deben identificarse con mucha anticipación durante el diseño de manera que no queden lagunas para que los atacantes invadan el sistema [29]. De hecho, respecto a las aplicaciones, permite al desarrollador plantear estrategias de mitigación para vulnerabilidades potenciales y lo ayuda a concentrar sus recursos en las partes del sistema que más lo requieren [30].

¿Cuándo realizar el modelado de amenazas? Independientemente de la complejidad del *software*, las cuestiones de seguridad deben abordarse tan temprano como sea posible, en cada fase del SDLC. El modelado de amenazas debe ser un proceso iterativo continuo por dos razones: Primero, es imposible identificar todas las amenazas en una sola vez y segundo, los requisitos cambiantes del negocio deben mejorarse y adaptarse a medida que se construye el sistema. El modelado de amenazas no solo se asocia con la fase de diseño, también se puede considerar una parte importante de la fase de requisitos y se puede ejecutar de forma continua en todo el SDLC. En la fase de diseño cubre las vulnerabilidades que podrían introducirse debido a la falta de especificación de requisitos de seguridad. En la fase de codificación, pueden surgir vulnerabilidades debido a una implementación deficiente [29].

c) MITRE ATT & CK

Este marco de trabajo intenta desglosar todo un conjunto de tareas específicas que un atacante debe realizar para lograr su objetivo. Sirve para ayudar a proponer las defensas que serían buenas para contrarrestar esas técnicas en particular que utilizan los atacantes.

d) STRIDE

El modelo de amenazas STRIDE (Modelo de Microsoft) es una forma de garantizar que las aplicaciones no sufran a la suplantación de identidad de sus usuarios, la manipulación de datos, el repudio, la revelación de información, la denegación de servicios y la elevación de privilegios [31]. Identificar los objetivos de seguridad asociados a los tipos de amenazas ayuda a la comprensión de los objetivos del atacante, lo cual permite explorar el componente que necesita una observación minuciosa para protegerlo contra el ataque [32]. En la siguiente tabla 2 se relacionan las amenazas con las propiedades deseables de un sistema que protegen contra ellas [31].

Tabla 2. Tipos de amenazas según modelo STRIDE con sus objetivos, propiedades o controles de seguridad asociados

SIGLA	TIPO DE AMENAZA	DESCRIPCIÓN	OBJETIVO, PROPIEDAD O CONTROL DE SEGURIDAD
S	Suplantación de identidad de un usuario (*Spoofing*)	Ataque para falsificar la identidad, acceder ilegalmente y usar la información de autenticación de otro usuario, como su nombre de usuario y clave.	Autenticación
T	Manipulación de datos (*Tampering*)	Ataque que implica modificar maliciosamente los datos, como los cambios no autorizados a los datos persistentes, es decir, los almacenados en una base de datos o la alteración de los datos que viajan entre dos dispositivos a través de una red abierta como internet.	Integridad
R	Repudio (*Repudiation*)	Las amenazas de repudio se asocian a los usuarios que niegan realizar una acción sin que otras partes tengan alguna forma de demostrar lo contrario, es decir, un usuario realiza una operación indebida en un sistema que no es capaz de rastrear las operaciones prohibidas. El no repudio es lo contrario, es decir, se refiere a cuando un sistema es capaz de contrarrestar las amenazas de repudio.	No repudio o auditabilidad

I	Divulgación de información, filtración o brecha de datos (*Information disclosure*)	Amenaza que involucra la exposición de información a personas no autorizadas, por ejemplo, un usuario capaz de leer un archivo sin haberle otorgado acceso o un intruso capaz de leer datos en tránsito entre dos dispositivos.	Confidencialidad
D	Denegación de servicio (*Denial of service*)	Ataque cuyo objetivo es dejar temporalmente inaccesible, sin servicio o inutilizable el servicio ofrecido por un servidor web a usuarios válidos.	Disponibilidad
E	Elevación o escalamiento de privilegios (*Elevation of privilege*)	Ataque orientado a obtener acceso privilegiado a recursos inaccesibles por defecto o a información no autorizada del sistema y puede comprometer o destruir todo el sistema. Incluye situaciones en las que el atacante penetra efectivamente todas las defensas del sistema y se convierte en parte de este.	Autorización

Fuente: Adaptación a partir de "Ciclo de vida de desarrollo ágil de software seguro" [31], "*Uncover Security Design Flaws using the STRIDE Approach*" [32]

— **Proceso o método de modelado de amenazas STRIDE [33]**

Identificación de activos: Consiste en identificar elementos en posible riesgo como servidores, bases de datos, páginas web, etc.

Visión de la arquitectura: Consiste en identificar qué hace la aplicación. Saber qué tecnologías están involucradas y documentar.

Descomponer la aplicación: Consiste en identificar los límites de confianza, el flujo de los datos y los puntos de entrada.

Identificar amenazas: Consiste en identificar las posibles amenazas para cada sección, usando STRIDE o árboles de ataque.

Documentar amenazas: Consiste en llevar una bitácora de estas amenazas y de cómo contrarrestarlas.

Cuantificar amenazas: Consiste en otorgar una valoración del posible impacto y daño de la amenaza mediante DREAD.

e) OTRAS TÉCNICAS DE REPRESENTACIÓN DE AMENAZAS

— **Árboles de ataque**

Schneier [AA] quien acuñó por primera vez este término, lo propuso como una manera formal de describir la seguridad del sistema basado en diversos ataques, donde el nodo raíz representa el objetivo de un atacante y los nodos hoja representan las diversas maneras por las cuales se podría lograr el objetivo del atacante. Estos nodos se descomponen por relación *AND* y *OR*. Se pueden asignar valores, como, por ejemplo, el costo que necesita ahorrarse para lograr el objetivo o la probabilidad de hacer una tarea. El valor del nodo raíz indica si el objetivo del sistema es vulnerable a un ataque. Las características del atacante deben analizarse para determinar qué parte del árbol de ataque necesita una especial atención del resto de las partes. La ventaja del árbol de ataque es que ayuda a estudiar el sistema desde el punto de vista del atacante y ayuda a analizar el sistema al evaluar el impacto de la aplicación de contramedidas. Las principales limitaciones son: el diseñador y el desarrollador deben tener un conocimiento sólido sobre las características del atacante para modelar el árbol. Además, Schneier no analiza cómo los árboles de ataque podrían vincularse con otros artefactos de desarrollo, como el diseñador, el desarrollador o el evaluador del *software*.

— **Árboles de mitigación**

Guifre Ruiz *et al.* [34] propusieron una nueva estructura de datos conocida como árbol de mitigación para detectar amenazas en los diseños de *software*, similar a los árboles de ataque, pero con una diferencia. Los árboles de ataque se construyen de manera destructiva, mientras que estos se construyen de manera constructiva. El árbol de mitigación tiene el objetivo de mitigar la amenaza determinada y cada rama contiene el conjunto de características o especificaciones de *software*, para las actividades de diseño e implementación necesarias para lograr el objetivo de la raíz. Además, cada característica contiene un costo estimado de llevarla a cabo.

— **Redes de ataque**

J. P. McDermott [35] propuso el enfoque de red de ataque para las pruebas de penetración. Las redes de ataque proporcionan un medio gráfico para mostrar cómo se puede combinar una colección de fallas para lograr una penetración significativa en el sistema. Una red de ataque conserva los beneficios esenciales del árbol de ataque y también proporciona las alternativas y el refinamiento del enfoque del árbol de ataque. Las redes de ataque pueden modelar ataques más sofisticados que pueden combinar varias fallas y se utilizan para organizar el desarrollo de un escenario de ataque plausible.

2.2.1.1.3. *CODIFICACIÓN*

2.2.1.1.3.1. *Buenas prácticas de codificación segura*

Las buenas prácticas de codificación segura son pautas o directrices de codificación que se implementan en el código fuente para evitar o mitigar las vulnerabilidades más comunes de seguridad en el *software*. Para ello, existen estándares bien establecidos como OWASP TOP 10, SEI CERT (Equipo de respuesta a emergencias informáticas), Seguridad y desarrollo de aplicaciones DISA STIG (Guía de implementación técnica de seguridad), entre otros [36], que al implementarlos ayudan a proteger contra los agentes de amenaza que explotan el *software* y a su vez contribuyen a reducir la superficie de ataque que estos suelen atacar con *malware*. Entre los componentes principales que conforman estos estándares encontramos a la validación de entrada de datos, control de acceso, prácticas criptográficas, gestión de errores y registros, el manejo de archivos, el manejo de memoria, protección de datos, entre otros [37], [38].

2.2.1.1.3.2. *Revisión manual de código*

Se revisa y evalúa el código fuente ya sea por el mismo desarrollador u otro con el objetivo de encontrar defectos o vulnerabilidades en este. Esta tarea requiere cierto nivel de conocimientos por parte del evaluador y es un poco más costosa debido al tiempo que toma.

2.2.1.1.3.3. *Revisión automática de código*

Se examina y analiza el código fuente por medio de alguna herramienta aplicando un conjunto de reglas establecidas para encontrar vulnerabilidades conocidas. Muchas de estas herramientas emplean estándares de codificación segura como las anteriormente mencionadas, lo cual facilita el proceso de revisión [36]. Sin embargo, pueden aparecer vulnerabilidades no reales o falsos positivos, que tras ser confirmados mediante el uso de otra herramienta o por la misma experiencia del evaluador, se descartan.

2.2.1.1.4. *PRUEBAS*

2.2.1.1.4.1. *Pruebas de penetración*

Es un método de prueba en el que la seguridad de un programa informático o una red se somete a un ataque simulado deliberado, con el objetivo de encontrar y aprovechar las vulnerabilidades del mismo. Estas son llevadas a cabo por una persona o equipo de personas

externas que tienen poco o ningún conocimiento previo sobre la seguridad del sistema para mostrar los puntos ciegos que los creadores del sistema no tuvieron en cuenta [39].

2.2.1.1.4.2. *Pruebas fuzz*

Es un método de prueba que hace que un programa de *software* consuma datos malformados deliberadamente para ver cómo reacciona este. De tal manera se puede verificar la seguridad de las entradas respecto a la validación de datos, detectar fallas de codificación y determinar qué tan susceptibles son a la explotación de inyección SQL, denegación de servicio y secuencia de comandos entre sitios. Asimismo, tiene como objetivo detectar vulnerabilidades conocidas, desconocidas y de día cero que los agentes de amenaza a menudo aprovechan. Estas pruebas no solo identifican el problema, también la causa y la forma en que un atacante puede interactuar con el *software*, sin tener que confirmar falsos positivos [40].

2.2.1.1.5. *DESPLIEGUE*

Consiste en poner en funcionamiento al *software* y a disposición de los usuarios. Para ello se debe planificar el entorno, considerando las dependencias existentes entre los diversos componentes del *software*. Debido a que es posible que estos funcionen bien independientemente, pero que al integrarlos provoquen problemas de compatibilidad que impidan el normal funcionamiento del *software*. Este despliegue puede realizarse de manera manual o automatizada, dependiendo de la complejidad o de las necesidades de la aplicación [41].

2.2.1.1.6. *OPERACIONES O MANTENIMIENTO*

Abarca el monitoreo, actualización y mantenimiento continuo para que el *software* siga funcionando de manera óptima. En el monitoreo, el equipo de operaciones se mantiene vigilante del funcionamiento del *software*, así como de nuevas vulnerabilidades que surgen y que pueden afectar al mismo, con la finalidad de que al detectarlas se enfoquen los esfuerzos y se reduzca el tiempo de exposición [17].

2.2.2. SEGURIDAD DE SOFTWARE

La seguridad del *software* se ha definido como “el desarrollo e implementación de métodos y procesos para garantizar que el *software* funcione según lo previsto y esté libre de defectos de diseño y fallas de implementación” [21]. A menudo, esta se analiza en relación

con la garantía de *software*, la cual se ha definido como "el nivel de confianza de que el *software* esté libre de vulnerabilidades, ya sea intencionalmente diseñado en el *software* o accidentalmente insertado en cualquier momento durante su ciclo de vida, y que funcione según lo previsto" [42]. Por tanto, la seguridad del *software* engloba lo que hace una organización de desarrollo de *software* para proteger un producto de *software* y los datos críticos asociados de vulnerabilidades, amenazas internas y externas, errores críticos o configuraciones incorrectas que pueden afectar el rendimiento o exponer los datos.

La seguridad del *software* comprende tanto los procesos organizacionales como las capacidades del producto, los cuales son elementos vitales de la seguridad del *software*. Los procesos organizacionales, incluyen estructuras de gobierno, estrategias, orientación y procedimientos claramente definidos que guíen el desarrollo de *software* de una manera que identifique e incorpore objetivos de seguridad a lo largo del ciclo de vida de un producto, proteja la integridad del entorno de desarrollo, aplique recursos para la gestión de incidentes y vulnerabilidades. y gestiona la cadena de suministro que soporta el proyecto de desarrollo de *software*. Las capacidades de seguridad del producto, son aspectos técnicos de productos de *software* específicos que son útiles para permitir que los productos aborden desafíos de seguridad comunes, como la protección de datos, la prevención del acceso o uso no autorizado, el seguimiento de incidentes y vulnerabilidades y la gestión de eventos imprevistos [43].

2.2.2.1. VULNERABILIDADES EN EL SOFTWARE

Una vulnerabilidad en nuestro contexto es una debilidad que puede tener un sistema de información, un procedimiento de seguridad del sistema, un control interno o una implementación que podría ser explotada por una fuente de amenaza [44]. Por consiguiente, una vulnerabilidad en el *software* se ha definido como una falla de seguridad, problema técnico o una debilidad encontrada en el código del *software* que puede ser explotada por un atacante, es decir, una fuente de amenaza [45]. Asimismo, existen diversas organizaciones dedicadas, entre otras actividades, a clasificar vulnerabilidades para diferentes sistemas informáticos, entre ellas tenemos:

2.2.2.1.1. *MITRE*

Es una organización sin fines de lucro que ha desarrollado una lista de más de 900 debilidades comunes de *software* y hardware llamada CWE (Enumeración de debilidades

comunes) que sirve como lenguaje común (por la taxonomía de clasificación relacionada), también como una vara para medir herramientas de seguridad y como línea base para identificar, mitigar y prevenir debilidades. De esta lista se genera el CWE TOP 25 que es la lista de las debilidades de *software* más peligrosas y actuales, la cual ayuda a identificar los errores de programación más usuales y críticos que podrían conllevar a vulnerabilidades de *software* graves [46]. Esta última se basa en los datos de las CVE (Vulnerabilidades y exposiciones comunes) que se encuentran en la NVD (Base de datos nacional de vulnerabilidades del NIST (Instituto Nacional de Estándares y Tecnología) y las puntuaciones del CVSS (Sistema de puntuación de vulnerabilidades comunes) [47].

2.2.2.1.2. *INSTITUTO DE TECNOLOGÍA SANS*

Es un instituto sin fines de lucro que ofrece capacitaciones, recursos y eventos a profesionales actuales o futuros en temas relacionados a la ciberseguridad. Al igual que CWE, SANS proporciona su propia lista del Top 25 de los errores más peligrosos del *software*, en donde cada identificador CWE mostrado en este listado, enlaza al sitio web del CWE relevante donde se encuentra la clasificación, los datos completos, las consecuencias de las vulnerabilidades, el costo de remediación, la facilidad de detección, entre otros [48].

2.2.2.1.3. *OWASP (OPEN WEB SECURITY PROJECT)*

Es una fundación sin fines de lucro que persigue el objetivo de mejorar la seguridad del *software* especialmente web y lo hace poniendo a disposición cientos de herramientas de código abierto y recursos para educar y ayudar a desarrolladores y tecnólogos a proteger la web. Uno de sus proyectos más emblemáticos es el OWASP TOP 10 el cual es un documento de referencia que presenta las categorías de los riesgos, vulnerabilidades o preocupaciones de seguridad más críticos de la seguridad de las aplicaciones web la cual recibe una actualización cada tres años. También existe un TOP 10 para APIs y dispositivos móviles con actualizaciones menos frecuentes [49].

2.2.2.1.4. *NIST (NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY)*

Es un laboratorio de ciencias físicas que está adscrito al Departamento de Comercio de los Estados Unidos que se encarga de promover la innovación y competitividad industrial en dicho país a través del avance de la ciencia, los estándares y la tecnología de medición para

aumentar la seguridad económica y mejorar la calidad de vida de sus ciudadanos [50]. Esta cuenta con una Base de datos nacional de vulnerabilidades (NVD) que proporciona una capacidad de búsqueda muy detallada y vincula a los usuarios con información acerca de vulnerabilidades y parches. Este instituto también cuenta con un *framework* de errores (BF) el cual proporciona una clasificación estructurada, completa, ortogonal e independiente del lenguaje que permite descripciones inequívocas sobre las vulnerabilidades del *software* [51].

2.2.2.2. MODELOS DE MADUREZ DE SEGURIDAD DE SOFTWARE

2.2.2.2.1. *MODELO DE MADUREZ BSIMM*

El Building Security in Maturity Model (BSIMM) [52] es un modelo industrial único que ayuda a las organizaciones a planificar, implementar y medir sus iniciativas de seguridad de *software* (SSI). Proporciona una forma de evaluar el estado actual de SSI, identificar brechas, priorizar cambios y determinar cómo y dónde aplicar los recursos para una mejora inmediata. También funciona como una hoja de ruta para un SSI, es decir, puede identificar sus propias metas y objetivos, para luego consultar el modelo y determinar qué actividades son importantes para aplicarlas en su contexto. Asimismo, en la décima segunda versión, el modelo está compuesto por cuatro dominios, doce prácticas y ciento veintidós actividades; en la que cada práctica cuenta con tres niveles de madurez.

2.2.2.2.2. *MODELO DE MADUREZ SAMM*

El Software Assurance Maturity Model (SAMM) [53] es un modelo flexible creado para ayudar a las organizaciones de cualquier tamaño, que puede ser aplicado en toda la organización, en una unidad de negocio o en un proyecto individual y es independiente del estilo de desarrollo. Asimismo, permite formular e implementar una estrategia de seguridad para el *software*. Este ayuda a

- Evaluar prácticas de seguridad en los *softwares* existentes de la organización.
- Construir programa de seguridad en iteraciones bien definidas.
- Demostrar las mejoras concretas en el programa de garantía de *software*.
- Definir y medir las actividades relacionadas a la seguridad en la organización.

Asimismo, en la segunda versión, el modelo está compuesto por cinco dominios, quince prácticas y noventa actividades; en la que cada práctica cuenta con tres niveles de madurez.

3. Materiales y métodos

3.1. *Tipo y nivel de investigación*

La presente investigación es de tipo experimental. Por consiguiente, el diseño de investigación o diseño experimental elegido es del tipo preexperimental, con diseño con preprueba-posprueba con un solo grupo [54]. En cuanto al nivel de investigación es del tipo aplicada [55]. Debido a que se buscó evaluar el efecto de aplicar la metodología para mejorar la seguridad del *software* como servicio en empresas de desarrollo.

Diagrama del diseño de investigación:

$$G\ O_1\ X\ O_2$$

Donde:

G = Grupo experimental.

X = Metodología de seguridad de *software* (variable independiente).

O_1 = Medición previa del nivel de seguridad del *software* del grupo experimental.

O_2 = Medición posterior del nivel de seguridad del *software* del grupo experimental.

3.2. *Población, muestra y muestreo*

La población identificada fueron los proyectos de *software* de empresas de desarrollo, que lo brindan como servicio, para sí mismas y que están preocupadas por la seguridad del mismo.

3.3. *Operacionalización de variables*

Tabla 3. Operacionalización de variables

VARIABLE	DIMENSIÓN	INDICADOR	DESCRIPCIÓN
Seguridad del *software*	Nivel de seguridad	— Porcentaje de vulnerabilidades resueltas del total de vulnerabilidades	— Es el porcentaje de vulnerabilidades resueltas respecto del total de vulnerabilidades encontradas
		— Porcentaje de verdaderos positivos del total de vulnerabilidades	— Es el porcentaje de verdaderos positivos respecto del total de vulnerabilidades

			encontradas
		— Porcentaje de falsos positivos del total de vulnerabilidades	— Es el porcentaje de verdaderos positivos respecto del total de vulnerabilidades no reales detectadas

3.4. *Técnicas e instrumentos de recolección de datos*

Tabla 4. Técnicas e instrumentos de recolección de datos

TÉCNICAS	INSTRUMENTOS	ELEMENTOS DE LA POBLACIÓN	PROPÓSITO
Encuesta	Cuestionario	Gerentes de proyectos de desarrollo	Conocer la percepción de los gerentes de proyectos de desarrollo respecto de la implementación de la metodología propuesta.
Encuesta	Cuestionario	Gerentes de TI	Conocer la percepción de los gerentes de TI respecto de la implementación de la metodología propuesta.
Observación	Notas de observación	Proyectos de *software*	Se observarán y documentarán los resultados del análisis del *software* para conocer de manera objetiva los indicadores que ayuden a determinar la mejora de la seguridad del mismo.

Tabla 5. Matriz de consistencia

FORMULACIÓN DEL PROBLEMA	METODOLOGÍA DE INVESTIGACIÓN			
¿Cómo mejorar la seguridad en el *software* como servicio que producen las empresas de desarrollo?	TIPO DE INVESTIGACIÓN			
	Tipo: experimental, diseño de investigación: preexperimental (diseño de preprueba-posprueba con un solo grupo), nivel de investigación: aplicada			
OBJETIVO GENERAL	MÉTODO	DESCRIPCIÓN		
Crear una metodología para mejorar la seguridad del *software* como servicio	Bibliográfico			
	Caso de estudio			
HIPÓTESIS	TÉCNICAS	INSTRUMENTOS	ELEMENTOS DE LA POBLACIÓN	PROPÓSITO
La implementación de una metodología de seguridad de *software* mejorará significativamente la seguridad del *software* como servicio en empresas de desarrollo que la apliquen	Encuesta	Cuestionario	Gerentes de proyectos de desarrollo	Conocer la percepción de los gerentes de proyectos de desarrollo respecto de la implementación de la metodología propuesta.
VARIABLE DEPENDIENTE Seguridad del *software* como servicio en empresas de desarrollo	Encuesta	Cuestionario	Gerentes de TI	Conocer la percepción de los gerentes de TI respecto de la implementación de la metodología propuesta.
VARIABLE INDEPENDIENTE Metodología de seguridad de *software*	Observación	Notas de observación	Proyectos de *software*	Se observarán y documentarán los resultados del análisis del *software* para conocer de manera objetiva los indicadores que ayuden a determinar la mejora de la seguridad del mismo.
OBJETIVOS ESPECÍFICOS	DESCRIPCIÓN DEL LOGRO DE LOS OBJETIVOS ESPECÍFICOS			INDICADORES
Armonizar modelos que sirven como base para la presente investigación.	Listar y comparar los pasos de los modelos, *frameworks*, guías existentes; y tomar un criterio en función de la situación problemática identificada para la creación de la metodología.			Rango de cercanía respecto a los pasos de los demás modelos, *frameworks*, guías existentes.
Realizar un piloto de la aplicación de la metodología mediante caso de estudio.	Realizar un piloto aplicando la metodología en un proyecto de desarrollo de una empresa mediante caso de estudio para obtener resultados y posibles mejoras a este para luego replicar a los demás.			Porcentaje de vulnerabilidades resueltas del total de vulnerabilidades, porcentaje de verdaderos positivos del total de vulnerabilidades, porcentaje de falsos positivos del total de vulnerabilidades.
Validar la metodología propuesta para mejorar la seguridad del *software* en función de las fases.	Evaluación mediante juicio expertos.			Coeficiente de concordancia W de Kendall
Validar la utilidad de la metodología de seguridad del *software*.	Evaluar mediante encuesta a Gerentes de proyecto - Gerentes de TI.			Escala de Likert

4. Resultados y discusión

A continuación, presentamos los resultados de acuerdo con el orden de los objetivos específicos. En primer lugar, se muestran los resultados para el objetivo específico 01: Análisis de modelos, Armonización de modelos y Metodología propuesta.

4.1. *Análisis de modelos relacionados con el tema de investigación*

¿Qué mérito o característica observamos para abordar estos modelos? Las características que se observaron fueron las siguientes: como primera característica fue que ambos modelos son comparables en cuanto a sus dominios o categorías de procesos, así como respecto a sus prácticas. Como segunda característica, ambos modelos se actualizan regularmente, es decir, la frecuencia de actualización es anualmente en el caso de BSIMM y cada 2 a 3 años en el caso de SAMM. En consecuencia, se retroalimentan con un nivel de frecuencia aceptable respecto de lo que las empresas de diversos sectores a nivel mundial hacen para mantener la seguridad de sus aplicaciones. Como tercera característica, ambos son

independientes del modelo de desarrollo, lenguaje de programación y tienen un enfoque preventivo de la seguridad dentro del SDLC. Así pues, comparten el fundamento de que es mejor resolver problemas de seguridad en etapas previas a la entrega del *software* porque el costo de solucionarlos es menor y se evitan mayores problemas de diversa índole. Como cuarta característica, dada la cantidad de actividades que contienen las prácticas de estos modelos, sostenemos que son suficientes para permitirnos escoger las más idóneas ante el problema identificado y plantear una solución acorde con este sector del desarrollo de *software*.

4.2. *Armonización de modelos*

En [56], hace una clara definición de aquello en qué consiste una armonización y lo que no. La armonización no consiste en (i) Crear un meta modelo maestro o un nuevo modelo único que abarque todos los demás modelos o (ii) Declarar cualquier combinación única de modelos como la mejor, o sugerir una combinación universal para adaptarse a todos. La armonización, en cambio, es el desarrollo de una solución adecuada que permita satisfacer los objetivos de una organización. Asimismo, en [56], define la homogenización como "el conjunto de pasos y herramientas mediante los cuales se tratan uno o más modelos para convertir las estructuras de sus elementos de proceso en estructuras homogéneas". Esta técnica apoya y facilita la implementación de otras como por ejemplo técnicas de comparación. Así pues, para la armonización de modelos se utilizó el método de Pardo [57], el cual consta de cuatro pasos que son la homogenización, la comparación, el análisis porcentual entre modelos y el análisis de resultados, los cuales se muestran a continuación.

En la tabla 6 realizamos una comparación a alto nivel de los modelos de manera tal que podamos establecer un emparejamiento de la información y de los elementos de proceso de los mismos bajo un esquema común que facilita y prepara los modelos para el proceso de la armonización.

Tabla 6. Estructura común de elementos de proceso

SECCIÓN	ESTEREOTIPOS Y ELEMENTOS	BSIMM12	SAMM2
Sección 1: Descripción (SD)	SD1. Categoría de proceso	x	x
	SD2. Procesos	x	x
	SD3. Actividades	x	x

		SD4. Tareas		
Sección 2: Roles y Recursos (SRR)		SRR1. Roles	x	x
		SRR2. Herramientas	x	
Sección 3: Control (SC)		SC1. Artefactos SC2. Objetivos SC3. Métricas		
Sección 3: Información Adicional (SIA)		SIA1.Procesos relacionados SIA2. Métodos		

En la tabla 7 se muestra la comparación entre las prácticas de ambos modelos, en la que es posible observar un resultado de 100 %, lo cual significa que no hay una sola práctica que quede sin relacionarse con alguna práctica del otro modelo, esto es debido a que las prácticas de ambos modelos guardan relación con la temática que tratan.

Tabla 7. Comparación entre modelos

			BSIMM12														
	Categoría de procesos		Gobernancia			Inteligencia			Puntos de contacto SSDL					Despliegue			
	Categoría de procesos	**Prácticas**	Estrategia y Métricas (SM)	Cumplimiento y Política (CP)	Entrenamiento (T)	Modelos de ataque (AM)	Características y diseño de seguridad (SFD)	Estándares y requisitos (SR)	Análisis de Arquitectura (AA)	Revisión de código (CR)	Pruebas de seguridad de sw (ST)	Pruebas de penetración (PT)	Entorno de software (SE)	Gestión de configuración y gestión de vulnerabilidades (CMVM)			
SAMM2	Gobernancia	Estrategia y Métricas	x														
		Política y Cumplimiento		x													
		Educación y Orientación			x												
	Diseño	Evaluación de amenazas				x											
		Requerimientos de seguridad					x	x									
		Arquitectura de seguridad							x								
	Implementación	Construcción segura								x							
		Implementación segura											x				
		Gestión de defectos												x			
	Verificación	Evaluación de arquitectura							x								
		Pruebas basadas en requisitos									x						
		Pruebas de seguridad									x	x					
	Operaciones	Administracion de incidentes												x			
		Gestión del entorno						x					x	x			
		Gestión operativa						x					x				
	SAMM2 -> BSIMM12 15 Procesos de 15: 100%		BSIMM12 -> SAMM2 12 Procesos de 12: 100%														

En la tabla 8 utilizamos la escala discreta de comparación definida por Pino [58] para expresar el grado de relación entre los modelos.

Tabla 8. Escala de comparación utilizada para el análisis porcentual

ACRÓNIMO	DESCRIPCIÓN	PORCENTAJE
S	Fuertemente relacionado	(86 % a 100 %)
L	Relacionado en gran medida	(51 % a 85 %)
P	Parcialmente relacionado	(16 % a 50 %)
W	Débilmente relacionado	(1 % a 15 %)
	No relacionado	0 %

Fuente: "*Harmonizing maturity levels from CMMI-DEV and ISO/IEC 15504*" [58]

En la tabla 9 se muestran las relaciones entre las prácticas de los modelos SAMM2 y BSIMM12. De las 180 posibles relaciones, 160 califican como no relacionados, esto es, el 88.89 %, mientras que el 11.11 % si lo está de alguna manera. Del 11.11 % que representan a aquellos que se relacionan de alguna manera, el 25 % corresponde a fuertemente relacionados, el 30 % a relacionados en gran medida, el 40 % a relacionados parcialmente y el 5 % a relacionados débilmente. Por lo tanto, se puede observar que existe una relación entre los modelos, en la cual SAMM2 soporta el cumplimiento del 11.11 % de las prácticas definidas por BSIMM12.

Tabla 9. Análisis porcentual entre modelos

			BSIMM12																
		Categoría de procesos	Gobernancia			Inteligencia			Puntos de contacto SSDL						Despliegue				
	Categoría de procesos	**Prácticas**	Estrategia y Métricas (SM)	Cumplimiento y Política (CP)	Entrenamiento (T)	Modelos de ataque (AM)	Características y diseño de seguridad (SFD)	Estándares y requisitos (SR)	Análisis de Arquitectura (AA)	Revisión de código (CR)	Pruebas de seguridad de sw (ST)	Pruebas de penetración (PT)	Entorno de software (SE)	Gestión de configuración y gestión de vulnerabilidades (CMVM)					
SAMM2	Gobernancia	Estrategia y Métricas	L																
		Política y Cumplimiento		S															
		Educación y Orientación			S														
	Diseño	Evaluación de amenazas				S													
		Requerimientos de seguridad					P	L											
		Arquitectura de seguridad							S										
	Implementación	Construcción segura								S									
		Implementación segura											P						
		Gestión de defectos												P					
	Verificación	Evaluación de arquitectura							L										
		Pruebas basadas en requisitos									L								
		Pruebas de seguridad									P	L							
	Operaciones	Administracion de incidentes												L					
		Gestión del entorno						P					P	W					
		Gestión operativa						P					P						

4.3. *Metodología propuesta*

En base al análisis realizado en la etapa anterior, se procedió a elaborar la metodología propuesta de cómo mejorar la seguridad del *software* en empresas de desarrollo que ofrecen

software como servicio y para su propio uso, planteando un enfoque integral, debido a que el *software* es mucho más que sólo el código fuente o la aplicación como tal, considerando actividades a lo largo del ciclo de vida del desarrollo de *software* en ciclos más cortos y en aquello que adolecen las empresas del sector identificado en el caso de estudio. A continuación, se muestran las fases y prácticas que conforman la metodología (véase Figura 1), así como las actividades que comprenden cada una de estas prácticas (véase Tabla 10). El detalle de la metodología propuesta se encuentra en el ANEXO 3.

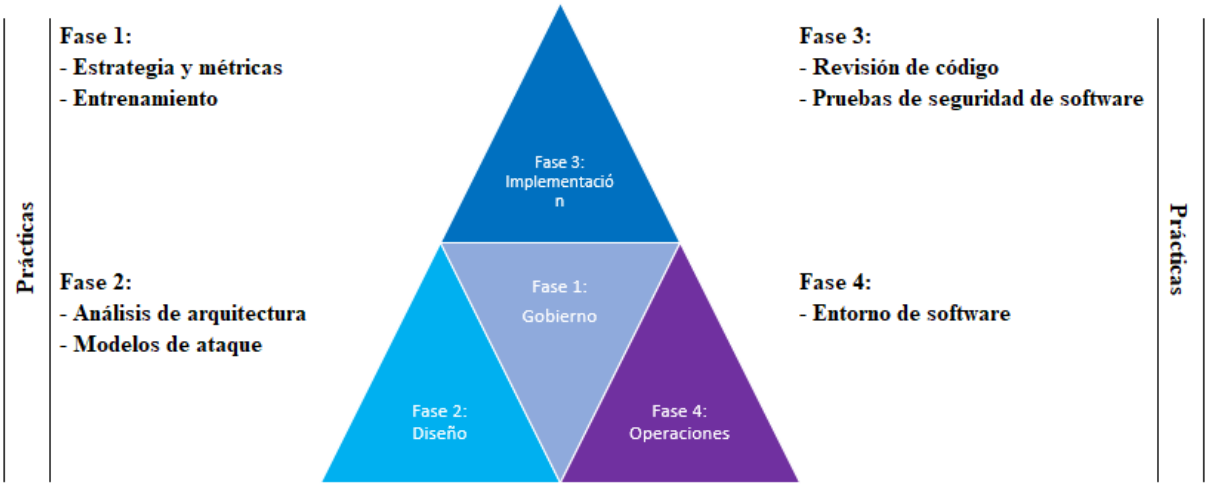


Figura 1. Metodología de seguridad de *software* propuesta

Tabla 10. Actividades por cada práctica

FASE	PRÁCTICAS	ACTIVIDADES	ESCENARIOS
Gobierno	Estrategia y métricas	Definir apetito, tolerancia y capacidad del riesgo	✓ ✓ ✓
		Realizar perfil de riesgo de la aplicación	✓ ✓ ✓
		Definir la estrategia	✓ ✓ ✓
		Establecer el proceso de seguridad de *software*	✓ ✓ ✓
		Definir y utilizar el proceso análisis de la arquitectura	✓ ✓ ✓
		Establecer el proceso de construcción	✓ ✓
		Identificar objetivos de seguridad de *software*	✓ ✓ ✓
		Determinar presupuestos	✓ ✓ ✓
		Asignar roles y responsabilidades	✓ ✓
		Establecer métricas y KPIs estratégicos	✓ ✓ ✓
		Establecer estándares de seguridad	✓ ✓ ✓
	Entrenamiento	Realizar capacitaciones o eventos	✓ ✓
		Identificar campeones de seguridad	✓ ✓
Diseño	Análisis de la arquitectura	Realizar una revisión de las características de seguridad	✓ ✓ ✓
		Utilizar lista de verificación (*checklist*) de principios de seguridad	✓ ✓ ✓
	Modelos de	Recopilar y utilizar inteligencia de ataque	✓

	ataque	Mantener y utilizar una lista de los N principales ataques posibles	✓ ✓
		Recopilar y publicar historias de ataques	✓
		Realizar modelado de amenazas	✓ ✓ ✓
		Identificar requisitos de seguridad	✓ ✓ ✓
		Realizar evaluaciones a sus proveedores de *software*	✓
Implementación	Revisión de código	Realizar una revisión de código oportuna y obligatoria	✓ ✓
		Utilizar herramientas automatizadas	✓ ✓ ✓
	Pruebas de seguridad de *software*	Integrar herramientas de seguridad de caja opaca en el proceso de QA	✓ ✓
Operaciones	Entorno de *software*	Usar monitoreo de entrada de la aplicación	✓ ✓
		Definir configuraciones y parámetros de implementación seguros	✓ ✓ ✓

Leyenda:

Actividades mínimas para empresa: ✓ Pequeña ✓ Mediana ✓ Grande

En segundo lugar, se muestran los resultados para el objetivo específico 02: Resultados de la realización de un piloto en un proyecto de desarrollo terminado.

4.4. *Resultados del piloto de la aplicación de la metodología en un proyecto desarrollo terminado*

Los detalles de la ejecución del piloto se presentan en el ANEXO 5.

Tabla 11. Resultados de vulnerabilidades antes y después de la aplicación de la metodología

TIPO DE REVISIÓN	CANTIDAD DE VULNERABILIDADES							
	PRETEST	VP[1]	FP[2]	% VP	% FP	POSTEST	RESUELTAS	% RESUELTAS
SAST	28	28	0	100	0	0	28	100.00
MANUAL	1	1	0	100	0	0	1	100.00
DAST	240	240	0	100	0	4	236	98.33
TOTAL	269	269	0	100	0	4	265	98.51

VP[1]=verdadero positivo; FP[2]=falso positivo

4.4.1. CONTRASTACIÓN DE HIPÓTESIS

Hipótesis Nula (H_0): La implementación de una metodología de seguridad de *software* no mejorará significativamente la seguridad del *software* como servicio en empresas de desarrollo que la apliquen.

Hipótesis Alternativa (H_1): La implementación de una metodología de seguridad de *software* mejorará significativamente la seguridad del *software* como servicio en empresas de desarrollo que la apliquen.

Nivel de significancia (α): 5 % = 0.05

Estadístico de prueba chi cuadrado (X^2): $X^2 = \sum \frac{(Oij - Eij)^2}{Eij}$

Donde:

Oij: son las frecuencias observadas; Eij: son las frecuencias esperadas

Tabla 12. Frecuencia de valores observados

VULNERABILIDADES	PRETEST	POSTEST	TOTAL
Total de vulnerabilidades encontradas	269	4	273
Total de vulnerabilidades resueltas	0	265	265
TOTAL	269	269	538

Frecuencia esperada

$$Eij = \frac{(total\ columna) * (total\ fila)}{suma\ total}$$

Tabla 13. Frecuencia de valores esperados

VULNERABILIDADES	PRETEST	POSTEST	TOTAL
Total de vulnerabilidades encontradas	136.5	136.5	273
Total de vulnerabilidades resueltas	132.5	132.5	265
TOTAL	269	269	538

Tabla 14. Cálculo del estadístico de prueba chi cuadrado

$X^2 = \sum \frac{(Oij - Eij)^2}{Eij}$	Oij	Eij	O-Eij	$(Oij - Eij)^2$	$(Oij - Eij)^2 / Eij$
Vulnerabilidades encontradas pretest	269	136.5	132.5	17,556.25	128.62
Vulnerabilidades resueltas pretest	0	132.5	132.5	17,556.25	132.50
Vulnerabilidades encontradas postest	4	136.5	132.5	17,556.25	128.62
Vulnerabilidades resueltas postest	265	132.5	132.5	17,556.25	132.50
TOTAL				$X^2 =$	522.23

Valor del estadístico de prueba $X^2 = 522.23$

Grados de libertad

gl = (f-1)*(c-1) = (2-1)*(2-1) = 1

Donde:

f = Número de filas; c = Número de columnas

Valor crítico

$X^2_{(\alpha, gl)}$ = INV.CHICUAD.CD(probabilidad; grados_de_libertad) = 3.84

Criterio de decisión: Se acepta la Hipótesis nula (H_0) si $X^2 \leq X^2$ crítico, caso contrario se rechaza H_0 y se acepta la Hipótesis alterna (H_1).

$X^2 = 522.23 > X^2$ crítico = 3.84

Decisión: $X^2 = 522.23$ pertenece a la región crítica. Por lo tanto, se rechaza H_0.

Conclusión: A un 95 % de confianza existe suficiente evidencia para indicar que la aplicación de la metodología de seguridad de *software* mejoró significativamente la seguridad del *software* como servicio.

En tercer lugar, se muestran los resultados para el objetivo específico 03: Resultados de la validación de contenido de la metodología propuesta.

4.5. *Validación de contenido de la metodología propuesta*

La metodología propuesta fue evaluada por tres expertos del área de seguridad de la información y de software, con el objetivo de medir su confiabilidad. Para medir el nivel de confiabilidad de la metodología, se procesó los resultados del juicio de expertos (véase ANEXO 7) utilizando el Alfa de Cronbach, obteniendo un nivel de confiabilidad de 67 %, calificado como “Muy confiable”, tal como se muestra en las siguientes tablas:

Tabla 15. Estadístico de confiabilidad Alfa de Cronbach

ESTADÍSTICAS DE FIABILLIDAD	
Alfa de Cronbach	N de elementos
.671	26

Tabla 16. Escala de interpretación del coeficiente Alfa de Cronbach

RANGO	INTERPRETACIÓN
0.53 a menos	Confiabilidad nula
0.54 a 0.59	Confiabilidad baja
0.60 a 0.65	Confiable
0.66 a 0.71	Muy Confiable
0.72 a 0.99	Excelente confiabilidad
1	Confiabilidad perfecta

Además, para determinar el grado de concordancia de las respuestas emitidas por los expertos, se aplicó el coeficiente de concordancia W de Kendall. Este coeficiente tiende a

variar entre 0 y 1, mientras más cercano sea el valor a 1 mayor será el grado de concordancia, por ende, mientras más cercano sea el valor a 0 menor será el grado de concordancia. Así pues, planteamos como hipótesis nula (H_0) que no existe concordancia cuando W=0 y como hipótesis alterna (H_1) que existe concordancia cuando W > 0.

Tabla 17. Coeficiente de concordancia W de Kendall

ESTADÍSTICOS DE CONTRASTE	SUFICIENCIA	CLARIDAD	COHERENCIA	RELEVANCIA
N	26	26	26	26
W de Kendall	.130	.115	.058	.154
Chi-cuadrado	6.750	6.000	3.000	8.000
gl	2	2	2	2
Sig. Asintót.	.034	.050	.223	.018

En base a los resultados obtenidos, aceptamos la hipótesis alterna, es decir, existe concordancia entre las opiniones de los expertos y esta es estadísticamente significativa ($p < 0.05$) para el criterio de suficiencia, relevancia, en cuanto a la claridad se encuentra justo en el umbral de decisión convencional, por lo tanto, la concordancia es marginalmente significativa y con respecto a la coherencia no es significativa, es decir, la concordancia para este criterio es debida al azar. Por último, se muestran los resultados para el objetivo específico 04: Resultados de la validación de la utilidad de la metodología propuesta.

4.6. *Validación de la utilidad de la metodología propuesta*

Se realizó una encuesta al gerente de TI y al gerente de proyectos de desarrollo para conocer su percepción respecto a la utilidad de la metodología propuesta, siendo esta calificada como totalmente de acuerdo por parte de ambos (véase ANEXO 8).

5. Conclusiones

Al finalizar la presente investigación, como primera conclusión podemos decir que mediante la implementación de la metodología de seguridad de *software* basada en los modelos BSIMM y SAMM en un caso de estudio, se mejoró la seguridad de un proyecto de *software* terminado.

Se logró el objetivo de crear una metodología de seguridad de *software* tomando como referencia los modelos BSIMM y SAMM, ambas armonizadas de tal manera que permitió mejorar la seguridad del *software* como servicio en empresas de desarrollo.

La evaluación de la metodología propuesta se realizó por expertos en el tema, con el objetivo de medir el nivel de confiabilidad de esta y posteriormente la concordancia entre ellos respecto de atributos como la claridad, la suficiencia, la coherencia y la relevancia, obteniendo resultados favorables, lo cual certifica que es válido para mejorar la seguridad del *software* que producen las empresas de desarrollo.

Así también la metodología propuesta fue evaluada por quienes estuvieron en contacto con la misma durante la realización del experimento en el caso de estudio, es decir, el equipo que conformó el proyecto de *software*, como resultado respecto de su utilidad obtuvimos el máximo nivel de calificación, lo cual indica que la metodología de seguridad de *software* les fue útil.

Recomendaciones

Se recomienda complementar el estudio, añadiéndole pruebas de penetración.

Realizar la implementación en más empresas de desarrollo de manera que la metodología reciba una mayor retroalimentación, así como desde el inicio del desarrollo de un proyecto de *software*.

Referencias

[1] C. Eng, “State of software security volume 9”. 2018. [Online]. Available: https://www.veracode.com/sites/default/files/pdf/resources/sossreports/state-of-software-security-volume-9-veracode-report.pdf [Accessed: 16-dic-2021]

[2] R. Roy and D. Spaniel, “Software Security is National Security”. Abril 2019. [Online]. Available: https://icitech.org/software-security-is-national-security. Accessed: 17-dic-2021]

[3] D. Smith, D. Villalba, M. Irvine, D. Stancke and N. Harvey, “Accelerate State of DevOps 2021”. 2021. [Online]. Available: https://cloud.google.com/devops/state-of-devops/ [Accessed: 20-dic-2021]

[4] R. Deraison, “Tenable’s 2020 threat landscape retrospective”. 2020. [Online]. Available: https://www.tenable.com/cyber-exposure/2020-threat-landscape-retrospective. [Accessed: 27-dic-2021]

[5] PROMPERÚ. Panorama de la Industria del Software y Servicios de Informática. 2021.

[6] R. A. Correa, J. R. Bermejo, J. Bermejo, J. Sicilia, M. Sánchez and Á. A. Magreñán, "Hybrid Security Assessment Methodology for Web Applications," Computer Modeling in Engineering & Sciences, vol. 126, (1), pp. 89-124, 2021. Available: https://www.proquest.com/scholarly-journals/hybrid-security-assessment-methodology-web/docview/2474506676/se-2. DOI: https://doi.org/10.32604/cmes.2021.010700.

[7] J. C. Sancho, A. Caro and P. García, "A Preventive Secure Software Development Model for a Software Factory: A Case Study," in IEEE Access, vol. 8, pp. 77653-77665, 2020, doi: 10.1109/ACCESS.2020.2989113.

[8] W. Shao-Fang, A. Shukla and B. Katt, "Developing Security Assurance Metrics to Support Quantitative Security Assurance Evaluation," Journal of Cybersecurity and Privacy, vol. 2, (3), pp. 587, 2022. Available: http://usat.lookproxy.com/scholarly-journals/developing-security-assurance-metrics-support/docview/2716553378/se-2. DOI: https://doi.org/10.3390/jcp2030030.

[9] R. A. Khan, S. U. Khan, H. U. Khan and M. Ilyas, "Systematic Literature Review on Security Risks and its Practices in Secure Software Development," in IEEE Access, vol. 10, pp. 5456-5481, 2022, doi: 10.1109/ACCESS.2022.3140181.

[10] S. Moyo and E. Mnkandla, "A Novel Lightweight Solo Software Development Methodology With Optimum Security Practices," in IEEE Access, vol. 8, pp. 33735-33747, 2020, doi: 10.1109/ACCESS.2020.2971000.

[11] S. S. Priya and S. S. Arya, "Threat Modeling for a Secured Software Development," International Journal of Advanced Research in Computer Science, vol. 7, (1), 2016. Available: http://usat.lookproxy.com/scholarly-journals/threat-modeling-secured-software-development/docview/1783652430/se-2.

[12] R. Desai and T. N. Nisha, “Best Practices for Ensuring Security in DevOps: A Case Study Approach,” Journal of Physics: Conference Series, vol. 1964, (4), 2021. Available: https://www.proquest.com/scholarly-journals/best-practices-ensuring-security-devops-case/docview/2555406591/se-2. DOI: http://dx.doi.org/10.1088/1742-6596/1964/4/042045.

[13] H. Nina, J. A. Pow-Sang and M. Villavicencio, "Systematic Mapping of the Literature on Secure Software Development," in IEEE Access, vol. 9, pp. 36852-36867, 2021, doi: 10.1109/ACCESS.2021.3062388.

[14] J. Romero, “Trabajo de investigación de mejores prácticas en desarrollo de sistemas web contra ataque de inyección”, tesis de maestría, Dpto. Posgrado, Univ. Tecnológica del Perú, LIMA, LIM, PERÚ, 2019. [Online]. Available: https://repositorio.utp.edu.pe/handle/20.500.12867/2078

[15] M. Souppaya, K. Scarfone and D. Dodson. NIST Special Publication 800-218. Secure Software Development Framework (SSDF) Version 1.1: Recommendations for Mitigating the Risk of Software Vulnerabilities. 2022. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-218.pdf

[16] M. Swanson, P. Bowen, A. Wohl, D. Gallup and D. Lynes. NIST Special Publication 800-34 Rev. 1 Contingency Planning Guide for Federal Information Systems. 2010. https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-34r1.pdf

[17] SYNOPSYS. “What is SDLC.” SYNOPSYS.com. https://www.synopsys.com/glossary/what-is-sdlc.html (Accessed Dec. 1, 2022).

[18] I. Sommerville. Software Engineering. 10th ed. London, United Kingdom: Pearson, 2016.

[19] M. Mardjan and A. Jahan, Security by design. [Online]. Available: https://nocomplexity.com/documents/securitybydesign/securityprinciples.html. [Accessed: Dec 27, 2022]

[20] J. H. Saltzer and M. D. Schroeder, "The protection of information in computer systems,".1974. [Online]. Available: http://web.mit.edu/Saltzer/www/publications/protection/

[21] SAFECode, Fundamental Practices for Secure Software Development, Third Edition, March 2018. [Online]. Available: https://safecode.org/wp-content/uploads/2018/03/SAFECode_Fundamental_Practices_for_Secure_Software_Development_March_2018.pdf

[22] C. Pfleeger and S. Pfleeger, Analyzing Computer Security: A Threat/vulnerability/countermeasure Approach, Prentice Hall Professional, 2012.

[23] R. E. Smith, "A Contemporary Look at Saltzer and Schroeder's 1975 Design Principles," in IEEE Security & Privacy, vol. 10, no. 6, pp. 20-25, Nov.-Dec. 2012, doi: 10.1109/MSP.2012.85.

[24] I. Arce, K. Clark-Fisher, N. Daswani, J. DelGrosso, D. Dhillon, C. Kern, et al., Avoiding the top 10 software security design flaws, Sep. 2014. [Online]. Available: https://ieeecs-media.computer.org/media/technical-activities/CYBSI/docs/Top-10-Flaws.pdf.

[25] S. A. Ebad, "Exploring How to Apply Secure Software Design Principles," in IEEE Access, vol. 10, pp. 128983-128993, 2022, doi: 10.1109/ACCESS.2022.3227434.

[26] M. Souppaya and K. Scarfone. Draft NIST Special Publication 800-154 Guide to Data-Centric System Threat Modeling. 2016. https://csrc.nist.gov/CSRC/media/Publications/sp/800-154/draft/documents/sp800_154_draft.pdf

[27] Dirección de seguridad y gestión del ciberriesgo. Ediciones de la U, 2022. [Online]. Available: https://books.google.com.pe/books?id=nANcEAAAQBAJ&pg=PA77&dq=modelado+de+amenazas&hl=es-419&sa=X&ved=2ahUKEwic3PyFtov8AhXrpJUCHZipAZ4Q6AF6BAgIEAI#v=onepage&q=modelado%20de%20amenazas&f=false. [Accessed: Dec 21, 2022]

[28] A. Agrawal and R. A. Khan, “A Framework to Detect and Analyze Software Vulnerabilities – Development Phase Perspective”, International Journal of Recent Trends in Engineering, Vol. 2, No. 2, November 2009.

[29] S. S. Priya and S. S. Arya, "Threat Modeling for a Secured Software Development," International Journal of Advanced Research in Computer Science, vol. 7, (1), 2016. Available: http://usat.lookproxy.com/scholarly-journals/threat-modeling-secured-software-development/docview/1783652430/se-2.

[30] Desarrollo seguro en ingeniería del software.: Aplicaciones seguras con Android, NodeJS, Python y C++. Marcombo, 2020. [Online]. Available: https://books.google.com.pe/books?id=wkxOEAAAQBAJ&printsec=frontcover&hl=es&source=gbs_ge_summary_r&cad=0#v=onepage&q&f=false. [Accessed: Dec 21, 2022]

[31] Ciclo de vida de desarrollo ágil de software seguro. Editorial Los Libertadores, 2020. [Online]. Available: https://books.google.com.pe/books?id=XdQ7EAAAQBAJ&pg=PA19&dq=modelado+de+amenazas&hl=es-419&sa=X&ved=2ahUKEwic3PyFtov8AhXrpJUCHZipAZ4Q6AF6BAgDEAI#v=onepage&q=modelado%20de%20amenazas&f=false

[32] Shawn Hernan, Scott Lambert, Tomasz Ostwald, Adam Shostack, "Uncover Security Design Flaws using the STRIDE Approach" msdn.microsoft.com, Nov. 2006. Available: https://learn.microsoft.com/en-us/archive/msdn-magazine/2006/november/uncover-security-design-flaws-using-the-stride-approach

[33] Desarrollador .NET. USERSHOP, Gradi S.A., 2008. [Online]. Available: https://books.google.com.pe/books?id=RYOqqtFaNS8C&pg=PT361&dq=modelado+de+amenazas&hl=es-419&sa=X&ved=2ahUKEwic3PyFtov8AhXrpJUCHZipAZ4Q6AF6BAgEEAI#v=onepage&q=modelado%20de%20amenazas&f=false

[34] S. Myagmar, A. J. Lee, and W. Yurcik, "Threat Modeling as a Basis for Security Requirements", IEEE Symposium on Requirements Engineering for Information Security (SREIS '05), Paris, France, Aug 2005.

[35] G. Ruiz, E. Heymann, E. César and B. P. Miller, "Automating Threat Modeling through the Software Development Life-Cycle," Sep. 2012. http://research.cs.wisc.edu/mist/papers/Guifre-sep2012.pdf

[36] K. Greene. "Estándares de codificación segura: aplicación de prácticas de codificación segura con SAST." ES.PARASOFT.com. https://es.parasoft.com/blog/secure-coding-standards-enforcing-secure-coding-practices-with-sast/ (Accessed Dec. 10, 2022).

[37] R. Seacord and R. Schiela. "Top 10 Secure Coding Practices – CERT secure coding – Confluence." WIKI.SEI.CMU.edu. https://wiki.sei.cmu.edu/confluence/display/seccode/Top+10+Secure+Coding+Practices (Accessed Dec. 10, 2022).

[38] C. Cuenca. "Desarrollo seguro: Principios y buenas prácticas." OWASP.org. https://owasp.org/www-pdf-

archive/Desarrollo_Seguro_Principios_y_Buenas_Pr%C3%A1cticas..pdf (Accessed Dec. 10, 2022).

[39] CLOUDFLARE. "¿Qué es una prueba de penetración?". CLOUDFLARE.com. https://www.cloudflare.com/es-es/learning/security/glossary/what-is-penetration-testing/ (Accessed Dec. 10, 2022).

[40] CIBERSEGURIDAD. "Guía completa sobre Fuzz Testing." CIBERSEGURIDAD.com. https://ciberseguridad.com/herramientas/fuzz-testing/ (Accessed Dec. 10, 2022).

[41] SERVICENOW. "¿En qué consiste el ciclo de vida del desarrollo de software (SDLC)?." SERVICENOW.com. https://www.servicenow.com/es/products/devops/what-is-sdlc.html (Accessed Dec. 10, 2022).

[42] Committee on National Security Systems. "National Information Assurance (IA) Glossary". CNSS Instruction No. 4009, April 26, 2010. https://www.hsdl.org/?view&did=7447

[43] The Software Alliance. "The BSA Framework for Secure Software: A new approach to securing the software lifecycle". 2019. https://www.bsa.org/files/reports/bsa_software_security_framework_web_final.pdf

[44] Committee on National Security Systems, Committee on National Security Systems (CNSS) Glossary, CNSSI No. 4009, April 6, 2015. https://rmf.org/wp-content/uploads/2017/10/CNSSI-4009.pdf

[45] K. Dempsey, P. Eavy, G. Moore, E. Takamura, "Automation support for security control assessments: software vulnerability management" NISTIR 8011 Vol.4, Apr. 2020. https://nvlpubs.nist.gov/nistpubs/ir/2020/NIST.IR.8011-4.pdf

[46] MITRE. "About CWE." CWE.MITRE.org. https://cwe.mitre.org/about/index.html (Accessed Dec. 22, 2022).

[47] MITRE. "2022 CWE Top 25 debilidades de software más peligrosas." CWE.MITRE.org. https://cwe.mitre.org/top25/archive/2022/2022_cwe_top25.html (Accessed Dec. 22, 2022).

[48] SANS. "CWE/SANS TOP 25 errores de software más peligrosos." SANS.org. https://www.sans.org/top25-software-errors/ (Accessed Dec. 22, 2022).

[49] OWASP. "Project Spotlight - Top 10." OWASP.org. https://owasp.org/projects/spotlight/ (Accessed Dec. 22, 2022).

[50] NIST. "About NIST." NIST.gov. https://www.nist.gov/about-nist (Accessed Dec. 22, 2022).

[51] NIST. “SAMATE.” NIST.gov. https://www.nist.gov/itl/ssd/software-quality-group/samate (Accessed Dec. 22, 2022).

[52] E. Erlikhman, J. Ewers, S. Migues, and K. Nassery. “BSIMM12 Insights & Trends Report”. 2021. [Online]. Available: https://www.bsimm.com/download.html. [Accessed: 18-jun-2022]

[53] OWASP. “OWASP SAMM v2.0”. 2020. [Online]. Available: https://owaspsamm.org/about/. [Accessed: 20-jun-2022]

[54] Hernández Sampieri, R., Fernández Collado, C., & Baptista Lucio, P. (2010). Metodología de la investigación científica. Quinta edición. McGrawHill: México

[55] J. Supo, “Niveles de investigación,” 7 Enero 2013. [Online]. Available: https://es.slideshare.net/josesupo/niveles-de-investigacion-15895478. [Accessed: 10-ene-2022].

[56] C. Pardo, F. Pino, F. García y M. Piattini. “Identifying Methods and Techniques for the Harmonization of Multiple Process Reference Models”. Dyna, vol.79, n.° 172, pp.85-93, 2012. [Online]. Available: http://www.scielo.org.co/scielo.php?script=sci_arttext&pid=S0012-73532012000200009. [Accessed: 15-may-2022]

[57] M. Cuéllar, C. Pardo, L. Herrera, y M. Correa, “Armonización de múltiples modelos para el gobierno de TI y el desarrollo de software”, Ventana Informática, n.° 30, pp. 43-53, 2014.

[58] F. J. Pino, M. T. Baldassarre, M. Piattini, and G. Visaggio, “Harmonizing maturity levels from CMMI-DEV and ISO/IEC 15504,” Journal of Software Maintenance and Evolution: Research and Practice, pp. 279-296, Sep. 2009, doi: https://doi.org/10.1002/spip.437.

Anexos

ANEXO 1 – Cuestionario para levantamiento de información de la situación actual de las empresas del sector de desarrollo de software respecto de la seguridad del software

Objetivo: El presente cuestionario tiene como objetivo identificar la situación actual de la seguridad del software en la empresa.

Indicaciones: Seleccione o marque con "X" según corresponda. Puede marcar más de una alternativa cuando se le indique entre paréntesis. Al marcar la alternativa "Otros", deberá especificar su respuesta sobre la línea.

1. ¿Qué tipo de aplicaciones crea en su empresa? (Puede marcar más de una alternativa)

☐Escritorio ☐Web ☐Mobile ☐Micro servicios ☐IoT ☐Otros:_____

2. ¿Dónde aloja su software?

☐Servidores propios ☐Nube ☐Ambos

3. ¿Cuáles de los siguientes problemas de seguridad han experimentado los softwares producidos? (Puede marcar más de una alternativa)

☐Sobrecargas de tráfico de fuerza bruta ☐Ataques que aprovechan las debilidades en la autenticación

☐Ataques dirigidos a una vulnerabilidad en particular ☐Manipulación de la lógica de negocio de la aplicación

☐ No lo sé

4. ¿Cuenta con un histórico de casos de ataques exitosos y fallidos al software de la empresa?

☐ Sí ☐ No

5. ¿Revisa y evalúa la información de nuevas amenazas potenciales de forma regular? (revisión de los consejos de seguridad de productos y servicios de proveedores)

☐ Sí ☐ No

6. Cuando subcontrata el desarrollo de software, incluyen el cumplimiento de requisitos referidos a desarrollo seguro como: (Puede marcar más de una alternativa o ninguna)

☐Validación de entradas ☐Administración de sesiones ☐Seguridad en base de datos ☐Criptografía
☐Control de acceso ☐Manejo de errores y logs

7. ¿Su personal realiza análisis de seguridad a su código fuente?

☐Sí ☐No

8. ¿Qué estándares de codificación segura emplean sus desarrolladores para la creación de software? (Puede marcar más de una alternativa o ninguna)

☐CERT (Equipo de respuesta a emergencias informáticas) ☐OWASP Top 10 ☐CWE (enumeración de debilidades comunes) ☐Seguridad y desarrollo de aplicaciones DISA STIGS ☐Otros: _____

9. ¿Realiza ponencias o conversatorios con especialistas para brindar capacitaciones en temas como técnicas de seguridad de software para desarrollo, operaciones?

☐Sí ☐No

10. Al ingresar nuevo personal ¿realizan inducción sobre cómo crear, operar e implementar código seguro, el ciclo de vida de desarrollo seguro y recursos de seguridad internos? (estándares de seguridad, requisitos de seguridad y otros proporcionados)

☐Sí ☐No

11. ¿La alta gerencia recibe o ha recibido capacitaciones en seguridad de software?

☐Sí ☐No ☐No lo sé

12. Tu personal se capacita regularmente en: (Puede marcar más de una alternativa)

☐Concientización de seguridad ☐Diseño seguro ☐Codificación segura ☐Respuesta segura ☐Seguridad en la nube ☐Modelado de amenazas ☐ Otros: _____

13. ¿Tu personal está certificado en desarrollo de software seguro?

☐Sí ☐No

14. ¿Qué herramientas de seguridad perimetral utiliza el personal de su área? (Puede marcar más de una alternativa o ninguna)

☐ WAF (Cortafuegos de aplicaciones web)

☐ IDS/IPS (Sistemas de Detección de Intrusos/ Sistemas de Prevención de Intrusos)

15. ¿Qué herramientas de seguridad de software utiliza el personal de su área? (Puede marcar más de una alternativa o ninguna)

☐ OSSM (Administración de software de código abierto)

☐ SCA (Análisis de composición de software) ☐ CSA (Análisis de seguridad de contenedores)

☐ SAST (Análisis de código fuente) ☐DAST (Análisis en tiempo de ejecución) ☐IAST(SAST+DAST)

☐RASP(Puntos de control en la aplicación) ☐ASTO (Todas las herramientas AST de manera centralizada)

☐Metadatos de automatización (Infraestructura como código)

16. ¿En qué etapa del ciclo de vida del desarrollo del software, realiza el análisis de seguridad de su software? (Puede marcar más de una alternativa o ninguna)

☐Diseño/Arquitectura ☐Desarrollo ☐Build(Construcción) ☐QA(Prueba) ☐Previo al despliegue

☐Producción

17.Si en la anterior pregunta no marcó alguna de las alternativas, ¿Por qué?

☐ Porque no conozco ☐ Porque no es necesario para mí

18. ¿Cuenta con un proceso para el análisis de la arquitectura?

☐Sí ☐No

19. Si la anterior respuesta es sí, ¿lo aplica en las revisiones de diseño para encontrar fallas en su software?

☐Sí ☐No

20. ¿Cuál es el principal desafío o los más comunes al implementar procesos de seguridad en su software? (Puede marcar más de una alternativa)

☐Velocidad/Agilidad ☐No está claro lo que se espera de nosotros ☐ Mala gestión de las dependencias

☐Colaboración entre personal de Seguridad/QA/Desarrollo ☐Integración en el uso de herramientas

☐Nos enteramos de los problemas de manera tardía en el proceso (Cuando no nos enteramos de los problemas en el análisis de seguridad de cada etapa)

ANEXO 2 – Resultados del procesamiento del cuestionario

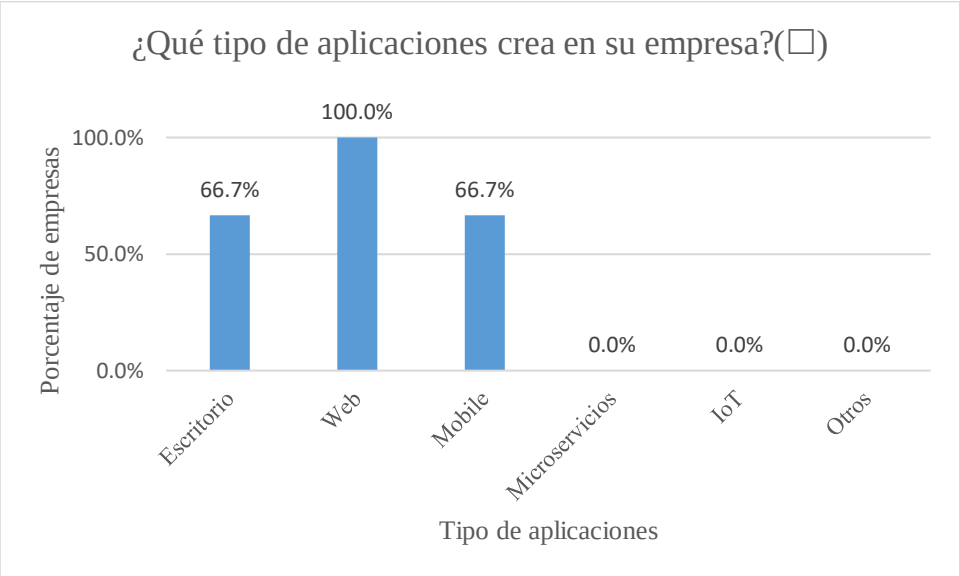


Figura 2. Tipo de aplicaciones que crean las empresas

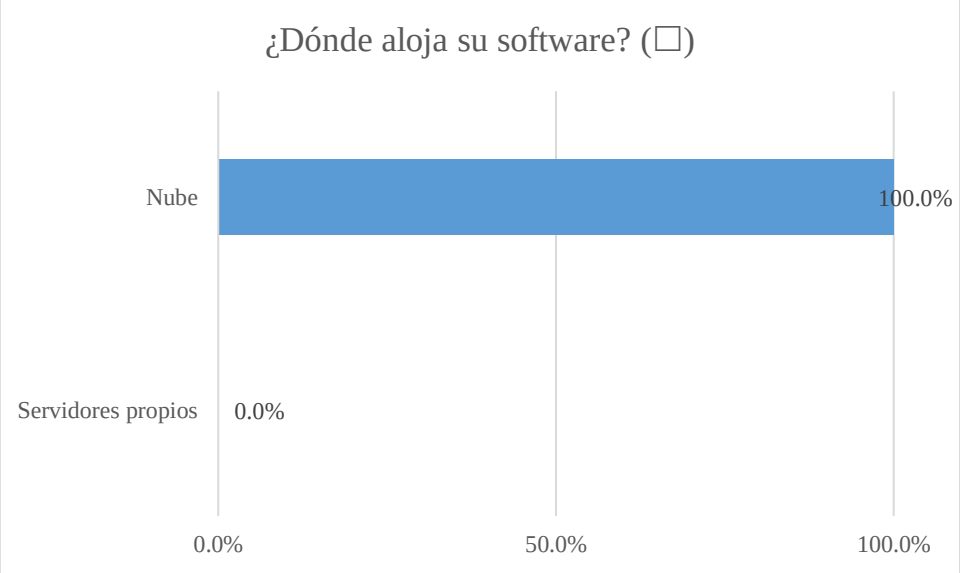


Figura 3. Lugar donde las empresas alojan su software

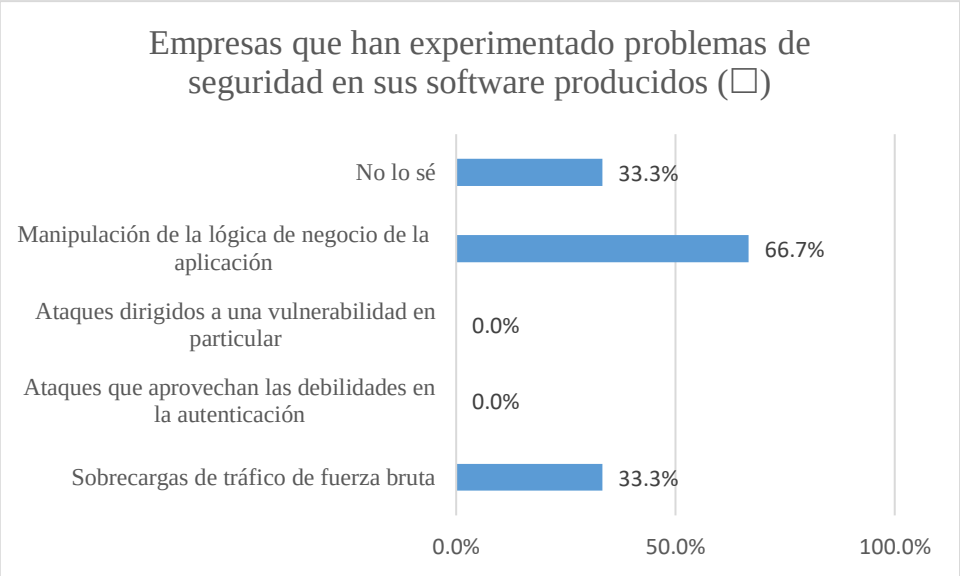


Figura 4. Empresas que han experimentado problemas de seguridad en sus *softwares* producidos

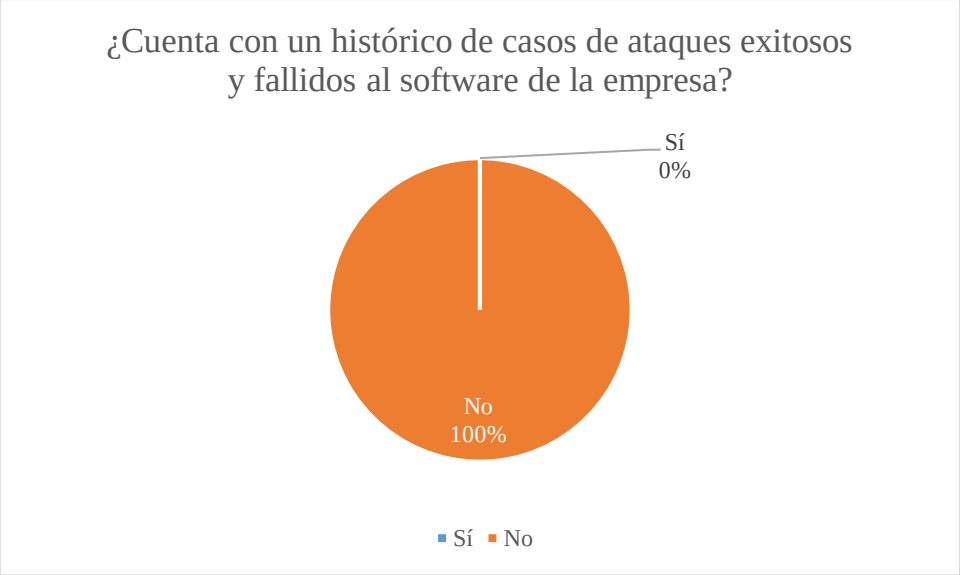


Figura 5. Empresas que cuentan con un histórico de casos de ataques exitosos y fallidos contra sus softwares

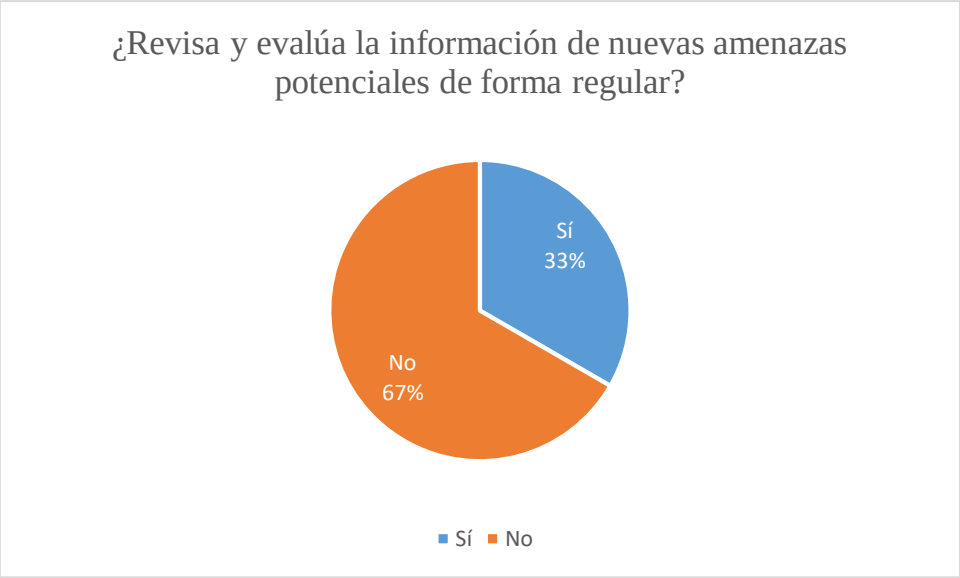


Figura 6. Empresas que revisan y evalúan la información de nuevas amenazas potenciales de forma regular

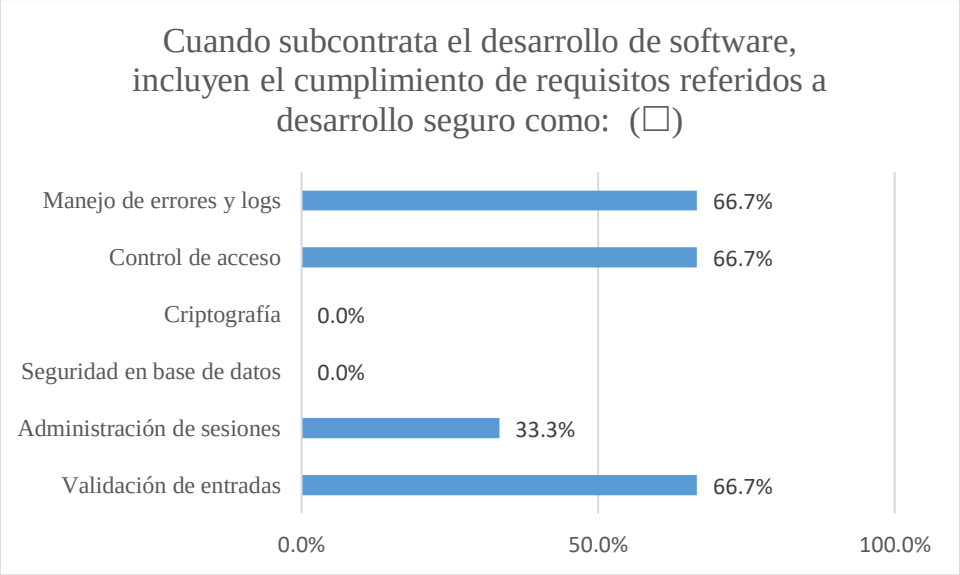


Figura 7. Empresas que incluyen el cumplimiento de requisitos referidos a desarrollo seguro en la subcontratación de software

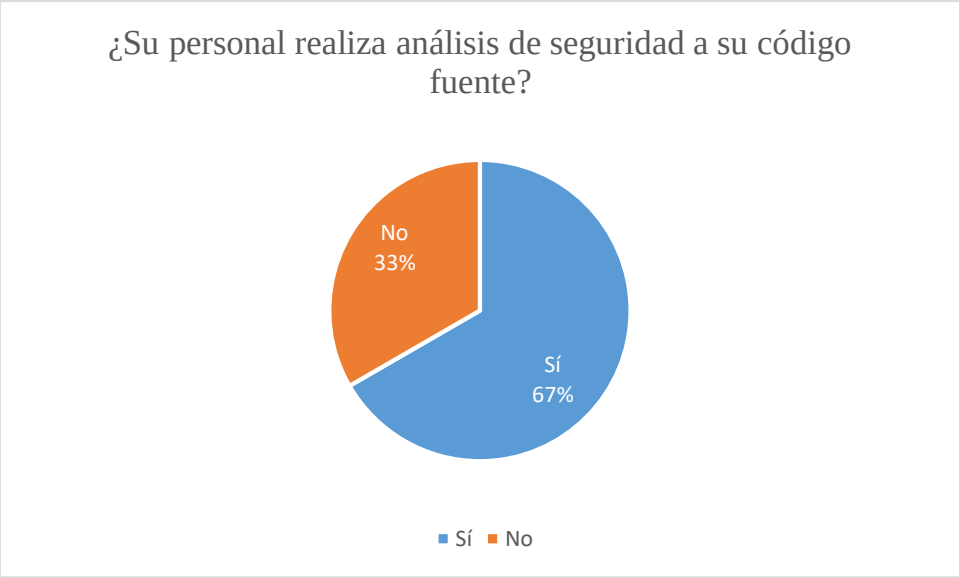


Figura 8. Empresas en las que su personal realiza análisis de seguridad a su código fuente

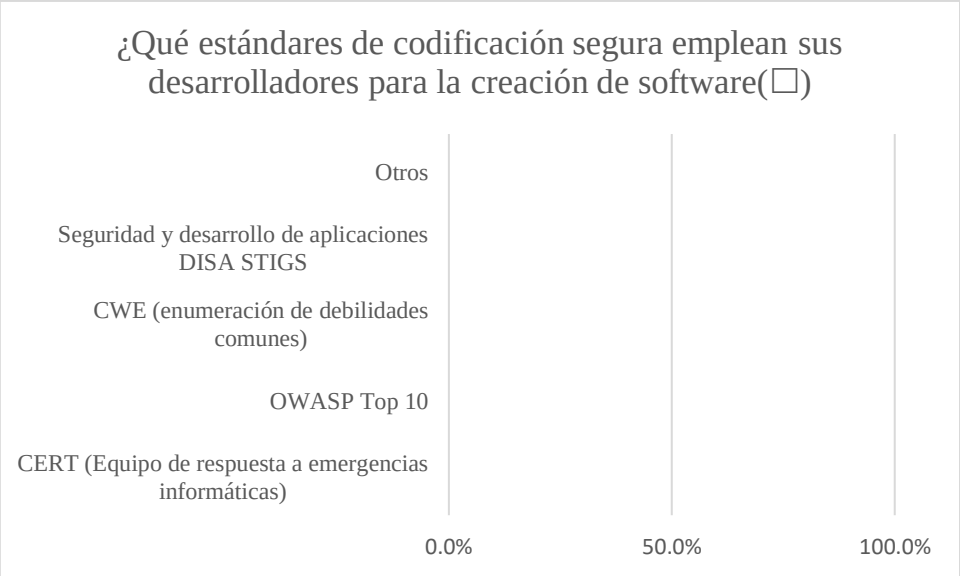


Figura 9. Empresas en las que sus desarrolladores emplean estándares de codificación segura para la creación de software

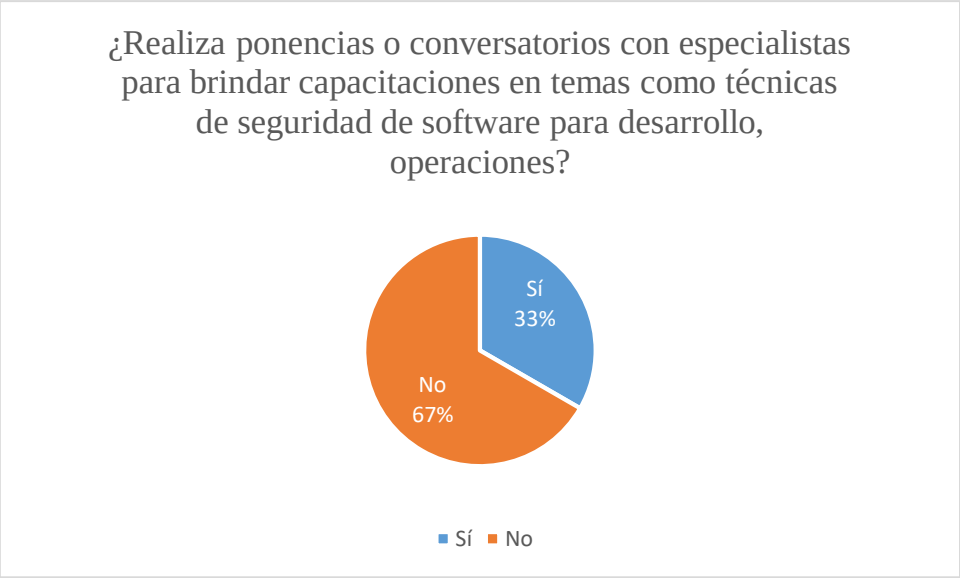


Figura 10. Empresas que realizan ponencias o conversatorios con especialistas para brindar capacitaciones en temas como técnicas de seguridad de software para desarrollo, operaciones

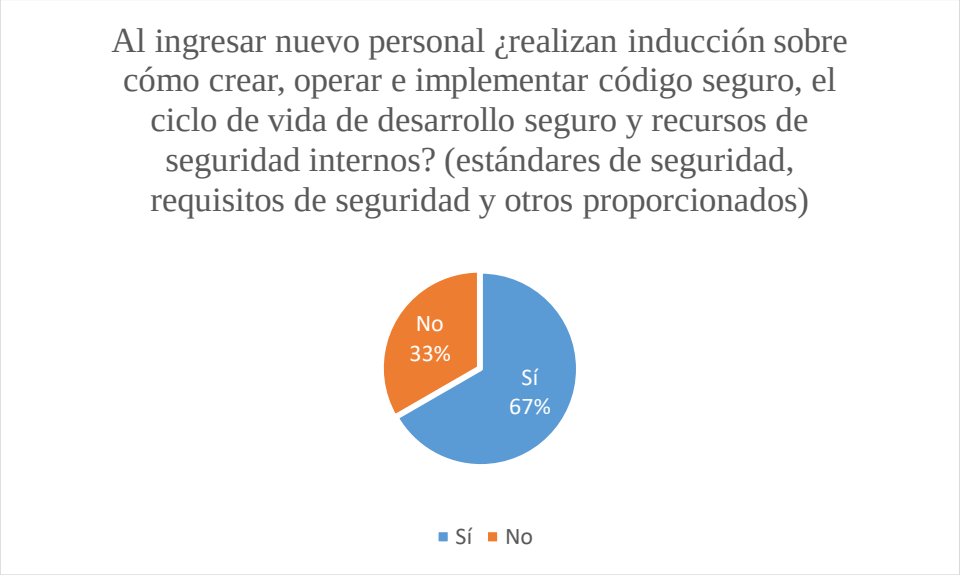


Figura 11. Empresas que realizan inducción al nuevo personal sobre cómo crear, operar e implementar código seguro, el ciclo de vida de desarrollo seguro y recursos de seguridad internos

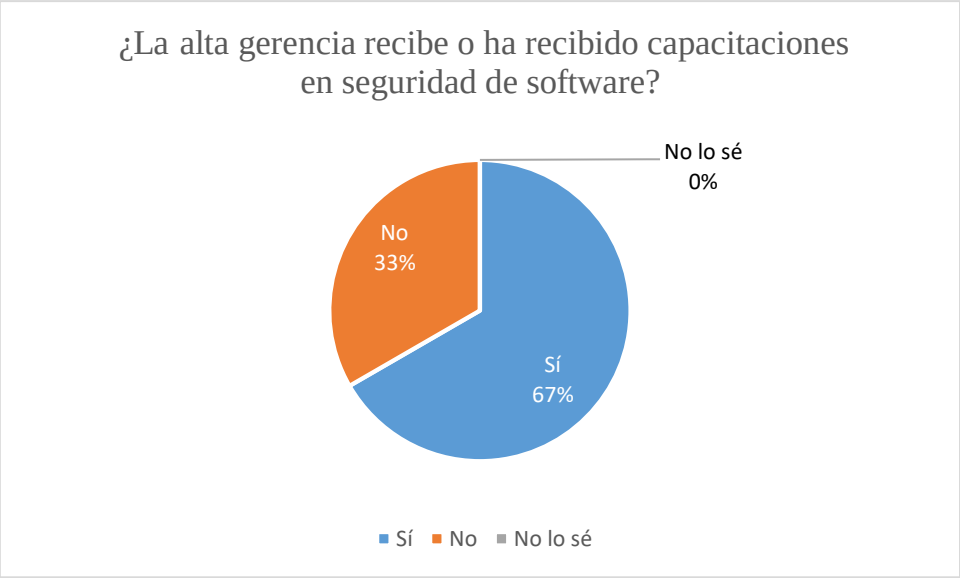


Figura 12. Empresas en las que la alta gerencia recibe o ha recibido capacitaciones en seguridad de software

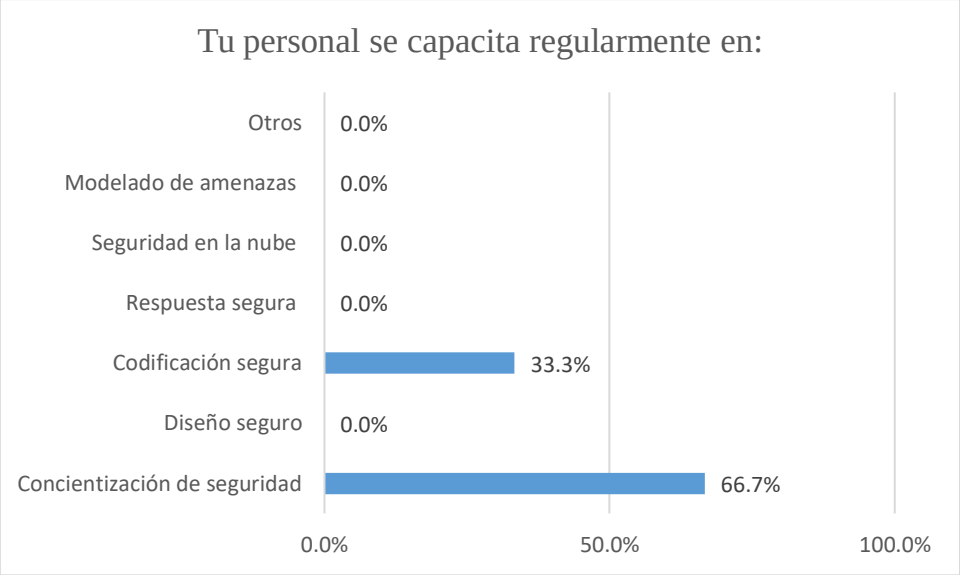


Figura 13. Temas en los que se capacita regularmente el personal de las empresas (□)

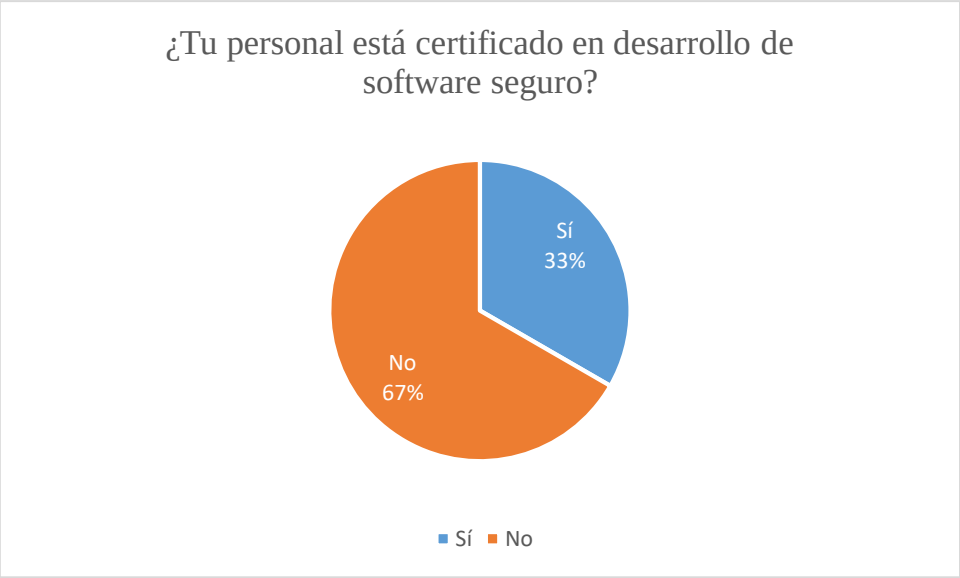


Figura 14. Empresas que tienen personal certificado en desarrollo de software seguro

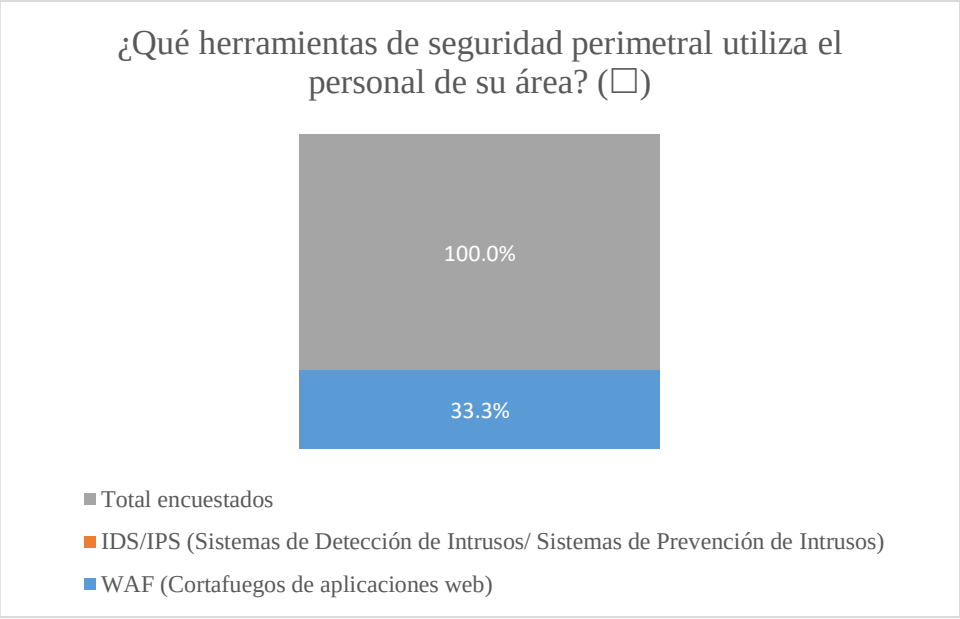


Figura 15. Herramientas de seguridad perimetral utilizadas por el personal de TI

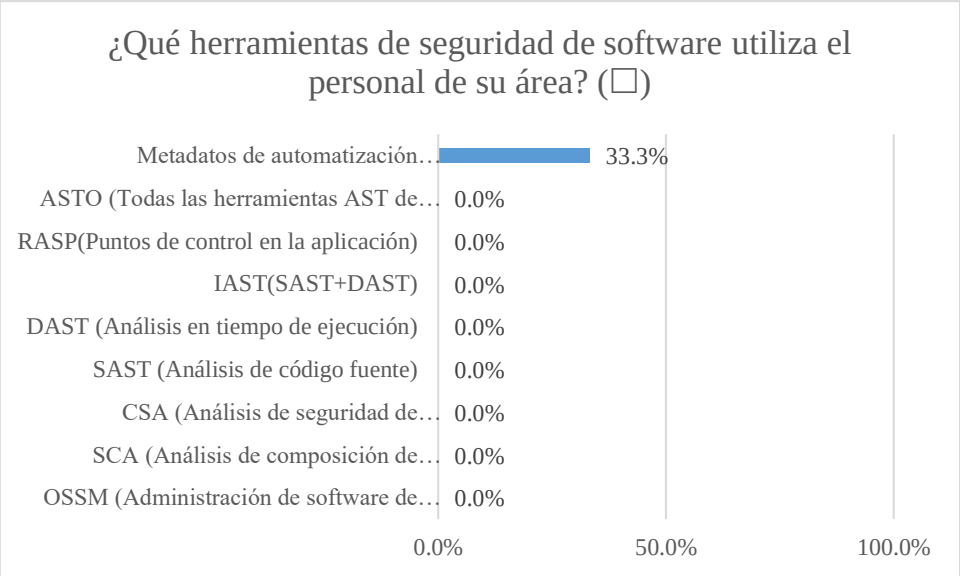


Figura 16. Herramientas de seguridad de software que utiliza el personal de las empresas

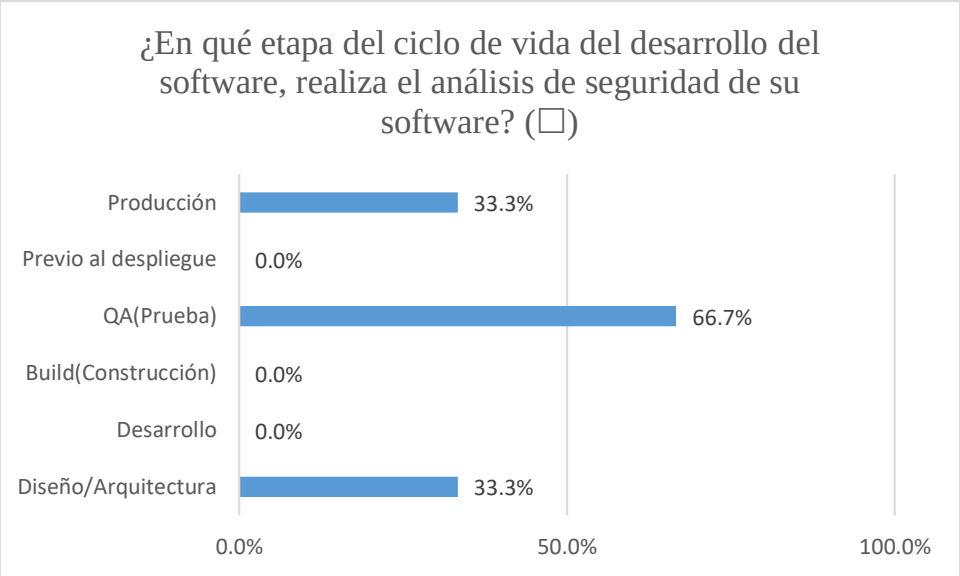


Figura 17. Etapas del ciclo de vida del desarrollo de software en las que realizan análisis de seguridad a su software

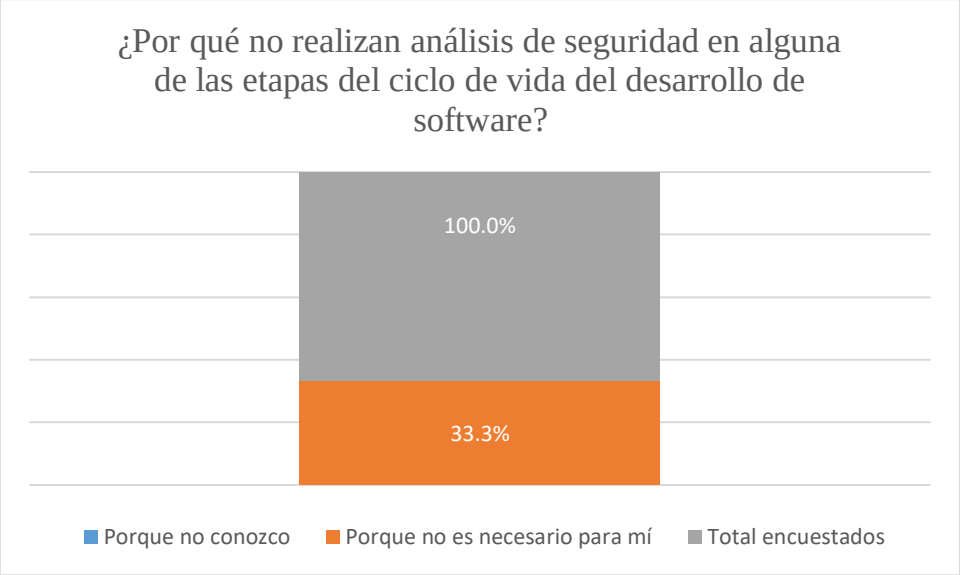


Figura 18. Empresas que consideran algunas de las razones para no realizar análisis de seguridad en alguna de las etapas del ciclo de vida del desarrollo de software

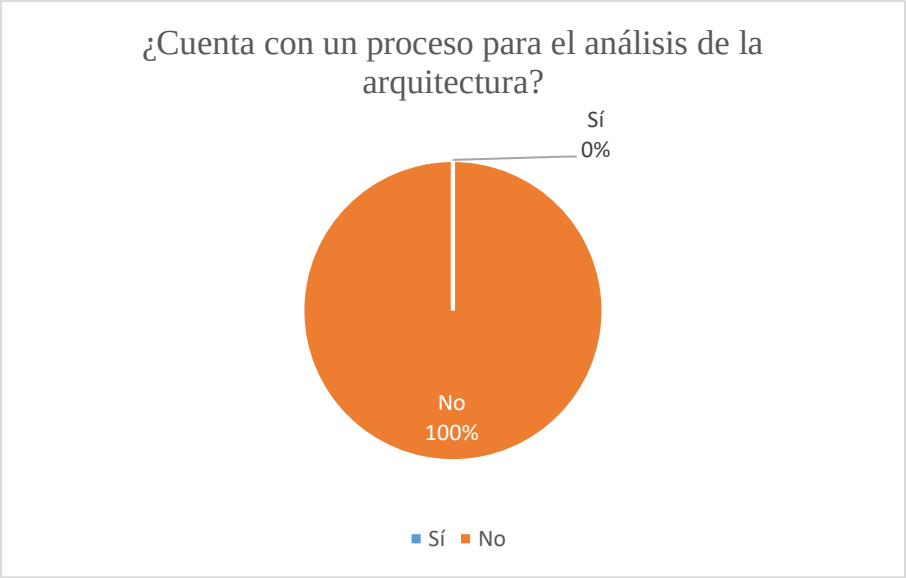


Figura 19. Empresas que cuentan con un proceso para el análisis de la arquitectura

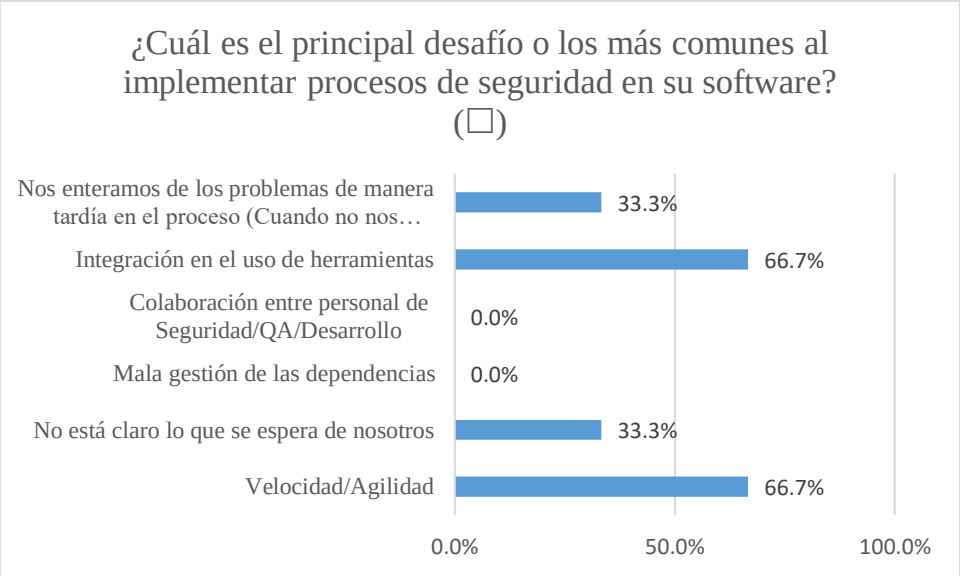


Figura 20. Desafíos comunes al implementar procesos de seguridad en su *software*

ANEXO 3 – Metodología propuesta detallada

5.1.1. FASE 1: GOBIERNO

Esta fase tiene como objetivo alinear los esfuerzos dedicados a la seguridad del *software* con la estrategia establecida junto a sus objetivos y métricas que ayudan a dar seguimiento para el cumplimiento de la misma. También es donde se definen las políticas, estándares, procesos y el entrenamiento para el talento humano.

5.1.1.1. Estrategia y métricas

5.1.1.1.1. Definir apetito, tolerancia y capacidad del riesgo

Tener definidos estos aspectos ayudan a establecer la matriz de criterios para impacto (véase tabla 13), que junto con la matriz de criterios para probabilidad (véase tabla 14) ayudan a determinar el perfil de riesgo de cada aplicación y a determinar qué riesgos son prioridad para la organización en la matriz de evaluación de riesgos (véase tabla 15).

El apetito de riesgo es una cifra monetaria que la organización está dispuesta a asumir, es decir, a perder, en caso de que el riesgo se materialice. Para definir el apetito se deben considerar tres factores: la capacidad objetiva de la organización para absorber pérdidas, la cultura o predisposición al riesgo, es decir, averso (>0 %, <6 %), moderado (≥6 %, ≤12 %), agresivo (>12 %, ≤25 %); y el sector en el que opera la organización. Otras consideraciones a nivel organización son el patrimonio, la reputación, el retorno, la facturación anual, la utilidad neta anual. Para fines prácticos utilizaremos los dos últimos. Para calcular el apetito se recomienda tomar el promedio de la utilidad neta anual de los 5 a 10 últimos años y multiplicarlo por el valor de la predisposición al riesgo. Este apetito no debe ser muy variable de un año a otro salvo excepciones como por una inflación, una recesión. Una vez definido el apetito, este sirve para 5 a 7 años.

La tolerancia al riesgo es una ligera variación del apetito, es decir, es la suma del apetito más un delta de ±5 % del apetito y es sólo para condiciones extraordinarias en las que consideraría arriesgarse un poco más o menos, es decir, si necesita alinearse con ciertas estrategias corporativas, políticas, enfoques de riesgo más agresivos o más conservadores que pueden condicionar nuevas inversiones.

La capacidad de riesgo es un nivel de impacto económico en las operaciones de tal manera que puede poner en controversia la viabilidad de la organización. Para calcular la capacidad de riesgo, obtenemos un promedio de la facturación anual de los últimos 5 a 10 años y dependiendo de la solidez financiera de la organización lo multiplicamos por un porcentaje mayor al 50 %, cuanto más alto sea el porcentaje mayor solidez financiera reflejará la organización.

Ahora bien, todo impacto económico de todo riesgo se compara contra el umbral del apetito. Si este está dentro del rango, se considera aceptable, si está por encima se considera inaceptable y se analizan las acciones y controles necesarios para que este sea aceptable; si está por debajo del apetito, la organización puede asumir más riesgo para optimizar su

rendimiento.

Tabla 18. Información financiera de la organización

ÍTEM	FACTOR	MILLONES
Facturación Anual		S/440
Utilidad Neta (U.N.)	0.3	S/132
Apetito	0.11 de U.N.	S/14.52
Tolerancia	1.05 del apetito	S/15.25
Capacidad	0.71 de facturación	S/312.4

Para definir la matriz de criterios para impacto (véase tabla 13), tipificamos escenarios de impacto para escenarios de riesgo. Se deben tener en cuenta tres consideraciones: los riesgos significativos de tecnologías de la información deben ser expresados en términos inequívocos de negocio, los interesados deben comprender el impacto en los objetivos estratégicos y la pérdida directa o indirecta; y conocer el vínculo entre los escenarios de riesgo de tecnologías de la información y el impacto en los objetivos estratégicos. La matriz comprende cuatro ítems: tipología, nivel, umbral empresa y escenarios.

La tipología debe expresar desde lo menos impactante hasta lo más impactante en términos económicos y se establece juntamente con el nivel que es una escala de 1 a n, siendo 1 lo menos impactante y n lo más impactante.

El umbral empresa expresa en un rango monetario lo que significa para la organización el tipo de impacto asociado. Para definirlos razonablemente comience por los extremos de la tipología de la siguiente manera: El umbral empresa para la tipología menos impactante puede calcularse con un porcentaje menor igual a la mitad de la predisposición al riesgo escogido pero mayor igual al 0.5 %, respecto de la utilidad neta. Mientras que el umbral empresa para la tipología más impactante o nivel más alto debe estar asociada con la capacidad de riesgo definida anteriormente, siendo mayor igual a esta. A continuación, el apetito y la tolerancia de riesgo pueden estar comprendidos en el nivel siguiente al más bajo. Para los rangos de los umbrales restantes defínalos de tal manera que guarden relación con el escenario descriptivo planteado y según el contexto de su organización, ya que no hay una fórmula con factores estáticos para ello, son sólo aproximaciones.

Los escenarios descriptivos expresan de manera descriptiva el impacto ya sea en los objetivos estratégicos de la organización, en los reclamos de los clientes o en el valor de la

acción si es que cotiza en bolsa.

Tabla 19. Matriz criterios para impacto

TIPOLOGÍA	INSIGNIFICANTE	MENOR	MODERADO	MAYOR	CATASTRÓFICO
NIVEL	1	2	3	4	5
UMBRAL EMPRESA	< 6.6 MM	6.6-15.25 MM	15.26-61 MM	61.1-312.4 MM	>312.4 MM
ESCENARIOS	Impacto insignificante en el logro de los objetivos	Impacto menor, fácilmente remediable	Se afectan algunos objetivos estratégicos	Algunos objetivos estratégicos no serán logrados	La mayoría de los objetivos estratégicos no serán logrados

Para definir la matriz de criterios para probabilidad (véase tabla 14), tipificamos escenarios de probabilidad para escenarios de riesgo. Esta matriz debe servir o adaptarse a cualquier riesgo no sólo a los ya identificados o planteados en la matriz de evaluación de riesgos. La matriz comprende cuatro ítems: tipología, nivel, tipo de escenario y probabilidad.

La tipología debe expresar desde lo más difícil hasta lo más fácil de darse y se establece juntamente con el nivel que es una escala de 1 a n, siendo 1 lo más difícil de darse y n lo más fácil de darse.

Respecto al tipo de escenario, existen tres: temporal, descriptivo y mixto. El escenario temporal describe la frecuencia de ocurrencia del evento cada cierto tiempo, este tiempo puede estar expresado sólo en años o en años, meses, días y horas. El escenario descriptivo expresa la frecuencia de ocurrencia del evento de manera descriptiva. El escenario mixto es una combinación de los dos previos. Deberá elegir el que mejor calce para su contexto. Para definirlo, el evaluador de riesgos debe considerar el historial de la organización. Para ello, se reúne con el personal idóneo que mayor conocimiento tenga sobre la organización y lo realiza mediante juicio a expertos para identificar la frecuencia de ocurrencia del evento y poder asignar un rango de probabilidad que sea razonable al escenario escogido y a su tipología.

La probabilidad establece que, en un contexto de ocurrencia, una amenaza definida explote la vulnerabilidad del activo y está expresada de 0 % a 100 %. No es una buena forma distribuir rangos de probabilidad uniformes, es decir, dividir el cien por ciento de probabilidad por el número de niveles establecidos.

Tabla 20. Matriz criterios para probabilidad

TIPOLOGÍA	RARO	IMPROBABLE	POSIBLE	PROBABLE	CASI SEGURO
NIVEL	1	2	3	4	5
ESCENARIO TEMPORAL	1 vez cada 5 años	1 vez cada 4 años	1 vez cada 3 años	1 vez cada 2 años	1 vez o más al año
ESCENARIO DESCRIPTIVO	El evento es teóricamente posible, pero nunca ha ocurrido en nuestra entidad ni en otras similares	El evento ocurrió alguna vez en otra entidad, pero nunca en la nuestra	Un evento así ocurrió en nuestra entidad una vez.	Un evento así ha ocurrido varias veces en nuestra entidad	El evento ocurre frecuentemente en nuestra entidad
PROBABILIDAD	<10%	10-40%	41-60%	61-85%	>85%

Realizar perfil de riesgo de la aplicación

Ayuda a reconocer aquellas aplicaciones que pueden representar una amenaza crítica para la organización si fueran atacadas o violadas.

Utilice un método simple para evaluar el riesgo por cada aplicación, estimando el impacto comercial potencial que representa para la organización en caso de un ataque. Para lograrlo, evalúe el impacto de una brecha en la confidencialidad, integridad y disponibilidad del dato o servicio. Considere usar un conjunto de 5 a 10 preguntas para entender las características importantes de la aplicación, como si la aplicación procesa datos financieros, si está orientada a internet o si se trata de datos relacionados con la privacidad. El perfil de riesgo de la aplicación le indica si estos factores son aplicables y si podrían afectar significativamente a la organización.

¿Cómo identificar nuevos riesgos? Para ello, ir a los OWASP top10 que tienen los riesgos más comunes para aplicaciones web, móviles, APIs, contenedores. Además, en el anexo 4 se ofrece una lista de 145 riesgos relacionados a la seguridad del software agrupados por cada etapa del ciclo de vida del desarrollo de software, escoger sólo los más significativos para su contexto.

Esta herramienta sirve como base para el modelado de amenazas y será retroalimentada posteriormente al uso de las herramientas de escaneo de vulnerabilidades.

Respecto a la asignación del nivel de probabilidad, está directamente relacionada con

alguno de los niveles de la matriz criterios para probabilidad y el nivel impacto está directamente relacionado con alguno de los niveles de la matriz criterios para impacto. La multiplicación de ambos da como resultado el riesgo, mientras mayor sea este, mayor prioridad tendrá.

Tabla 21. Matriz de evaluación de riesgos

	Activo de *software* y Riesgos	Amenaza	Vulnerabilidad	Nivel de Probabilidad(P)	Nivel de Impacto(I)	Riesgo (PxI)	Prioridad
N.°	**Aplicación 1**						
1	R1						
2	R2						
	Artefacto *software* 1						
3	R1						

Definir la estrategia

La estrategia es un conjunto integrador de opciones que nos posicionan en el campo de juego de nuestra elección de tal manera que ganemos. La estrategia tiene una teoría de cómo podríamos ser mejores que nuestros competidores y esa teoría debe ser coherente, factible y debe poder traducirse en acciones. Por ejemplo, la estrategia del cambio en todas partes (pasar de pruebas de seguridad grandes y que consumen mucho tiempo, a pruebas de seguridad más pequeñas, más rápidas e impulsadas por canalizaciones realizadas para mejorar el rendimiento del equipo de ingeniería), otro ejemplo de estrategia a definir es la seguridad del *software* como prioridad.

Establecer el proceso de seguridad de *software*

Incluye la definición de metas, roles, responsabilidades y actividades. Puede basarse en metodología SDL (*security development lifecycle*) de Microsoft o Synopsys *touchpoints*.

El proceso incluye condiciones para lanzamiento (o también llamadas puertas, puntos de control, barandillas o medidas de seguridad (reglas de alto nivel que proporcionan gobierno continuo para el entorno general), hitos, etc.) en uno o más puntos del SDLC.

Antes de establecer las condiciones para el lanzamiento debe:

i) Identificar ubicaciones compatibles con las prácticas de desarrollo existentes.

ii) Recopilar información necesaria respecto a los umbrales de clasificación de riesgo o de los datos de defectos para tomar la decisión si continua o no.

Deben verificarse en cada proyecto de *software*, incluso los aparentemente inocuos y cada proyecto debe cumplir con alguna regla establecida (generalmente relacionadas con regulaciones, acuerdos contractuales y otras obligaciones) u obtener alguna exención para continuar o permanecer en producción. El encargado de rastrear las exenciones es el SSG (*software security group*). Lo mismo aplica para APIs (*application programming interface*), *frameworks*, bibliotecas o dependencias, código hecho a medida, microservicios, configuraciones de contenedores, etc. Son *softwares* que deben de cumplir con las condiciones de liberación de seguridad. Se debe verificar tanto antes como después del propio proceso de desarrollo.

Por ejemplo, el SSG puede recolectar los resultados de las pruebas de seguridad para cada proyecto previo al lanzamiento y posteriormente dar su opinión informada sobre lo que constituye pruebas suficientes o resultados de pruebas aceptables sin tratar de detener un proyecto. En entornos CI/CD (integración continua /despliegue continuo), los lanzamientos son más frecuentes y generalmente dependen de la automatización como enfoque para recopilar la evidencia apropiada.

Asimismo, como organización se define un proceso para la aceptación del riesgo de seguridad y se documenta dicha responsabilidad, es decir, existe un propietario del riesgo que aprueba el estado del *software* previo a su lanzamiento. Por ejemplo, se requiere que el jefe de la unidad de negocios a la que se le hizo el desarrollo, apruebe de manera informada las vulnerabilidades críticas que no se mitigaron o los pasos de SSDL (*secure software development lifecycle*) que se omitieron, ya sea por un carácter de urgencia o porque se culminó el proyecto según lo acordado. Esto debe aplicarse tanto a proyectos internos, subcontratados y a aquellos que se implementarán en entornos externos como en la nube. Lo hace con una firma, enviando un formulario o similar y se almacena como evidencia para su uso posterior de necesitarse.

— establecer condiciones para lanzamiento de *software*

— verificar condiciones para lanzamiento de *software*

— **requerir aprobación de seguridad previo al lanzamiento del *software***

Definir y utilizar el proceso análisis de la arquitectura

Define y documenta el proceso de análisis de la arquitectura y lo aplica en las revisiones de diseño para encontrar fallas. Esto incluye un enfoque estandarizado para pensar en los ataques, las vulnerabilidades y varias propiedades de seguridad, las discusiones de impacto técnico, el riesgo asociado, el análisis de frecuencia o probabilidad.

Documentar tanto la arquitectura como las fallas de seguridad descubiertas.

Puede usar la metodología STRIDE de Microsoft. La metodología ARA (análisis de riesgo de la arquitectura) de Synopsys es otra opción, sin embargo, no se consideró para atender a la problemática del sector identificado porque es una versión de inmersión más profunda, por lo que se recomienda sólo si realiza *software* sensible como para dispositivos médicos implantables o para autos sin conductor. A continuación, definimos el proceso para el análisis usando STRIDE.

- **Identificación de activos.** Consiste en identificar elementos en posible riesgo como servidores, bases de datos, páginas web, etc.
- **Visión de la arquitectura.** Consiste en identificar qué hace la aplicación. Saber qué tecnologías están involucradas y documentar.
- **Descomponer la aplicación.** Consiste en identificar los límites de confianza, el flujo de los datos y los puntos de entrada.
- **Identificar amenazas.** Consiste en identificar las posibles amenazas para cada sección, usando STRIDE o árboles de ataque.
- **Documentar amenazas.** Consiste en llevar una bitácora de estas amenazas y de cómo contrarrestarlas.
- **Cuantificar amenazas.** Consiste en otorgar una valoración del posible impacto y daño de la amenaza.

Establecer el proceso de construcción

Se describen instrucciones claras de todo el proceso que una persona o herramienta debe seguir de forma coherente y produzca siempre el mismo resultado. Debe estar almacenado en un repositorio central y ser accesible a quienes intervienen en este proceso. Evitar manejar múltiples versiones o copias de este proceso, manejar las actualizaciones siempre en un solo lugar.

Revisar que las herramientas usadas para la construcción se encuentren mantenidas activamente por el proveedor y debidamente actualizadas con parches de seguridad. Reforzar la configuración de cada herramienta con las pautas brindadas por el proveedor.

Identificar objetivos de seguridad de *software*

Al revisar los requisitos funcionales del proyecto de *software*, se identifican los requisitos de seguridad relevantes (expectativas) para dicha funcionalidad razonando sobre la confiabilidad, integridad o disponibilidad deseada del servicio o los datos ofrecidos por el proyecto de *software*.

Los requisitos establecen el objetivo, por ejemplo, en un proceso de registro, los datos personales se deben transferir y almacenar de manera segura, mas no la medida real para lograr el objetivo, por ejemplo, usar TLSv1.2 para la transferencia segura.

Los objetivos de seguridad pueden relacionarse con la funcionalidad de seguridad específica que necesita agregar a la aplicación, por ejemplo, identificar al usuario de la aplicación en todo momento o con la calidad y el comportamiento general de la aplicación, por ejemplo, garantizar que los datos personales estén debidamente protegidos en tránsito, lo que no necesariamente conduce a una nueva funcionalidad.

Determinar presupuestos

Para las capacitaciones, herramientas, eventos, talento humano.

Tabla 22. Presupuesto

N.°	DESCRIPCIÓN	MONTO S/
1	Capacitaciones	S/ X,000
2	Herramientas de *software* para escaneo de vulnerabilidades	S/ X,000
3	Realización de eventos	S/ X00
	Total	S/ X,000

Asignar roles y responsabilidades

Se describen los roles con sus respectivas responsabilidades, por ejemplo, rol de grupo de seguridad de *software*, campeones de seguridad, etc.

Tabla 23. Roles y responsabilidades

ROL	RESPONSABILIDAD
Grupo de seguridad de *software* (SSG)	Es el responsable de mantener actualizado el sitio web interno o wiki interactivo, el cual

	es conocido por todos para informarse acerca de la seguridad del *software*.
Campeón de seguridad	Es el responsable de brindar revisiones periódicas de los problemas relacionados con la seguridad para el equipo del proyecto.

Establecer métricas y KPIs estratégicos

Las métricas se definen en tres categorías. La primera son métricas de esfuerzo dedicado a la seguridad, por ejemplo, horas de capacitación, tiempo dedicado a revisar código y cantidad de aplicaciones escaneadas en busca de vulnerabilidades. La segunda categoría son métricas de resultado, que miden los resultados de los esfuerzos de seguridad, por ejemplo, cantidad de defectos de seguridad que aún no han sido resueltos y cantidad de incidentes de seguridad que involucran vulnerabilidades en la aplicación; y la tercera son métricas de entorno, que miden el entorno donde se llevan a cabo los esfuerzos de seguridad, por ejemplo, número de aplicaciones o líneas de código como medida de dificultad o complejidad. Algunas otras a considerar pueden ser la cantidad de defectos críticos por aplicación, densidad de defectos (número de defectos encontrados dentro de las líneas de código de un *software*), cuya reducción podría utilizarse para dar a conocer que el costo de solucionarlo tiende a disminuir en el tiempo, en el supuesto que se han continuado realizando con el mismo nivel de profundidad las pruebas de los cambios efectuados en el software. Es mejor recopilar las métricas tempranamente a través de procesos basados en eventos con telemetría (telemetría de seguridad). Lo crucial aquí es relacionar los resultados de seguridad a los objetivos comerciales de forma clara para sustentar el financiamiento.

Pueden mostrarse en un tablero sólo para los ejecutivos relevantes y a los gerentes de desarrollo de *software* que ayuden a impulsar el cambio. Si la cultura organizacional promueve la competencia interna entre grupos, esta información puede agregar una dimensión de seguridad, podría inicialmente centrarse menos en las tendencias históricas (por ejemplo, errores por versión) y más sobre la velocidad (tiempo de reparación) y la calidad (densidad de defectos). Además, puede compartir estos datos a los grupos de ingeniería a través de los tableros de la plataforma de canalización (si se usa), para democratizar las métricas y promover la superación personal de los desarrolladores.

Los KPIs (*key performance indicator*) se componen de las métricas más significativas y efectivas. Ver los KPI como indicadores definitivos del éxito de toda la estrategia y

considérelos procesables. Debe desarrollar KPIs con la aceptación del liderazgo y los equipos responsables de la seguridad del *software*. Pueden estar disponibles para los equipos de desarrollo e incluyen umbrales de aceptabilidad y orientación en caso los equipos necesiten tomar acciones al respecto. Por ejemplo, el tiempo que trascurre desde que se detecta una vulnerabilidad hasta que se remedia. Dentro de una estrategia empresarial puede tener definido como uno de sus objetivos que la remediación de vulnerabilidades sea inferior a un número de días según el tipo de criticidad de esta. Con este KPI puede controlar si está cumpliendo con este plazo.

Tabla 24. Definición de métricas

N.°	Aplicación	Criticidad	Periodo	Horas de capacitación	Horas de revisión de código	N.° de defectos de seg.		N.° de incidentes de seg. debido a vulnerabilidades de seg.
						No resueltos	Resueltos	
1	App1	Crítica	Trim1 Trim2 Trim3 Trim4					
2	App2	Alto						
3	App3	Medio						
4	App4	Bajo						

Establecer estándares de seguridad

— Crear junta de revisión de estándares

La creación de juntas de revisión de estándares denota que lleva un proceso formal para el desarrollo de estándares y garantiza que todas las partes interesadas puedan opinar. La responsabilidad de crear y administrar estas las pueden asumir los grupos de arquitectura empresarial o de riesgo empresarial. Para cualquier estándar propuesto, puede operar designando a un campeón quien tiene la responsabilidad de demostrar que el estándar cumple con sus objetivos y posteriormente obtener la aprobación y aceptación de dicha junta. Cuando los estándares se implementan directamente como *software*, el campeón responsable puede ser un administrador de DevOps (*development and IT operations*), un ingeniero de lanzamiento o el mismo propietario del artefacto de implementación asociado.

— Para principales controles de seguridad (validación de entradas, autenticación)

Explica cómo regirse a la política y realizar operaciones específicas de seguridad, por

ejemplo, podría describir cómo realizar la autenticación de aplicaciones basada en la identidad.

Estos estándares no sólo están dirigidos a cubrir el código de una aplicación, sino al *software* en general, esto incluye la construcción de contenedores, orquestación (infraestructura como código) y la configuración de la red de servicios.

Se pueden implementar de diferentes maneras para mantenerlos procesables y relevantes, por ejemplo, se pueden automatizar en entornos de desarrollo IDE, en una cadena de herramientas, o enlazar directamente a ejemplos de código y artefactos de implementación, por ejemplo, contenedores. Para esta actividad, nos basamos en las prácticas fundamentales para desarrollo de software seguro de SAFECode, la guía de revisión de código y el Cheat Sheet Series Project de OWASP.

— Para las tecnologías en uso

Tiene como objetivo reducir el esfuerzo al no tener que examinar nuevos riesgos tecnológicos para cada proyecto nuevo. Para ello, puede crear una configuración base segura para cada pila de tecnología (comúnmente en forma de imágenes doradas, AMIs (*amazon machine images*), etc.), lo cual disminuye aún más el esfuerzo para usar la pila de manera segura. En las implementaciones tradicionales dentro de las instalaciones de la organización o a nivel local, una pila generalmente puede estar comprendida por un sistema operativo, una base de datos, un servidor de aplicaciones y un entorno de tiempo de ejecución (por ejemplo, una pila LAMP). Cuando se reutilicen contenedores, microservicios o código de orquestación, la frontera de seguridad es ideal para obtener tracción. Para entornos en la nube, las configuraciones más robustas probablemente abarquen actualizaciones de parches de seguridad, configuración de seguridad y servicios de seguridad, como el registro y monitoreo. Para ello utilizamos la guía de implementación técnica de seguridad DISA STIG para servidores web, bases de datos, contenedores, entre otros.

4.4.1.2. Entrenamiento

Realizar capacitaciones o eventos

Está dirigida a los roles involucrados en el SDLC como la gerencia, el desarrollo, las pruebas o la auditoría del *software*. El objetivo es aumentar la conciencia sobre las amenazas y los riesgos de seguridad del *software*, las mejores prácticas de seguridad y los principios de diseño de *software* seguro. Puede incluir en los temas de capacitación a las listas de errores,

vulnerabilidades, riesgos comunes de *software* como OWASP TOP10, CWE (*common weakness enumerations*), CVE (*common vulnerabilities and exposures*), BF (*bugs framework*). Además, se pueden hacer entrenamientos de 30 min antes de iniciar las labores durante un mes.

Cómo mínimo es recomendable hacer reuniones o eventos cara a cara una vez al año con ponentes en temas avanzados como técnicas más recientes en seguridad del *software* para DevOps o tecnología nativa de la nube. Para aquellos que se encuentran trabajando a distancia, el estar todos con sus cámaras encendidas y promover su participación logra una diferencia importante.

Con el lema, "entrenamiento que no se usa, no genera ningún cambio", puede incluir en este tipo de capacitaciones material como el histórico de casos relevantes de ataques exitosos y fallidos al *software* de la empresa, de esta manera es más probable que el personal entienda la relevancia para su trabajo, entendiendo cuándo y cómo aplicarlo.

Si es necesario generar una mayor motivación en el equipo puede realizarse mediante dinámicas de juego con puntuaciones y recompensas.

Incluir recursos de seguridad en la incorporación de nuevo personal con la finalidad de contribuir a la cultura de seguridad lo más pronto posible es vital. Además de cómo crear, implementar y operar código seguro, en qué consiste el SSDL y los recursos de seguridad internos como mejores estándares actuales y requisitos de seguridad, puede hacerlo a través de un sitio web interno o wiki interactivo conocido por todos para obtener información acerca de la seguridad del *software*. Este puede estar mantenido por el SSG.

Identificar campeones de seguridad

Crear un equipo satélite o campeones de seguridad, identificando a sus miembros que lo conformarán a aquellos que destaquen luego de las capacitaciones recibidas, con el objetivo de que en cada equipo de desarrollo tenga un intermediario entre seguridad de la información y los desarrolladores, puede ser un desarrollador, evaluador o gerente de producto.

Los objetivos del puesto son:

i) Aumentar la eficacia y la eficiencia de la seguridad y el cumplimiento de las aplicaciones.

ii) Fortalecer la relación entre varios equipos y la seguridad de la información.

Para lograr estos objetivos, los campeones de seguridad ayudan con la investigación, verificación y priorización de defectos de *software* relacionados con la seguridad y el

cumplimiento.

Están involucrados en todas las evaluaciones de riesgos, evaluaciones de amenazas y revisiones arquitectónicas para ayudar a identificar oportunidades para remediar los defectos de seguridad al hacer que la arquitectura de la aplicación sea más resistente y reducir la superficie de amenaza de ataque.

Además de ayudar a la seguridad de la información, los campeones de seguridad brindan revisiones periódicas de todos los problemas relacionados con la seguridad para el equipo del proyecto, de modo que todos estén al tanto de los problemas y de los esfuerzos de remediación actuales y futuros. Estas revisiones se aprovechan para ayudar a pensar en soluciones a problemas más complejos al involucrar a todo el equipo de desarrollo.

5.1.2. FASE 2: DISEÑO

Esta fase tiene como objetivo pensar en la seguridad del *software* desde el diseño, analizando la arquitectura y utilizando la inteligencia de la organización para pensar como un atacante.

5.1.2.1. Análisis de la arquitectura

Realizar una revisión de las características de seguridad

Se identifican las características de seguridad y su configuración de implementación (autenticación, control de acceso, uso de criptografía, etc.), luego se inspecciona el diseño y los parámetros de tiempo de ejecución para buscar problemas que podrían ocasionar que estas funciones fallen en su propósito o resulten insuficientes.

Deben considerar la aplicabilidad y el uso correcto de características de seguridad en todos los niveles que constituyen la arquitectura y entorno operativo, porque muchas de las aplicaciones de hoy en día ya no son de solo tres niveles, involucran componentes diseñados con una variedad de niveles como navegador, endpoints, SaaS de terceros, microservicios, motores de orquestación, etc. En algunos casos este proceso se puede agilizar utilizando componentes de seguridad por diseño.

Mayormente esta revisión se hace con análisis basados en listas de verificación y procedimientos de modelado de amenazas.

Utilizar lista de verificación (*checklist*) de principios de seguridad

Se realiza durante el diseño, como, por ejemplo, defensa en profundidad, separación de privilegios, uso de valores predeterminados seguros, simplicidad en el diseño de la funcionalidad de seguridad, fallas seguras, equilibrio entre seguridad y usabilidad, ejecución con privilegios mínimos, protección del eslabón más débil, evasión de la seguridad por oscuridad, etc.

Limítelos de manera que sólo requieran una pequeña cantidad de esfuerzo adicional más allá del costo normal de implementación de los requisitos funcionales. Todo lo que sea más grande, puede programarlo para versiones futuras.

5.1.2.2. Modelos de ataque

Recopilar y utilizar inteligencia de ataque

El SSG garantiza que la organización esté al día con los nuevos tipos de ataques, vulnerabilidades y la forma en que se explotan. Lo hace revisando la matriz empresarial de MITRE ATT&CK que muestra las tácticas y técnicas de los tipos de ataque en sus diferentes etapas, donde es posible encontrar ejemplos, referencias (fecha en que sucedió, en la que se detectó y las plataformas que son propensas) y sugerencias para su detección y mitigación, también asistiendo a conferencias técnicas, aunque menos económico y monitoreando los foros de los atacantes, luego correlaciona esa información con lo que acontece en la organización (puede aprovechar la automatización para obtener registros operativos y telemetría). Usar un servicio comercial de suscripción puede ser otra de las formas para recolectar inteligencia de ataque relacionada a las aplicaciones, API, contenedorización, orquestación, entornos en la nube, etc. Independientemente de la fuente de información que se emplee, la información debe adecuarse a las necesidades de la organización y esta debe ser digerible y útil para los diseñadores, arquitectos, desarrolladores, evaluadores, ingenieros DevOps e ingenieros de confiabilidad.

Mantener y utilizar una lista de los N principales ataques posibles

El SSG transforma periódicamente una lista creciente que contiene diversos tipos de ataques en una lista prioritaria más corta para enfocar los esfuerzos de prevención de la organización en esta última, a la cual se le denomina los N principales, y la usa para impulsar el cambio. Esta lista inicial se nutre tanto de fuentes internas como externas a la organización. La priorización lo hace en base a una posible pérdida comercial o en base a ataques exitosos

ocurridos anteriormente hacia su *software*. No es necesario actualizar muy frecuentemente la lista de los N principales, y la clasificación de los ataques puede realizarse de manera aproximada. Por ejemplo, el SSG podría presentar dos veces al año la lista de ataques que la organización debería estar preparada para contrarrestarlos "ahora", "pronto" o "algún día".

Recopilar y publicar historias de ataques

Para propagar el conocimiento y obtener el mayor provecho de las lecciones en la materia, el SSG recopila y publica las historias de los ataques hacia el *software* de la organización. Lo hace incluyendo la discusión en un portal de seguridad o en las reuniones de fin de semana. Todos los ataques caben ser mencionados, tanto los fallidos como los exitosos y el hecho de analizar la información histórica de estos ayuda a justificar la importancia de la seguridad del *software* en el contexto de la organización. Esto último es útil para las capacitaciones, porque ayuda a no enfocarse mucho en la lista de las 10 principales de OWASP, ya que esta muestra la realidad de otras organizaciones. Exponer de manera muy reservada la información acerca de los ataques no contribuye a enriquecer al personal a través de la experiencia.

Realizar modelado de amenazas

Esta es una actividad estructurada para identificar, evaluar y administrar amenazas del sistema, fallas de diseño arquitectónico y mitigaciones de seguridad recomendadas. Debe usarse en entornos donde existe un riesgo de seguridad significativo. Se puede aplicar a nivel de componente, aplicación o sistema. Está destinado a ayudar a los equipos de desarrollo a comprender qué riesgos existen en lo que se está construyendo, qué podría salir mal y cómo se pueden mitigar o remediar los riesgos.

Por lo general, se realiza como parte de la fase de diseño o como parte de una evaluación de seguridad, aunque también es útil durante las etapas de identificación de requisitos seguros, la revisión de código seguro y las pruebas de penetración. Se realiza entre propietarios de productos, arquitectos, campeones de seguridad y evaluadores de seguridad.

Empiece exponiendo al modelado de amenazas a los equipos y las partes interesadas para aumentar la conciencia de seguridad y crear una visión compartida acerca de la seguridad del sistema.

Realícelos para aplicaciones de alto riesgo y utiliza listas de verificación de amenazas simples, como STRIDE. Evite talleres extensos y listas demasiado detalladas de amenazas de

baja relevancia. Si agrega una nueva funcionalidad a una aplicación existente, mire solo las funciones recién agregadas en lugar de tratar de cubrir todo el alcance. Un buen punto de partida son los diagramas existentes que anota durante los talleres de discusión.

Siempre debe conservar el resultado de una discusión de modelado de amenazas para su uso posterior. Para este ejercicio puede usar una hoja de papel o una pizarra blanca o inteligente. A continuación, se detallan los pasos para realizar el modelado de amenazas.

i) Definir el alcance y la profundidad del análisis con las partes interesadas. Esto significa establecer parámetros y límites para los componentes que deben incluirse en la revisión. Este paso proporciona una comprensión del esfuerzo general y el cronograma de entrega esperado para la evaluación.

ii) Recopilar datos. El equipo de modelado de amenazas lee y revisa los artefactos proporcionados por el equipo de desarrollo, como, descripciones arquitectónicas, documentos de diseño de software, especificaciones de API, documentación del usuario final, código fuente y artefactos relacionados. Este análisis preliminar se utiliza para identificar áreas de alto riesgo en la plataforma, que formarán la base para todas las actividades futuras en el modelo de amenaza. Asimismo, el equipo seleccionará a los miembros clave de los equipos comerciales y de aplicaciones para entrevistarlos. Las entrevistas se centrarán en proporcionar una comprensión más profunda de la aplicación para evaluar las áreas de riesgo. Estas entrevistas brindan a los analistas de amenazas conocimientos clave sobre el diseño y los detalles de implementación de la aplicación.

iii) Modelar el sistema. Los analistas de amenazas crean un diagrama simplificado de todos los componentes dentro del alcance, adyacentes o de conexión (algunos que pueden estar fuera del alcance), así como las conexiones y sus protocolos.

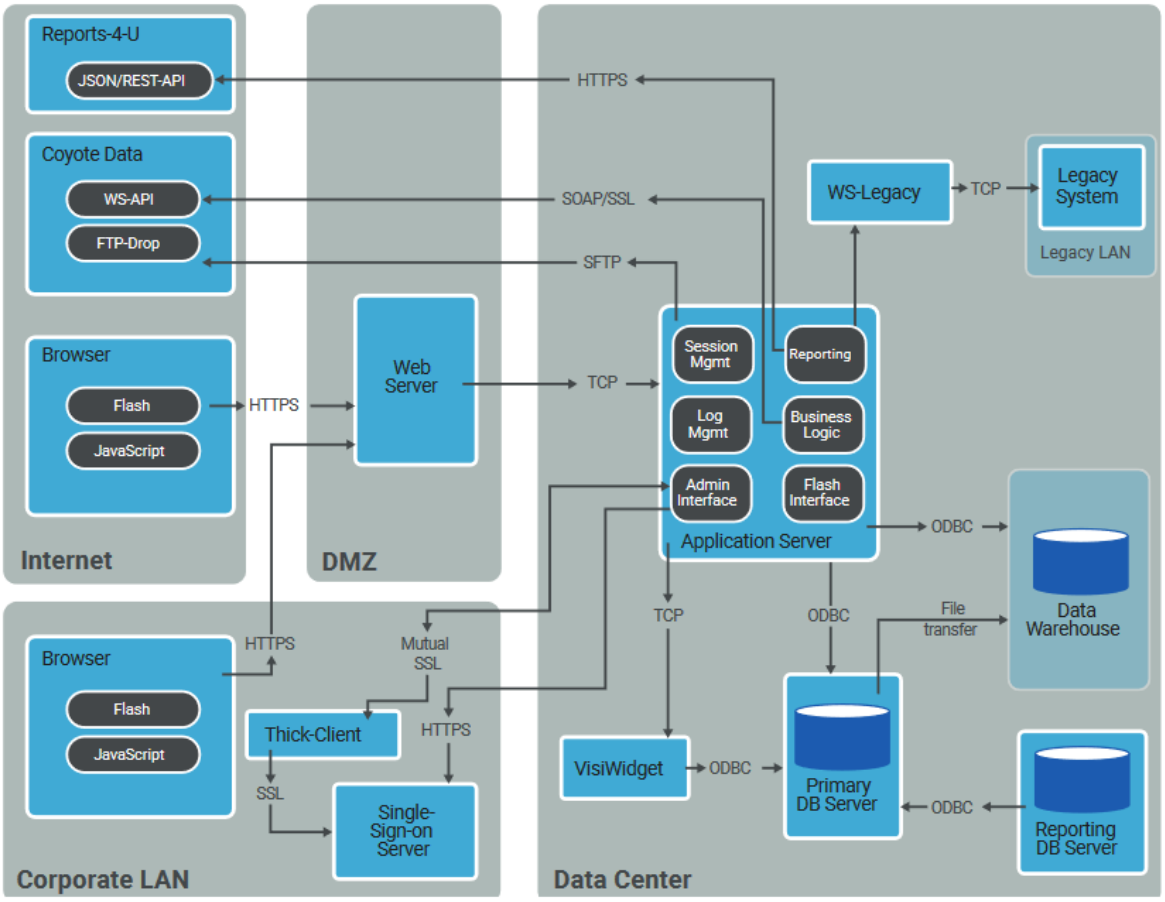
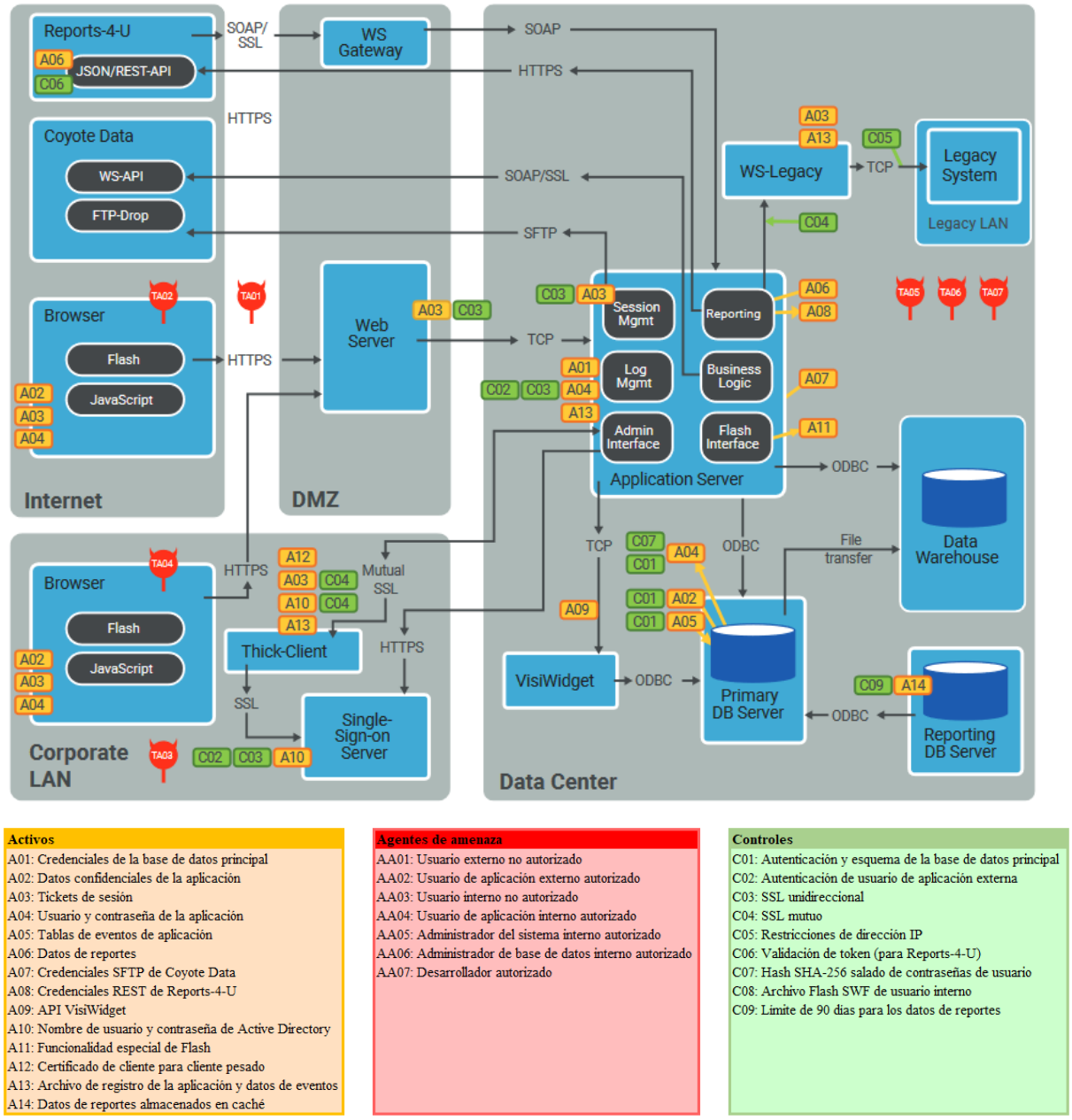


Figura 21. Diagrama de componentes principales del sistema

iv) Realizar análisis de amenazas. Los analistas de amenazas agregan activos, controles, agentes de amenazas y zonas de confianza para completar el diagrama de componentes principales del sistema. Este diagrama debe enumerar todas las posibles amenazas realistas para el sistema, centrándose en los datos, el flujo de datos, la plataforma y la seguridad operativa. Además, debe identificar controles débiles o faltantes que pueden conducir a comprometer la seguridad. Una vez identificados, marcan la ubicación de cada amenaza, como lo ilustra la Figura 22.



Activos
A01: Credenciales de la base de datos principal
A02: Datos confidenciales de la aplicación
A03: Tickets de sesión
A04: Usuario y contraseña de la aplicación
A05: Tablas de eventos de aplicación
A06: Datos de reportes
A07: Credenciales SFTP de Coyote Data
A08: Credenciales REST de Reports-4-U
A09: API VisiWidget
A10: Nombre de usuario y contraseña de Active Directory
A11: Funcionalidad especial de Flash
A12: Certificado de cliente para cliente pesado
A13: Archivo de registro de la aplicación y datos de eventos
A14: Datos de reportes almacenados en caché

Agentes de amenaza
AA01: Usuario externo no autorizado
AA02: Usuario de aplicación externo autorizado
AA03: Usuario interno no autorizado
AA04: Usuario de aplicación interno autorizado
AA05: Administrador del sistema interno autorizado
AA06: Administrador de base de datos interno autorizado
AA07: Desarrollador autorizado

Controles
C01: Autenticación y esquema de la base de datos principal
C02: Autenticación de usuario de aplicación externa
C03: SSL unidireccional
C04: SSL mutuo
C05: Restricciones de dirección IP
C06: Validación de token (para Reports-4-U)
C07: Hash SHA-256 salado de contraseñas de usuario
C08: Archivo Flash SWF de usuario interno
C09: Límite de 90 días para los datos de reportes

Figura 22. Diagrama modelo de seguridad del sistema

Una vez realizado este, identifique qué podría salir mal (es decir, las amenazas) utilizando métodos como STRIDE para todos los flujos de datos que cruzan un límite de confianza.

v) Realizar análisis de riesgos. Revise el flujo de datos y las conexiones componente por componente para producir una lista enumerada de escenarios de amenazas realistas (posibles ataques), haciéndose las siguientes preguntas: ¿Existen rutas por las que un agente de amenaza pueda llegar a un activo sin pasar por un control? ¿Podría un agente de amenaza burlar este control de seguridad? ¿Qué debe hacer un agente de amenaza para burlar este control?

Si se puede acceder a un activo sin pasar por un control, puede haber un riesgo potencial o una falla de diseño. Al pasar por un control, considere si el control detendría a un agente de amenaza y si ese agente de amenaza puede tener métodos para eludir el control de seguridad.

Tabla 25. Análisis de riesgos

Activos / Servicios	Amenaza	Escenario de ataque	Impacto	Probabilidad	Riesgo	Prioridad	Controles (Mitigación)
AGROCORE SAAS	*Spoofing*	EA01: Usuario externo no autorizado suplanta a otro usuario	2	3	6	6	MFA, SSL mutuo, protección de secretos
		EA02: Usuario externo no autorizado suplanta a un usuario con un rol de administrador	5	2	10	5	MFA, SSL mutuo, protección de secretos
	Tampering	EA03: Usuario externo no autorizado manipula datos	4	3	12	4	Controles de acceso en todas las páginas, protocolos resistentes a la manipulación
	Repudiation	EA04: Un usuario de aplicación interno autorizado niega sus acciones	2	2	4	7	Bitácoras replicadas
	Information disclosure	EA05: Un usuario de aplicación interno autorizado puede acceder a los datos de otro usuario	4	4	16	2	Sesión segura, *Zero Trust*
		EA06: Un usuario externo no autorizado accede a la base de datos principal	5	3	15	3	Encriptar datos en la base de datos
	Denial of service	EA07: Un usuario externo no autorizado impide a los usuarios acceder a la aplicación	5	5	25	1	*WAF*, redundancia, *failover*
	Elevation of privilege	EA08: Un usuario externo no autorizado actuando como administrador	4	3	12	4	Sesión segura

Identificar requisitos de seguridad

Al revisar los requisitos funcionales del proyecto de *software*, se identifican los requisitos de seguridad relevantes (expectativas) para dicha funcionalidad razonando sobre la

confiabilidad, integridad o disponibilidad deseada del servicio o los datos ofrecidos por el proyecto de *software*.

Los requisitos establecen el objetivo (por ejemplo, en el proceso de registro los datos personales deben transferirse y almacenarse de manera segura), mas no la medida real para lograr el objetivo (por ejemplo, usar TLSv1.2 para la transferencia segura.

Los objetivos de seguridad pueden relacionarse con la funcionalidad de seguridad específica que necesita agregar a la aplicación (por ejemplo, identificar al usuario de la aplicación en todo momento) o con la calidad y el comportamiento general de la aplicación (por ejemplo, garantizar que los datos personales estén debidamente protegidos en tránsito), lo que no necesariamente conduce a una nueva funcionalidad.

Siga las buenas prácticas para redactar los requisitos de seguridad. Hágalos específicos, relevantes y con límites de tiempo (SMART). Tener cuidado con agregar requisitos demasiado generales, por ejemplo, la aplicación debe protegerse contra OWASP Top 10. Si bien pueden ser ciertas, no agregan valor a la discusión.

Realizar evaluaciones a sus proveedores de *software*

Realice una evaluación a sus proveedores para comprender las fortalezas y debilidades de estos. Use una lista de verificación básica o realice entrevistas (puede usar cuestionarios) para revisar sus prácticas y entregas típicas. Esto le da una idea de cómo se organizan y de otros aspectos para evaluar si necesita tomar acciones adicionales para mitigar los riesgos potenciales. Idealmente, hable con diferentes roles en la organización, o incluso establezca una pequeña evaluación de madurez para este fin. Los proveedores sólidos ejecutan su propio programa de garantía de *software* y pueden responder a la mayoría de sus preguntas. Si los proveedores tienen competencias débiles en seguridad de *software*, discuta con ellos cómo y en qué medida planean trabajar en esto y evalúe si es suficiente para su organización. Un proveedor de *software* podría estar trabajando en un proyecto de bajo riesgo, pero ello podría cambiar. Es importante que sus proveedores entiendan y se alineen con su apetito al riesgo y puedan cumplir con sus requisitos en esa área. Haz explícito lo que esperas de ellos y discútelo claramente. Por ejemplo, podría hacer preguntas como: ¿Realiza un análisis de la arquitectura de su software? ¿Tiene un procedimiento definido para este? ¿Realiza un análisis para detectar vulnerabilidades tanto al software como a sus componentes? ¿Capacita a su personal en temas como técnicas de seguridad de software? ¿Sus desarrolladores cuentan con alguna certificación en prácticas de codificación segura? ¿Realizan inducción al nuevo

personal sobre cómo crear, operar e implementar código seguro, el ciclo de vida de desarrollo seguro y recursos de seguridad internos? ¿Revisa y evalúa la información de productos y servicios de sus proveedores acerca de nuevas amenazas potenciales de forma regular?

5.1.3. FASE 3: IMPLEMENTACIÓN

Esta fase tiene como objetivo descubrir defectos al *software* antes del lanzamiento al entorno de producción, lo cual incluye hacer una revisión de código y las pruebas de seguridad con herramientas de tipo SAST y DAST.

5.1.3.1. Revisión de código

Realizar una revisión de código oportuna y obligatoria

Debe empezar dándole prioridad a la revisión de las aplicaciones de alto riesgo para que sean atendidas de manera oportuna, por ejemplo, al buscar problemas de seguridad lo hace mediante la revisión del diseño con la revisión del código fuente, sus dependencias y además en la configuración de artefactos de implementación y metadatos de automatización, esto quiere decir, en los contenedores e infraestructura como código, respectivamente. Esta revisión se puede hacer usando herramientas o de manera manual, en todos los casos debe ser un proceso proactivo. Cuando emergen nuevas tecnologías puede ser necesario el uso de nuevos enfoques.

Recordar que es obligatorio para todos los proyectos, inclusive los aparentemente inocuos deben cumplir las condiciones de seguridad prescritas para continuar hacia las siguientes etapas o permanecer en producción. Para aquellos proyectos que son de bajo riesgo pueden depender más de la automatización, mientras que los de alto riesgo pueden no tener restricciones en la cantidad de horas que le dedican sus revisores. Contar con un estándar mínimo aceptable obliga a que los proyectos que no pasan sean corregidos y reevaluados.

Utilizar herramientas automatizadas

Existen múltiples herramientas para la revisión de código de manera automatizada, una de ellas es mediante el análisis estático con herramientas de tipo SAST, haciendo que el proceso sea más eficiente y consistente. La automatización no reemplaza al juicio humano, pero brinda definición al proceso de revisión y experiencia en seguridad a los revisores que

comúnmente no son expertos en seguridad. Hay que tener en cuenta que una herramienta específica puede no cubrir todos los lenguajes existentes.

Puede avanzar hacia la automatización mediante la instrumentación del análisis estático en los flujos de trabajo de administración de código fuente o control de versiones (por ejemplo, *pull requests* o también conocidos como solicitudes de incorporación de cambios) y flujos de trabajo de canalización de entrega (*build, package and deploy*) para que la revisión sea más eficiente, consistente y acorde con la frecuencia de publicación. Ya sea para revisar una parte del código fuente de forma incremental (como cuando un desarrollador confirma código nuevo o cambios pequeños) o para realizar un análisis completo del programa escaneando todo el código fuente.

Con el tiempo puede personalizar estas reglas automatizadas por ejemplo eliminando los falsos positivos repetidos de tal manera que ayude a descubrir defectos de seguridad específicos de los estándares de codificación propios de la organización o del middleware que utiliza, basado en el marco o el proporcionado por la nube. Asimismo, puede asignar mentores de herramientas que estén disponibles para mostrar a los desarrolladores cómo aprovechar al máximo las herramientas de revisión de código, estableciendo la configuración correcta o ayudando a interpretar los resultados. Tanto para las reglas personalizadas como para el desenvolvimiento como mentores, pueden ser los campeones de seguridad.

Categorías de herramientas de prueba de seguridad de aplicaciones

En el mercado de herramientas de pruebas de seguridad de aplicaciones (AST) existen productos y servicios que analizan y prueban aplicaciones con el objetivo de encontrar vulnerabilidades de seguridad. Este mercado es altamente dinámico y evoluciona rápidamente debido a los cambios de las arquitecturas de las aplicaciones y de las tecnologías. Así pues, podemos encontrar herramientas con capacidades de pruebas básicas, entre ellas, las pruebas estáticas (SAST), dinámicas (DAST), interactivas (IAST) y análisis de composición de *software* (SCA); y herramientas con capacidades especializadas opcionales que complementan a las básicas. Asimismo, los proveedores las ofrecen como *software* local o mayormente como *software* como servicio.

Entre las categorías de herramientas con capacidades básicas existen las siguientes:

AST estático (SAST): realiza un análisis del código fuente, código de bytes o el código binario de una aplicación para buscar vulnerabilidades, por lo general durante la fase de desarrollo o de pruebas dentro del SDLC.

AST dinámico (DAST): realiza un análisis de las aplicaciones en estado de ejecución durante la fase de pruebas o de operaciones, por lo general simula ataques contra aplicaciones web, aunque cada vez más también contra APIs y analiza sus reacciones para determinar si es vulnerable.

AST interactivo (IAST): este tipo de herramienta es una combinación de SAST y DAST que prueba a nivel de código si es que las vulnerabilidades conocidas son verdaderamente explotables mientras la aplicación se encuentra en ejecución, también llamada instrumentación de una aplicación en ejecución, y lo hace a través de *Java Virtual Machine* (JVM) o .NET *Common Language Runtime* (CLR). Asimismo, son buenas reduciendo el número de falsos positivos y una buena opción para entornos ágiles y DevOps, ya que usar SAST y DAST por separado puede tomar algo de tiempo como para cumplir el ciclo de entrega en este tipo de entornos.

SCA: realizan una identificación de los componentes de código abierto y algunas de ellas los componentes comerciales utilizados en una aplicación, identificando vulnerabilidades conocidas, probables problemas de licencia y riesgos operativos.

Entre las categorías de herramientas con capacidades especializadas opcionales existen las siguientes:

Pruebas de API: Dado que es posible que las herramientas AST tradicionales no las prueben totalmente, entran en juego aquellas con capacidades especializadas como las que analizan API en entornos tanto de desarrollo como de producción y prueban el código fuente de esta, así también ingieren el tráfico registrado o las definiciones de API para respaldar la prueba de esta en ejecución, se encuentran como funciones típicas.

Orquestación y correlación de seguridad de aplicaciones (ASOC): facilitan las pruebas y corrección de vulnerabilidades al automatizar los flujos de trabajo y procesar los hallazgos. Automatizan las pruebas de seguridad en el SDLC y durante los proyectos, incorporando datos de diversas fuentes. Estas herramientas correlacionan y analizan los hallazgos para concentrar los esfuerzos y facilitar su interpretación, clasificación y corrección. Actúan como una capa de gestión y orquestación entre el desarrollo y las pruebas de seguridad.

AST crítico para el negocio: Esta categoría se refiere a las aplicaciones de negocio de gran escala como SAP, Oracle, Salesforce, las cuales identifican vulnerabilidades dentro del código de la aplicación en lenguajes personalizados específicos del proveedor por ejemplo ABAP u otros, así como las configuraciones incorrectas, las vulnerabilidades conocidas y los errores que resultan en exposiciones de seguridad.

Seguridad del contenedor: analiza las imágenes del contenedor o un contenedor completamente instanciado previa a la implementación para buscar problemas de seguridad. Se centran en evaluar vulnerabilidades y fortalecer la configuración. Además, pueden funcionar como parte del proceso de implementación de aplicaciones o integrarse con repositorios de contenedores, de tal forma que puedan realizar evaluaciones de seguridad a medida que las imágenes se almacenan para un uso posterior.

Habilitación del desarrollador: ayudan a los desarrolladores y miembros del equipo de ingeniería a crear código seguro, capacitándolos en seguridad y guiándolos en la remediación de vulnerabilidades, ya sea de manera independiente o dentro del mismo entorno de desarrollo.

Fuzzing: También conocida como prueba no determinista. Inyectan datos aleatorios, mal formados o inesperados a un programa para reconocer posibles vulnerabilidades como el bloqueo de la aplicación o el comportamiento anormal, la pérdida de memoria o el desbordamiento de búfer u otras condiciones que dejen al programa en un estado indeterminado. Se puede usar con la mayoría de los tipos de programas, aunque es mayormente útil con aquellos sistemas que dependen de cantidades significativas de procesamiento de entrada como aplicaciones y servicios web, APIs.

Pruebas de infraestructura como código (IaC): IaC es la creación, aprovisionamiento y configuración de la infraestructura de computación definida por *software*, red y almacenamiento, como código fuente. Estas herramientas ayudan a reconocer problemas de seguridad relacionados con entornos operativos específicos, a asegurar que se cumplan los estándares comunes que fortalecen la configuración, a identificar secretos incluidos y a realizar otras pruebas que respaldan los estándares específicos de la organización, así como los requisitos de cumplimiento.

AST Móvil (MAST): tienen características especializadas que se centrar en probar las vulnerabilidades y los problemas de seguridad específicos de las aplicaciones móviles como manejo y validación de certificados, prevención de fuga de datos, conexiones wifi falsas, jailbreak o rooting, etc. que se ejecutan en sistemas operativos como iOS, Android y otros. Generalmente son una combinación de SAST y DAST que se han optimizado para admitir lenguajes y marcos utilizados en el desarrollo de aplicaciones móviles y de internet de las cosas (IoT).

Figura 23. Cuadrante mágico para pruebas de seguridad de aplicaciones. Fuente: Gartner abril 2022

A continuación, mostramos el método para evaluar la seguridad de las aplicaciones el cual consta de tres subfases: (1) Definición; (2) Ejecución; y (3) Resultados y están compuestas de la siguiente manera.

1. Definición

1.1 Establecer los objetivos de evaluación. Establecemos cuál es la aplicación objetivo y el alcance de la evaluación, es decir, el tipo de vulnerabilidades a evaluar clasificadas por las herramientas como: críticas, altas, medias o bajas.

1.2 Identificar características del objetivo. Identificamos todos los detalles técnicos de la aplicación, como, lenguaje de programación, entorno de ejecución e implementación; y corroboramos el contexto y el funcionamiento de la aplicación.

1.3 Seleccionar herramientas de análisis. Identificamos las herramientas y verificamos su compatibilidad con el entorno de evaluación.

1.4 Definir métricas. Definimos métricas para clasificar los resultados, como, número de verdaderos positivos, número de falsos positivos, porcentaje de verdaderos positivos, porcentaje de falsos positivos.

2. Ejecución

2.1 Análisis estático semiautomático. Obtenemos los resultados de la ejecución de la herramienta SAST. Luego auditamos y clasificamos manualmente los resultados como falsos positivos, verdaderos positivos.

2.2 Análisis dinámico semiautomático. Identificamos vulnerabilidades que el análisis estático no cubre, con ello obtenemos nuevos resultados para ampliar grado de cobertura. Es independiente del paso 2.1, por lo que puede ejecutarse en paralelo al paso anterior.

2.3 Resultados. Analizamos, correlacionamos e identificamos el grado de cobertura del análisis de seguridad en base a las vulnerabilidades y métricas identificadas entre SAST y DAST.

3. Resultados

3.1 Clasificar resultados. Clasificamos las vulnerabilidades de seguridad detectadas aplicando las métricas seleccionadas a los resultados de la fase de ejecución.

5.1.3.2. Pruebas de seguridad de *software*

Integrar herramientas de seguridad de caja opaca en el proceso de QA

Las herramientas de análisis dinámico o también llamadas de caja negra son importantes porque ayudan a tener una mirada desde el punto de vista del atacante, aunque de forma genérica. Estas herramientas son relevantes para las aplicaciones web, pero también se pueden encontrar para entornos en la nube, contenedores, aplicaciones móviles, etc. Además, pueden conectarse a cadenas de herramientas internas, ser proporcionadas por cadenas de herramientas basadas en la nube o ser utilizadas directamente por el equipo de ingeniería. Esta actividad está cubierta en la subfase de ejecución de la actividad anterior.

5.1.4. FASE 4: OPERACIONES

5.1.4.1. Entorno de *software*

Usar monitoreo de entrada de la aplicación

Supervisar la entrada al *software* cobra importancia cuando se trata de detectar ataques. Para ello existen sistemas de monitoreo que escriben archivos de registro, estos pueden ser útiles cuando una persona o un robot los revisa periódicamente y toma acciones. Para las aplicaciones web, los WAF (cortafuegos de aplicaciones web) pueden hacer esta tarea. Para otras pilas de *software* y tecnología, como los dispositivos móviles, probablemente

necesiten soluciones de monitoreo de entrada específicas para estas. Para los casos de *software* sin servidor y contenedores puede ser necesario interactuar con el *software* del proveedor para obtener los registros y datos de monitoreo propicios. Es probable que otros tipos de *software* necesiten otros enfoques, incluida la instrumentación (medir el desempeño de un *software*) en tiempo de ejecución. En esta actividad, el SSG puede ser el responsable del cuidado y alimentación del sistema de monitoreo, cabe señalar que la respuesta a los incidentes no forma parte de esta actividad.

Por ejemplo, utilizamos el *ModSecurity Core Rule Set* (CRS) que es un proyecto de OWASP que contiene un conjunto de más de 200 reglas para la detección de ataques. Esto nos sirve como primera línea de defensa frente a una amplia gama de ataques, incluidos los del OWASP Top Ten y con un mínimo de falsos positivos, ya que estos se han reducido en más del 90 % en su versión 3 con una instalación por defecto. Además, se puede usar con cortafuegos de aplicaciones web como ModSecurity o compatibles como es el caso de los cortafuegos de los proveedores en la nube Google Cloud Platform, Azure, Oracle Cloud Infrastructure. Cabe resaltar que, aunque en un inicio sólo estuvo disponible para el servidor web apache ahora también lo está para Nginx, IIS y otros.

Ahora bien, en el primer paso necesitamos definir un plan de supervisión que incluya lo siguiente: i) Definir los objetivos de la monitorización. Por ejemplo, el 99.99% de las solicitudes deben pasar por el conjunto de reglas sin falsos positivos, esto es, sólo 1 de cada 10 000 solicitudes puede ser falso positivo. ii) Definir los recursos a monitorizar. Por ejemplo, servidor web apache. iii) Definir la frecuencia con la que monitorizará dichos recursos. Por ejemplo, semanal. iv) Definir las herramientas de monitoreo que utilizará. Por ejemplo, cloudwatch, cloud monitoring (off-premise), Nikto (escáner de seguridad, on-premise). v) Definir el encargado de realizar las tareas de monitoreo. Por ejemplo, el SSG. vi) Definir la persona que recibirá las notificaciones cuando surjan problemas. Por ejemplo, líder SSG.

En el segundo paso, necesitamos tener un método para afrontar los falsos positivos y así protegernos realmente de los atacantes. Para ello, necesitamos i) Conocer las formas para reducir los falsos positivos ii) Establecer la configuración inicial iii) Identificar y reducir los falsos positivos iv) Reducir el umbral de anomalía por iteraciones.

En el tercer paso, necesitamos establecer un punto de referencia del desempeño normal del entorno, por ejemplo, con un umbral de anomalías de 10. Para ello debemos monitorear como mínimo el número de solicitudes web permitidas y bloqueadas. Luego medimos el desempeño en diferentes momentos y bajo diferentes condiciones de carga. Finalmente, obtenemos datos históricos y comparamos con los datos de desempeño actuales

para identificar los patrones de desempeño normales y anómalos y determinar si continuamos optimizando las reglas.

Definir configuraciones y parámetros de implementación seguros

Es necesario crear o contar con una guía de instalación o de configuración y que esta sea clara para implementar los artefactos de *software*, ya sea de manera manual o automatizada (mediante infraestructura como código), para de esta manera ayudar a los miembros del equipo a instalar y a configurar el *software* de forma segura. De requerir algunos pasos especiales para garantizar una implementación segura, se pueden describir en una guía de configuración o directamente en la automatización de la implementación, incluida la información sobre proveedores y componentes de servicios en la nube. En algunos casos, este conocimiento no queda internamente, sino que también puede compartirlo con los clientes que adquieren el *software*. En caso de que la implementación se realice mediante automatización, esta debe ser comprensible no sólo para las máquinas sino también para las personas. Esta codificación puede hacerlo mediante secuencias de comandos de infraestructura, como, por ejemplo, Terraform, Cloud formation, Open stack heat Helm, Ansible o Chef. Para esta actividad, consultamos el Cheat Sheet Series Project de OWASP, el cual proporciona buenas prácticas de seguridad para diversos lenguajes de programación, tecnologías, con sus respectivas configuraciones y parámetros de implementación seguros, con el objetivo de ayudar a los desarrolladores, defensores, constructores a proteger sus aplicaciones.

ANEXO 4 – Catálogo de riesgos de seguridad del software agrupados por fase del SDLC

N.°	Riesgos de seguridad del software en la fase de ingeniería de requisitos
1	Los requisitos de seguridad a menudo se descuidan o se consideran como un requisito no funcional
2	Falta de negociación y gestión de requisitos de seguridad.
3	Falta de validación de requisitos de seguridad.
4	Evaluación de riesgos inadecuada
5	Falta de análisis de riesgos de seguridad
6	Falta de experiencia, conocimiento, orientación y capacitación en seguridad durante la documentación de los requisitos de seguridad
7	Falta de desarrollo de modelos de amenazas.
8	Falta de actividad de elicitación de requisitos de seguridad
9	Identificación e inicio de requisitos de seguridad inadecuados
10	Falta de documentación de requisitos seguros

11	Falta de priorización, gestión y categorización de los requisitos de seguridad.
12	Falta de lista de verificación de documentos de seguridad
13	Falta de comprensión compartida de las definiciones de requisitos
14	Plan inadecuado para la autenticación, autorización y privacidad de requisitos seguros
15	Falta de desarrollo de los correspondientes artefactos de requisitos de seguridad.
16	Falta de conciencia de los requisitos de seguridad en los clientes/usuarios
17	Asignación incorrecta de requisitos de seguridad
18	Deserialización insegura
19	Identificación incorrecta de dependencias de requisitos de seguridad
20	La empresa de desarrollo de software tiene poca capacidad para implementar características de seguridad en pequeños incrementos.
21	Identificación incorrecta de activos críticos y vulnerables
22	Cambio de requerimiento descansos
23	Falta de actualización del repositorio de requisitos de seguridad.
24	Los ajustes de los requisitos hacen que sea imposible conectar las especificaciones de los requisitos con los objetivos de seguridad
25	Falta de prototipo de seguridad.
	Riesgos de seguridad del software en la fase de diseño
26	Falta de desarrollo de modelos de amenazas durante la fase de diseño.
27	Falta de atención para seguir los principios de diseño de seguridad.
28	Falta de conocimiento, orientación y capacitación sobre el diseño de seguridad
29	Documentación de diseño segura inadecuada
30	Falta de creación y mantenimiento de modelos de casos de abuso y patrones de ataque
31	Revisión del diseño de seguridad inadecuado y su verificación.
32	Falta de desarrollo de diagrama de flujo de datos.
33	Realización incorrecta de la revisión de seguridad del diseño y la arquitectura
34	Restricción incorrecta para compartir el acceso a los recursos
35	Falta de revisión de especificaciones de diseño de seguridad
36	Falta de establecimiento de requisitos de diseño de seguridad.
37	Falta de implementación de decisiones de diseño de seguridad: (Protocolos criptográficos, estándares, servicios, marcos y mecanismos)
38	Falta de defensa en profundidad
39	Falta de control de acceso y trazabilidad
40	Falta de uso de patrones de diseño de seguridad.
41	Falta de características de encriptación y validación de datos de diseño
42	Funciones de registro de auditoría de diseño inadecuado
43	Error al manejar el error
44	Evaluación incorrecta de los riesgos de los componentes de terceros
45	El software parece tener más errores a medida que se vuelve más complejo
46	Condiciones de carrera
47	Uso de componentes vulnerables y detalles confidenciales de aplicaciones
48	Las prácticas de refactorización rompen las restricciones de seguridad
49	Falta de diversificación y ofuscación.
50	Uso de componentes con vulnerabilidades conocidas
	Riesgos de seguridad del software en la fase de codificación
51	Manipulación: es la modificación no autorizada de datos.
52	Inyección SQL.

53 Scripts de sitios cruzados, falsificación de solicitudes de sitios cruzados.
54 Denegación de servicios: es el proceso de hacer que un sistema o aplicación no esté disponible.
55 Repudio: es la capacidad de los usuarios (legítimos o no) de negar que hayan realizado acciones o transacciones específicas.
56 Revelación de información: es la exposición no deseada de datos privados.
57 Elevación de privilegios: ocurre cuando un usuario con privilegios limitados asume la identidad de un usuario privilegiado.
58 Suplantación de identidad: un intento de obtener acceso a un sistema utilizando una identidad falsa.
59 Conjetura de contraseña: falta de aplicación de la complejidad de la contraseña.
60 Desbordamiento de búfer y matriz.
61 Cifrado débil, comunicación insegura.
62 Código desordenado, código con malos olores, código muerto.
63 Código, inyección de comandos.
64 Problemas de cadena de formato.
65 ID de sesión vulnerable, robo de ID de sesión.
66 Hackear.
67 Hombre en el medio: ataque Man-in-the-middle: este ataque intercepta las comunicaciones entre dos componentes.
68 Desreferencia de puntero nulo.
69 Programación de aplicaciones inseguras, falta de lenguaje de programación de seguridad.
70 Reproducir defectos de ataque.
71 La seguridad del software, a menudo, falla porque su desarrollo generalmente se basa en bases ad-hoc o sigue procesos de desarrollo tradicionales.
72 Información sensible en código fuente.
73 Subprocesamiento inseguro
74 Uso de Ancho de Banda
75 Error al restringir el acceso a la URL
76 Falta de diferencias entre los roles del desarrollador y el rol del revisor de seguridad para tener resultados objetivos.
77 Redireccionamientos y reenvíos invalidados
78 Base de datos accesible
79 Aplicación HTTP en lugar de HTTPS
80 Phishing a través del marco
81 Marca de tiempo de respuesta de TCP
82 Los continuos cambios de código dificultan la realización de las actividades de aseguramiento.
83 Referencias a objetos directos inseguros
84 secuestro de DNS
85 Enviar parámetros sísmicos falsos
86 El cambio continuo de los procesos de desarrollo (para respaldar la lección aprendida) entra en conflicto con la necesidad de auditoría para uniformar los procesos estables.
87 Atributo de autocompletar no habilitado
88 Solicitud de cambio POST para GET
89 Directivas POST con parámetros invalidados
90 Página del servidor predeterminado
91 Inyecciones de enlaces
92 Certificados SSL débiles

93	Desviación del estado BPEL y ataques de inundación
94	Ataques de corte
95	Envenenamiento de cookie
	Riesgos de seguridad del software en la fase de prueba
96	Falta de pruebas de seguridad de análisis de penetración
97	Falta de pruebas de seguridad de análisis estático y dinámico.
98	Falta de revisión de seguridad final
99	Falta de pruebas de fuzz
100	Ataque de fuerza bruta
101	Falta de desarrollo de modelos de amenazas: ya que ayuda a desarrollar casos de prueba o planes de prueba.
102	Falta de pruebas funcionales y no funcionales.
103	Falta de revisión manual del código.
104	Falta de generación automática de parches
105	Falta de pruebas unitarias
106	Varios tipos de ataques (virus), malware, virus troyanos: un tipo de virus que es bien conocido por causar problemas y destrucción a las computadoras es un virus troyano.
107	Falta de desarrollo de un plan de prueba para describir el alcance y las actividades de las pruebas de software.
108	Parámetros sísmicos ilegales, validación de parámetros incompleta/inconsistente.
109	Uso correcto no válido de las herramientas de prueba de seguridad.
110	Prueba son, en general, todos los casos de vulnerabilidades.
111	Las pruebas no cubren en general, todos los casos de vulnerabilidades
112	Las pruebas de seguridad son, en general, difíciles de automatizar.
113	Falta de análisis de redundancia.
	Riesgos de seguridad del software en la fase de implementación
114	Falta de configuración de software predeterminada
115	Cierre de sesión implementado incorrectamente
116	Servicios y puertos mal habilitados
117	Ignorar interrupciones de seguridad
118	Falta de validación de salida
119	Uso inadecuado de API seguras
120	Falta de retroalimentación de seguridad
121	Falta de respuesta en la planificación y ejecución
122	Conexiones simultáneas desde diferentes IPS
123	Firma SMB (Bloque de mensajes del servidor) No se requiere
124	Falta de certificación en versión final y archivo.
125	Falta de garantías para especificar las expectativas y requisitos del cliente.
126	Firma de código incorrecta
127	Desinfección inadecuada de datos y eliminación segura
128	Falta de actualización de modelos de amenazas.
	Riesgos de seguridad del software en la fase de mantenimiento
129	Falta de confianza en la seguridad.
130	Falta de métodos adecuados para descubrir nuevas amenazas en el sistema.
131	Falta de encontrar el área de superficie de ataque para las nuevas amenazas
132	No desarrollar parches de seguridad para las amenazas.
133	Configuración, gestión de vulnerabilidades y control de cambios inadecuados

134	Las actividades de seguridad aumentan el costo del software.
135	Ataques de tiempo
136	Incapacidad para ejecutar actualizaciones de software o cambiar nombres de usuario y contraseñas
137	Falta de optimización de registro
138	Falta de educación de los usuarios en el uso de la aplicación de software de manera segura
139	Necesidad de verificar la corrección de la vulnerabilidad
140	Falta de mantenimiento de la seguridad del software.
141	Falta de gestión de relaciones en el informe de errores.
142	Falta de monitoreo continuo y mejora de la evaluación de la seguridad.
143	Falta de colaboración con organizaciones externas en la promoción de la seguridad del sistema.
144	Falta de pilotaje de ideas innovadoras, tecnologías y herramientas de seguridad para mejorar la seguridad organizacional.
145	Falta de asistencia del gobierno para las reglas adecuadas para el delito cibernético.

Fuente: "Systematic Literature Review on Security Risks and its Practices in Secure Software Development" [9]

ANEXO 5 – Implementación parcial de la metodología

Definición del estándar de seguridad

N°	Verificaciones de seguridad	Severidad	Identificación	Corrección
1	Same Site Cookie	crítico	En el archivo config/session.php la opción llamada 'same_site' se encuentra establecida como 'none' ó null	En su archivo config/session.php, configure la opción de configuración same_site como "lax" o "estrict"
2	Secure Cookie	crítico	En el archivo config/session.php la opción llamada 'secure' se encuentra establecida como false, ya que esto expone la aplicación a ataques de intermediarios.	En su archivo config/session.php (o en la variable env correspondiente SESSION_SECURE_COOKIE), configure la opción de configuración secure como true (las cookies de sesión solo se enviarán al servidor si el navegador tiene una conexión HTTPS. Recomendable si su app es sólo https) ó null (se habilita automáticamente en las conexiones HTTPS)
3	Ext	c	Buscar en todos los archivos,	Cambiar el método all, por only o validated

	ract Function	rítico	extract(request()->all()); ó cualquiera de sus formas como extract($request->all()) Esta función importa variables de una matriz asociativa. Esta función trata las claves como nombres de variables. Podría conducir potencialmente al secuestro de variables superglobales como $_COOKIE o $_SERVER, o incluso variables establecidas en el alcance de la llamada a la función, por ejemplo, si los datos de la solicitud contienen una variable _SERVER[HTTP_USER_AGENT], entonces puede cambiar el secuestro del agente de usuario para la solicitud.	extract(request()->only(['search', 'id']));
4	Session Timeout	alto	En el archivo config/session.php la opción de configuración 'lifetime' excede 1440 min. Esta condición se omite si la sesión está configurada para que caduque automáticamente al cerrarse o si la aplicación no tiene estado (no usa sesiones).	En su archivo config/session.php (o en el valor env SESSION_LIFETIME), configure la opción de configuración lifetime a un valor menor a un día (1440 min). El valor predeterminado de Laravel es de 2 horas, parece una opción inteligente para la mayoría de las aplicaciones: 'lifetime' => env('SESSION_LIFETIME', 120), Alternativamente, puede configurar la sesión para que caduque automáticamente cuando el navegador se cierre, en su archivo config/session.php: 'expire_on_close' => true,
5	Arbitrary File Upload	alto	Si permite que los usuarios carguen archivos, también debe validar los tipos de archivos que se están cargando. El no hacerlo puede resultar en ataques de ejecución de código arbitrario. Estos ataques implican primero cargar archivos ejecutables maliciosos (como archivos PHP) y luego activar su código malicioso visitando la URL del archivo (si es público).	Agregue reglas de validación para validar también el tipo o extensión MIME del archivo: $request->validate(['filename' => 'file|mimes:jpg,png,bmp']);
6	Clickjacking	alto	Si no procesa páginas en cualquiera de las etiquetas <frame>, <iframe>, embed> o <object> debe establecer el encabezado de seguridad X-Frame-Options para protegerse contra los ataques de clickjacking. Este encabezado indica si se debe permitir que un navegador represente una página en cualquiera de las etiquetas mencionadas.	Puede agregar el encabezado X-Frame-Options en la configuración de su servidor web. Para Nginx, puede usar la directiva add_header en su bloque server ó location: add_header X-Frame-Options "SAMEORIGIN"; Para Apache, puede usar la directiva Header en su contenedor <VirtualHost>, <Directory> ó <Location>: Header always set X-Frame-Options "SAMEORIGIN". Alternativamente, si no desea establecer el encabezado en el nivel del servidor web, puede agregar el middleware FrameGuard

				al grupo web de middleware en su clase App\Http\Kernel: protected $middlewareGroups = ['web' => [\Illuminate\Http\Middleware\FrameGuard::class, \App\Http\Middleware\EncryptCookies::class, \Illuminate\Cookie\Middleware\AddQueuedCookiesToResponse::class, \Illuminate\Session\Middleware\StartSession::class, // \Illuminate\Session\Middleware\AuthenticateSession::class, \Illuminate\View\Middleware\ShareErrorsFromSession::class, \App\Http\Middleware\VerifyCsrfToken::class, \Illuminate\Routing\Middleware\SubstituteBindings::class,], 'api' => ['throttle:api', \Illuminate\Routing\Middleware\SubstituteBindings::class,],];
7	Command Injection	alto	Si su aplicación ejecuta un comando de shell creado con datos proporcionados por el usuario sin escape, puede exponer su aplicación a ataques de inyección de comandos. Buscar en todos los archivos: exec(, shell_exec(, system(, passthru(.Por ejm: su aplicación necesita realizar un whois en un nombre de dominio proporcionado por el usuario: exec('whois '.$request->input('domain')); o el mismo argumento con los siguientes comandos shell_exec(, system(, passthru(	Debe escapar del comando o escapar de los argumentos usando escapeshellcmd o escapeshellarg así: exec(escapeshellcmd('whois '.$request->input('domain'))); exec('whois '.escapeshellarg($request->input('domain')));
8	Host Injecti	alto	En este ataque, el atacante puede cambiar el host de una URL firmada utilizando el encabezado X-Forwarded-Host o Host. Esto permitiría a un atacante generar un	Asegúrese de configurar un bloque de servidor general (Nginx) o VirtualHost (Apache) para capturar todas las solicitudes con encabezados de Host no reconocidos,

	on		correo electrónico malicioso para restablecer la contraseña con un enlace a un sitio web controlado por el atacante. Los ataques de inyección de host pueden ocurrir a través del encabezado X-Forwarded-Host o del encabezado Host: Si su aplicación está detrás de un balanceador de carga o un proxy inverso y no configura hosts confiables, estará expuesta a ataques de inyección de host a través del encabezado X-Forwarded-Host. Si ya configuró el encabezado X-Forwarded-Host en su nivel de proxy inverso, puede ignorar este. Si la configuración de su servidor web no es segura, puede permitir ataques de inyección de host a través del encabezado del host. Esto puede suceder si utiliza comodines en el nombre del servidor o no tiene configuraciones generales de servidor para capturar todas las solicitudes con encabezados de host no reconocidos. Este vídeo (https://www.youtube.com/watch?v=KGTTlzZiihw) muestra una demostración en vivo de la vulnerabilidad. Tenga en cuenta que el envenenamiento por restablecimiento de contraseña es sólo uno de los posibles ataques. Otros ataques que surgen de la inyección de encabezados de host incluyen el envenenamiento de la caché web, la elusión de la autenticación, SSRF y la fuerza bruta del host virtual. Puedes aprender más sobre esto (https://portswigger.net/web-security/host-header/exploiting) Para confirmar que se trata de un problema, puede utilizar curl para activar las dos solicitudes siguientes a su aplicación: curl https://myapp.com -H "Host: evil.com" curl https://myapp.com -H "X-Forwarded-Host: evil.com" Si ve evil.com en cualquiera de los encabezados de respuesta (por ejemplo, encabezado de ubicación) o en cualquiera de las URL del cuerpo de la respuesta, entonces es vulnerable.	especifique nombres de servidores que no sean comodines y active la directiva UseCanonicalName (para Apache). En caso de que no tenga control sobre la configuración de su servidor web, puede agregar el middleware TrustHosts a su middleware global. Si utiliza CDN de proxy inverso como Cloudflare, ellos deberían encargarse automáticamente de esto por usted. Opción 1: agregar el middleware TrustHosts Simplemente agregue el middleware TrustHosts a su middleware global como se indica a continuación: protected $middleware = [\App\Http\Middleware\TrustHosts::class, \App\Http\Middleware\TrustProxies::class, \Fruitcake\Cors\HandleCors::class, \App\Http\Middleware\PreventRequestsDuringMaintenance::class, \Illuminate\Foundation\Http\Middleware\ValidatePostSize::class, \App\Http\Middleware\TrimStrings::class, \Illuminate\Foundation\Http\Middleware\ConvertEmptyStringsToNull::class,]; Opción 2: incluir en la lista blanca sus encabezados TrustedProxy De forma predeterminada, el middleware TrustProxies incluye en la lista blanca todos los encabezados, incluido X-Forwarded-Host. Puede configurar específicamente los encabezados de la lista blanca solo en X-Forwarded-For en su clase App\Http\Middleware\TrustProxies, de modo que incluso si el atacante configura el encabezado X-Forwarded-Host, su aplicación no confiará en él: protected $headers = Request::HEADER_X_FORWARDED_FOR;
9	Mime Sni	alt	En su servidor apache o nginx no está establecido el encabezado X-Content-Type-Options. Este encabezado les dice a los	Puede configurar el encabezado X-Content-Type-Options en su servidor web. Para Nginx, puede usar la directiva add_header

	ffing	o	navegadores que no ignoren los tipos de contenido definidos explícitamente y ayuda a prevenir ataques de detección de MIME. En un ataque de rastreo MIME, un atacante disfraza un archivo HTML como un tipo de archivo diferente y carga el archivo en el servidor web. En consecuencia, el navegador lo representará como un archivo HTML y, por lo tanto, brindará al atacante la posibilidad de ejecutar XSS. Esta condición se omite para aplicaciones sin estado (por ejemplo, solo API).	en su bloque server o location ej: add_header X-Content-Type-Options "nosniff"; Para Apache puede usar la directiva Header en su contenedor <VirtualHost>, <Directory>, o <Location> ej: Header set X-Content-Type-Options "nosniff"

10	Object Injection	alto	Buscar en todos los archivo cualquier llamada a la función unserialize($request en datos de entrada de usuarios que no sean de confianza porque puede exponer su aplicación a ataques de inyección de objetos. Dado que PHP permite la serialización de objetos, los atacantes podrían pasar cadenas serializadas ad-hoc a una llamada unserialize() vulnerable, lo que resultaría en una inyección arbitraria de objetos PHP en el alcance de la aplicación.
			Ej: Tiene una clase llamada SomeClass con un método destructor
			class SomeClass
			{
			public $cache_file;
			
			function __destruct()
			{
			$file = "/var/www/cache/tmp/{$this->cache_file}";
			if (file_exists($file)) @unlink($file);
			}
			}
			y tiene un controlador que deserializa los datos de solicitud del usuario: $user_data = unserialize($request->input('data'));
			
			El atacante ahora podrá eliminar un archivo arbitrario configurando la cadena de consulta de datos en O:8:"SomeClass":1:{s:10:"cache_file";s:15:"../../index.php";}. Cuando esta cadena no se serializa, se crea el objeto SomeClass que elimina el archivo de caché en __destruct. Esto significa que el atacante puede proporcionar la ruta a cualquier archivo en una cadena serializada personalizada para eliminar el archivo.
			
			Esto es sólo un ejemplo. Esta vulnerabilidad puede conducir a una amplia variedad de ataques, incluida la inyección
				Elimine o retire todas las llamadas a la función unserialize en los datos de solicitud del usuario (user request data)

			de código, la inyección de SQL, el cruce de rutas y la denegación de servicio de la aplicación, según el contexto	

11	Column Name SQL Injection	crítico	Buscar en todos los archivos del controlador, que en las clausulas ::where, ::orderBy, ->having , etc se esté dictando dinámicamente el nombre de la columna a traves de $request sin especificar los campos permitidos en validate y en caso se dicte el campo estaticamente, que dentro del ->get() se dicte todas las columnas del $request.
			
			PDO no admite el enlace de nombres de columna, por lo que si se usa la entrada del usuario para dictar nombres de columna.
			
			Considere un ejemplo de un cuadro de búsqueda, donde le da una opción al usuario para buscar libros por un conjunto de campos:
			
			use App\Models\Book;
			
			public function searchBooks(Request $request) {
			// Request input dictates column name in where clause.
			return Book::where($request->input('search_field'), $request->input('query'))
			->get();
			}
			El código anterior es vulnerable a los ataques de inyección SQL porque la entrada del usuario dicta el nombre de la columna para la consulta de búsqueda.
			
			Algunos otros ejemplos de código vulnerable incluyen:
			
			use App\Models\Book;
			
			// Request input dictates column name in order by clause.
			$books = Book::orderBy($request->input('sort_column'))->get();
			
			// Request input dictates column names in select clause.
			$books = Book::where('published', true)->get($request->all());
			
			// Request input dictates column names in having clause.
				En lugar de que el usuario dicte directamente el nombre de la columna, debe validar que el nombre de la columna esté presente en un conjunto de columnas incluidas en la lista blanca, es decir, dentro de validate:
				use App\Models\Book;
				
				public function searchBooks(Request $request) {
				$validated = $request->validate([
				'search_field' => 'in:author,title,isbn',
				]);
				
				return Book::where($validated['search_field'], $request->input('query'))
				->get();
				}

			$books = Book::where('author', $request->input('author')) ->having($request->input('someColumn'))->get();	

12	Directory Traversal	crítico	Un ataque transversal de directorio tiene como objetivo acceder a archivos y directorios que se almacenan fuera de la carpeta raíz web mediante la manipulación de variables que hacen referencia a archivos con secuencias de "punto-punto-barra (../)" y sus variaciones o mediante el uso de rutas de archivo absolutas.
			
			use Illuminate\Http\Request;
			
			public function downloadDocument(Request $request) {
			return response()->download(storage_path('/').$request->input('filename'));
			}
			
			El código anterior es vulnerable a los ataques de cruce de directorios. Si el usuario proporciona un nombre de archivo como ../../../../etc/passwd, entonces el usuario puede obtener acceso a su archivo /etc/passwd dependiendo de la ruta raíz de su aplicación. Otros ejemplos de código vulnerable incluyen:
			
			use Illuminate\Filesystem\Filesystem;
			use Illuminate\Support\Facades\Storage;
			
			response()->file($request->input('path').'.xml');
			file_get_contents($request->post('path'));
			(new Filesystem())->get($request->input('path'));
			(new Filesystem())->copy($request->input('path'), 'sometarget');
			Storage::download($request->input('path'));
			Storage::get($request->input('path'));
				Opción 1: usar la funcion basename
				La mejor manera de solucionar esto es usar la función basename, si la ruta del directorio está predeterminada (no es variable):
				
				response()->download(storage_path('somedirectory/').basename($request->input('filename')));
				
				Opción 2: Validación de ruta usando Realpath
				Si la ruta del directorio puede tener subdirectorios, puede usar la función realpath en su lugar y validar que pertenece al directorio deseado:
				
				use Illuminate\Support\Str;
				use Illuminate\Http\Request;
				
				public function downloadDocument(Request $request) {
				$path = realpath(storage_path('somedir/'.$request->input('path')));
				if (! Str::startsWith($path, storage_path())) {
				abort(403);
				}
				return response()->download($path);
				}
				
				ADVERTENCIA
				realpath convierte el directorio en una ruta real sin ningún tipo de archivo ../. El solo uso de realpath no protegerá su aplicación. Deberá validar el resultado de la función realpath como se indicó anteriormente.
				
				Ajustes de configuración segura de PHP
				Para una mayor seguridad, puede establecer los ajustes de configuración de PHP doc_root y open_basedir en el directorio raíz de su aplicación. Esto limita los archivos a los que puede acceder PHP al directorio especificado. Asegúrese de consultar la documentación de PHP sobre estos ajustes de configuración (https://www.php.net/manual/es/ini.core.php).

				ADVERTENCIA
				El uso de open_basedir deshabilitará la caché de realpath. Esto puede afectar negativamente el rendimiento de su aplicación.
				
				Otro ejm:
				 Simply visit /download?filename=../.env in the address bar above.
				
				 Then, try the same with the safe route: /safeDownload?filename=../.env
				
				<?php
				use Illuminate\Http\Request;
				use Illuminate\Support\Facades\Route;
				
				Route::get('/download', function(Request $request) {
				  return response()->download(base_path('resources/').$request->input('filename'));
				});
				
				Route::get('/safeDownload', function(Request $request) {
				  return response()->download(base_path('resources/').basename($request->input('filename')));
				});
				
				Route::view('/', 'demo');

13	Open Redirection	crítico	Si su aplicación acepta una entrada controlada por el usuario que especifica un enlace a un sitio externo y usa ese enlace en una redirección, está expuesta a una vulnerabilidad de redirección abierta. Al modificar el valor de la URL de un sitio malicioso, un atacante puede lanzar con éxito una estafa de phishing y robar las credenciales del usuario. Debido a que el nombre del servidor en el enlace modificado es idéntico al del sitio original, los intentos de phishing tienen una apariencia más confiable. Este es un ejemplo clásico de una vulnerabilidad de redirección abierta.:	Elimine cualquier redireccionamiento a enlaces externos proporcionados por el usuario

			return redirect($request->input('link')); Entonces, si su sitio web es example.com, un atacante puede crear un enlace como www.example.com/redirect?link=evil.com/confirm-passwordel que redirige a un sitio web externo. Esto facilita las estafas de phishing porque la víctima pensaría que el enlace en realidad pertenece a example.com.
			
			Otros ejemplos de código vulnerable son los siguientes:
			use Illuminate\Routing\Redirector;
			use Illuminate\Support\Facades\Redirect;
			
			redirect()->to($request->input('path'));
			(new Redirector(url()))->away('/somewhere/'.$request->query('path'));
			return Redirect::to(request()->post('path'));	

14	Raw SQL Injection	crítico	Buscar en todos los archivos, whereRaw('author = '.$request->input('author')); , ::fromRaw($request->get('query'))->get(); , ::insert, ::update concatenado directamente con parámetros de entrada y ::unprepared	Use enlaces o vinculación de parámetros de consulta para los datos de entrada del usuario: use App\Models\Book; Book::whereRaw('author = ?', [$request->input('author')]); o también puede usar enlaces de parámetros con nombre: use App\Models\Book; Book::whereRaw('author = :author', ['author' => $request->input('author')]);
15	Regex DOS	crítico	En los ataques de DOS de expresiones regulares (ReDOS), los atacantes construyen patrones de expresiones regulares "malvados" que pueden hacer que se cuelguen o funcionen muy lentamente. Tenga en cuenta las siguientes situaciones: 1.El patrón regex lo construye el usuario: Esto es muy peligroso y fácil de explotar. Debe busca cualquier código que obtenga patrones de expresiones regulares a partir de los datos de entrada del usuario. 2.El patrón de expresiones regulares es vulnerable en el código: tal vez, el código en sí contiene algunos patrones de expresiones regulares que son "malvados". Considere el siguiente código: preg_match($request->input('pattern'), $request->input('query')); (El analizador	Elimine todos los patrones de expresiones regulares que se obtienen de los datos de entrada del usuario

			solo detecta este tipo de expresiones regulares donde el patrón de expresión regular se obtiene de los datos de entrada del usuario.) Dado que tanto el patrón de expresión regular como la cadena de asunto se obtienen de la entrada del usuario, es extremadamente peligroso y fácil de explotar. Ahora, consideremos este código: preg_match('/(.*a){x}/', $request->input('query')); (El analizador no detecta este tipo expresiones regulares, tener más cuidado con estos) El código anterior también es vulnerable a los ataques de DOS de expresiones regulares. Dada la cadena de entrada aaa...a(con x o más del carácter a), la complejidad de tiempo de la coincidencia de expresiones regulares es O(2x).	

16	Unrestricted File Upload	crítico	Si su aplicación permite que los datos controlados por el usuario construyan la ruta de carga de un archivo, esto puede resultar en la sobrescritura de un archivo crítico o el almacenamiento del archivo en una ubicación incorrecta. Por ejm:
			
			use Illuminate\Http\Request;
			use Illuminate\Support\Facades\Storage;
			
			public function upload(Request $request) {
			Storage::put($request->input('path'), $request->file('uploadFile'));
			}
			
			El código anterior es vulnerable a sobrescribir cualquier archivo porque se utilizan datos de entrada del usuario que no son de confianza para determinar la ruta del archivo. Por lo tanto, si el usuario establece la ruta en algo como ../.env, el usuario puede sobrescribir su archivo .env según los permisos del archivo y el controlador de almacenamiento predeterminado.
			
			Otros ejemplos de código vulnerable incluyen:
			$request->file('avatar')->storeAs($request->input('path'), auth()->user()->id);
			Storage::putFile($request->input('path'), $request->file('somefile'));
				Opción 1: usar la función basename
				La mejor manera de solucionar esto es usar la función basename, si la ruta del directorio está predeterminada (no es variable):
				Storage::put('somedir/'.basename($request->input('path')), $request->file('uploadFile'));
				
				Opción 2: Validación de ruta usando Realpath
				Si la ruta del directorio puede tener subdirectorios, puede usar la función realpath en su lugar y validar que pertenece al directorio deseado:
				
				use Illuminate\Support\Str;
				use Illuminate\Http\Request;
				
				public function uploadDocument(Request $request)
				{
				$path = realpath(storage_path('somedir/'.$request->input('path')));
				if (! Str::startsWith($path, storage_path())) {
				abort(403);
				}
				$request->file('avatar')->storeAs($path, auth()->user()->id);
				}
				ADVERTENCIA

				realpath convierte el directorio en una ruta real sin ningún tipo de archivo ../. El solo uso de realpath no protegerá su aplicación. Deberá validar el resultado de la función realpath como se indicó anteriormente. Ajustes de configuración segura de PHP Para una mayor seguridad, puede establecer los ajustes de configuración de PHP doc_root y open_basedir en el directorio raíz de su aplicación. Esto limita los archivos a los que puede acceder PHP al directorio especificado. Asegúrese de consultar la documentación de PHP sobre estos ajustes de configuración (https://www.php.net/manual/es/ini.core.php). ADVERTENCIA El uso de open_basedir deshabilitará el caché de ruta real. Esto puede afectar negativamente el rendimiento de su aplicación.
17	Validation SQL Injection	crítico	Buscar en todos los archivos ->ignore, es decir, Rule::unique('users')->ignore($request->input('user_id')); ó Rule::unique('users')->ignore(5, $request->get('userid_col'));	No pase ningún dato de entrada del usuario al método ignore Rule::unique('users')->ignore(auth()->id());
18	Debug Statement	alto	Si su aplicación contiene declaraciones de depuración como 'var_dump', 'dump', 'dd', 'print_r', 'var_export', 'debug_print_backtrace', 'debug_backtrace', 'debug_zval_dump', es posible que genere el resultado de la respuesta y exponga su aplicación a numerosos riesgos de seguridad, incluido el volcado de variables de entorno confidenciales o secretos y la exposición de variables de PHP que pueden resultar en ataques de inyección de código	Elimine las declaraciones de depuración en su aplicación como 'var_dump', 'dump', 'dd', 'print_r', 'var_export', 'debug_print_backtrace', 'debug_backtrace', 'debug_zval_dump'
19	Eval Function	crítico	Buscar en todos los archivos, eval(.Se podría argumentar que eval está bien si no se proporciona la entrada del usuario. Pero eso es una falacia. eval es pura maldad (incluso si no se proporcionan datos de entrada del usuario) y he aquí por qué: 1.Es un gran riesgo de seguridad cuando se proporciona información del usuario. Los ataques de inyección de código son el tipo de ataque más desagradable. 2.Agrega otro vector de ataque. Entonces,	Eliminar todas las llamadas a funciones y sentencias eval()

			por ejemplo, digamos que eval no se proporciona la entrada del usuario y simplemente ejecuta el código en una tabla de base de datos que es inaccesible para los usuarios. Si su aplicación tiene una vulnerabilidad de inyección SQL, puede escalar rápidamente a inyección de código. Por lo tanto, es un riesgo de seguridad incluso si no se proporciona la entrada del usuario. 3.Es difícil de depurar (porque el código está literalmente en una cadena). 4.El uso eval puede evitar que Opcache y/o un compilador JIT optimicen el código evaluado y el código que lo rodea. 5.El código eval es ilegible y su IDE no reconocerá ni formateará el código. En pocas palabras, eval es malo para la seguridad, el rendimiento, la legibilidad del código y la depuración.	
20	SQL Injection	crítico	Buscar en todos los archivos cualquier código de bd php nativo como: 1. Interacción directa con el objeto PDO. 2. funciones de bd php nativas como ('mysqli_connect', 'mysqli_execute', 'mysqli_stmt_execute', 'mysqli_stmt_close', 'mysqli_stmt_fetch', 'mysqli_stmt_get_result', 'mysqli_stmt_more_results', 'mysqli_stmt_next_result', 'mysqli_stmt_prepare', 'mysqli_close', 'mysqli_commit', 'mysqli_begin_transaction', 'mysqli_init', 'mysqli_insert_id', 'mysqli_prepare', 'mysqli_query', 'mysqli_real_connect', 'mysqli_real_query', 'mysqli_store_result', 'mysqli_use_result', 'mysqli_multi_query', 'pg_connect', 'pg_close', 'pg_affected_rows', 'pg_delete', 'pg_execute', 'pg_fetch_all', 'pg_fetch_result', 'pg_fetch_row', 'pg_fetch_all_columns', 'pg_fetch_array', 'pg_fetch_assoc', 'pg_fetch_object', 'pg_flush', 'pg_insert', 'pg_get_result', 'pg_pconnect', 'pg_prepare', 'pg_query', 'pg_query_params', 'pg_select', 'pg_send_execute', 'pg_send_prepare', 'pg_send_query', 'pg_send_query_params', 'pg_affected_rows',) 3. Uso del método DB::unprepared de facade	Reemplazar las funciones sql inseguras y el método de facades unprepared por las funciones de protección de inyección sql disponibles con los modelos de Eloquent o los generadores de consultas.

21	CSRF	alto	Además de verificar el uso de token en formularios post, put, patch o delete; si la aplicación utiliza ajax a través de una biblioteca como jQuery, por ejemplo, debe estar presente el encabezado X-CSRF-TOKEN para cada solicitud. Esta condición puede excluirse en algunos casos, como, por ejemplo, si está utilizando Stripe para procesar pagos y está utilizando su sistema de webhook, deberá excluir su ruta de controlador de webhook de Stripe de la protección CSRF, ya que Stripe no sabrá qué token CSRF enviar a sus rutas. Por lo general, debe colocar este tipo de rutas fuera del grupo web de middleware App\Providers\RouteServiceProvider que se aplica a todas las rutas del archivo routes/web.php. Sin embargo, también puede excluir las rutas agregando sus URI a la propiedad $except del middleware VerifyCsrfToken:
			<?php
			namespace App\Http\Middleware;
			use Illuminate\Foundation\Http\Middleware\VerifyCsrfToken as Middleware;
			
			class VerifyCsrfToken extends Middleware
			{
			protected $except = [
			'stripe/*',
			'http://example.com/foo/bar',
			'http://example.com/foo/*',
			];
			}
			
			También se omite para aplicaciones sin estado o aplicaciones que no usan cookies.
				Si la aplicación utiliza ajax a través de una biblioteca como jQuery, por ejemplo, primero almacene el token en una etiqueta HTML meta:
				<meta name="csrf-token" content="{{ csrf_token() }}">
				
				y luego puede indicarle que agregue automáticamente el token a todos los encabezados de solicitud:
				$.ajaxSetup({
				headers: {
				'X-CSRF-TOKEN': $('meta[name="csrf-token"]').attr('content')
				}
				});
				ó como parámetro en el envío post :
				$.post("{{route('tu_ruta')}}", { _token: $('meta[name="csrf-token"]').attr('content'), param2: value2 })
				.done(function(data) { ... },"json");

22	PHP Ini	alto	allow_url_fopen: Debería estar deshabilitado. Deshabilitarlo minimiza el riesgo de escalar LFI (https://www.acunetix.com/blog/articles/local-file-inclusion-lfi/) a RFI (https://www.acunetix.com/blog/articles/remote-file-inclusion-rfi/) y reduce el riesgo de ejecución remota de código, divulgación de información y secuencias de comandos entre sitios (XSS). allow_url_include: Debería estar deshabilitado. Esto tiene un efecto similar a allow_url_fopen. expose_php: Debería estar deshabilitado. Si esta configuración está activada, un atacante puede ver la versión de PHP ejecutándose en el servidor de aplicaciones.	[allow_url_fopen: Off or 0], [allow_url_include: Off or 0], [expose_php: Off or 0], [display_errors: Off or 0], [display_startup_errors: Off or 0], [log_errors: On or 1] and [ignore_repeated_errors: Off or 0].

			display_errors: Debe deshabilitarse para evitar exponer mensajes de error detallados de la aplicación que pueden incluir información confidencial. display_startup_errors: Debe deshabilitarse para evitar exponer errores que ocurren durante la secuencia de inicio de PHP. log_errors: Debe estar habilitado para registrar mensajes de error en el archivo de registro de errores del servidor. ignore_repeated_errors: Debería estar deshabilitado.	
23	File Permissions	crítico	Los archivos o directorios inseguros incluyen: [], [\app], [\resources], [\storage], [\public], [\config], [\database], [\routes], [\bootstrap], [\bootstrap\cache], [\bootstrap\app.php] y [\public\index.php].	Para confirmar si su aplicación establece permisos de archivos y directorios seguros. En general, todos los directorios de Laravel deben configurarse con un nivel de permiso máximo de 775 y los archivos no ejecutables con un nivel de permiso máximo de 664. Los archivos ejecutables como Artisan o scripts de implementación deben proporcionarse con un nivel de permiso máximo de 775. Si sus permisos son más laxos que los anteriores, esto expondría su aplicación a verse comprometida si otra cuenta en el mismo servidor se ve comprometida. chmod -R 775 folder_name. Establece permisos para que el (U)ser/propietario pueda leer, escribir y ejecutar. (G)roup puede leer, escribir y ejecutar. (O)otros pueden leer, no escribir y ejecutar. chmod -R 664 folder_name. Establece permisos para que el (U)ser/propietario pueda leer, escribir y no ejecutar. (G)roup puede leer, escribir y no ejecutar. (O)otros pueden leer, no pueden escribir y no pueden ejecutar. 'allowed_permissions' => [base_path() => '775', app_path() => '775', resource_path() => '775', storage_path() => '775', public_path() => '775',

				config_path() => '775', database_path() => '775', base_path('routes') => '775', app()->bootstrapPath() => '775', app()->bootstrapPath('cache') => '775', app()->bootstrapPath('app.php') => '664', base_path('artisan') => '775', public_path('index.php') => '664', public_path('server.php') => '664',],
24	XSS Cross-Site Scripting	crítico	Verifique que su aplicación establece un encabezado Content-Security-Policy apropiado para protegerse contra ataques XSS. Asegúrese de leer la documentación de la política de seguridad de contenido para comprender qué directivas y fuentes son válidas para su aplicación: https://web.dev/articles/csp?hl=es-419 y https://cheatsheetseries.owasp.org/cheatsheets/Content_Security_Policy_Cheat_Sheet.html Esta verificación se omite si su aplicación no tiene estado (no utiliza el StartSessionmiddleware).	Puede agregar el encabezado Content-Security-Policy en la configuración de su servidor web. Tenga en cuenta que esto es solo un ejemplo básico de poderla agregar. Para Nginx, puede usar la directiva add_header en su bloque server o location: add_header Content-Security-Policy "default-src 'self'; script-src 'self';"; Para Apache, puede usar la directiva Header en su contenedor <VirtualHost> <Directory>o <Location> Header always set Content-Security-Policy "default-src 'self'; script-src 'self';"
25	Cookie HttpOnly	alto	Una cookie con un atributo HttpOnly es inaccesible desde Javascript. La opción de configuración http_only de su archivo config/session.php determina si su cookie de sesión debe establecer el atributo HttpOnly. Si no está habilitada, su aplicación podría verse expuesta a ataques de secuencias de comandos entre sitios (XSS). A menos que tenga un caso de uso muy específico que requiera el acceso a las cookies de sesión desde Javascript, se recomienda habilitar esta opción. Tenga en cuenta que el valor predeterminado HttpOnly para todas las cookies es true y la opción de configuración http_only en config/session.php solo establece este atributo para la cookie de sesión y no para las otras cookies en su aplicación. Esta verificación se omite si su aplicación no tiene estado (no utiliza sesiones)	Establezca el atributo http_only en su archivo config/session.php como verdadero. 'http_only' => true,

26	Cookie Domain	alto	El atributo domain"cookie" especifica qué hosts pueden recibir la cookie. Si no se especifica, se usa el mismo origen que la configuró, excluyendo los subdominios. Este analizador confirma que su atributo domain de cookie de sesión (este también es el valor predeterminado para todas las cookies) está configurado como nulo si no tiene registros de rutas de subdominio. Si se especifica un atributo de dominio, siempre se incluyen los subdominios, lo que potencialmente hace que su aplicación sea menos segura. Esta verificación se omite si hay registros de ruta relacionados con más de un dominio o subdominio único en su aplicación.	Establezca su opción de configuración domain en su archivo config/session.php en nulo. 'domain' => null,
27	Encrypted Cookies	alto	En general, es recomendable cifrar todas las cookies. Como mínimo, reduce la vulnerabilidad de ataque de la aplicación. Sin embargo, resulta absolutamente fundamental utilizar el cifrado de cookies para los siguientes casos de uso: 1.Si utiliza el almacén de sesiones cookie, es fundamental cifrar sus cookies; de lo contrario, los datos de sesión específicos del usuario no solo serían legibles, sino que también podrían ser manipulados por el robo de cookies o ataques de intermediario. Además, algunas jurisdicciones tienen normativas para cifrar los datos personales confidenciales del usuario. 2.Si almacena algún tipo de datos en cookies que no deben ser legibles por los clientes (por ejemplo, secretos o credenciales), debe cifrar sus cookies. 3.Si almacena datos en cookies que no deben manipularse, debe cifrar sus cookies; de lo contrario, los clientes podrán cambiar fácilmente los datos. Incluso si no almacena datos personales confidenciales o del usuario en cookies, se recomienda utilizar el cifrado de cookies solo para reducir la superficie de ataque y para su tranquilidad. Esta verificación se omite en aplicaciones sin estado o que no utilizan cookies.	De forma predeterminada, Laravel añade el middleware EncryptCookies al grupo de middleware web de tu clase Kernel. Si deseas cifrar las cookies para tus rutas web, puedes añadir este middleware al grupo de middleware web de tu clase App\Http\Kernel

28	Hard Coded Credentials	crítico	Tener credenciales codificadas en el código fuente es una mala práctica por las siguientes razones: 1. Permite a todos los desarrolladores del proyecto ver la contraseña. 2. Si la cuenta protegida por la contraseña se ve comprometida, usted se verá obligado a elegir entre seguridad y disponibilidad. 3. Una vez que el código está en producción, la contraseña no se puede cambiar sin parchear el software.	Elimine todas las referencias a credenciales codificadas y guárdelas en la base de datos o en su archivo .env, al que se puede hacer referencia mediante un archivo de configuración.
29	Web Server Fingerprinting	bajo	curl -I -L https://myapp.com	Puede desactivar la versión del servidor usando las siguientes directivas. Para Nginx: serverTokens off; Para Apache: ServerTokens Prod

Evaluación de seguridad utilizando herramientas automatizadas SAST, DAST y comprobación manual

- Aplicación objetivo de evaluación: GVOTE
- Alcance de la evaluación: Vulnerabilidades críticas, altas, medias y bajas
- Características de la aplicación tipo: Página única
- Lenguaje de programación y versión: PHP 8.1
- Servidor web y versión: Apache 2.4
- Sistema operativo del servidor y versión: CentOS Linux 8.3.2011
- Servidor de administración de base de datos y versión: SQL Server 2019
- Herramientas de análisis seleccionadas SAST: Enlightn DAST: OWASP ZAP
- Las métricas a utilizar son:
  - Enumeración y clasificación de vulnerabilidades usando la enumeración de debilidades comunes CWE
  - Verdadero positivo: es la detección en cualquier tipo de análisis de que es una vulnerabilidad de seguridad real
  - Falso positivo: es la detección en cualquier tipo de análisis de que no es una vulnerabilidad de seguridad real

PRE TEST

Capturas escaneo SAST

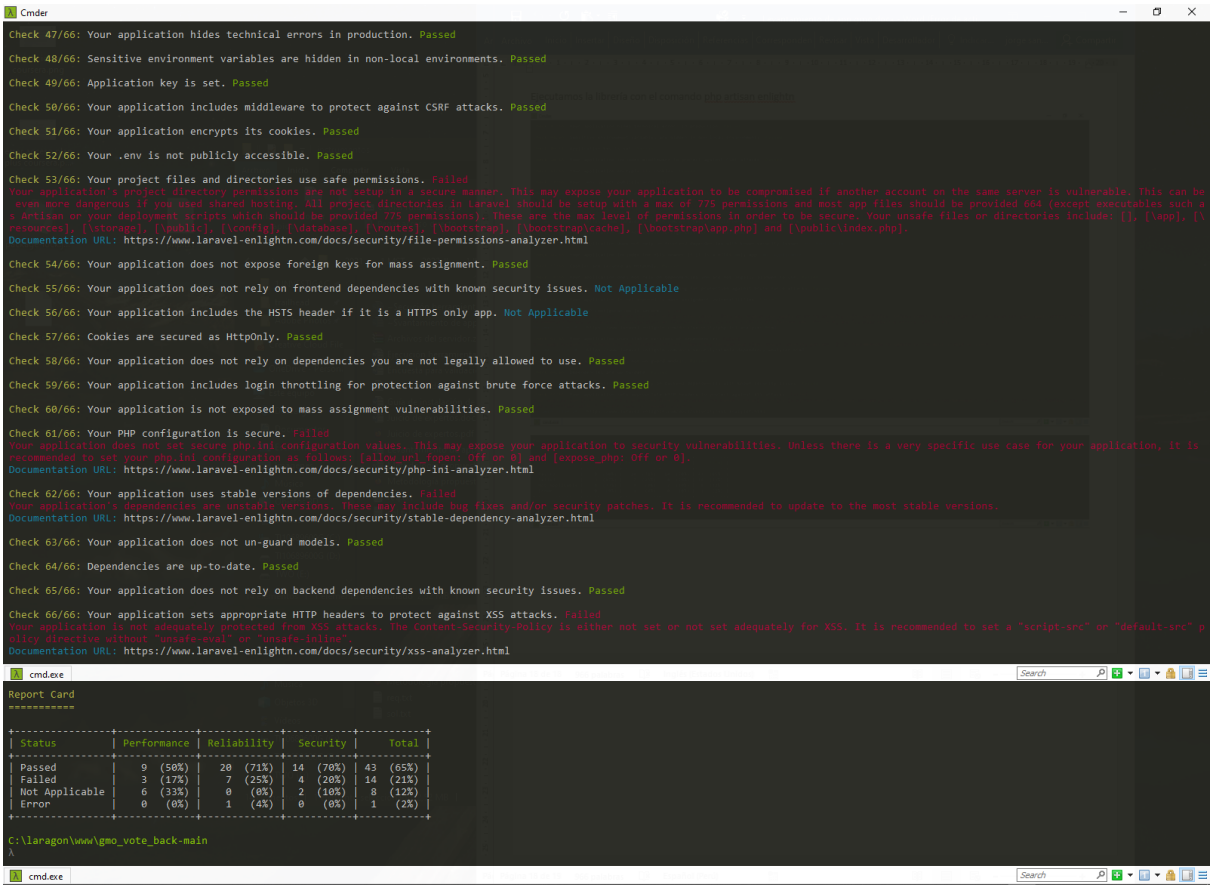


Figura 24. Reporte de escaneo SAST

Verificación 53. Revisión del uso de permisos seguros en archivos y directorios en el servidor de producción

Captura de comandos backend

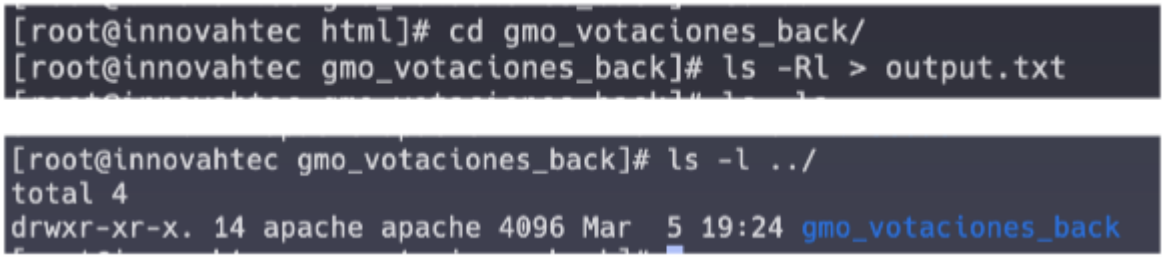


Figura 25. Extracción del listado de todos los archivos y subdirectorios de manera recursiva con detalles de permisos, propietario, grupo, tamaño y fecha, del directorio del proyecto

base_path() => '775', como se observa el directorio del Proyecto gmo_votaciones_back, tiene permisos más restrictivos del recomendado, es decir, se encuentra establecido en ‘755’.

```
[root@innovahtec gmo_votaciones_back]# ls -l ../
total 4
drwxr-xr-x. 14 apache apache 4096 Mar  5 19:24 gmo_votaciones_back
```

Figura 26. Permisos del directorio principal del proyecto

app_path() => '775', resource_path() => '775', storage_path() => '775', public_path() => '775', config_path() => '775', database_path() => '775', base_path('routes') => '775', app()->bootstrapPath() => '775'. Como se observa los directorios tienen permisos más restrictivos de los recomendados, es decir, se encuentran establecidos en '755'.

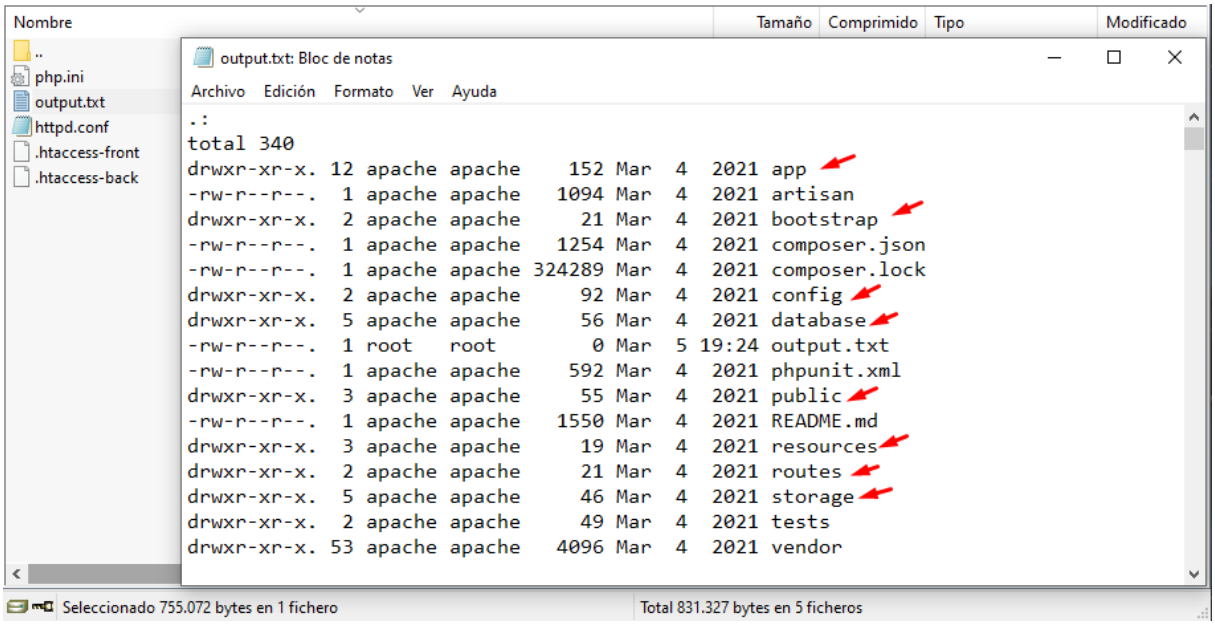


Figura 27. Permisos de los subdirectorios del proyecto principal

app()->bootstrapPath('cache') => '775', no se encontró directorio cache

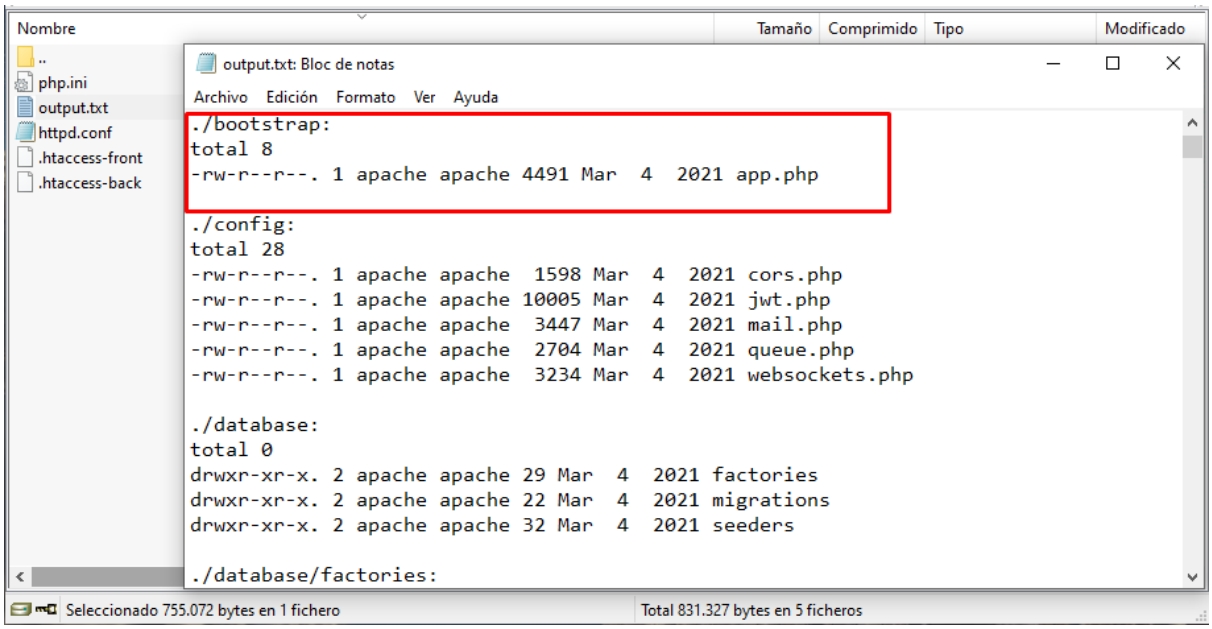


Figura 28. Revisión de permisos del directorio cache

app()->bootstrapPath('app.php') => '664', como se observa el archivo app.php tiene un permiso más restrictivo del recomendado.

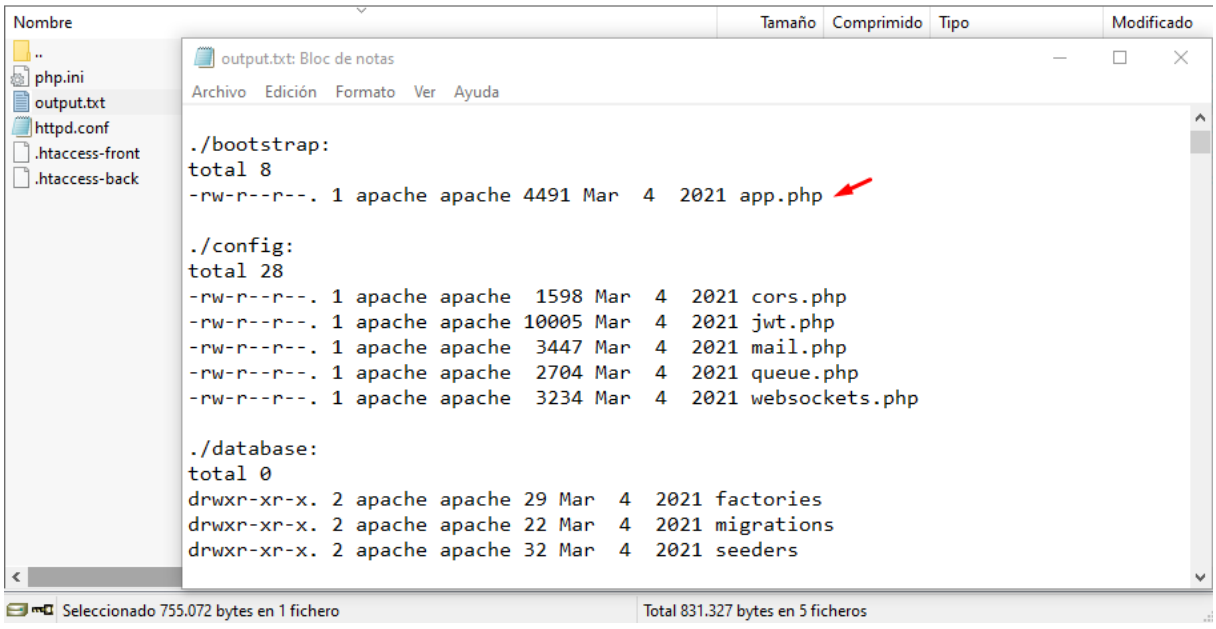


Figura 29. Revisión de permisos del archivo app.php

base_path('artisan') => '775', como se observa el archivo artisan tiene un permiso más restrictivo del recomendado.

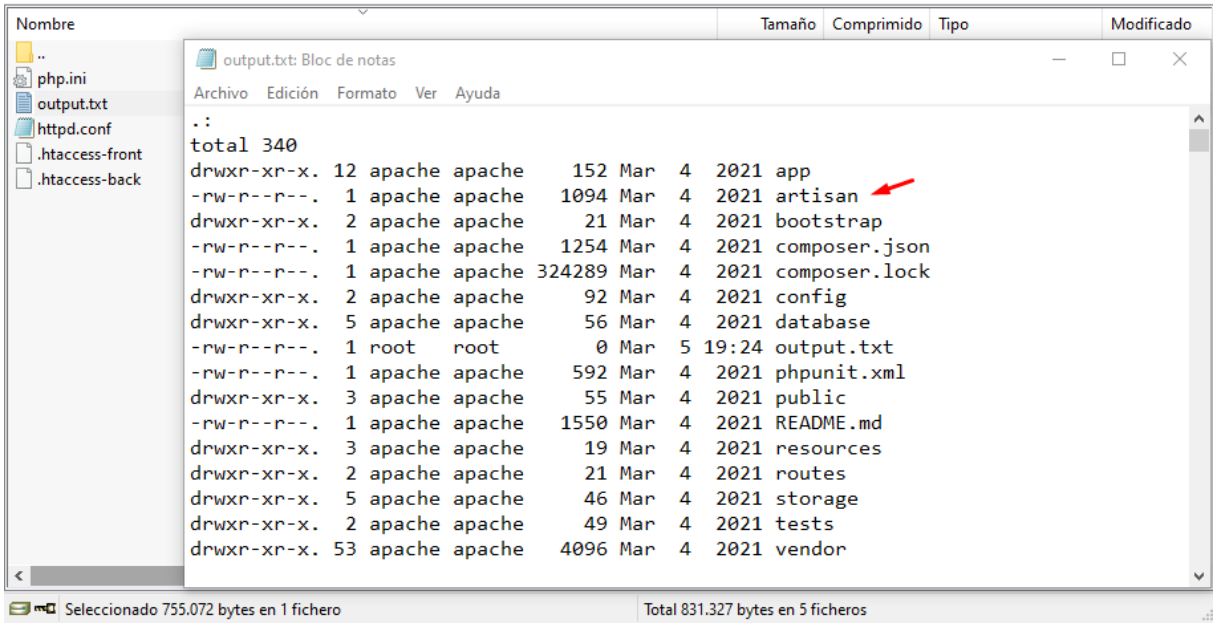


Figura 30. Revisión de permisos del archivo artisan

public_path('index.php') => '664', como se observa el archivo index.php tiene un permiso más restrictivo del recomendado.

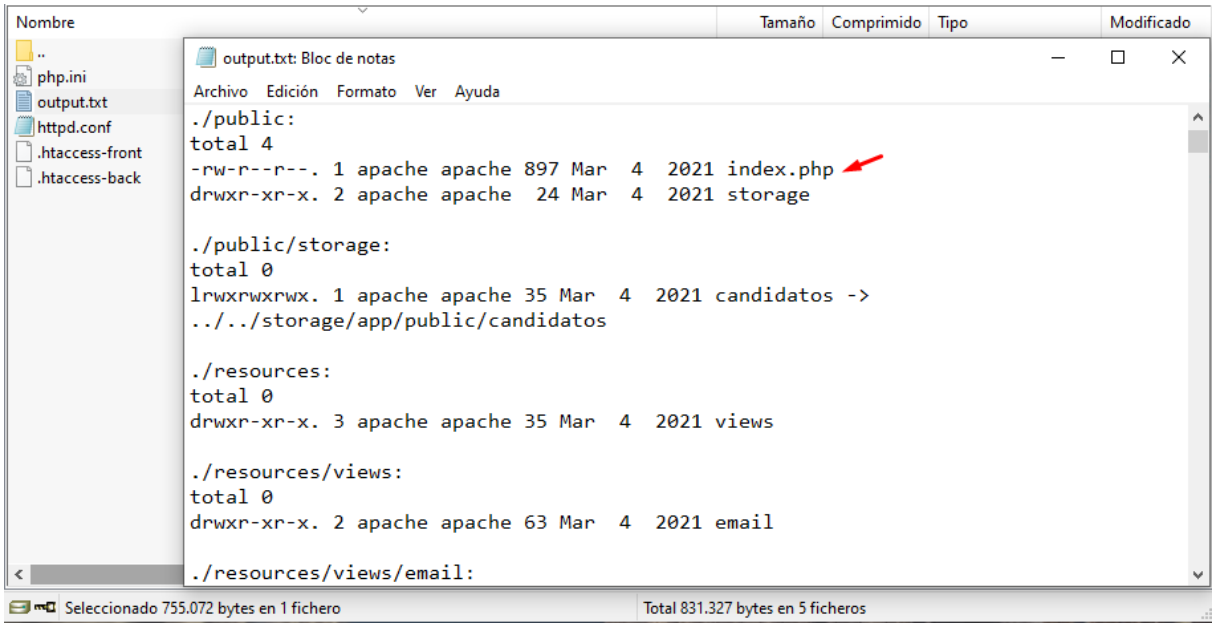


Figura 31. Revisión de permisos del archivo index.php

public_path('server.php') => '664', no se encontró archivo server.php

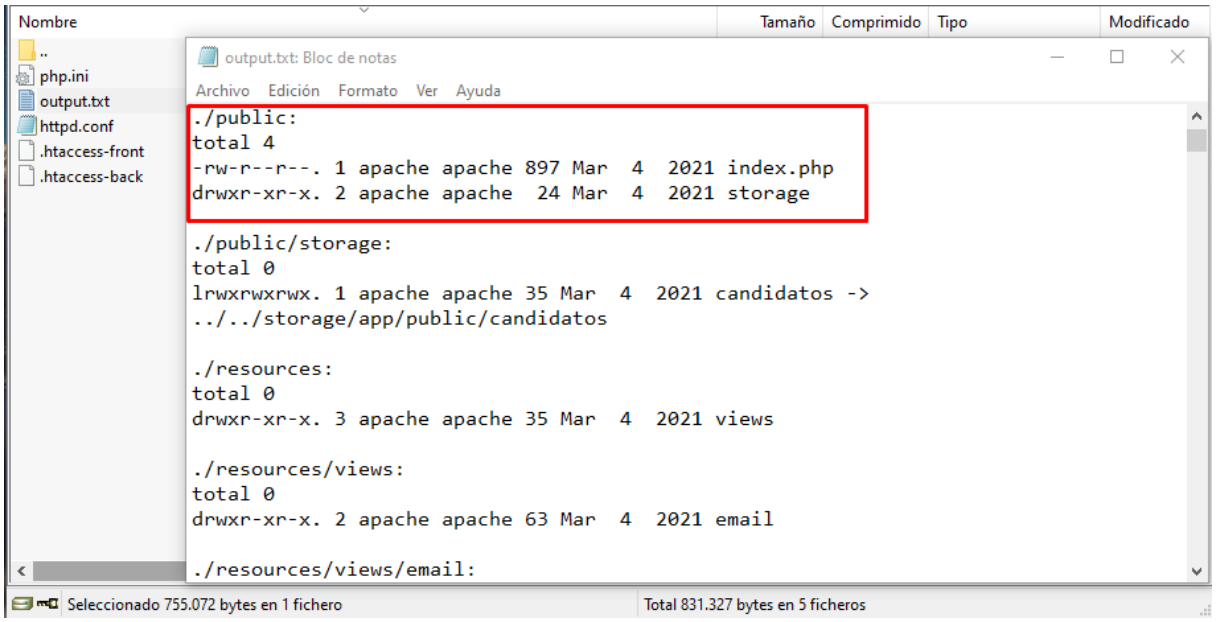


Figura 32. Revisión de permisos del archivo server.php

Verificación 61. Revisión de la configuración de php

Se observa que el parámetro allow_url_fopen está activo, lo cual no es lo recomendado.

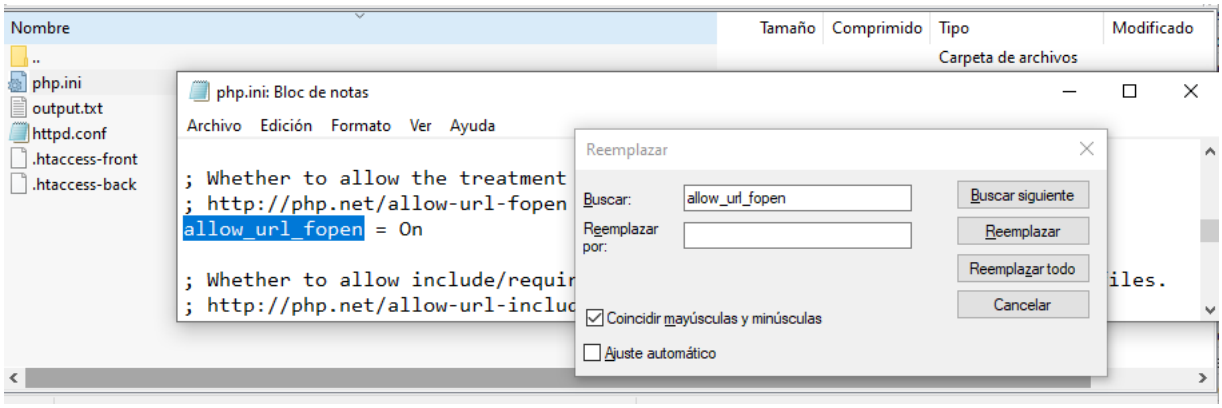


Figura 33. Revisión del parámetro allow_url_fopen

Se observa que el parámetro está desactivado, lo cual es lo recomendado.

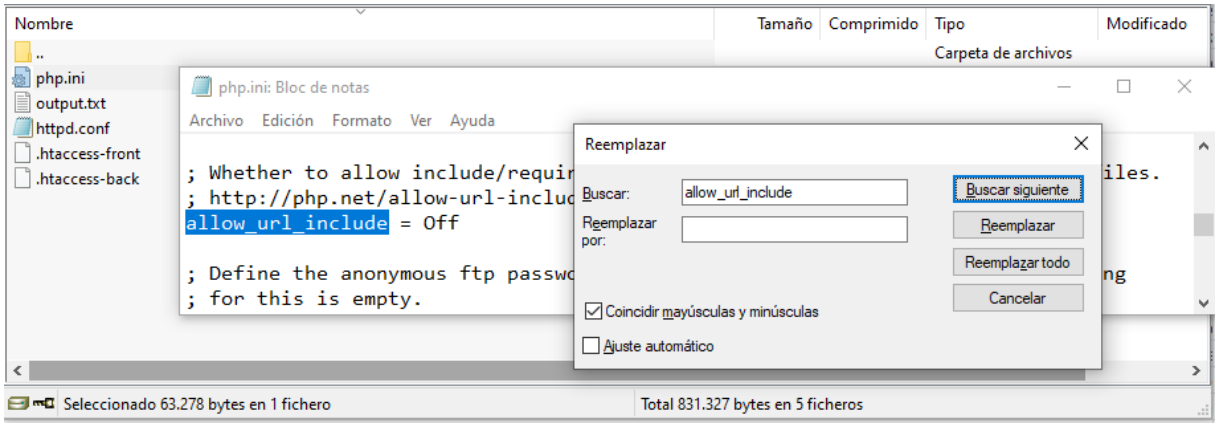


Figura 34. Revisión del parámetro allow_url_include

Se observa el parámetro expose_php activo, lo cual no es lo recomendado.

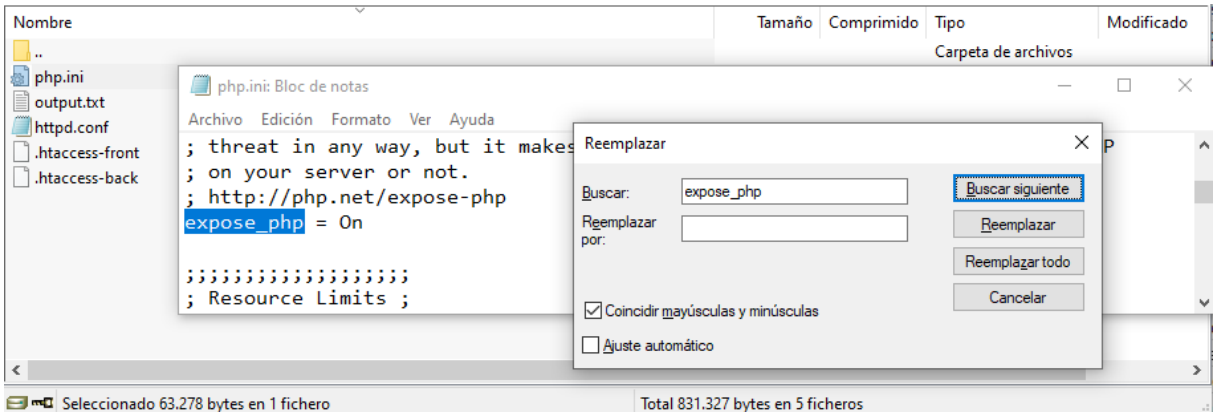


Figura 35. Revisión del parámetro expose_php

Se observa que el parámetro está desactivado, lo cual es lo recomendado.

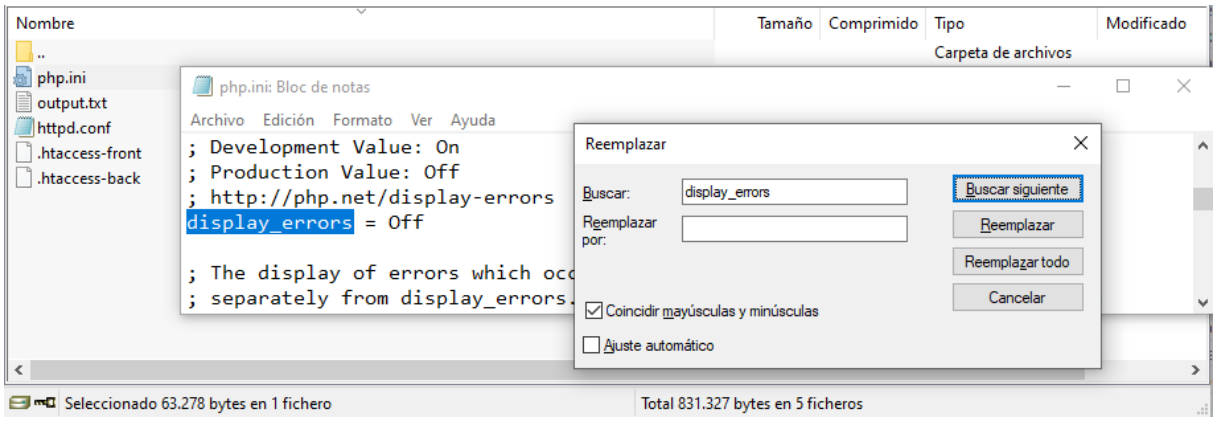


Figura 36. Revisión del parámetro display_errors

Se observa que el parámetro está desactivado, lo cual es lo recomendado.

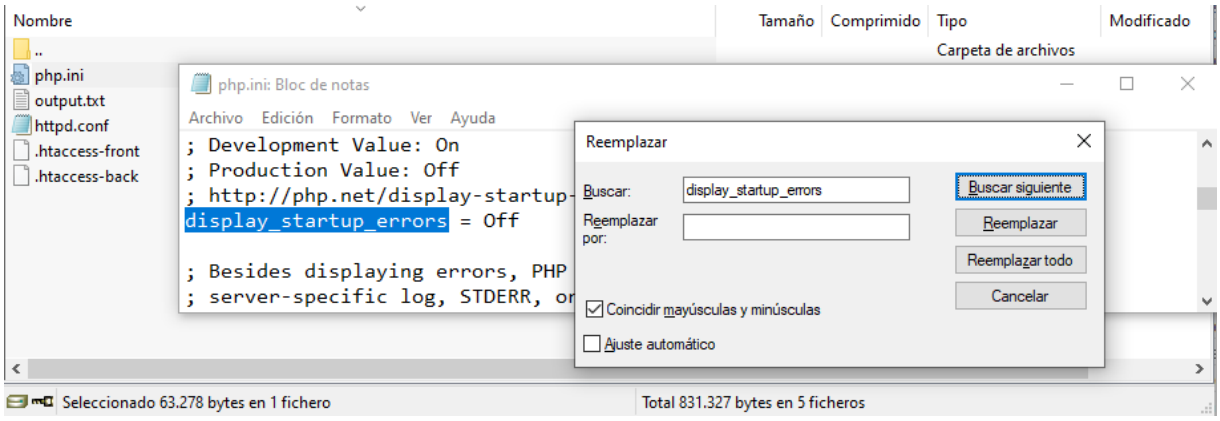


Figura 37. Revisión del parámetro display_startup_errors

Se observa que el parámetro está activado, lo cual es lo recomendado.

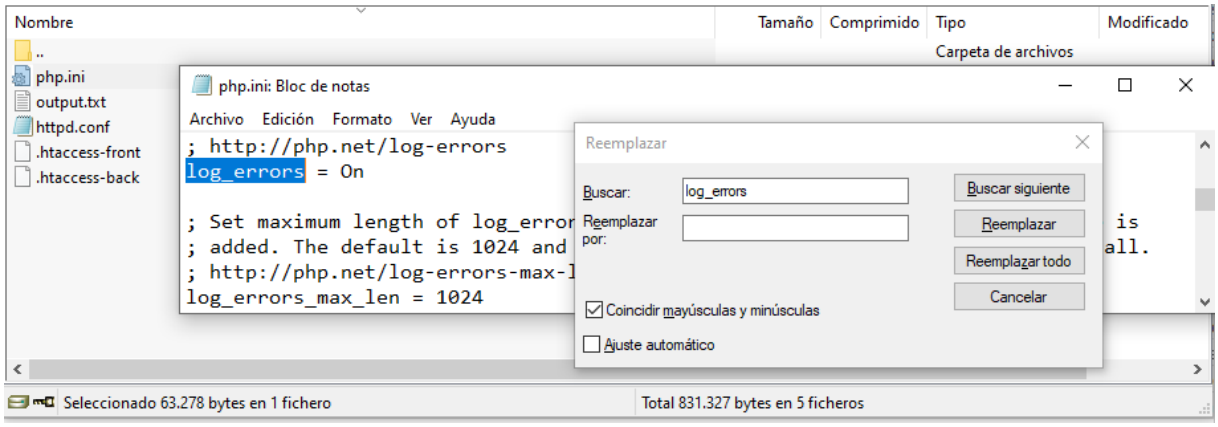


Figura 38. Revisión del parámetro log_errors

Se observa que el parámetro está desactivado, lo cual es lo recomendado.

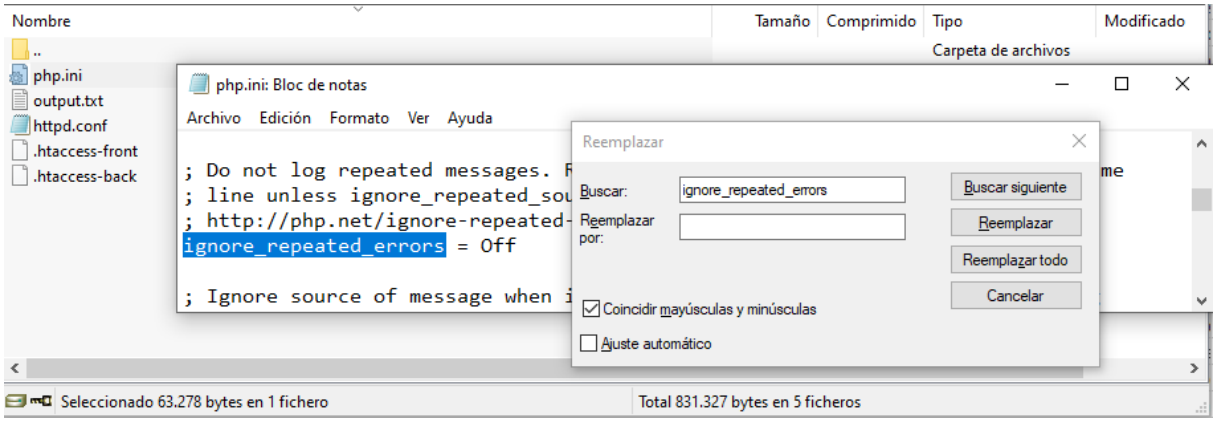


Figura 39. Revisión del parámetro ignore_repeated_errors

Verificación 66. Revisión de cabeceras HTTP CSP

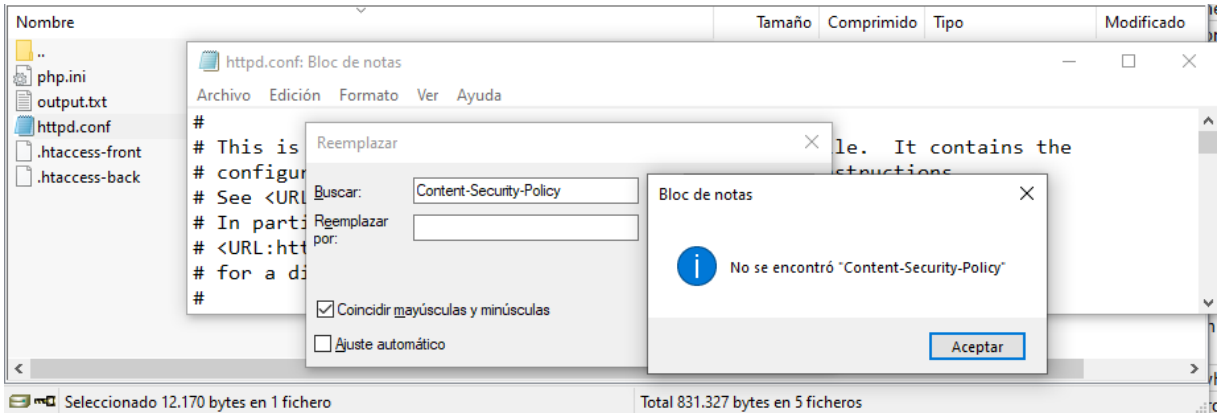


Figura 40. Revisión de cabeceras http content-security-policy

Capturas escaneo DAST

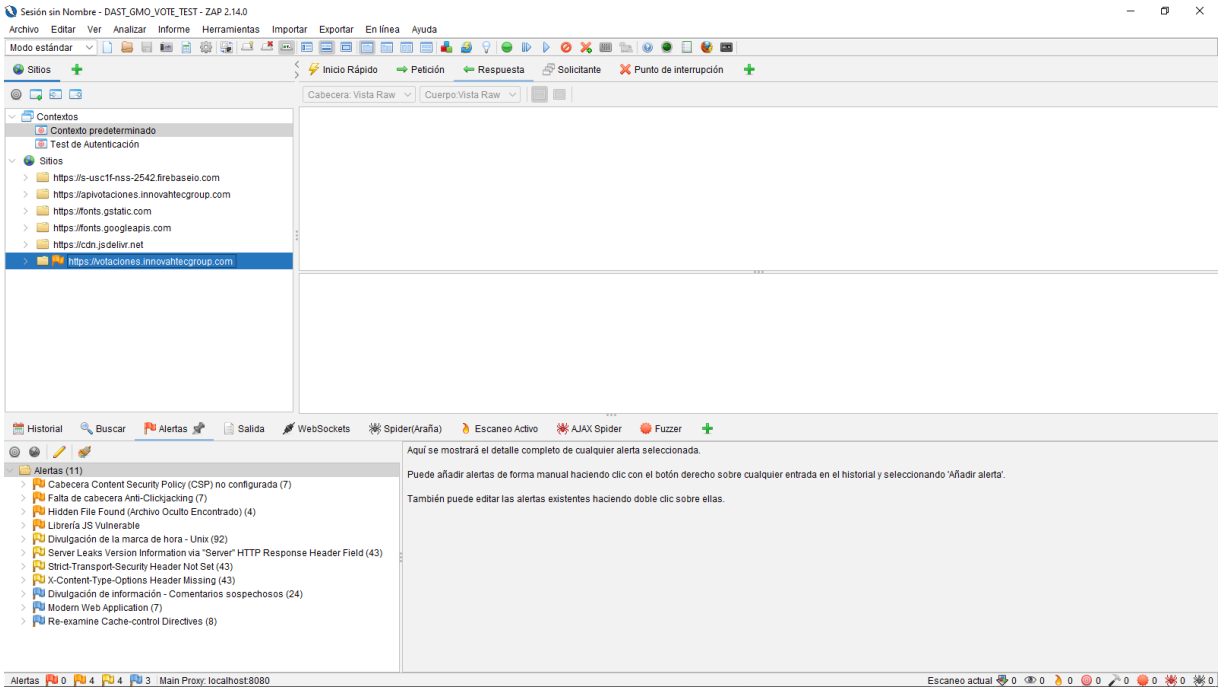


Figura 41. Reporte de escaneo DAST

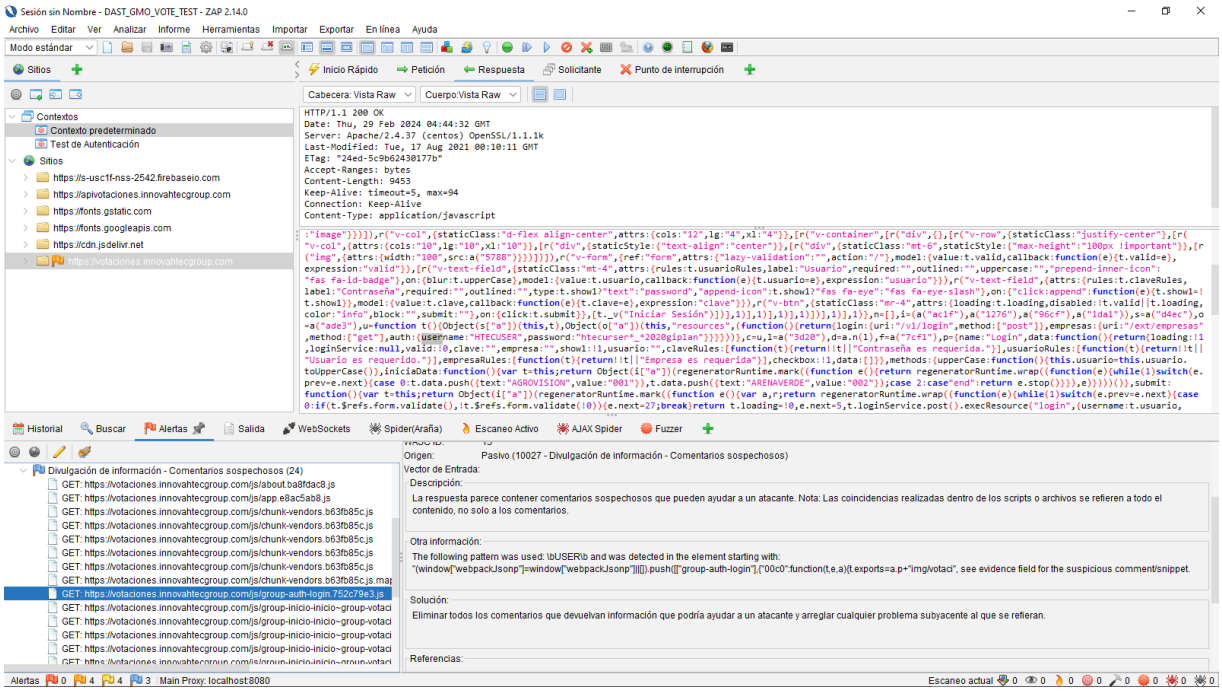


Figura 42. Detección de credenciales de autenticación

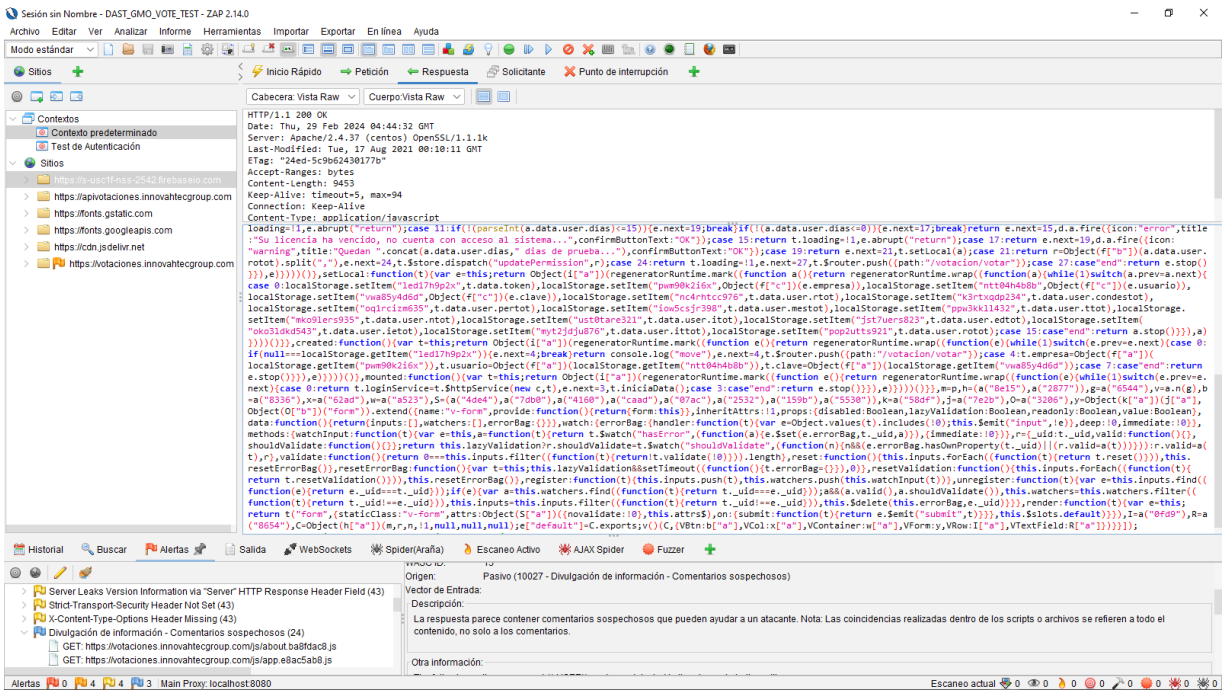


Figura 43. Exposición de datos sensibles en localstorage como token y clave

No se encontró cabecera X-Frame-Options, anti-clickjacking

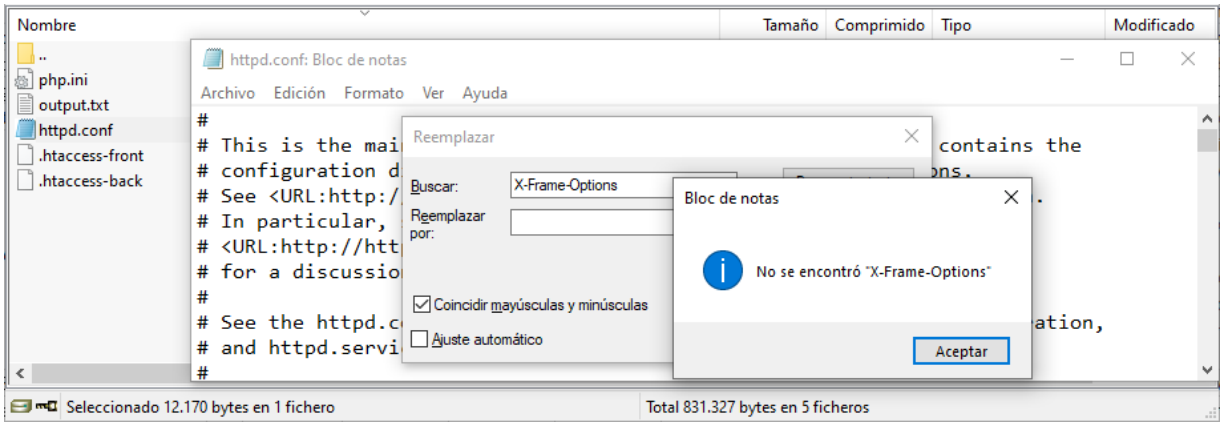


Figura 44. Revisión del parámetro X-Frame-Options

No se encontró cabecera Strict-Transport-Security

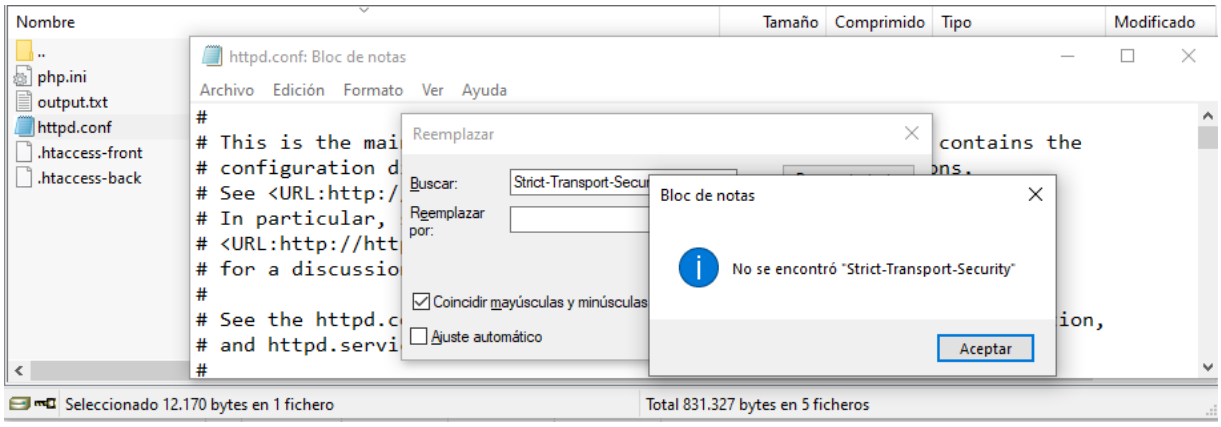


Figura 45. Revisión del parámetro Strict-Transport-Security dentro de httpd.conf

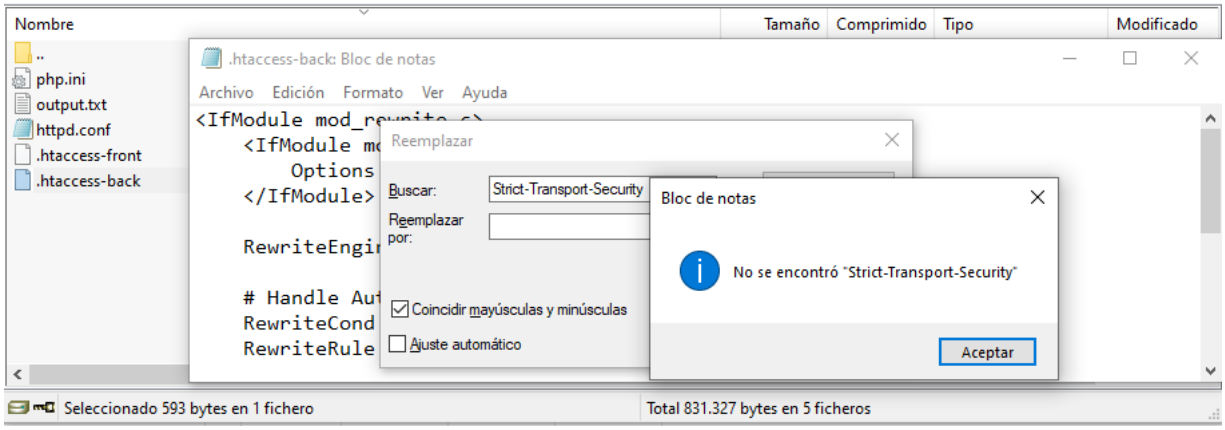


Figura 46. Revisión del parámetro Strict-Transport-Security dentro de htaccess

No se encontró cabecera X-Content-Type-Options

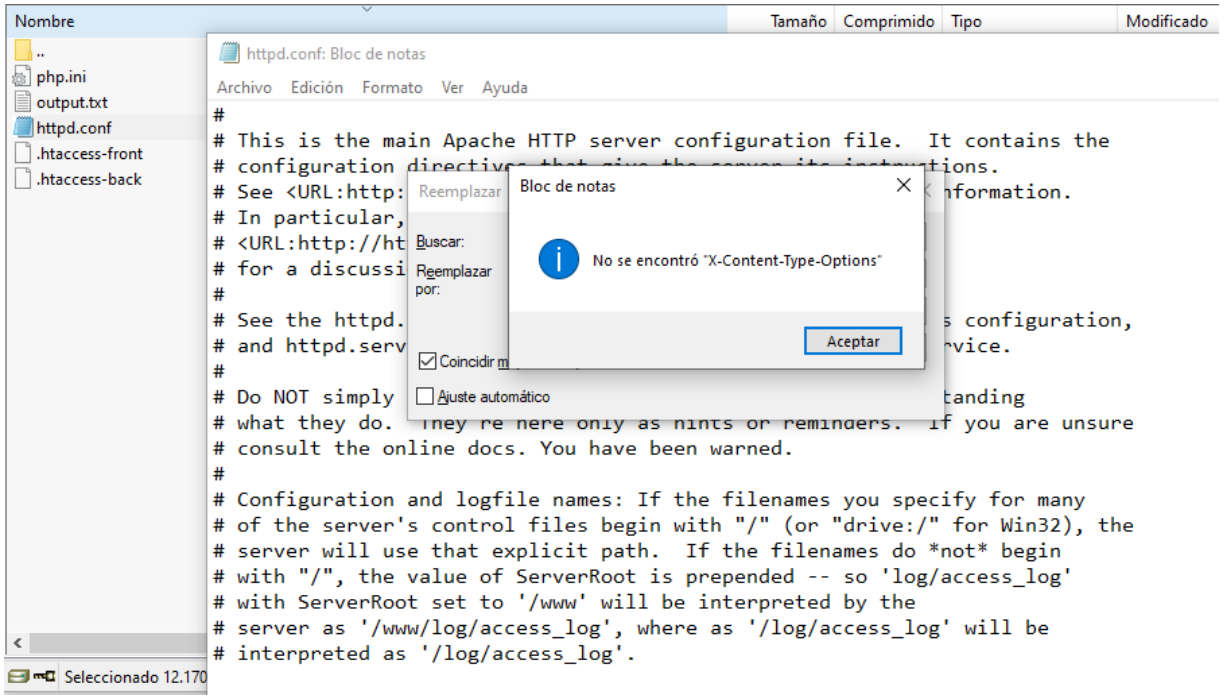


Figura 47. Revisión de parámetro X-Content-Type-Options

Se observó que el reporte de seguridad para el subdominio de la aplicación de votaciones obtuvo la calificación más baja.

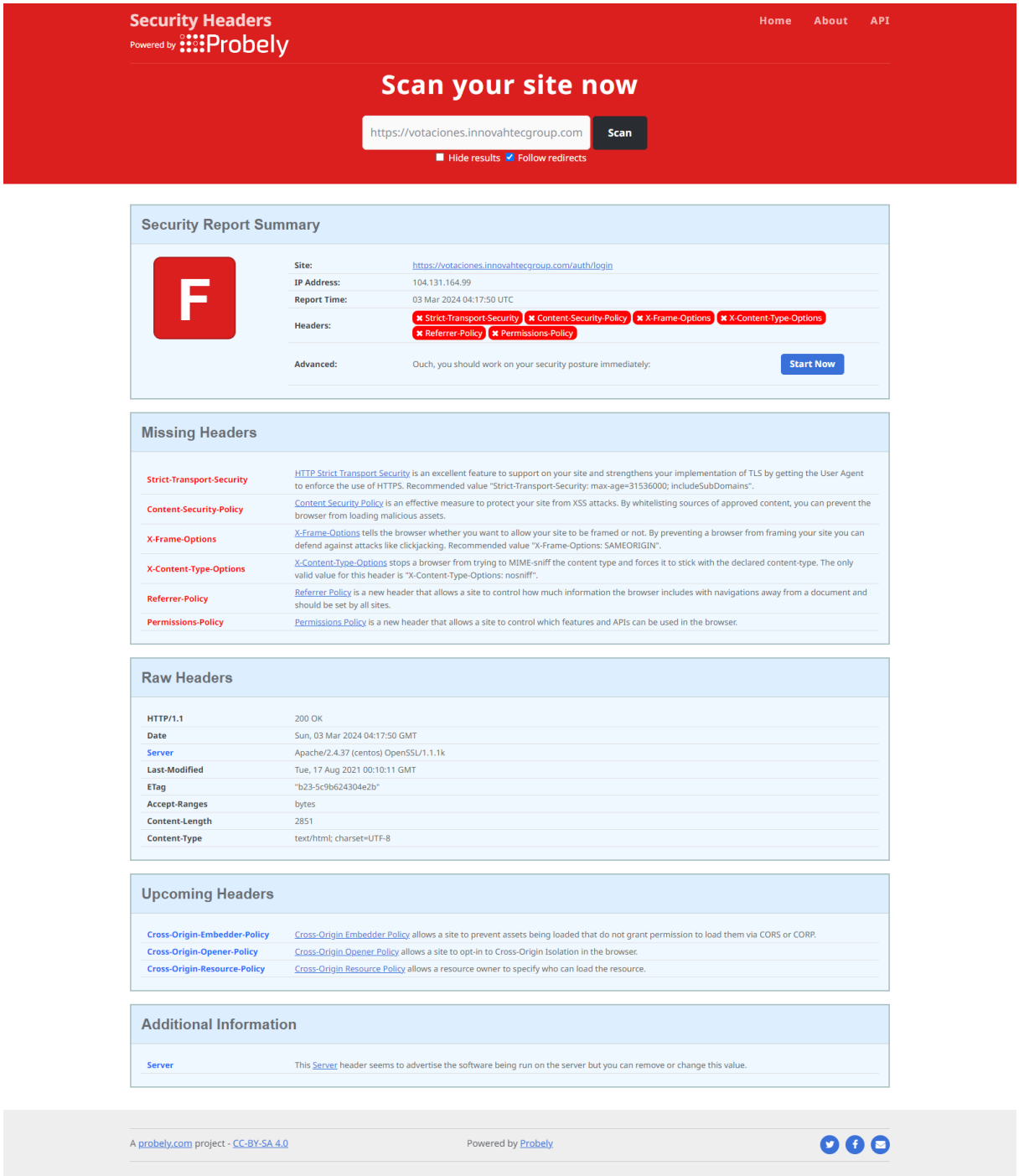


Figura 48. Reporte de cabeceras de seguridad

Capturas comprobación manual

La opción de configuración same_site se encontró establecida según lo recomendado.

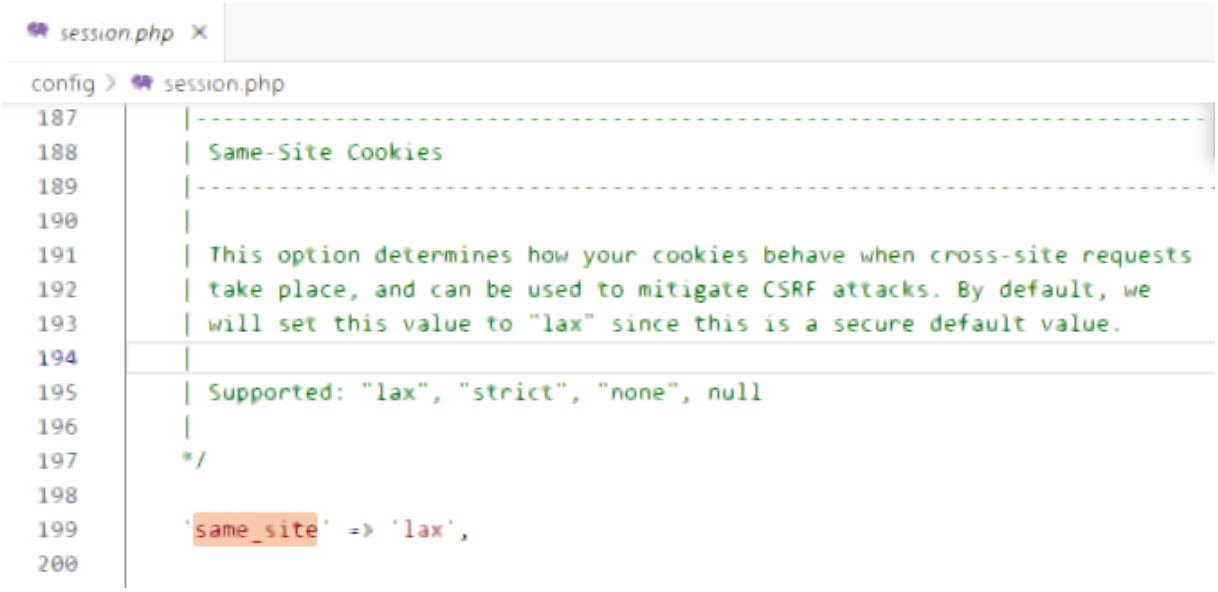


Figura 49. Verificación N°1 Same Site Cookie

Verificación N°2 La aplicación no utiliza cookies para almacenar la sesión, utiliza JWT (Json Web Token).

No se encontró el uso de la función extract

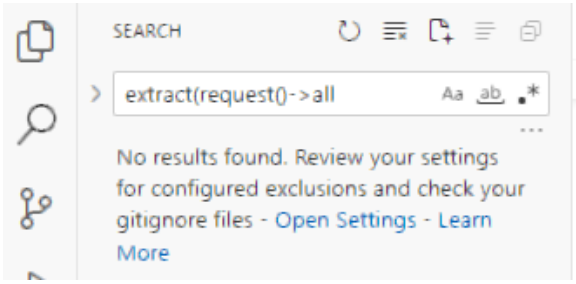


Figura 50. Verificación N°3 Extract Function

La opción de configuración lifetime se encontró establecida dentro del valor recomendado.

```
23      /*
24      |--------------------------------------------------------------------------
25      | Session Lifetime
26      |--------------------------------------------------------------------------
27      |
28      | Here you may specify the number of minutes that you wish the session
29      | to be allowed to remain idle before it expires. If you want them
30      | to immediately expire on the browser closing, set that option.
31      |
32      */
33
34      'lifetime' => env('SESSION_LIFETIME', 120),
35
36      'expire_on_close' => false,
```

Figura 51. Verificación N°4 Session Timeout

Verificación N°5 La aplicación no permite que los usuarios carguen archivos.

Se observa que no está incluido el middleware FrameGuard al grupo web middleware, por lo que debe verificarse la inclusión del encabezado X-Frame-Options en la configuración del servidor web, la cual no se encontró establecida.

```
app > Http > Kernel.php > Kernel > $middlewareGroups
33      protected $middlewareGroups = [
34          'web' => [
35              \App\Http\Middleware\EncryptCookies::class,
36              \Illuminate\Cookie\Middleware\AddQueuedCookiesToResponse::class,
37              \Illuminate\Session\Middleware\StartSession::class,
38              \Illuminate\View\Middleware\ShareErrorsFromSession::class,
39              \App\Http\Middleware\VerifyCsrfToken::class,
40              \Illuminate\Routing\Middleware\SubstituteBindings::class,
41          ],
42
43          'api' => [
44              // \Laravel\Sanctum\Http\Middleware\EnsureFrontendRequestsAreStateful::class,
45              \Illuminate\Routing\Middleware\ThrottleRequests::class.':api',
46              \Illuminate\Routing\Middleware\SubstituteBindings::class,
47              //\Tymon\JWTAuth\Http\Middleware\Authenticate::class,
48              //Tymon\JWTAuth\Facades\JWTAuth::class,
49          ],
50      ];
51

FrameGuard::class
```

Figura 52. Verificación N°6 Clickjacking

No se encontró el uso de funciones exec, shell_exec, system y passthru

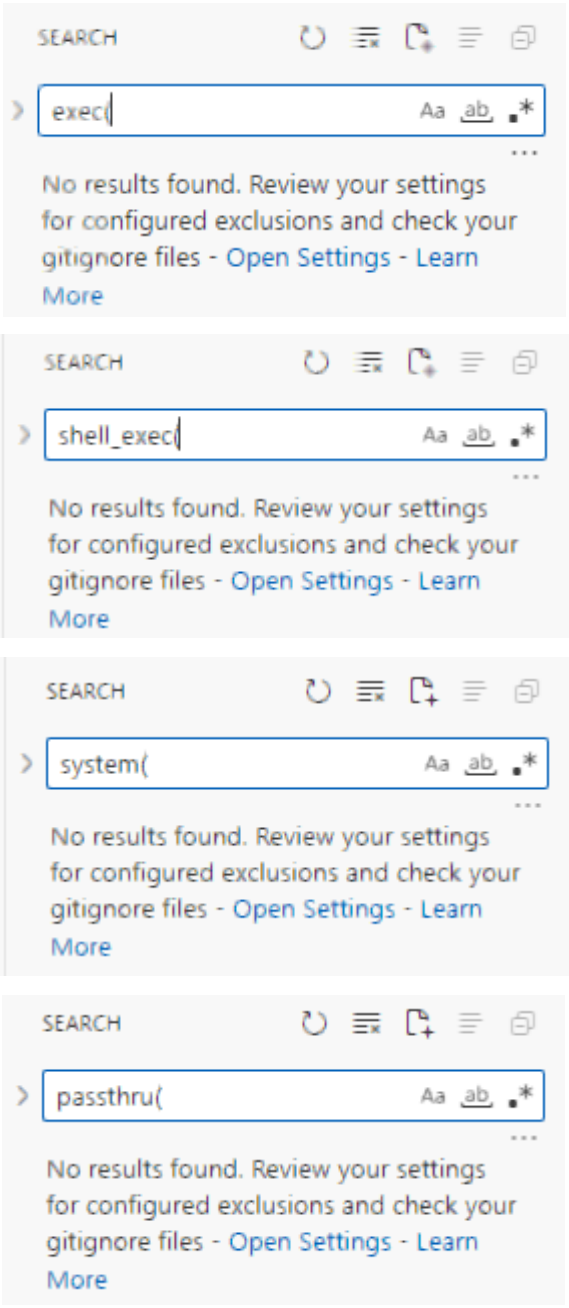


Figura 53. Verificación N°7 Command Injection, uso de funciones exec, shell_exec, system y passthru

Se encontró la línea comentada. Sin embargo, otra alternativa es incluir en la lista blanca sus encabezados TrustedProxy, por lo que se procedió a inspeccionar dichas cabeceras, evidenciándose la cabecera HEADER_X_FORWARDED_FOR, lo cual cumple.

Kernel.php

app > Http > Kernel.php > Kernel > $middleware

TrustHosts::class

```php
<?php

namespace App\Http;

use App\Http\Middleware\JwtMiddleware;
use Illuminate\Foundation\Http\Kernel as HttpKernel;

class Kernel extends HttpKernel
{
    /**
     * The application's global HTTP middleware stack.
     *
     * These middleware are run during every request to your application.
     *
     * @var array<int, class-string|string>
     */
    protected $middleware = [
        // \App\Http\Middleware\TrustHosts::class,
        \App\Http\Middleware\TrustProxies::class,
        \Illuminate\Http\Middleware\HandleCors::class,
        \App\Http\Middleware\PreventRequestsDuringMaintenance::class,
        \Illuminate\Foundation\Http\Middleware\ValidatePostSize::class,
        \App\Http\Middleware\TrimStrings::class,
        \Illuminate\Foundation\Http\Middleware\ConvertEmptyStringsToNull::class,
        \Illuminate\Http\Middleware\HandleCors::class,
    ];

```

Figura 54. Verificación N°8 Host Injection en archivo kernel.php

Kernel.php | TrustProxies.php

app > Http > Middleware > TrustProxies.php > TrustProxies > $headers

ADER_X_FORWARDED_FOR

```php
<?php

namespace App\Http\Middleware;

use Illuminate\Http\Middleware\TrustProxies as Middleware;
use Illuminate\Http\Request;

class TrustProxies extends Middleware
{
    /**
     * The trusted proxies for this application.
     *
     * @var array<int, string>|string|null
     */
    protected $proxies;

    /**
     * The headers that should be used to detect proxies.
     *
     * @var int
     */
    protected $headers =
        Request::HEADER_X_FORWARDED_FOR |
        Request::HEADER_X_FORWARDED_HOST |
        Request::HEADER_X_FORWARDED_PORT |
        Request::HEADER_X_FORWARDED_PROTO |
        Request::HEADER_X_FORWARDED_AWS_ELB;
}

```

Figura 55. Verificación N°8 Host Injection en archivo TrustProxies.php

Revisión en servidor web, no se encontró establecida dicha cabecera

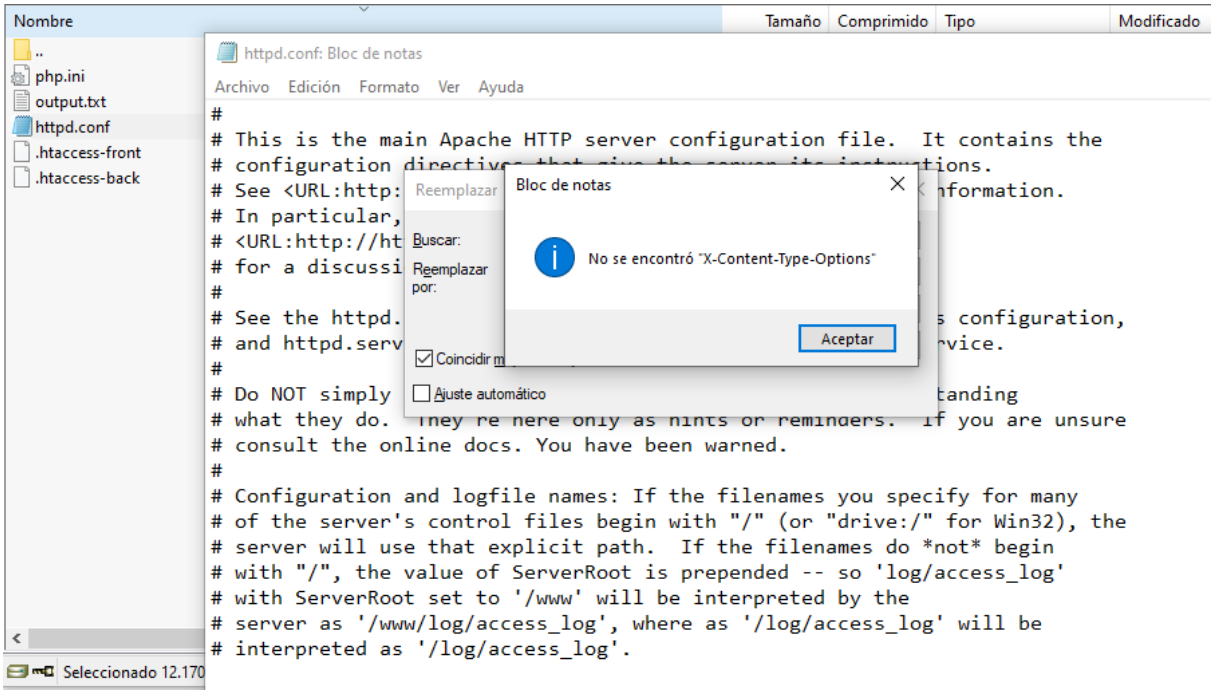


Figura 55. Verificación N°9 Mime Sniffing

No se encontró el uso de la función unserialize

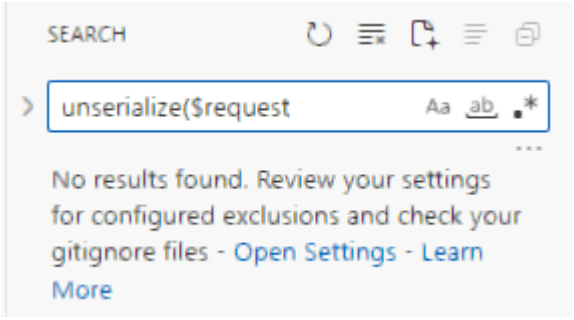


Figura 56. Verificación N°10 Object Injection

Verificación N°11 No se evidenciaron cláusulas ni funciones de búsqueda, ni de ordenamiento en las que se le permita al usuario realizarlas por un conjunto de campos tanto de manera dinámica o estática.

La aplicación no permite al usuario proporcionar un nombre de archivo para la exportación de reportes, los directorios y nombres se generan dentro de la función.

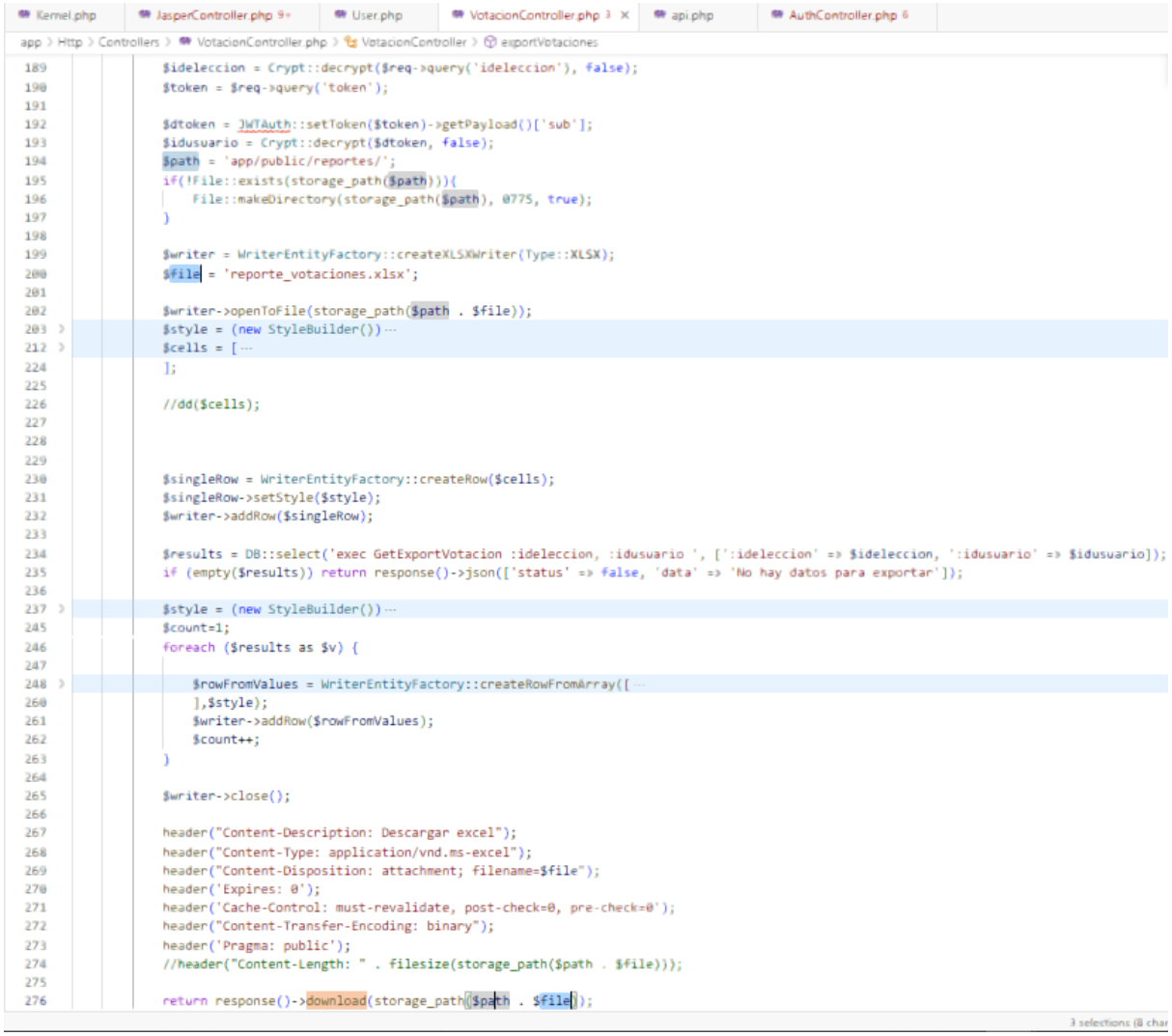


Figura 57. Verificación N°12 Directory Traversal

Figura 58. Verificación N°12 Directory Traversal

Sólo existe una redirección y no es controlada por el usuario, únicamente redirige al home si está autenticado

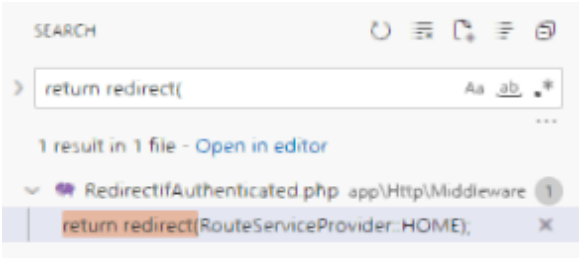
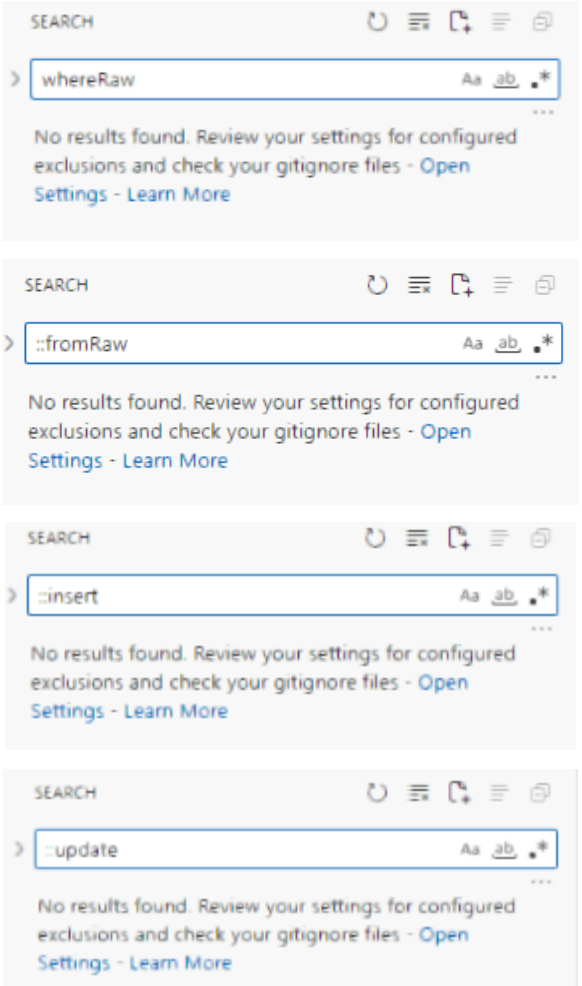


Figura 59. Verificación N°13 Open Redirection

No se encontró el uso de métodos que permitan la ejecución de sentencias SQL en bruto o directas sin el uso de sentencias preparadas. Sin embargo, se encontró el uso del método select pero este invoca la ejecución de un procedimiento almacenado y de forma parametrizada, lo cual cumple.



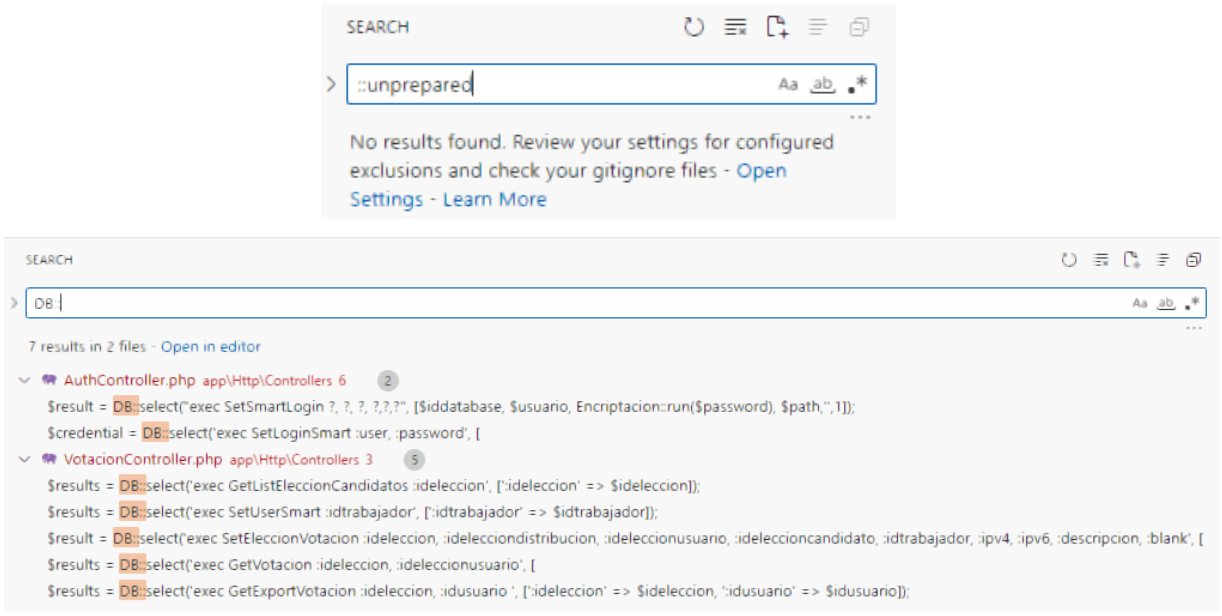


Figura 60. Verificación N°14 Raw SQL Injection

No se encontró el uso de expresiones regulares

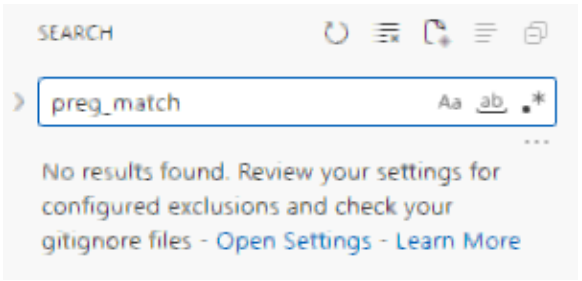


Figura 61. Verificación N°15 Regex DOS

La aplicación no cuenta con la funcionalidad de importación o carga de archivos

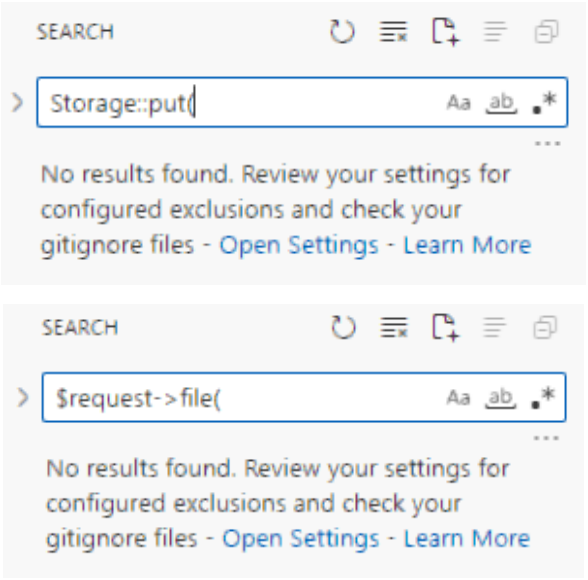


Figura 62. Verificación N°16 Unrestricted File Upload

No se encontró el uso del método ignore

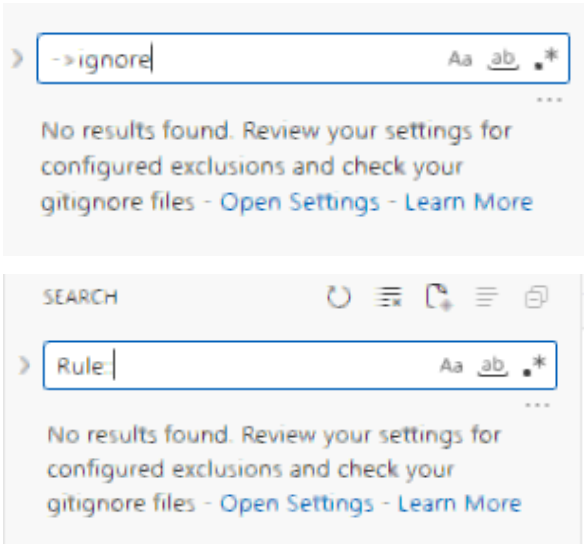
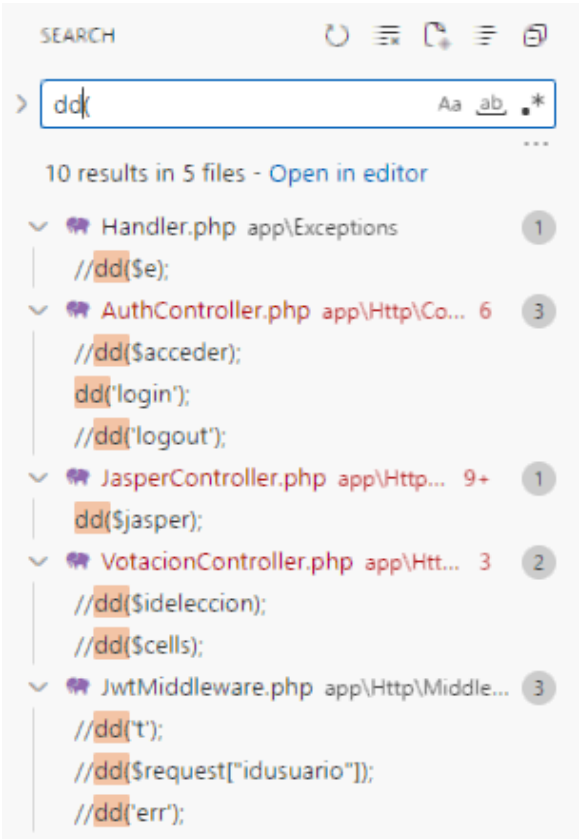


Figura 63. Verificación N° 17 Validation SQL Injection

Se encontraron dos usos de declaraciones de depuración en AuthController.php y en JasperController.php



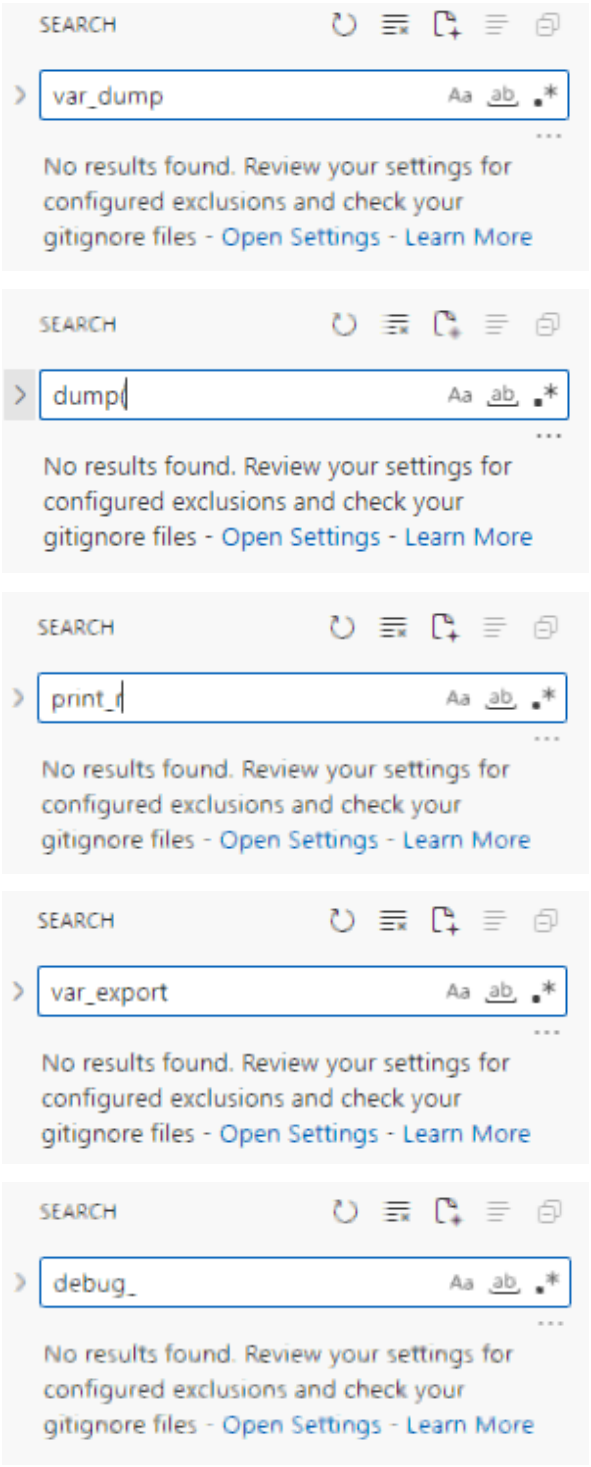


Figura 64. Verificación N°18 Debug Statement

No se encontró uso de la función eval

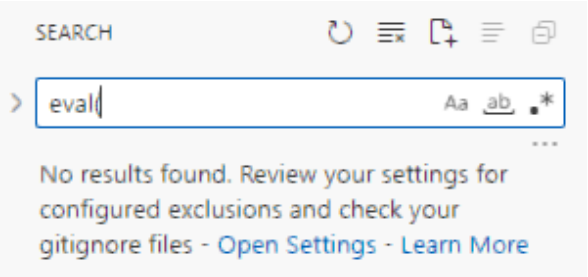


Figura 65. Verificación N°19 Eval Function

Verificación N°20 No se observó interacción directa con el objeto PDO, ni el uso de funciones de bd php nativas, ni el uso del método DB::unprepared de facade.

Verificación N°21 No se encontró el uso de ajax en la aplicación.

Verificación N°25 Se observó el atributo http_only establecido según lo recomendado.

Verificación N°26 La aplicación no utiliza cookies para almacenar la sesión, utiliza JWT (Json Web Token).

Verificación N°27 Se observó el middleware EncryptCookies agregado de forma predeterminada al grupo middleware según lo recomendado.

POST TEST

Capturas escaneo SAST

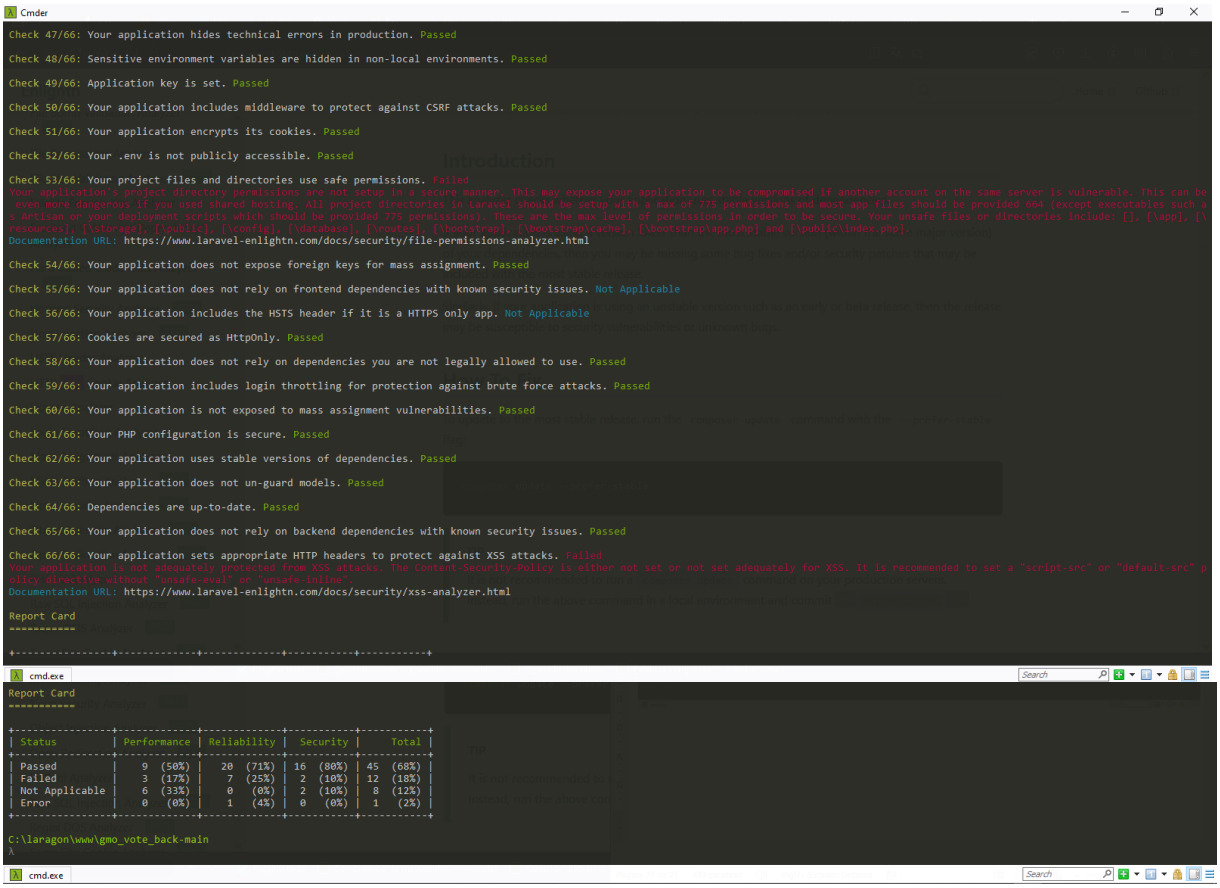


Figura 66. Reporte de escaneo SAST

Verificación 53. Esta verificación se realizó directamente en el servidor de producción, en donde se evidencia que los permisos aplicados son más estrictos que los recomendados.

Verificación 61. Se establecieron los parámetros adecuados en el archivo de configuración de php del servidor.

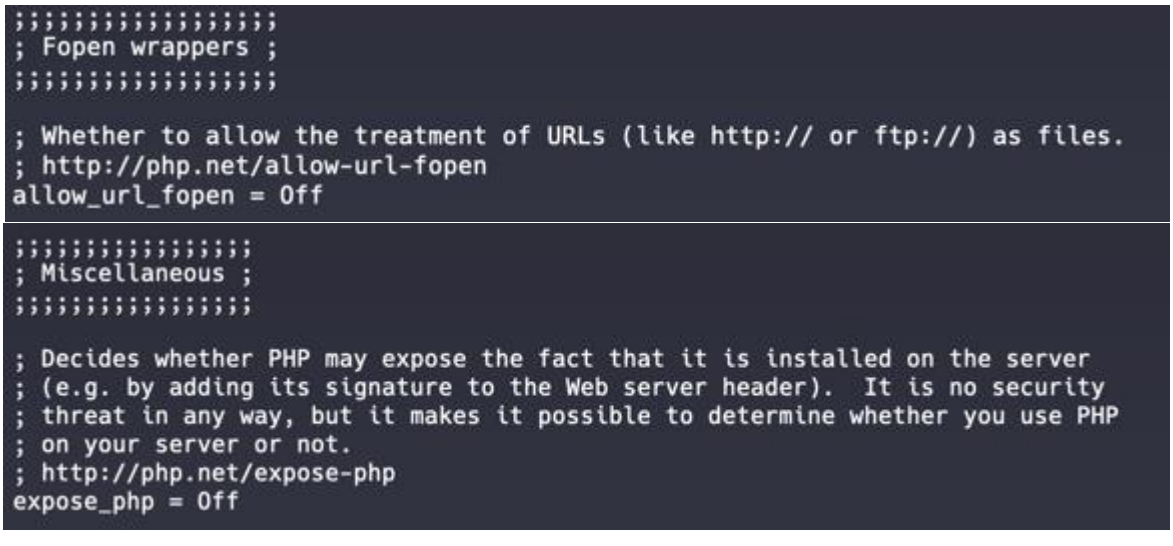


Figura 67. Verificación N°22 PHP ini

Verificación 62. Se actualizaron las dependencias a versiones estables en las que no se encontraron avisos relacionados a vulnerabilidades de seguridad.

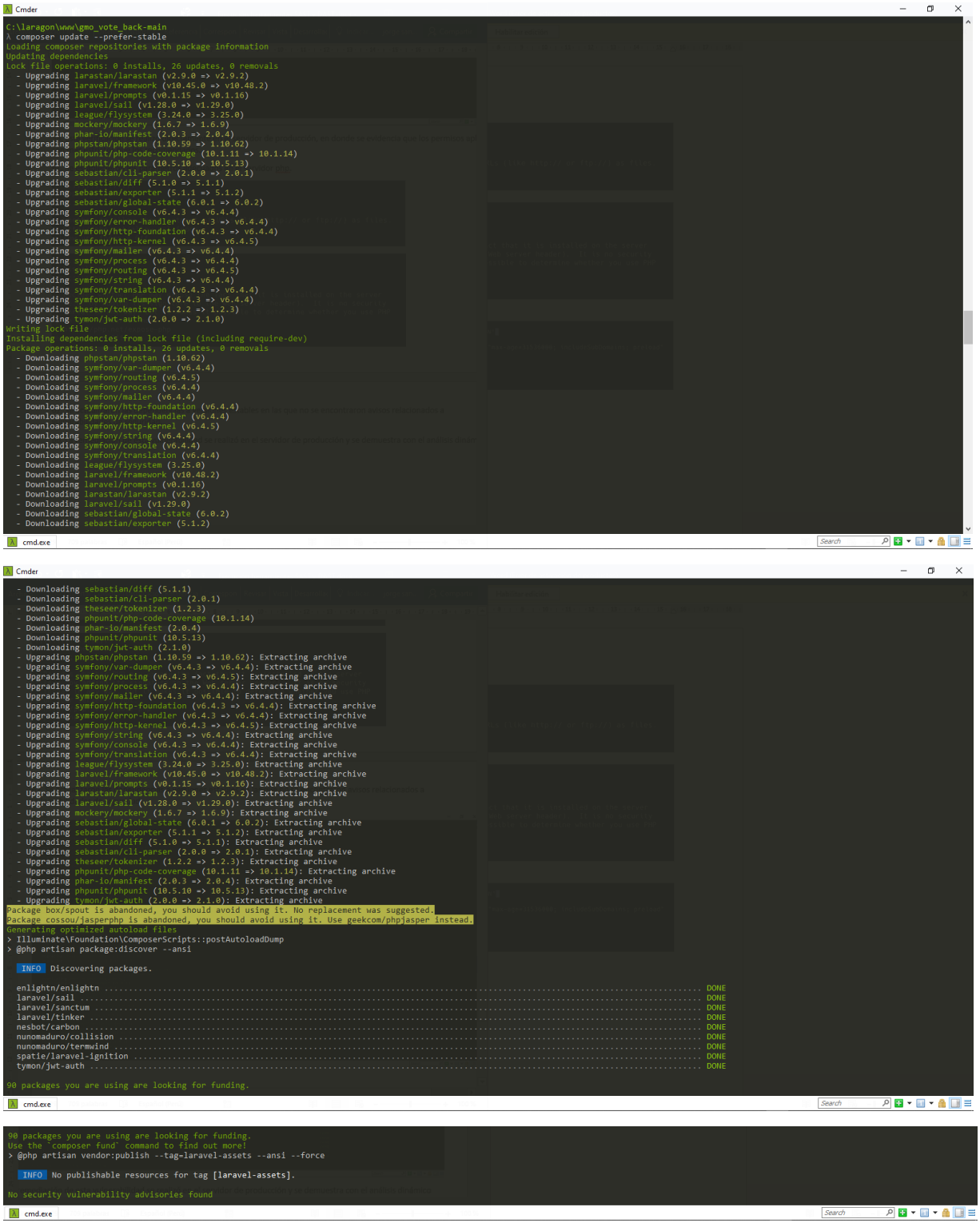


Figura 68. Actualización de dependencias

Verificación 66. La remediación de esta vulnerabilidad se realizó en el servidor de producción y se demuestra con el análisis dinámico.

```
<IfModule mod_ssl.c>
<VirtualHost *:443>
#Protocols h2 http/1.1
ServerName votaciones.innovahtecgroup.com
ServerAlias www.votaciones.innovahtecgroup.com
ServerAdmin jmendozadev@innovahtecgroup.com
ServerSignature Off
DocumentRoot /var/www/votaciones.innovahtecgroup.com/html/votaciones/dist

<Directory "/var/www/votaciones.innovahtecgroup.com/html/votaciones/dist">
        Options Indexes FollowSymLinks
        AllowOverride All
        Require all granted
</Directory>

RewriteEngine On
# Some rewrite rules in this file were disabled on your HTTPS site,
# because they have the potential to create redirection loops.

# RewriteRule ^ https://%{SERVER_NAME}%{REQUEST_URI} [END,NE,R=permanent]

#ErrorLog /var/log/httpd/redirect.error.log

ErrorLog /var/www/votaciones.innovahtecgroup.com/log/redirect.error.log
CustomLog /var/www/votaciones.innovahtecgroup.com/log/requests.log combined

LogLevel warn

Header always set Content-Security-Policy "default-src 'self'; script-src 'self'; style-src 'self' https://cdn.jsdelivr.net https://font
s.googleapis.com; font-src 'self' https://cdn.jsdelivr.net https://fonts.gstatic.com; img-src 'self'; frame-src 'self'; connect-src 'sel
f' https://apivotaciones.innovahtecgroup.com wss://s-usc1f-nss-2542.firebaseio.com; object-src 'none'; base-uri 'self'; manifest-src 'se
lf';media-src 'self'; worker-src 'none' "
```

Figura 69. Inclusión de cabecera Content-Security-Policy

Capturas escaneo DAST

Se observó en la prueba posterior que prevalecen cuatro vulnerabilidades respecto a archivos ocultos encontrados que no se pudieron remediar con el estándar de seguridad definido.

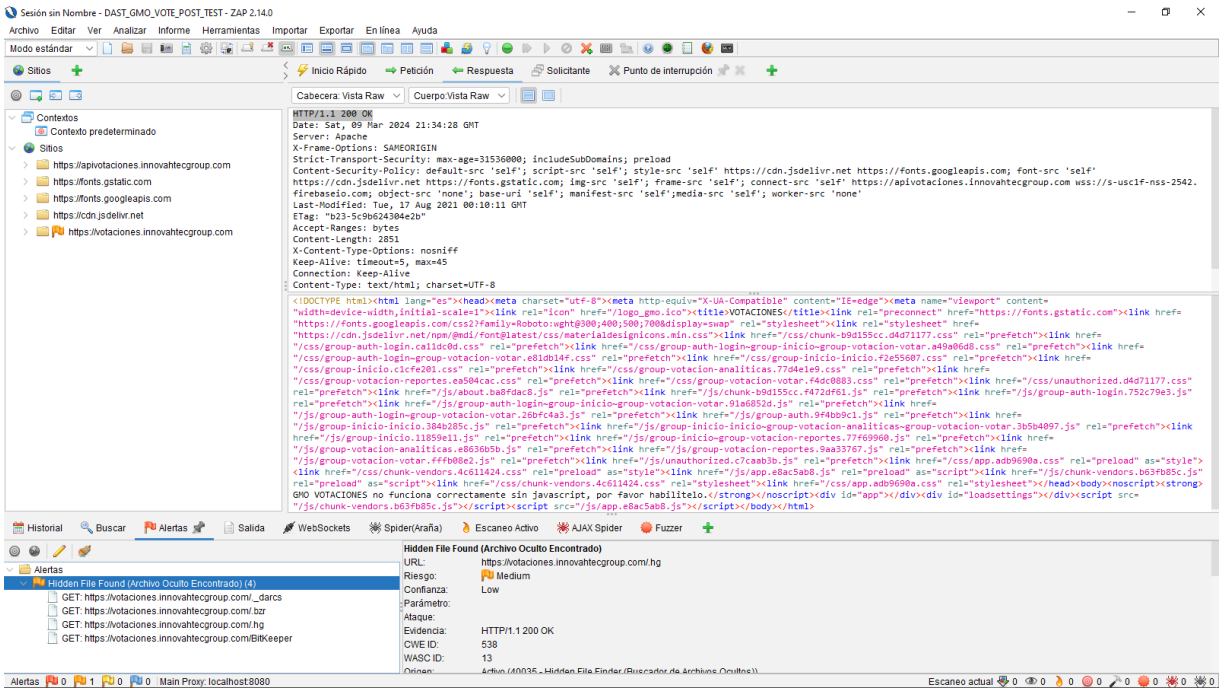


Figura 70. Reporte escaneo DAST

Se observó que el reporte de seguridad para el subdominio de la aplicación de votaciones mejoró, posicionándose en la categoría más alta.

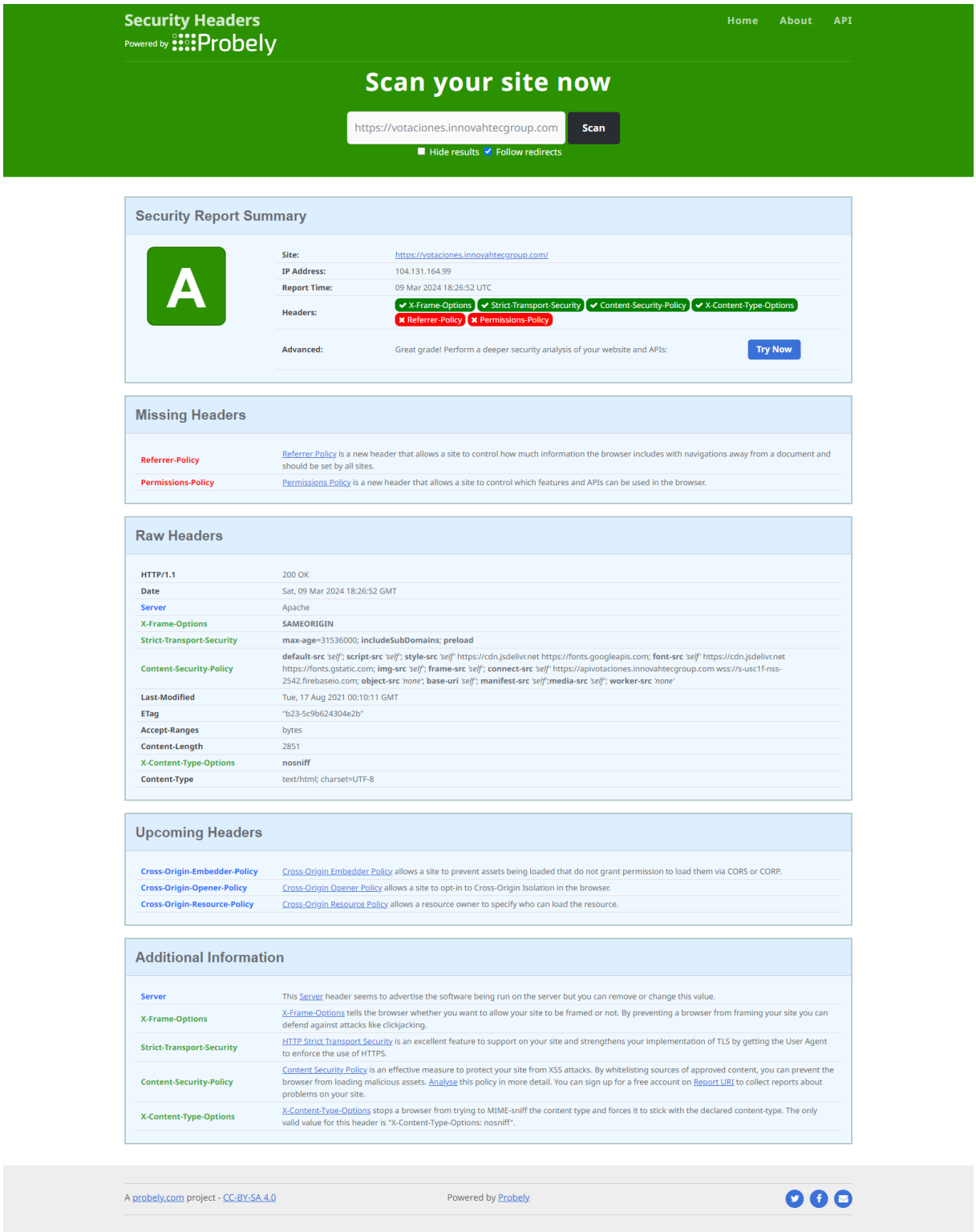


Figura 71. Reporte de cabeceras de seguridad

Tabla 26. Resumen de cantidad de vulnerabilidades por categoría de vulnerabilidad y tipo de revisión

N°	Tipo de revisión	CWE	Categorías de vulnerabilidades	Severidad	VP /FP	Cantidad de vulnerabilidades	Estado
1	SAST	98	Configuración de php insegura	alta	VP	2	Resuelta
2	SAST	-	Uso de versiones inestables de dependencias	alta	VP	26	Resuelta
3	MANUAL	-	Uso de declaraciones de depuración	alta	VP	1	Resuelta
4	DAST	693	Cabecera CSP no configurada	media	VP	7	Resuelta
5	DAST	1021	Falta de cabecera anti-clickjacking	media	VP	7	Resuelta
6	DAST	538	Archivo oculto encontrado	media	VP	4	No resuelta
7	DAST	829	Librería js vulnerable	media	VP	1	Resuelta
8	DAST	200	Divulgación de la marca de hora - UNIX	baja	VP	92	Resuelta
9	DAST	200	Servidor divulga información a través de campo server en la cabecera de respuesta HTTP	baja	VP	43	Resuelta
10	DAST	319	Cabecera Strict-Transport-Security no configurada	baja	VP	43	Resuelta
11	DAST	693	Cabecera X-Content-Type-Options no configurada	baja	VP	43	Resuelta
			Total			269	

ANEXO 6 – Instrumento de evaluación para la validación de contenido mediante juicio de expertos

Respetado juez: _____, Usted ha sido seleccionado para evaluar la metodología propuesta de la investigación: "METODOLOGÍA DE SEGURIDAD DE SOFTWARE PARA MEJORAR LA SEGURIDAD DEL SOFTWARE COMO SERVICIO EN EMPRESAS DE DESARROLLO". Agradecemos su valiosa colaboración.

FORMACIÓN ACADÉMICA: _____

AREAS DE EXPERIENCIA PROFESIONAL: _____

_____

TIEMPO DE EXPERIENCIA: _____

CARGO ACTUAL: _____

ORGANIZACIÓN: _____

OBJETIVO DE LA INVESTIGACIÓN: Crear una metodología para mejorar la seguridad del software como servicio_____

OBJETIVO DEL JUICIO DE EXPERTOS: Lograr una validación de contenido_____

De acuerdo con los siguientes indicadores califique cada uno de los ítems según corresponda.

CATEGORÍA	CALIFICACIÓN	INDICADOR
SUFICIENCIA Los ítems que pertenecen a una misma dimensión bastan para obtener la medición de esta.	1. No cumple con el criterio	Los ítems no son suficientes para medir la dimensión.
	2. Bajo nivel	Los ítems miden algún aspecto de la dimensión, pero no corresponden con la dimensión total.
	3. Moderado nivel	Se deben incrementar algunos ítems para poder evaluar la dimensión completamente.
	4. Alto nivel	Los ítems son suficientes.
CLARIDAD El ítem se comprende fácilmente, es decir, su sintáctica y semántica son adecuadas.	1. No cumple con el criterio	El ítem no es claro.
	2. Bajo nivel	El ítem requiere bastantes modificaciones o una modificación muy grande en el uso de las palabras de acuerdo con su significado o por la ordenación de las mismas.
	3. Moderado nivel	Se requiere una modificación muy específica de algunos de los términos del ítem.
	4. Alto nivel	El ítem es claro, tiene semántica y sintaxis adecuada.
COHERENCIA El ítem tiene relación lógica con la dimensión o indicador que está midiendo.	1. No cumple con el criterio	El ítem no tiene relación lógica con la dimensión.
	2. Bajo nivel	El ítem tiene una relación tangencial con la dimensión.
	3. Moderado nivel	El ítem tiene una relación moderada con la dimensión que está midiendo.
	4. Alto nivel	El ítem se encuentra completamente relacionado con la dimensión que está midiendo.
RELEVANCIA El ítem es esencial o importante, es decir, debe ser incluido.	1. No cumple con el criterio	El ítem puede ser eliminado sin que se vea afectada la medición de la dimensión.
	2. Bajo nivel	El ítem tiene alguna relevancia, pero otro ítem puede estar incluyendo lo que mide este.
	3. Moderado nivel	

	4. Alto nivel	El ítem es relativamente importante. El ítem es muy relevante y debe ser incluido.

DIMENSIÓN	PRÁCTICA	ITEM	SUFICIENCIA*	CLARIDAD	COHERENCIA	RELEVANCIA	OBSERVACIONES
Gobierno	Estrategia y métricas	Definir apetito, tolerancia y capacidad del riesgo					
		Realizar perfil de riesgo de la aplicación					
		Definir la estrategia					
		Establecer el proceso de seguridad de software					
		Definir y utilizar el proceso análisis de la arquitectura					
		Establecer el proceso de construcción					
		Identificar objetivos de seguridad de software					
		Determinar presupuestos					
		Asignar roles y responsabilidades					
		Establecer métricas y KPIs estratégicos					
		Establecer estándares de seguridad					

	Entrenamiento	Realizar capacitaciones o eventos					
		Identificar campeones de seguridad					
Diseño	Análisis de la arquitectura	Realizar una revisión de las características de seguridad					
		Utilizar lista de verificación (checklist) de principios de seguridad					
	Modelos de ataque	Recopilar y utilizar inteligencia de ataque					
		Mantener y utilizar una lista de los N principales ataques posibles					
		Recopilar y publicar historias de ataques					
		Realizar modelado de amenazas					
		Identificar requisitos de seguridad					
		Realizar evaluaciones a					

		sus proveedores de software					
Implementación	Revisión de código	Realizar una revisión de código oportuna y obligatoria					
		Utilizar herramientas automatizadas					
	Pruebas de seguridad de software	Integrar herramientas de seguridad de caja opaca en el proceso de QA					
Operaciones	Entorno de software	Usar monitoreo de entrada de la aplicación					
		Definir configuraciones y parámetros de implementación seguros					

¿Hay alguna dimensión que hace parte del constructo y no fue evaluada? ¿Cuál?

_____
_____
_____

*Para los casos de equivalencia semántica se deja una casilla por ítem, ya que se evaluará si la traducción o el cambio en vocabulario son suficientes.

RESULTADOS

Opinión: Marque con un aspa (X) en alguna de las casillas correspondientes.

	FAVORABLE		DEBE MEJORAR		DESFAVORABLE

_____
FIRMA DEL EXPERTO

ANEXO 7 – Resultados de la validación de contenido mediante juicio de expertos de la metodología propuesta

			Experto 1				Experto 2				Experto 3			
Fase	Prácticas	Actividades	Suficiencia	Claridad	Coherencia	Relevancia	Suficiencia	Claridad	Coherencia	Relevancia	Suficiencia	Claridad	Coherencia	Relevancia
Gobierno	Estrategia y métricas	Definir apetito, tolerancia y capacidad del riesgo	4	4	4	4	4	4	4	4	4	4	4	4
		Realizar perfil de riesgo de la aplicación	4	4	4	4	4	4	4	4	4	4	4	4
		Definir la estrategia	4	4	4	3	4	4	4	4	4	4	4	4
		Establecer el proceso de seguridad de software	4	4	4	4	4	4	4	4	4	3	4	4
		Definir y utilizar el proceso análisis de la arquitectura	4	4	4	4	4	4	4	4	4	4	4	4
		Establecer el proceso de construcción	4	4	4	4	4	4	4	4	4	3	4	4
		Identificar objetivos de seguridad de software	4	4	4	4	4	4	4	4	4	4	4	4
		Determinar presupuestos	3	4	4	4	4	4	4	4	4	4	4	4
		Asignar roles y responsabilidades	4	4	4	4	4	4	4	4	4	4	4	4
		Establecer métricas y KPIs estratégicos	4	4	4	4	4	4	4	4	4	4	4	4
		Establecer estándares de seguridad	3	4	3	3	4	4	4	4	4	4	3	4
	Entrenamiento	Realizar capacitaciones o eventos	4	4	4	4	4	4	4	4	4	4	4	4
		Identificar campeones de seguridad	4	4	4	4	4	4	4	4	4	4	4	4
Diseño	Análisis de la arquitectura	Realizar una revisión de las características de seguridad	4	4	4	4	4	4	4	4	4	3	4	4
		Utilizar lista de verificación (checklist) de principios de seguridad	4	4	4	4	4	4	4	4	3	4	4	4
	Modelos de	Recopilar y utilizar inteligencia de ataque	4	4	4	4	4	4	4	4	3	4	4	4

	ataque	Mantener y utilizar una lista de los N principales ataques posibles	3	4	4	3	4	4	4	4	3	4	4	4
		Recopilar y publicar historias de ataques	4	4	4	4	4	4	4	4	4	4	3	4
		Realizar modelado de amenazas	4	4	4	4	4	4	4	4	4	4	4	4
		Identificar requisitos de seguridad	4	4	4	3	4	4	4	4	4	4	4	4
		Realizar evaluaciones a sus proveedores de software	4	4	4	4	4	4	4	4	4	4	4	4
Implementación	Revisión de código	Realizar una revisión de código oportuna y obligatoria	4	4	4	4	4	4	4	4	3	4	4	4
		Utilizar herramientas automatizadas	4	4	4	4	4	4	4	4	3	4	4	4
	Pruebas de seguridad de software	Integrar herramientas de seguridad de caja opaca en el proceso de QA	4	4	4	4	4	4	4	4	3	4	4	4
Operaciones	Entorno de software	Usar monitoreo de entrada de la aplicación	4	4	4	4	4	4	4	4	4	4	4	4
		Definir configuraciones y parámetros de implementación seguros	4	4	4	4	4	4	4	4	4	4	4	4

Las recomendaciones por parte de los expertos y que fueron actualizadas son: la agrupación de los items métricas y kpis, la creación de escenarios por tamaño de empresa de desarrollo, para lo cual se consideraron las actividades mínimas que deben hacer manteniendo el enfoque preventivo y finalmente eliminar el item plan estratégico de seguridad debido a su amplitud y porque escapa al ámbito de la tesis que solamente es la de ser una metodología de seguridad de software.

Experto 1

INSTRUMENTO DE EVALUACIÓN PARA LA VALIDACIÓN DE CONTENIDO MEDIANTE JUICIO DE EXPERTOS

Respetado juez: Gonzalo Martin Valdivia Benites, Usted ha sido seleccionado para evaluar la metodología propuesta de la investigación: "METODOLOGÍA DE SEGURIDAD DE SOFTWARE PARA MEJORAR LA SEGURIDAD DEL SOFTWARE COMO SERVICIO EN EMPRESAS DE DESARROLLO". Agradecemos su valiosa colaboración.

FORMACIÓN ACADÉMICA: Magister en Dirección de Tecnologias de Informacion

AREAS DE EXPERIENCIA PROFESIONAL: Seguridad de la Información, Ciber seguridad, Control, Gobierno de TI, Auditoria de Sistemas

TIEMPO DE EXPERIENCIA: 32 años

CARGO ACTUAL: Director Corporativo de Seguridad TI (CISO)

ORGANIZACIÓN: Yanbal

OBJETIVO DE LA INVESTIGACIÓN: Crear una metodología para mejorar la seguridad del software como servicio

OBJETIVO DEL JUICIO DE EXPERTOS: Lograr una validación de contenido

De acuerdo con los siguientes indicadores califique cada uno de los ítems según corresponda.

CATEGORÍA	CALIFICACIÓN	INDICADOR
SUFICIENCIA Los ítems que pertenecen a una misma dimensión bastan para obtener la medición de esta.	1. No cumple con el criterio	Los ítems no son suficientes para medir la dimensión.
	2. Bajo nivel	Los ítems miden algún aspecto de la dimensión, pero no corresponden con la dimensión total.
	3. Moderado nivel	Se deben incrementar algunos ítems para poder evaluar la dimensión completamente.
	4. Alto nivel	Los ítems son suficientes.
CLARIDAD El ítem se comprende fácilmente, es decir, su sintáctica y semántica son adecuadas.	1. No cumple con el criterio	El ítem no es claro.
	2. Bajo nivel	El ítem requiere bastantes modificaciones o una modificación muy grande en el uso de las palabras de acuerdo con su significado o por la ordenación de las mismas.
	3. Moderado nivel	Se requiere una modificación muy específica de algunos de los términos del ítem.
	4. Alto nivel	El ítem es claro, tiene semántica y sintaxis adecuada.
COHERENCIA El ítem tiene relación lógica con la dimensión o indicador que está midiendo.	1. No cumple con el criterio	El ítem no tiene relación lógica con la dimensión.
	2. Bajo nivel	El ítem tiene una relación tangencial con la dimensión.
	3. Moderado nivel	El ítem tiene una relación moderada con la dimensión que está midiendo.
	4. Alto nivel	El ítem se encuentra completamente relacionado con la dimensión que está midiendo.
RELEVANCIA El ítem es esencial o importante, es decir, debe ser incluido.	1. No cumple con el criterio	El ítem puede ser eliminado sin que se vea afectada la medición de la dimensión.
	2. Bajo nivel	El ítem tiene alguna relevancia, pero otro ítem puede estar incluyendo lo que mide este.
	3. Moderado nivel	El ítem es relativamente importante.
	4. Alto nivel	El ítem es muy relevante y debe ser incluido.

DIMENSIÓN	PRÁCTICA	ITEM	SUFICIENCIA*	CLARIDAD	COHERENCIA	RELEVANCIA	OBSERVACIONES
Gobierno	Estrategia y métricas	Definir apetito, tolerancia y capacidad del riesgo	4	4	4	4	Todo depende de ello
		Realizar perfil de riesgo de la aplicación	4	4	4	4	Idem
		Definir la estrategia	4	4	4	3	En la practica muchas veces no se tiene estrategia
		Establecer el proceso de seguridad de software	4	4	4	4	
		Definir y utilizar el proceso análisis de la arquitectura	4	4	4	4	La Arquitecutura es algo pocas veces visto
		Establecer el proceso de construcción	4	4	4	4	
		Identificar objetivos de seguridad de software	4	4	4	4	Esto es crítico
		Determinar presupuestos	3	4	4	4	Super relevante, aunque difícil de calcular con exactitud
		Asignar roles y responsabilidades	4	4	4	4	
		Establecer métricas y KPIs estratégicos	4	4	4	4	Esta es la base para la mejra del modelo
		Establecer estándares de seguridad	3	4	3	3	Los estandares deben usarse con cuidado En Latinoamercia deben personalizarse
	Entrenamiento	Realizar capacitaciones o eventos	4	4	4	4	Super relevante

		Identificar campeones de seguridad	4	4	4	4	
Diseño	Análisis de la arquitectura	Realizar una revisión de las características de seguridad	4	4	4	4	
		Utilizar lista de verificación (checklist) de principios de seguridad	4	4	4	4	
	Modelos de ataque	Recopilar y utilizar inteligencia de ataque	4	4	4	4	Si esto no hay visibilidad
		Mantener y utilizar una lista de los N principales ataques posibles	3	4	4	3	Es muy difícil mantener una lista actualizada, por que los ataques varian muy rapido
		Recopilar y publicar historias de ataques	4	4	4	4	Esto es muy efectivo y permanentemente se deben actualizar
		Realizar modelado de amenazas	4	4	4	4	
		Identificar requisitos de seguridad	4	4	4	3	Los requisitos deben ser tomados con cuidado de los estandares
		Realizar evaluaciones a sus proveedores de software	4	4	4	4	Super relevante Muchos de los ataques son a los proveedores
Implementación	Revisión de código	Realizar una revisión de código oportuna y obligatoria	4	4	4	4	La tendencia es Shift-Left
		Utilizar herramientas automatizadas	4	4	4	4	Super importante. El dia de hoy no es posible hacerlo a mano

	Pruebas de seguridad de software	Integrar herramientas de seguridad de caja opaca en el proceso de QA	4	4	4	4	
Operaciones	Entorno de software	Usar monitoreo de entrada de la aplicación	4	4	4	4	
		Definir configuraciones y parámetros de implementación seguros	4	4	4	4	

¿Hay alguna dimensión que hace parte del constructo y no fue evaluada? ¿Cuál?

Todas las dimensiones que se me propusieron fueron evaluadas

*Para los casos de equivalencia semántica se deja una casilla por ítem, ya que se evaluará si la traducción o el cambio en vocabulario son suficientes.

RESULTADOS

Opinión: Marque con un aspa (X) en alguna de las casillas correspondientes.

X	FAVORABLE		DEBE MEJORAR		DESFAVORABLE

FIRMA DEL EXPERTO

Experto 2

INSTRUMENTO DE EVALUACIÓN PARA LA VALIDACIÓN DE CONTENIDO MEDIANTE JUICIO DE EXPERTOS

Respetado juez: Juliana del Pilar Alva Zapata., Usted ha sido seleccionado para evaluar la metodología propuesta de la investigación: "METODOLOGÍA DE SEGURIDAD DE SOFTWARE PARA MEJORAR LA SEGURIDAD DEL SOFTWARE COMO SERVICIO EN EMPRESAS DE DESARROLLO". Agradecemos su valiosa colaboración.

FORMACIÓN ACADÉMICA: Ingeniería en Computación e Informática.
AREAS DE EXPERIENCIA PROFESIONAL: Area de Sistemas, Area de Riesgos, auditoría, seguridad y cumplimiento.
TIEMPO DE EXPERIENCIA: 14 años
CARGO ACTUAL: Docente universitario.
ORGANIZACIÓN: Universidad Tecnológica del Perú.
OBJETIVO DE LA INVESTIGACIÓN: Crear una metodología para mejorar la seguridad del software como servicio
OBJETIVO DEL JUICIO DE EXPERTOS: Lograr una validación de contenido

De acuerdo con los siguientes indicadores califique cada uno de los ítems según corresponda.

CATEGORÍA	CALIFICACIÓN	INDICADOR
SUFICIENCIA	1. No cumple con el criterio	Los ítems no son suficientes para medir la dimensión.
Los ítems que pertenecen a una misma dimensión bastan para obtener la medición de esta.	2. Bajo nivel	Los ítems miden algún aspecto de la dimensión, pero no corresponden con la dimensión total.
	3. Moderado nivel	Se deben incrementar algunos ítems para poder evaluar la dimensión completamente.
	4. Alto nivel	Los ítems son suficientes.
CLARIDAD	1. No cumple con el criterio	El ítem no es claro.
	2. Bajo nivel	El ítem requiere bastantes modificaciones o una modificación muy grande en el uso de las palabras de acuerdo con su significado o por la ordenación de las mismas.
El ítem se comprende fácilmente, es decir, su sintáctica y semántica son adecuadas.	3. Moderado nivel	Se requiere una modificación muy específica de algunos de los términos del ítem.
	4. Alto nivel	El ítem es claro, tiene semántica y sintaxis adecuada.
COHERENCIA	1. No cumple con el criterio	El ítem no tiene relación lógica con la dimensión.
El ítem tiene relación lógica con la dimensión o indicador que está midiendo.	2. Bajo nivel	El ítem tiene una relación tangencial con la dimensión.
	3. Moderado nivel	El ítem tiene una relación moderada con la dimensión que está midiendo.
	4. Alto nivel	El ítem se encuentra completamente relacionado con la dimensión que está midiendo.
RELEVANCIA	1. No cumple con el criterio	El ítem puede ser eliminado sin que se vea afectada la medición de la dimensión.
El ítem es esencial o importante, es decir, debe ser incluido.	2. Bajo nivel	El ítem tiene alguna relevancia, pero otro ítem puede estar incluyendo lo que mide este.
	3. Moderado nivel	El ítem es relativamente importante.
	4. Alto nivel	El ítem es muy relevante y debe ser incluido.

DIMENSIÓN	PRÁCTICA	ITEM	SUFICIENCIA*	CLARIDAD	COHERENCIA	RELEVANCIA	OBSERVACIONES
Gobierno	Estrategia y métricas	Definir apetito, tolerancia y capacidad del riesgo	4	4	4	4	
		Realizar perfil de riesgo de la aplicación	4	4	4	4	
		Definir la estrategia	4	4	4	4	
		Establecer el proceso de seguridad de software	4	4	4	4	
		Definir y utilizar el proceso análisis de la arquitectura	4	4	4	4	
		Establecer el proceso de construcción	4	4	4	4	
		Identificar objetivos de seguridad de software	4	4	4	4	
		Determinar presupuestos	4	4	4	4	
		Asignar roles y responsabilidades	4	4	4	4	
		Establecer métricas y KPIs estratégicos	4	4	4	4	
		Establecer estándares de seguridad	4	4	4	4	
	Entrenamiento	Realizar capacitaciones o eventos	4	4	4	4	

		Identificar campeones de seguridad	4	4	4	4	
Diseño	Análisis de la arquitectura	Realizar una revisión de las características de seguridad	4	4	4	4	
		Utilizar lista de verificación (checklist) de principios de seguridad	4	4	4	4	
	Modelos de ataque	Recopilar y utilizar inteligencia de ataque	4	4	4	4	
		Mantener y utilizar una lista de los N principales ataques posibles	4	4	4	4	
		Recopilar y publicar historias de ataques	4	4	4	4	
		Realizar modelado de amenazas	4	4	4	4	
		Identificar requisitos de seguridad	4	4	4	4	
		Realizar evaluaciones a sus proveedores de software	4	4	4	4	
Implementación	Revisión de código	Realizar una revisión de código oportuna y obligatoria	4	4	4	4	
		Utilizar herramientas automatizadas	4	4	4	4	

	Pruebas de seguridad de software	Integrar herramientas de seguridad de caja opaca en el proceso de QA	4	4	4	4	
Operaciones	Entorno de software	Usar monitoreo de entrada de la aplicación	4	4	4	4	
		Definir configuraciones y parámetros de implementación seguros	4	4	4	4	

¿Hay alguna dimensión que hace parte del constructo y no fue evaluada? ¿Cuál?

Ninguna

*Para los casos de equivalencia semántica se deja una casilla por ítem, ya que se evaluará si la traducción o el cambio en vocabulario son suficientes.

RESULTADOS

Opinión: Marque con un aspa (X) en alguna de las casillas correspondientes.

X	FAVORABLE		DEBE MEJORAR		DESFAVORABLE

Juliana Díaz

FIRMA DEL EXPERTO

Experto 3

INSTRUMENTO DE EVALUACIÓN PARA LA VALIDACIÓN DE CONTENIDO MEDIANTE JUICIO DE EXPERTOS

Respetado juez: Franklin Ebenezer Toro Santa Cruz, Usted ha sido seleccionado para evaluar la metodología propuesta de la investigación: "METODOLOGÍA DE SEGURIDAD DE SOFTWARE PARA MEJORAR LA SEGURIDAD DEL SOFTWARE COMO SERVICIO EN EMPRESAS DE DESARROLLO". Agradecemos su valiosa colaboración.

FORMACIÓN ACADÉMICA: Mg en Ingeniería de Sistemas en Gerencia de TI y Gest. Software
AREAS DE EXPERIENCIA PROFESIONAL: Desarrollo de software
Liderazgo de Proyectos Docente Universitario
TIEMPO DE EXPERIENCIA: 22 años
CARGO ACTUAL: Catedrático universitario
ORGANIZACIÓN: Universidad Nacional Pedro Ruiz Gallo
OBJETIVO DE LA INVESTIGACIÓN: Crear una metodología para mejorar la seguridad del software como servicio
OBJETIVO DEL JUICIO DE EXPERTOS: Lograr una validación de contenido

De acuerdo con los siguientes indicadores califique cada uno de los ítems según corresponda.

CATEGORÍA	CALIFICACIÓN	INDICADOR
SUFICIENCIA	1. No cumple con el criterio	Los ítems no son suficientes para medir la dimensión.
Los ítems que pertenecen a una misma dimensión bastan para obtener la medición de esta.	2. Bajo nivel	Los ítems miden algún aspecto de la dimensión, pero no corresponden con la dimensión total.
	3. Moderado nivel	Se deben incrementar algunos ítems para poder evaluar la dimensión completamente.
	4. Alto nivel	Los ítems son suficientes.
CLARIDAD	1. No cumple con el criterio	El ítem no es claro.
	2. Bajo nivel	El ítem requiere bastantes modificaciones o una modificación muy grande en el uso de las palabras de acuerdo con su significado o por la ordenación de las mismas.
El ítem se comprende fácilmente, es decir, su sintáctica y semántica son adecuadas.	3. Moderado nivel	Se requiere una modificación muy específica de algunos de los términos del ítem.
	4. Alto nivel	El ítem es claro, tiene semántica y sintaxis adecuada.
COHERENCIA	1. No cumple con el criterio	El ítem no tiene relación lógica con la dimensión.
El ítem tiene relación lógica con la dimensión o indicador que está midiendo.	2. Bajo nivel	El ítem tiene una relación tangencial con la dimensión.
	3. Moderado nivel	El ítem tiene una relación moderada con la dimensión que está midiendo.
	4. Alto nivel	El ítem se encuentra completamente relacionado con la dimensión que está midiendo.
RELEVANCIA	1. No cumple con el criterio	El ítem puede ser eliminado sin que se vea afectada la medición de la dimensión.
El ítem es esencial o importante, es decir, debe ser incluido.	2. Bajo nivel	El ítem tiene alguna relevancia, pero otro ítem puede estar incluyendo lo que mide este.
	3. Moderado nivel	El ítem es relativamente importante.
	4. Alto nivel	El ítem es muy relevante y debe ser incluido.

DIMENSIÓN	PRÁCTICA	ITEM	SUFICIENCIA*	CLARIDAD	COHERENCIA	RELEVANCIA	OBSERVACIONES
Gobierno	Estrategia y métricas	Definir apetito, tolerancia y capacidad del riesgo	4	4	4	4	
		Realizar perfil de riesgo de la aplicación	4	4	4	4	
		Definir la estrategia	4	4	4	4	
		Establecer el proceso de seguridad de software	4	3	4	4	
		Definir y utilizar el proceso análisis de la arquitectura	4	4	4	4	
		Establecer el proceso de construcción	4	3	4	4	
		Identificar objetivos de seguridad de software	4	4	4	4	
		Determinar presupuestos	4	4	4	4	
		Asignar roles y responsabilidades	4	4	4	4	
		Establecer métricas y KPIs estratégicos	4	4	4	4	
		Establecer estándares de seguridad	4	4	3	4	
	Entrenamiento	Realizar capacitaciones o eventos	4	4	4	4	

		Identificar campeones de seguridad	4	4	4	4	
Diseño	Análisis de la arquitectura	Realizar una revisión de las características de seguridad	4	3	4	4	
		Utilizar lista de verificación (checklist) de principios de seguridad	3	4	4	4	
	Modelos de ataque	Recopilar y utilizar inteligencia de ataque	3	4	4	4	
		Mantener y utilizar una lista de los N principales ataques posibles	3	4	4	4	
		Recopilar y publicar historias de ataques	4	4	3	4	
		Realizar modelado de amenazas	4	4	4	4	
		Identificar requisitos de seguridad	4	4	4	4	
		Realizar evaluaciones a sus proveedores de software	4	4	4	4	
Implementación	Revisión de código	Realizar una revisión de código oportuna y obligatoria	3	4	4	4	
		Utilizar herramientas automatizadas	3	4	4	4	

	Pruebas de seguridad de software	Integrar herramientas de seguridad de caja opaca en el proceso de QA	3	4	4	4	
Operaciones	Entorno de software	Usar monitoreo de entrada de la aplicación	4	4	4	4	
		Definir configuraciones y parámetros de implementación seguros	4	4	4	4	

¿Hay alguna dimensión que hace parte del constructo y no fue evaluada? ¿Cuál?

_____

_____

*Para los casos de equivalencia semántica se deja una casilla por ítem, ya que se evaluará si la traducción o el cambio en vocabulario son suficientes.

RESULTADOS

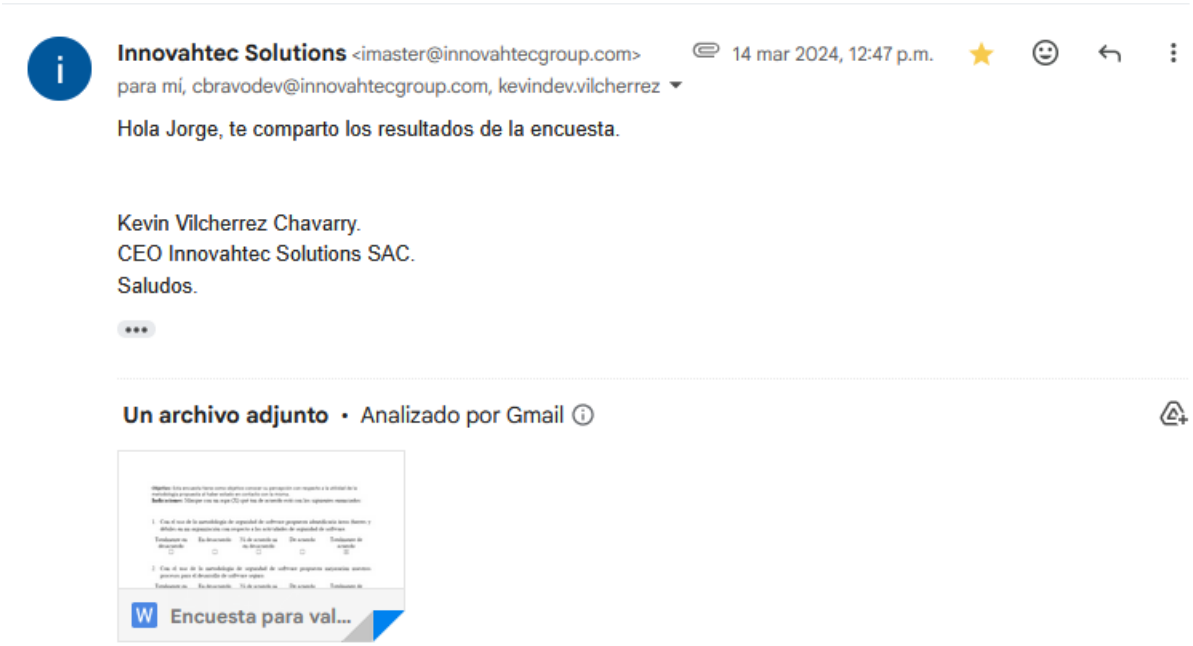
Opinión: Marque con un aspa (X) en alguna de las casillas correspondientes.

X	FAVORABLE		DEBE MEJORAR		DESFAVORABLE

FIRMA DEL EXPERTO

ANEXO 7 – Resultados de la encuesta para validar la utilidad de la metodología propuesta

Gerente de TI



Objetivo: Esta encuesta tiene como objetivo conocer su percepción con respecto a la utilidad de la metodología propuesta al haber estado en contacto con la misma.
Indicaciones: Marque con un aspa (X) qué tan de acuerdo está con los siguientes enunciados:

1. Con el uso de la metodología de seguridad de software propuesta identificaría áreas fuertes y débiles en mi organización con respecto a las actividades de seguridad de software.

Totalmente en desacuerdo	En desacuerdo	Ni de acuerdo ni en desacuerdo	De acuerdo	Totalmente de acuerdo
☐	☐	☐	☐	☒

2. Con el uso de la metodología de seguridad de software propuesta mejorarían nuestros procesos para el desarrollo de software seguro.

Totalmente en desacuerdo	En desacuerdo	Ni de acuerdo ni en desacuerdo	De acuerdo	Totalmente de acuerdo
☐	☐	☐	☐	☒

3. Si la metodología de seguridad de software propuesta estuviera disponible para mi trabajo, predigo que lo usaría regularmente en el futuro.

Totalmente en desacuerdo	En desacuerdo	Ni de acuerdo ni en desacuerdo	De acuerdo	Totalmente de acuerdo
☐	☐	☐	☐	☒

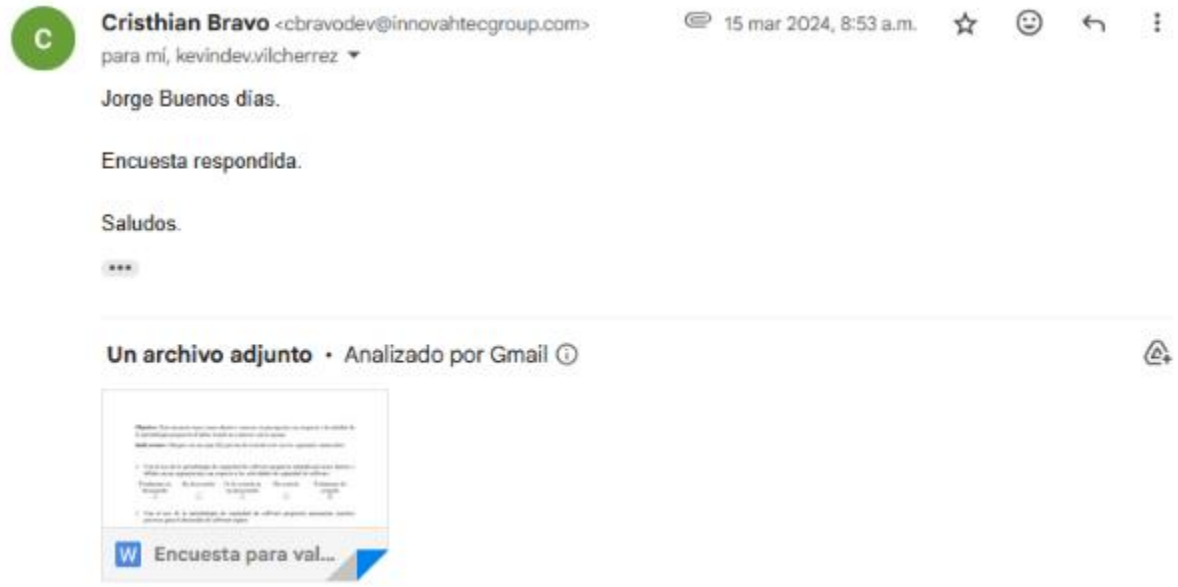
4. Estoy de acuerdo con los problemas de garantía de seguridad identificados por la metodología de seguridad de software propuesta.

Totalmente en desacuerdo	En desacuerdo	Ni de acuerdo ni en desacuerdo	De acuerdo	Totalmente de acuerdo
☐	☐	☐	☐	☒

5. La metodología de seguridad de software propuesta es una herramienta útil de garantía de seguridad de software para mi organización de desarrollo de software.

Totalmente en desacuerdo	En desacuerdo	Ni de acuerdo ni en desacuerdo	De acuerdo	Totalmente de acuerdo
☐	☐	☐	☐	☒

Gerente de proyectos de desarrollo



Objetivo: Esta encuesta tiene como objetivo conocer su percepción con respecto a la utilidad de la metodología propuesta al haber estado en contacto con la misma.

Indicaciones: Marque con un aspa (X) qué tan de acuerdo está con los siguientes enunciados:

1. Con el uso de la metodología de seguridad de software propuesta identificaría áreas fuertes y débiles en mi organización con respecto a las actividades de seguridad de software.

Totalmente en desacuerdo	En desacuerdo	Ni de acuerdo ni en desacuerdo	De acuerdo	Totalmente de acuerdo
☐	☐	☐	☐	☒

2. Con el uso de la metodología de seguridad de software propuesta mejorarían nuestros procesos para el desarrollo de software seguro.

Totalmente en desacuerdo	En desacuerdo	Ni de acuerdo ni en desacuerdo	De acuerdo	Totalmente de acuerdo
☐	☐	☐	☐	☒

3. Si la metodología de seguridad de software propuesta estuviera disponible para mi trabajo, predigo que lo usaría regularmente en el futuro.

Totalmente en desacuerdo	En desacuerdo	Ni de acuerdo ni en desacuerdo	De acuerdo	Totalmente de acuerdo
☐	☐	☐	☐	☒

4. Estoy de acuerdo con los problemas de garantía de seguridad identificados por la metodología de seguridad de software propuesta.

Totalmente en desacuerdo	En desacuerdo	Ni de acuerdo ni en desacuerdo	De acuerdo	Totalmente de acuerdo
☐	☐	☐	☐	☒

5. La metodología de seguridad de software propuesta es una herramienta útil de garantía de seguridad de software para mi organización de desarrollo de software.

Totalmente en desacuerdo	En desacuerdo	Ni de acuerdo ni en desacuerdo	De acuerdo	Totalmente de acuerdo
☐	☐	☐	☐	☒