CARRERA DE DESARROLLO DE SOFTWARE

TEMA:

“IMPLEMENTACIÓN DE METODOLOGIAS DE SEGURIDAD PARA EL DESARROLLO SEGURO DE LA APLICACIÓN WEB ANI DEL INSTITUTO SUDAMERICANO”

AUTOR:
MIGUEL ANGEL CRIOLLO VASQUEZ
KEVIN ISMAEL AVILA CAMPOS

TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE:
TECNÓLOGO EN DESARROLLO DE SOFTWARE

TUTORES:
• PROF. JONNATHAN D. VALLEJO

CUENCA – ECUADOR, 2025

DECLARACIÓN DE AUTORÍA DEL TRABAJO

Yo, **AVILA CAMPOS KEVIN ISMAEL**, estudiante del **Instituto Tecnológico Superior Particular Sudamericano** de la ciudad de Cuenca - Ecuador, que cursó la Tecnología en **Desarrollo de Software,** declaro en forma libre y voluntaria que la presente investigación que versa sobre **"IMPLEMENTACIÓN DE METODOLOGÍAS DE SEGURIDAD PARA EL DESARROLLO SEGURO DE LA APLICACIÓN WEB ANI DEL INSTITUTO SUDAMERICANO"** así como las expresiones vertidas en la misma, son autoría de la compareciente, quien ha realizado en base a recopilación bibliográfica, consultas de internet y consultas de campo.

En consecuencia, asumo la responsabilidad de la originalidad de la misma y el cuidado al remitirme a las fuentes bibliográficas respectivas para fundamentar el contenido expuesto.

Atentamente,

AVILA CAMPOS KEVIN ISMAEL

Cédula: 010570622-0

DECLARACIÓN DE AUTORÍA DEL TRABAJO

Yo, **MIGUEL ANGEL CRIOLLO VASQUEZ**, estudiante del **Instituto Tecnológico Superior Particular Sudamericano** de la ciudad de Cuenca - Ecuador, que cursó la Tecnología en **Desarrollo de Software,** declaro en forma libre y voluntaria que la presente investigación que versa sobre **"IMPLEMENTACIÓN DE METODOLOGÍAS DE SEGURIDAD PARA EL DESARROLLO SEGURO DE LA APLICACIÓN WEB ANI DEL INSTITUTO SUDAMERICANO"** así como las expresiones vertidas en la misma, son autoría de la compareciente, quien ha realizado en base a recopilación bibliográfica, consultas de internet y consultas de campo.

En consecuencia, asumo la responsabilidad de la originalidad de la misma y el cuidado al remitirme a las fuentes bibliográficas respectivas para fundamentar el contenido expuesto.

Atentamente,

MIGUEL ANGEL CRIOLLO VASQUEZ

Cédula: 010556544-4

DERECHOS DE AUTOR

INSTITUTO DE TECNOLOGIAS SUDAMERICANO
GENUINA EDUCACION SUPERIOR
ESTABLECIDO EN 1995

CARRERA DE DESARROLLO DE SOFTWARE

CERTIFICACIÓN DEL TUTOR

Aprobación del Trabajo de Titulación

Doy fe que el trabajo desarrollado por los estudiantes: AVILA CAMPOS KEVIN ISMAEL, CRIOLLO VASQUEZ MIGUEL ANGEL, con el título **"IMPLEMENTACIÓN DE METODOLOGÍAS DE SEGURIDAD PARA EL DESARROLLO SEGURO DE LA APLICACIÓN WEB ANI DEL INSTITUTO SUDAMERICANO",** cumple con los requisitos y méritos suficientes para ser sometido a la presentación pública y evaluación por parte del jurado examinador que se designe.

Atentamente,

Tnlgo. Jonnathan D. Vallejo

C.I: 0107213092

IMPLEMENTACIÓN DE METODOLOGÍAS DE SEGURIDAD PARA EL DESARROLLO SEGURO DE LA APLICACIÓN WEB ANI DEL INSTITUTO SUDAMERICANO

Trabajo presentado para optar al título de Tecnólogo Superior en Desarrollo de Software

Proyecto de grado presentado por: Kevin Ismael Ávila Campos - Miguel Angel Criollo.

Carrera: Desarrollo de Software

Tutor académico: Ing. Jonnathan D. Vallejo.

Cuenca, 25 de febrero de 2025

ÍNDICE

ÍNDICE DE TABLAS

ÍNDICE DE ILUSTRACIONES

ACRÓNIMOS

SAMM *SOFTWARE ASSURANCE MATURITY MODEL*

T-PS *THIRD-PARTY SOFTWARE*

APPSEC *APPLICATION SECURITY*

SDLC *SOFTWARE DEVELOPMENT LIFE CYCLE*

SSDLC *SAFE SOFTWARE DEVELOPMENT LIFE CYCLE*

OWASP *OPEN WEB APPLICACTION SECURITY PROJECT*

SAST *STATIC APPLICATION SCANNING TESTING*

DAST *DYNAMIC APPLICATION SCANNING TESTING*

IAST *INTERACTIVE APPLICATION SECURITY TESTING*

DEVSECOPS *DEVELOPMENT, SECURTY AND OPERATIONS*

DEVOPS *DEVELOPMENT AND OPERATIONS*

BSIMM *BUILDING SECURITY IN MATURITY MODEL*

XSS *CROSS-SITE SCRIPTING*

AppSec *APPLICATION SECURITY*

OASV *OWASP APLICACTION SECURITY VERIFICATION*

BSIMM *BUILDING SECURITY IN MATURITY MODEL*

CID *CONTINUOUS INTEGRATION AND DEPLOYMENT*

Threat Assessment *THREAT ASSESSMENT*

CI *CONTINUOUS INTEGRATION*

SCA *SOFTWARE COMPOSITION ANALYSIS*

CD *CONTINUOUS DEPLOYMENT*

ZAP *ZED ATTACK PROXY*

MTMT *MICROSOFT TREAHT MODELING TOOL*

MobSF *Mobile Security Framework*

STRIDE *Spoofing, Tampering, Repudiation, Information disclosure, Denial of service and Elevation of privilege*

WB-WEB *AniWeb*

AP-API *AniApi*

WS-WEBSOCKET *AniWebsocket*

MV-MOVIL *AniMovile*

TC-TRANSCRIPTION *AniTranscription*

PV-PROVIDER *AniProvider*

Glosario

API Interfaz de Programación de Aplicaciones (Application Programming Interface, API); conjunto de reglas y protocolos que permiten la comunicación entre diferentes sistemas de software, facilitando la integración y el intercambio de datos..

AppSec Seguridad de aplicaciones; conjunto de prácticas, herramientas y metodologías diseñadas para proteger las aplicaciones de amenazas y vulnerabilidades a lo largo de su ciclo de vida..

Ataque de Fuerza Bruta Ataque de Fuerza Bruta (Brute Force Attack); técnica de ataque en la que un atacante prueba sistemáticamente todas las combinaciones posibles de credenciales hasta encontrar la correcta, explotando la falta de restricciones en los intentos de autenticación..

Autenticación Proceso de verificar la identidad de un usuario, sistema o entidad, generalmente mediante credenciales como contraseñas, tokens o biometría.

CI/CD Práctica de desarrollo de software que combina la **Integración Continua (CI, Continuous Integration)** y la **Entrega/Despliegue Continuo (CD, Continuous Delivery/Deployment)**. CI se enfoca en automatizar la integración de cambios en el código en un repositorio compartido, ejecutando pruebas y verificaciones automáticamente. CD extiende este proceso automatizando la entrega del software a entornos de producción o preproducción, asegurando despliegues frecuentes y confiables. Herramientas populares incluyen Jenkins, GitHub Actions, GitLab CI/CD, y TeamCity..

CIA Confidentiality, Integrity, Availability; principios fundamentales de la seguridad de la información que garantizan la confidencialidad, integridad y disponibilidad de los datos.

DAST Dynamic Application Security Testing; técnica de prueba de seguridad que analiza aplicaciones en tiempo de ejecución para identificar vulnerabilidades.

Framework de Seguridad Conjunto estructurado de políticas, procedimientos, guías y herramientas diseñadas para mejorar la seguridad de sistemas y aplicaciones. Los frameworks de seguridad proporcionan un enfoque sistemático para gestionar riesgos, implementar controles

y cumplir con normativas de seguridad. Ejemplos incluyen **OWASP SAMM**, **NIST Cybersecurity Framework**, **ISO/IEC 27001**, **CIS Controls** y **Zero Trust Architecture**..

Fuzz Testing Técnica de prueba de seguridad que inyecta datos aleatorios o malformados en una aplicación para detectar vulnerabilidades o comportamientos inesperados.

Fuzzing El fuzzing es una técnica de pruebas de seguridad que consiste en enviar datos inesperados, aleatorios o maliciosos a una aplicación para detectar vulnerabilidades como inyecciones, fallos de validación o errores en la gestión de excepciones..

GitHub Plataforma de desarrollo colaborativo basada en Git que permite a los desarrolladores gestionar código fuente, colaborar en proyectos de software, realizar control de versiones y automatizar flujos de trabajo con CI/CD.

JSON Web Token (JWT) Estándar abierto (`RFC 7519`) que define un formato compacto y seguro para la transmisión de información entre partes en formato JSON. Se utiliza principalmente en autenticación y autorización, permitiendo la verificación de identidad sin necesidad de almacenar estado en el servidor. Un JWT consta de tres partes; **Header** (encabezado), **Payload** (carga útil) y **Signature** (firma)..

Least Privilege Principle Principio de Mínimos Privilegios; Concepto de seguridad que establece que un usuario, proceso o sistema solo debe tener los permisos mínimos necesarios para realizar sus tareas, reduciendo así el riesgo de abuso o acceso no autorizado.

Ley Orgánica de Protección de Datos Personales Legislación que regula el tratamiento y protección de datos personales para garantizar la privacidad y seguridad de los individuos.

Logs Registros de eventos del sistema; archivos o bases de datos que almacenan información sobre las acciones realizadas por los usuarios, el sistema y las aplicaciones, incluyendo cuándo y dónde ocurrieron, con el objetivo de auditoría, monitoreo y detección de incidentes de seguridad.

MASVS Mobile Application Security Verification Standard o MASVS es un estándar que establece criterios de seguridad específicos para aplicaciones móviles, ayudando a desarrolladores y evaluadores a mejorar su protección..

MFA Autenticación Multifactor (Multi-Factor Authentication, MFA); método de verificación de identidad que requiere dos o más factores de autenticación independientes, como contraseña, biometría y un código enviado por SMS o aplicación de autenticación, para mejorar la seguridad..

Microsoft SDL El Microsoft Security Development Lifecycle (SDL) es un marco de seguridad creado por Microsoft para integrar buenas prácticas de seguridad en cada fase del desarrollo de software..

Microsoft Threat Modeling Tool (MTMT) Herramienta de Microsoft para facilitar el modelado de amenazas en aplicaciones y sistemas.

Moodle Plataforma de aprendizaje en línea (Learning Management System, LMS); software de código abierto utilizado para crear cursos virtuales, gestionar contenido educativo y facilitar la comunicación entre estudiantes y profesores..

OASV Open Application Security Verification; estándar para verificar la seguridad de aplicaciones web y móviles.

OWASP API Security Top 10 Lista de las 10 principales vulnerabilidades de seguridad en APIs, identificadas por OWASP. Se enfoca en los riesgos específicos que afectan a las interfaces de programación de aplicaciones (APIs), como la exposición excesiva de datos, fallos en la autenticación y autorización, abuso de recursos y falta de restricciones en la velocidad de solicitudes..

OWASP Mobile Security Top 10 Lista de las 10 principales vulnerabilidades de seguridad en aplicaciones móviles, publicada por OWASP. Identifica los riesgos más críticos que afectan a las

apps móviles en plataformas como Android e iOS, incluyendo almacenamiento inseguro, autenticación inadecuada, comunicación insegura y problemas de criptografía..

OWASP Top 10 Lista de las 10 principales vulnerabilidades de seguridad en aplicaciones web identificadas por la Open Web Application Security Project (OWASP). Se actualiza periódicamente con base en las tendencias de seguridad y ataques más frecuentes. Las vulnerabilidades incluyen problemas como inyección, autenticación rota, exposición de datos sensibles y fallos en la seguridad del diseño..

OWASP Web Security top 10 El OWASP Web Security Top 10 es una lista publicada por la Open Web Application Security Project (OWASP) que identifica y clasifica las 10 vulnerabilidades de seguridad más críticas en aplicaciones web, sirviendo como referencia para desarrolladores, equipos de seguridad y organizaciones para identificar y mitigar riesgos comunes en aplicaciones web..

OWASP ZAP Zed Attack Proxy; herramienta de seguridad de código abierto desarrollada por OWASP para encontrar vulnerabilidades en aplicaciones web.

OWASP Open Web Application Security Project; organización sin fines de lucro dedicada a mejorar la seguridad del software a través de recursos y herramientas de código abierto.

OpenAI Organización de investigación en inteligencia artificial que desarrolla modelos avanzados de IA, como ChatGPT, DALL·E y Whisper, con el objetivo de crear inteligencia artificial segura y beneficiosa para la humanidad.

RBAC Role-Based Access Control; modelo de control de acceso que asigna permisos a usuarios en función de roles predefinidos, facilitando la gestión de privilegios.

RESTful API API RESTful (RESTful API); interfaz de programación de aplicaciones que sigue los principios de la arquitectura REST (Representational State Transfer), permitiendo la comunicación entre sistemas a través de peticiones HTTP utilizando métodos estándar como GET, POST, PUT y DELETE..

SAMM Software Assurance Maturity Model; marco de trabajo para evaluar y mejorar la madurez de los procesos de seguridad en el desarrollo de software.

SAST Static Application Security Testing; técnica de prueba de seguridad que analiza el código fuente de una aplicación para identificar vulnerabilidades sin ejecutarla.

SDLC Ciclo de vida de desarrollo de software; conjunto de fases necesarias para desarrollar software de calidad.

SQL Injection Inyección SQL; vulnerabilidad que permite a un atacante manipular consultas SQL en una base de datos a través de entradas no validadas en una aplicación, pudiendo acceder, modificar o eliminar datos.

SSDLC Secure Software Development Life Cycle; es un enfoque sistemático para integrar prácticas de seguridad en cada fase del ciclo de vida del desarrollo de software (SDLC). SSDLC abarca desde la planificación, diseño, desarrollo, pruebas y despliegue hasta el mantenimiento del software, asegurando que se identifiquen y mitiguen los riesgos de seguridad en todas las etapas. Esto incluye prácticas como análisis de amenazas, pruebas de penetración, revisión de código, análisis estático y dinámico, y la aplicación de principios de seguridad como el principio de menor privilegio y el de defensa en profundidad. SSDLC ayuda a crear software más seguro y confiable, minimizando vulnerabilidades y mejorando la protección de los datos..

STRIDE Modelo de clasificación de amenazas que incluye Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service y Elevation of Privilege.

Scrum Marco de trabajo ágil para la gestión y desarrollo de proyectos, especialmente en software. Scrum se basa en ciclos de desarrollo iterativos e incrementales llamados *sprints*, en los que se priorizan y entregan funcionalidades de manera continua. Incluye roles clave como el **Scrum Master**, el **Product Owner** y el **Development Team**, así como eventos como la *Daily Stand-up*, la *Sprint Review* y la *Retrospective*..

Security Champion Persona dentro de un equipo de desarrollo que actúa como enlace entre el equipo y los especialistas en seguridad, promoviendo mejores prácticas y facilitando la adopción de medidas de seguridad dentro del ciclo de desarrollo de software. En OWASP SAMM v2, el rol del Security Champion es clave para integrar la seguridad en las actividades diarias del equipo..

Sistema Integrado ANI Conjunto de aplicaciones y servicios desarrollados en el Instituto Tecnológico Superior Sudamericano sin fines de lucro que buscan resolver una problemática, siendo esta los debates realizados en un aula estudiantil de Universidades, ofreciendo control y gestión de estudiantes y generando grafos sobre los temas discutidos entre estudiantes, brindando mayor aserción sobre los temas discutidos..

Stakeholder Individuo o grupo que tiene un interés en un proyecto, producto o sistema, y que puede verse afectado por su desarrollo, implementación o resultados. En el contexto de desarrollo de software, los **stakeholders** pueden incluir clientes, usuarios finales, desarrolladores, testers, gerentes de proyecto, inversionistas y reguladores, entre otros..

Threat Assessment Evaluación de amenazas; proceso de identificar y analizar posibles amenazas a un sistema o aplicación.

Threat Modeling Modelado de amenazas; proceso sistemático para identificar, priorizar y mitigar riesgos de seguridad en una aplicación.

Whisper Modelo de reconocimiento automático de voz (ASR) desarrollado por OpenAI. Whisper es un sistema de transcripción y traducción multilingüe basado en inteligencia artificial, capaz de convertir audio en texto con alta precisión y soportando múltiples idiomas y acentos. Se utiliza en aplicaciones de subtitulado, asistentes de voz y accesibilidad..

XSS Cross-Site Scripting; vulnerabilidad de seguridad que permite a un atacante inyectar scripts maliciosos en páginas web vistas por otros usuarios, robando información o manipulando contenido.

ASVS Application Security Verification Standard o por sus siglas ASVS es un estándar que define niveles de verificación de seguridad para aplicaciones, proporcionando requisitos claros para evaluar su seguridad..

RESUMEN

La seguridad en el desarrollo de aplicaciones web se ha convertido en un desafío crítico en un entorno tecnológico caracterizado por el incremento de ciberataques y amenazas avanzadas. En este contexto, el *Sistema Integrado ANI* del Instituto Sudamericano enfrenta vulnerabilidades que podrían comprometer la confidencialidad, integridad y disponibilidad de los datos, tanto de los usuarios como la reputación de la institución. La falta de un enfoque estructurado en *AppSec* ha dejado expuesta la aplicación a riesgos. Esta situación destaca la necesidad de adoptar estrategias preventivas y correctivas para garantizar la protección de la información. Este estudio propone el diseño e implementación de un modelo de seguridad basado en el *SSDLC*), siendo este la integración del framework de seguridad *SAMM*, junto con herramientas automatizadas de análisis estático *SAST*, dinámico *DAST* y creación del modelo de amenazas *Threat Modeling*. Para su aplicación, se desarrollaron y formalizaron todas las políticas, procedimientos y documentos necesarios para la adopción de *SAMM*, garantizando un enfoque estructurado para abordar las vulnerabilidades desde las primeras etapas del desarrollo hasta el despliegue y mantenimiento. La investigación busca fortalecer la resiliencia del sistema frente a amenazas e incidentes, designar pautas y obligaciones para el rol de *Security Champion*, encargado de realizar el análisis de amenazas y establecer un marco de seguridad escalable y adaptable el contexto del equipo de desarrollo. Finalmente, el modelo propuesto ofrece una guía integral para implementar controles de seguridad efectivos, asegurando la protección de datos críticos y promoviendo una cultura organizacional centrada en la ciberseguridad.

Palabras clave: Seguridad, privacidad, amenazas, control de acceso, software, *SDLC*, *SAMM Threat Assessment*.

ABSTRACT

Web application security has become a critical challenge in a technological environment characterized by increasing cyberattacks and advanced threats. In this context, the *Sistema Integrado ANI* of the Instituto Sudamericano faces vulnerabilities that could compromise the confidentiality, integrity, and availability of data, as well as the institution's reputation. The lack of a structured approach to *AppSec* has left the application exposed to risks, highlighting the need for preventive and corrective strategies to ensure information protection. This study proposes the design and implementation of a security model based on the *SSDLC*, integrating the security framework *SAMM* along with automated static analysis (*SAST*), dynamic analysis (*DAST*), and threat modeling (*Threat Modeling*). For its application, all necessary policies, procedures, and documents for the adoption of *SAMM* were developed and formalized, ensuring a structured approach to addressing vulnerabilities from the early development stages to deployment and maintenance. The research aims to strengthen the system's resilience against threats and incidents, define guidelines and responsibilities for the *Security Champion* role—responsible for threat analysis—and establish a scalable and adaptable security framework tailored to the development team's context. Finally, the proposed model provides a comprehensive guide for implementing effective security controls, ensuring the protection of critical data, and fostering an organizational culture focused on cybersecurity.

Keywords: Security, privacy, threats, access control, software, *SDLC*, *SAMM*, *Threat Assessment*.

Dedicatoria

Este trabajo va dedicado, en primer lugar, a nuestros familiares, cuyo apoyo incondicional, amor y confianza han sido nuestras mayores fuentes de inspiración y fortaleza a lo largo de este proceso. Su motivación constante y palabras de aliento nos han impulsado a seguir adelante incluso en los momentos más difíciles.

A nuestro tutor, Jonnathan D. Vallejo, cuyo conocimiento, guía y dedicación fueron fundamentales para nuestro crecimiento académico y profesional. Su compromiso con la enseñanza y su pasión por la investigación nos han servido de modelo a seguir.

Los Autores

INTRODUCCIÓN

En la actualidad, el desarrollo de *Sistema Integrado ANI* se enfrenta numerosos desafíos en materia de seguridad derivados del aumento de ciberataques y del crecimiento exponencial de las amenazas digitales, debido a esto la protección de la información se ha convertido en una prioridad crítica para los desarrolladores, pues las vulnerabilidades en los sistemas pueden comprometer la *CIA* (Confidencialidad, Integridad y Disponibilidad) de los datos, afectando tanto a los usuarios como a la institución.

Duggineni (2023) señala que ”La falta de análisis de amenazas, pruebas de penetración y entrenamiento del equipo de desarrollo contribuye a la existencia de amenazas no detectadas, las cuales perjudican directamente a un negocio, afectando la confianza del usuario y dañando la reputación de la empresa”.

Durante el análisis de seguridad de; *Sistema Integrado ANI*, se detectaron múltiples vulnerabilidades que pueden comprometer la integridad del sistema. Entre los principales errores identificados se encuentran la falta de validación de entradas, lo que incrementa el riesgo de ataques de *SQL Injection* y ; el uso de credenciales almacenadas en texto plano, lo que expone información sensible a accesos no autorizados; y la ausencia de autenticación multifactor (*MFA*), lo que debilita los mecanismos de control de acceso. Estos problemas no solo ponen en riesgo la confidencialidad y disponibilidad de los datos, sino que también pueden afectar la reputación de la institución al exponerla a posibles brechas de seguridad y al incumplimiento de normativas sobre protección de datos. La falta de protocolos de respuesta ante incidentes agrava aún más la situación, ya que dificulta la detección y mitigación de ataques en tiempo real, aumentando la vulnerabilidad del sistema ante amenazas persistentes.

Por lo tanto el Instituto Sudamericano, a través de su *Sistema Integrado ANI*, no está exento de estos riesgos. A pesar de haber adoptado especificas buenas prácticas de desarrollo, la falta de un enfoque estructurado en *AppSec* ha incrementado la exposición a diversas amenazas como incidentes. Este escenario podría derivar en consecuencias graves, daños a la reputación institucional y el incumplimiento de normativas sobre protección de datos.

Uno de los principales factores que han contribuido a este problema es la estructura del equipo

de desarrollo, conformado por 11 miembros: ocho desarrolladores, un líder técnico y un *Stakeholder*.

- **Desarrolladores**
  - 3 desarrolladores trabajan en la versión Móvil, Web y Websocket.
  - 3 desarrolladores trabajan con los componentes de IA, API y proveedores que utilizan en la aplicación.
- **Líder de grupo**
  - 1 desarrollador es líder de grupo de los que trabajan en la aplicación
  - 1 desarrollador es líder de grupo de los que trabajan con los componentes.
- **Líder técnico, stackeholder:** Cargos ocupados por profesores encargados del club de desarrollo

Con el objetivo de evaluar sus conocimientos en desarrollo seguro y el marco *SAMM*, los resultados mostraron que el 65 % desconocía cómo prevenir inyecciones SQL, el 80 % no había utilizado herramientas SAST/DAST, y solo el 20 % tenía formación en autenticación multifactor (*MFA*). Este hallazgo puso en evidencia la necesidad de una capacitación utilizando la plataforma *Moodle*, ya que el equipo enfrenta múltiples desafíos, entre ellos, la presión para reducir los tiempos de desarrollo, la adopción de frameworks que no priorizan la seguridad y la errónea percepción de que las herramientas/dependencias utilizadas cuentan con protección integrada por defecto.

Además, se identificaron deficiencias críticas en la implementación de medidas de seguridad, como la ausencia de reportes de vulnerabilidades, políticas de acción frente a vulnerabilidades e incidentes de seguridad y documentos de cumplimiento de estándares, lo que aumenta significativamente el riesgo de explotación de fallos. Esta limitación deja expuesta a la aplicaciones a una gran cantidad de ataques que van desde la parte web como *XSS* o *SQL Injection*, hasta riesgos generados por dependencias y configuraciones.

Frente a esto, se planteo la necesidad de adoptar el framework de seguridad *SAMM* con el fin de implementar *SSDLC* dentro del ciclo de desarrollo *SDLC* actual, complementado con Herramientas de Análisis Estático *SAST* y Dinámico *DAST*, y *Threat Modeling* así como programas de capacitación continua para los desarrolladores, las cuales surgen como estrategia clave para dar a conocer vulnerabilidades existentes, debido a que el ciclo de desarrollo se encontraba en etapa de despliegue en el momento de realizar la investigación.

Problemática

El desarrollo de software seguro sigue siendo un desafío en muchas organizaciones, y el Instituto Sudamericano no es la excepción. A pesar de los avances en metodologías de seguridad, la implementación de prácticas efectivas sigue siendo limitada, lo que deja expuestos a los sistemas a múltiples vulnerabilidades. En el caso del Sistema Integrado ANI, se ha identificado que la seguridad no ha sido una prioridad desde las primeras fases del desarrollo, lo que ha resultado en la ausencia de controles adecuados para mitigar amenazas comunes.

Según Assal et al. (2018), ”la mayoría de los equipos de desarrollo encuestados reportaron que las prácticas de seguridad fueron adoptadas únicamente tras la culminación del proceso de desarrollo”. Este enfoque era el mas utilizado, ya que ayudaba en la detección y corrección de vulnerabilidades en fases avanzadas, pero esto provoco que no se tomara en cuenta estas amenazas desde el inicio del proyecto, algo con lo que nos hemos topado con el proyecto *Sistema Integrado ANI*.

Este panorama revela que los entornos de desarrollo, en su mayoría, no están concebidos para priorizar la seguridad de las aplicaciones o de paso no cuentan con suficientes empleados para implementar estas medidas de seguridad. Entre las razones más frecuentes Assal et al. (2018) destaca las siguientes:

- **Algunos empleados optan por no reportar vulnerabilidades:** esto para evitar el esfuerzo adicional que implica su corrección, lo que puede dejar expuestas áreas críticas del sistema a posibles ataques.
- **La seguridad no se considera un criterio importante:** al momento de seleccionar un fra-

mework de desarrollo, priorizan en muchos casos factores como la facilidad de uso, la rapidez de implementación o la flexibilidad, lo que puede derivar en la adopción de herramientas que carecen de controles de seguridad robustos o actualizados.

- **Los desarrolladores a asumen que la seguridad está implícitamente garantizada:**, en lugar de reconocer esto como una responsabilidad esencial que debe abordarse de manera proactiva durante todo el ciclo de desarrollo, los desarrolladores, en especial los nuevos, creen que no es necesario o simplemente desconocen las formas de proteger sus trabajos.

Esto nos permite entender que la seguridad no es un aspecto explícitamente descrito en el desarrollo de sistemas. Usualmente, el *Stakeholder* o dueños del negocio asumen que está garantizada. Esta inferencia, en su mayoría, puede deberse al desconocimiento de las metodologías planeadas o a la falta de comunicación con los desarrolladores. En el caso del *Sistema Integrado ANI*, antes de las investigaciones a profundidad se realizaron acercamientos sencillos sobre el estado de la misma, las primeras conclusiones (motivo por la cual se realizo este documento) fueron que no aplicaban seguridad en ningún momento, por lo cual es evidente que una evaluación, planificación e integración de la misma es esencial, aun mas tomando en cuenta que no va a ser utilizada en ambientes controlados sino en otras instituciones que tengan los convenios necesarios.

Según Zambrano et al. (2019), el desarrollo de aplicaciones de software en los últimos años ha permitido dinamizar el trabajo en las organizaciones, ya que ahora solo se requiere acceso a la red para ingresar a los sistemas y obtener información relevante y útil ya que de esta forma se gestiona ágil y rápidamente los datos de las entidades. Sin embargo, el uso inadecuado de los medios tecnológicos, junto con deficiencias en la implementación, fallos de seguridad y pérdidas de información, ha llevado a que las organizaciones prioricen la adopción de métodos de seguridad en sus sistemas de información.

Este enfoque ha impulsado un crecimiento acelerado en la adopción de estrategias destinadas a garantizar la confidencialidad, integridad y disponibilidad de los recursos informáticos, convirtiendo la seguridad de las aplicaciones en una prioridad fundamental. En este contexto, *Sistema Integrado ANI* al gestionar información sensible de usuarios y datos institucionales, se encuen-

tra expuesta a vulnerabilidades que podrían comprometer la seguridad de los datos de estudiantes y profesores. Estas vulnerabilidades no solo representan riesgos operacionales, sino que también pueden afectar la reputación del instituto al derivar en posibles incumplimientos de normativas de protección de datos, siendo específicos al territorio ecuatoriano esta incluye a *Ley Orgánica de Protección de Datos Personales*, la cual mediante el primer acercamiento se comunico que no se la esta tomando en consideración.

Según Bhaharin et al. (2019), el cumplimiento de políticas de seguridad es de los puntos mas importantes debido al incremento de ataques cibernéticos cada vez mas complejos, esto es debido a la complejidad que implica adaptarse a los cambios de tecnologías que avanzad rápidamente. También remarca que es necesario el entendimiento de los empleados que el cumplimiento de políticas de seguridad es vital, sugiere integrar comportamiento de seguridad como elementos humanos, los cuales serán soportados por políticas y tecnologías.

Es necesario revisar el estado de las políticas existentes (si existe alguna) y crear faltantes, es evidente que la creación de políticas no puede surgir de la nada, sino debe existir una razón de cumplimiento para describirlas, dentro de estos motivos pueden ser:

- **Por cumplimiento de requisitos de framework de seguridad:** como los recomendado por *SAMM*, estos pueden ser:
    - Gestión de dependencias
    - Gestión de datos sensibles
    - Reporte y gestión de vulnerabilidades e incidentes
- **Por cumplimiento de leyes:** como *Ley Orgánica de Protección de Datos Personales*

No tener políticas implementadas genera desconocimiento para actuar frente a problemas, por ejemplo la incapacidad de tomar decisiones cuando se encuentran amenazas de *Autenticación*, esto puede implicar la insuficiente mitigación o como sugiere Assal et al. (2018), no llegar a reportarse la misma debido a que implica esfuerzo extra para el desarrollador.

La implementación de frameworks de seguridad basados en *SSDLC* es uno de los puntos mas importantes al momento de convalidar la seguridad con un ciclo *SDLC* existente, así como en

programación es mejor usar código que ya resuelve un problema (refiriendose a librerias)que crear un el código necesario desde cero, se aplica de manera similar a *SSDLC*, siendo este un modelo de proceso, no especifica políticas, documentos, ni pasos a seguir específicos, el encargado del mismo es *Framework de Seguridad*. Según de Vicente Mohino et al. (2019), la implementación de cualquier metodología *SSDLC* no asegura al 100 % la seguridad de una aplicación, pero si la minimiza las amenazas, incrementa las posibilidades tanto en calidad como en seguridad y añade valor de beneficio al software.

Otros beneficios de la implementación de *SSDLC* que describe de Vicente Mohino et al. (2019) explica que las ventajas de ser aplicado desde el inicio del desarrollo son: menor riesgo a la organización, reducción de costos por detección temprana, y software mas confiable y robusto, estos puntos aportan grandes características al software desarrollado y siendo el caso actual, mejoraría en gran medida la seguridad sin perjudicar otros aspectos como tiempo y recursos.

El desarrollo de sistemas de aplicaciones fuera de entornos locales y controlados debe implementar pautas de seguridad, estas pueden ser mediante aplicación de estándares, políticas, análisis amenazas, entre otros. Crear medidas correctas para cada uno es una tarea extensa y difícil de definir sin ayuda de expertos, por lo tanto la adopción de un framework de desarrollo basado en *SSDLC* es esencial en organizaciones que no brindan presupuesto suficiente o necesitan altos niveles de madurez para crear tales normas.

Tomando en cuenta que el *Sistema Integrado ANI* usa *SDLC*, según de Vicente Mohino et al. (2019), la naturaleza principal del mismo indica que las iteraciones al ser entregadas lo mas rápido posible, no toman en consideración la seguridad ni sus actividades como verificación o análisis de amenazas, por lo cual la necesidad de implementación de un modelo *SSDLC* es sumamente necesaria.

Otro punto a considerar es el ambiente donde toma lugar el desarrollo del sistema, al ser un instituto tecnológico, los desarrolladores se conforman unicamente por estudiantes que aun no concluyen el proceso educativo que corresponde a la carrera de desarrollo de software, los mismos no tienen experiencia con lo que respecta a seguridad, por lo cual es necesario medir sus niveles de conocimientos y determinar cursos de capacitación depende del modelo *SSDLC* escogido.

En conclusión, conociendo que se usa *SDLC*, los desarrolladores no tienen mucha experiencia por ser estudiantes de la institución educativa, y que de antemano se entiende que la seguridad no es aplicada de manera completa y global, es necesario realizar un análisis del mismo para implementar un modelo *SSDLC* que mejor se adapte a las características extraídas, este debería cumplir aspectos como políticas, análisis de amenazas e incidentes de seguridad.

CAPÍTULO I PLANTEAMIENTO DEL PROBLEMA

2.1. Justificación

Para comprender las prácticas realizadas en el sistema integrado ANI se realizo una entrevista con el *Stakeholder*, líder técnico y lideres del grupo de desarrollo, donde se selecciono un grupo de preguntas 8.1 a resolver los respectivos integrantes, se realizo de manera presencial.

Las preguntas seleccionadas se realizaron en base a las sugerencias de *SAMM*, en el módulo de *Application Risk Profile*, el cual sugiere temas a resolver como el nivel de exposición de las aplicaciones referente al internet, si existen datos de privacidad envueltos, etc. El objetivo de la primera encuesta fue el de conocer las practicas, avances y metodología que se aplica para el desarrollo del *Sistema Integrado ANI*, según la misma 8.1.1, se obtuvieron los siguientes puntos:

- No existen políticas aplicadas
    - No se han creado políticas especificas para el sistema.
    - No se han creado políticas basadas en estándares.
    - No se han creado políticas que busquen el cumplimiento de *Ley Orgánica de Protección de Datos Personales*.
- Usan *SDLC* para el desarrollo.
- Usan *Scrum* para planificación.
- No aplican estándares, guías, ni frameworks enfocado a la seguridad.
- Ellos controlan el despliegue.
- El despliegue es manual, no se usan *CI/CD*.
- Almacenan datos sensibles con respeto a *Ley Orgánica de Protección de Datos Personales*.
    - Como nombre completo, edad, sexo y contraseñas.
    - Los datos no se mueven a otros servicios.

    - Estos datos sirven unicamente para identificar al usuario y dar detalles, no se usan para procesamiento.
    - No se espera ni permite guardar datos masivos.

- El enfoque del sistema es para Instituciones Universitarios en la ciudad de Cuenca.
- Si tienen tres roles principales:
    - **Admin:** encargado de gestionar creación de profesores, estudiantes.
    - **Profesor:** Encargado de gestionar/crear salas y agregar estudiantes al mismo.
    - **Estudiante**:
    - No se espera ni permite guardar datos masivos.
- El sistema de autenticación y autorización se basa en *JSON Web Token (JWT)*.
- Usan servicios de terceros como *Whisper* Y *OpenAI*.
- Tienen contrato con proveedor.
- No sea han realizado actividades de verificación de seguridad.
- El sistema no representa un servicio critico para las instituciones que lo usan, esta pensado para utilización esporádica pero no constante.
- Se tiene planeado escalar el sistema si los condiciones se dan.
- Usan *GitHub* como repositorio para todos los proyectos.
- No existe operaciones transaccionales.
- Tiene practicas de código seguro mínimas.
- El ciclo de desarrollo *SDLC* ya ha iniciado.

Mediante la entrevista realizada, se puede concluir los diversos puntos:

2.1.1. Riesgos Existentes

El *Sistema Integrado ANI*, enfrenta riesgos de seguridad significativos que podrían comprometer la seguridad de la información de los usuarios, disponibilidad de los servicios dispuestos y perjudicar la reputación del Instituto. Las practicas implementadas son mínimas (incluyen autenticación y autorización con *JSON Web Token (JWT)* y *Least Privilege Principle* sin revisiones) la falta de un enfoque estructurado y formal en materia de **appSec!** (**appSec!**) aumenta por sí sola la exposición a posibles ataques.

En el contexto actual, los hackers emplean técnicas avanzadas, como inyecciones de código, ataques de fuerza bruta, *phishing* dirigido y explotación de vulnerabilidades en bibliotecas o *frameworks* ampliamente utilizados en varios proyectos, lo que facilita la explotación de vulnerabilidades para extraer información.

Siendo así, se busca establecer un marco que no solo mitigue las vulnerabilidades existentes en los métodos que han utilizado hasta ahora en el proyecto, sino que también prepare al equipo de desarrollo para abordar de manera proactiva las amenazas que pueden ir apareciendo con el paso del tiempo. De este modo, se pretende garantizar aplicaciones más seguras y resilientes frente a los desafíos que plantea el entorno digital actual.

2.1.2. Características para implementación *SSDLC*

Implementar *SSDLC* no se lo realiza mediante una guía fija, exacta y aplicable para cualquier negocio según de Vicente Mohino et al. (2019), sino cada empresa tiene sus diferentes posibilidades y requerimientos, por lo cual los siguientes puntos son los mas importantes para determinar como y con que se implementara la seguridad:

- **Tiempo:** No es lógico aplicar las mismas métricas de tiempo de trabajo de empresas a los miembros del equipo de desarrollo del instituto, a pesar de que ellos tienen actas que ratifican horas semanales a cumplir, sigue siendo un club opcional sin remuneración, con un numero de horas semanales cortas que varían dependiendo del estudiantes y sus posibilidades.

    - Tienen dificultades para incluir nuevos conocimientos debido al tiempo.

    - El tiempo disponible es usado para desarrollo o reuniones.

- **Conocimientos:** Al ser todavía estudiantes, no tienen conocimientos de seguridad avanzados y cuentan con los básicos

- **Estado Actual:** Muy importante destacar que al momento de realizar las entrevistas, el desarrollo ya ha comenzado y un primer despliegue esta por ocurrir, ademas usan *SDLC*.

- **Escalabilidad:** El alcance del sistema es centrado dentro de instituciones educativas superiores, usadas aproximadamente una o dos veces por semana, se plantea la posibilidad de escalar el sistema a mayores niveles pero en un futuro lejano dependiendo de las posibilidades y éxito del mismo.

- **Reputación:** Al no existir puntos de negocio, el recurso mas valioso es el de la reputación del sistema, buenos niveles de la misma pueden significar mayor reconocimiento e impulso a proyectos similares dentro del instituto, la mala reputación afecta a la imagen del instituto y perjudicar en otras categorías, por lo cual se debe reducir a la menor cantidad de amenazas posibles.

Tomando en cuenta los puntos anteriores, la implementación de un modelo *SSDLC* no es suficiente para cubrir las necesidades de seguridad redactadas, el modelo a seleccionar debe cumplir pautas especificas y también se deben implementar capacitación a los estudiantes

Implementación del *SSDLC*

El proyecto al momento del primer análisis, presenta casi nulas implementaciones de seguridad. Por ello es necesaria la implementación de un modelo *SSDLC* que se integre con el actual *SDLC*, es necesaria la implementación del mismo ya que usar otro modelo, con nuevos requerimientos y practicas, genera mayores tiempos y esfuerzo al equipo de desarrollo, y tomando en cuenta las circunstancias del mismo, esta idea es descartada.

El modelo a seleccionar debería cumplir con todas, o al menos las mayorías de características basadas en la información obtenida

- **Implementación a SDLC:** Debe usar el ciclo de desarrollo *SDLC* para acotar en lo posible el tiempo en el que lo implementa y apegarse a lo que los desarrolladores ya conocen. Ademas debe cubrir todas las faces del ciclo, desde diseño hasta despliegue.

- **Niveles Madurez:** Debe tener niveles de madurez a aplicarse, debido a que se inicia con los niveles mas bajos, esto debido a que el sistema no tiene ninguna métrica de seguridad implementada y se debe comenzar desde cero, ademas de una posible escalabilidad implica poder fácilmente aumentar los niveles de madurez de ser necesario.

- **Guías Flexibles:** Existen diversos modelos que implican seguir al pie de la letra las especificaciones, implementar uno flexible es el mejor acercamiento ya que permite seleccionar otras herramientas, estándares, métricas que se adapten a la empresa, y tomando en cuenta que el modelo a aplicar es sobre el entorno de desarrollo de un instituto de educación, no todos serán compatibles para el mismo.

- **Rol de Seguridad:** Debe ser uno que implemente un nuevo rol de seguridad casi centralizada, los estudiantes por los conocimientos y tiempos pueden contribuir pasiva en actividades como reporte de amenazas, pero no realizar análisis de amenazas o pruebas, por lo cual se necesita un modelo que centralice lo mas posible este rol y que delegue actividades de seguridad en la menor medida al equipo de desarrollo.

Evaluación y Capacitación a través de *Moodle*

Como siguiente parte de la fase inicial, se aplicó un cuestionario al Club de Desarrollo para evaluar su nivel de conocimiento en seguridad de aplicaciones. A partir de los resultados obtenidos, se identificaron deficiencias en varias áreas clave, lo que motivó la creación de un curso de capacitación en *Moodle*.

A continuación, se presentaran las preguntas formuladas en la evaluación, junto con su respectiva justificación:

2.2. Análisis primera encuesta

Se realizaron diversas encuestas para evaluar el nivel de conocimiento y preparación del equipo de desarrollo en relación con las prácticas de seguridad informática aplicadas durante el ciclo de desarrollo de software. El propósito principal de las encuestas fue identificar las fortalezas y debilidades del equipo en áreas clave, como la detección y prevención de vulnerabilidades, el uso de herramientas automatizadas (*SAST* y *DAST*) y la implementación de metodologías de desarrollo seguro como el *SSDLC*.

El análisis de los datos obtenidos permite determinar los requisitos necesarios para diseñar el tipo de capacitación adaptado a las necesidades específicas del equipo de desarrollo. Los resultados fueron organizados en gráficos y porcentajes, lo que facilita su interpretación visual y contribuye a la toma de decisiones informadas sobre las medidas a implementar para fortalecer la seguridad en el proceso de desarrollo. Estos resultados también sirven de base para proponer planes de capacitación y estrategias que optimicen el desempeño del equipo, asegurando la adopción de prácticas proactivas para la protección contra amenazas cibernéticas.

Respuestas de la Primera Encuesta

En la *Figura* 2.1, se presenta un diagrama que resume las respuestas obtenidas en la primera pregunta de la encuesta, la cual evalúa el nivel de conocimiento de los encuestados sobre el concepto de *SQL Injection* y las estrategias para prevenir este tipo de vulnerabilidad.

Figura 2.1

Distribución de conocimientos sobre inyección SQL en el equipo de desarrollo

Los resultados indican que el 42.9 % del equipo desconoce cómo prevenir *SQL Injection* y cómo mitigarla mediante técnicas como la validación de entradas, el uso de consultas parametrizadas y la implementación de procedimientos almacenados. Sin embargo, también se identificó un grupo minoritario que mostró buena comprensión o aplicación de estas medidas preventivas, lo que resalta la necesidad de reforzar la capacitación continua en prácticas de codificación segura. Esto refleja que pueden existir malas practicas que permitan el éxito del ataque descrito.

En la *Figura* 2.2, se presentan los resultados de la segunda pregunta de la encuesta, la cual evalúa el nivel de familiaridad de los encuestados con el concepto de desarrollo seguro y su aplicación en el flujo de trabajo.

Figura 2.2

Nivel de familiaridad con el concepto de desarrollo seguro

Estos datos reflejan la necesidad de fortalecer la capacitación del equipo en metodologías de desarrollo seguro, especialmente porque solo el 28.6 % integra prácticas de desarrollo seguro, mientras que el 57.1 % solo tiene conocimiento teórico. Como estudiantes de carrera de software, se explica el alto porcentaje que ha escuchado del mismo, pero no lo ha aplicado, motivo por el cual es evidente que no aplican practicas de seguridad, o al menos no todos.

En la *Figura* 2.3, se presentan los resultados de la tercera pregunta de la encuesta, la cual analiza el nivel de familiaridad de los encuestados con las mejores prácticas para manejar contraseñas de manera segura en una aplicación. Los resultados indican que una parte significativa del equipo no tiene conocimientos generales sobre la gestión segura de contraseñas, como:

Figura 2.3

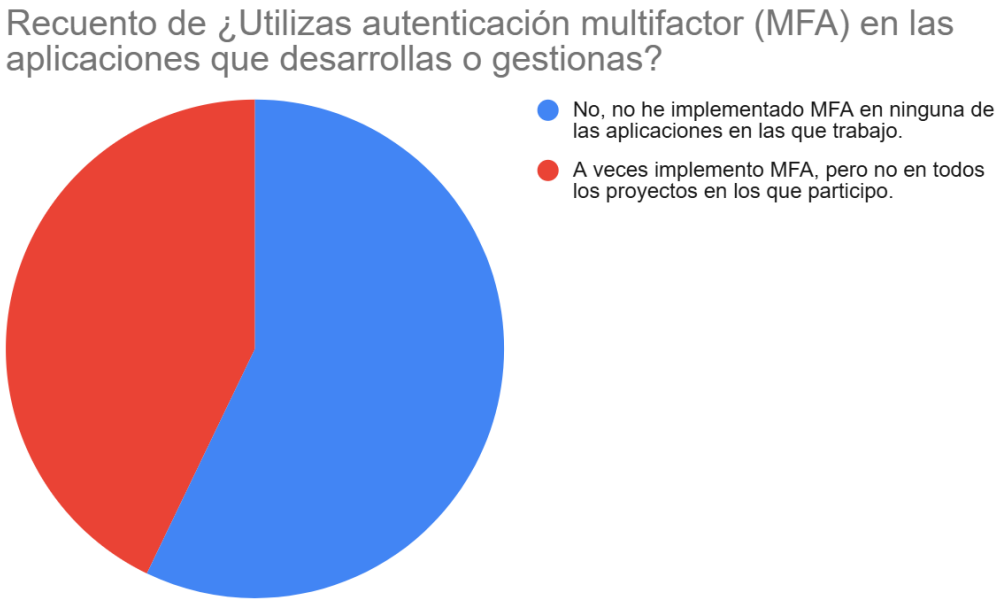
Conocimiento del equipo sobre gestión segura de contraseñas

Sin embargo, los datos sugieren que el 42.9 % de desarrolladores no aplica políticas de contraseñas seguras de manera consistente en el desarrollo, a pesar de ser una practica común, por la delegación de tareas no todos los desarrolladores cubren el concepto.

En la *Figura* 2.4, se presentan los resultados de la cuarta pregunta de la encuesta, la cual analiza si los encuestados utilizan autenticación *MFA* en las aplicaciones que desarrollan o gestionan.

Figura 2.4

Implementación de autenticación MFA en proyectos del equipo



El 57.1 % de los desarrolladores no implementa *MFA* en sus aplicaciones, lo que expone a riesgos como suplantación de identidad. La falta de uso generalizado de esta práctica esencial sugiere la necesidad de reforzar las políticas de autenticación segura.

En la *Figura* 2.5, se presentan los resultados de la quinta pregunta de la encuesta, la cual evalúa el conocimiento de los encuestados sobre los ataques de *CROSS-SITE SCRIPTING* (XSS).

Figura 2.5

Conocimiento sobre ataques Cross-Site Scripting XSS

Los resultados muestran que el 42.9 % del equipo no tiene nociones básicas sobre este tipo de vulnerabilidad, mientras que otro grupo presenta lagunas en su comprensión o aplicación de medidas para prevenirla. Este hallazgo es relevante, ya que los ataques de XSS están entre las vulnerabilidades más comunes y peligrosas según el *OWASP Top 10*.

En la *Figura* 2.6, se presentan los resultados de la sexta pregunta de la encuesta, la cual evalúa si los encuestados consideran las amenazas relacionadas con la configuración de seguridad en los servicios de terceros utilizados en el desarrollo de sus aplicaciones.

Figura 2.6

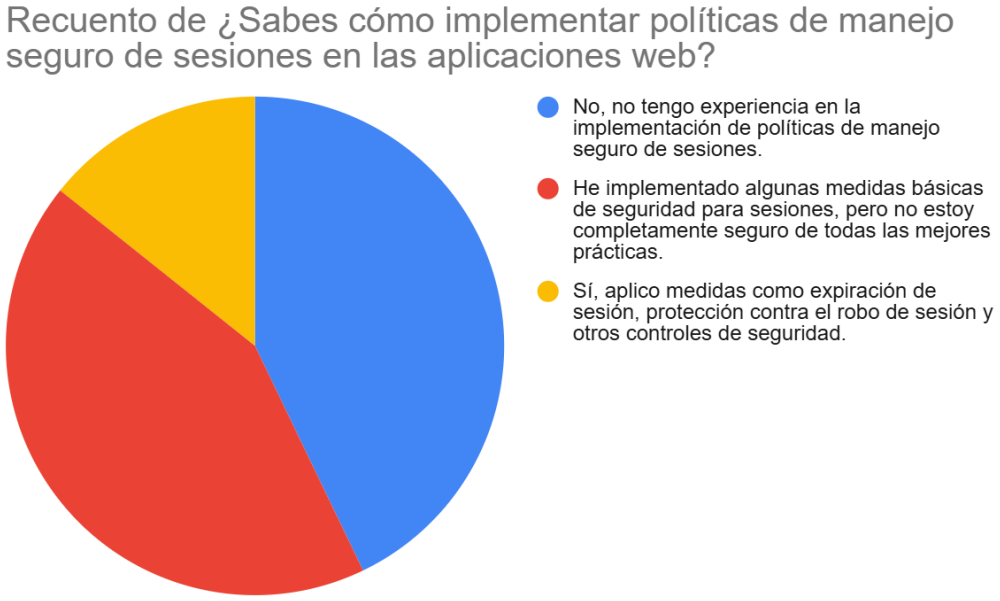
Revisión de configuraciones de seguridad en servicios de terceros

Los resultados reflejan un panorama mixto, en el que algunos miembros del equipo reconocen la importancia de revisar y asegurar las configuraciones en servicios externos, mientras que el 42.9 % no prestan suficiente atención a este aspecto crítico de la seguridad, o que generó una filtración de datos. Por lo tanto la falta de revisión de configuraciones es probable.

En la *Figura* 2.7, se presentan los resultados de la séptima pregunta de la encuesta, la cual analiza si los encuestados utilizan cifrado para proteger la información sensible almacenada en sus aplicaciones.

Figura 2.7

Uso de cifrado para proteger datos sensibles

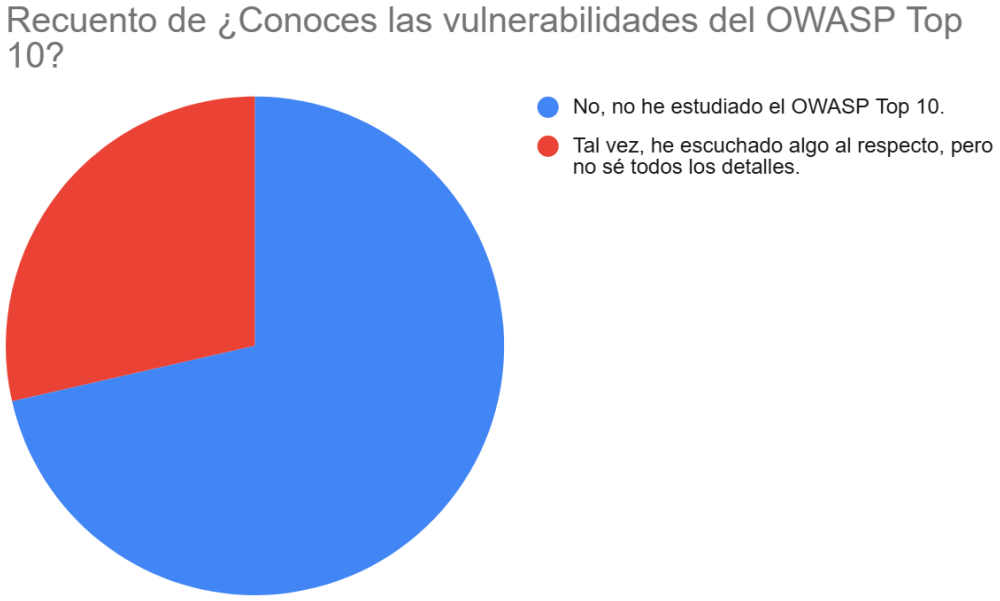


Los resultados reflejan que, si bien algunos desarrolladores emplean técnicas de cifrado, un porcentaje considerable no aplica o lo hacen de forma irregular esta medida de seguridad de manera consistente. Esto representa un riesgo importante, ya que la falta de cifrado adecuado puede exponer información confidencial, como contraseñas, datos personales y credenciales, a accesos no autorizados en caso de una violación de seguridad.

En la *Figura* 2.8, se presentan los resultados de la octava pregunta de la encuesta, la cual evalúa si los encuestados saben cómo implementar políticas de manejo seguro de sesiones en las aplicaciones web.

Figura 2.8

Conocimiento sobre políticas de manejo seguro de sesiones

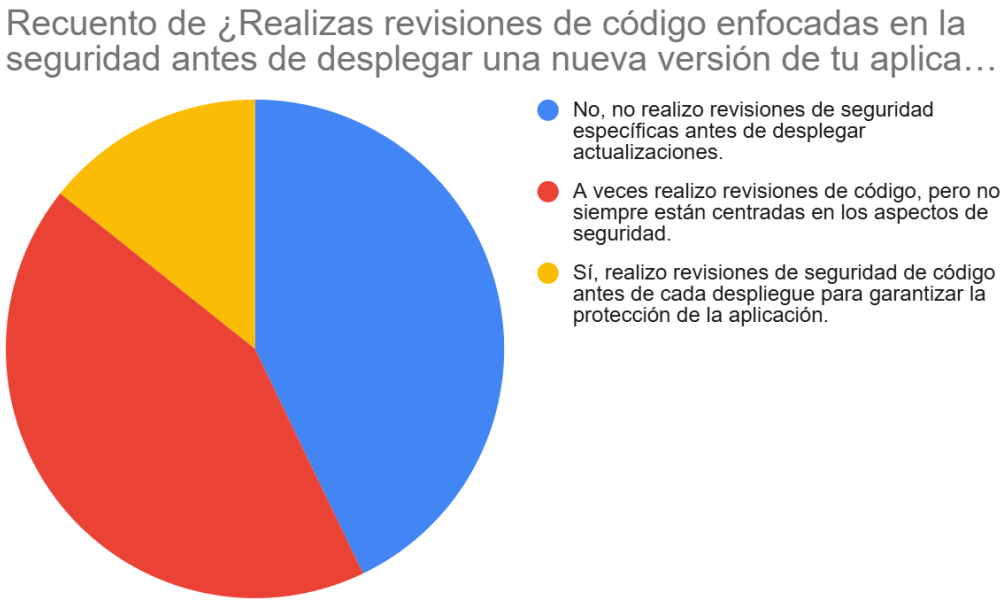


Los resultados indican que, aunque algunos desarrolladores han escuchado o les han comentado sobre las prácticas recomendadas para la gestión segura de sesiones, mientras que otra parte significativa del equipo presenta conocimientos limitados o incompletos en esta área. Teniendo en cuenta que existen dos aplicaciones principales, una web 9.0.2 y móvil 9.0.3, existe un gran riesgo de que las sesiones se manejen de manera incorrecta.

En la *Figura* 2.9, se presentan los resultados de la novena pregunta de la encuesta, la cual evalúa si los encuestados conocen las vulnerabilidades del *OWASP Top 10*.

Figura 2.9

Familiaridad con las vulnerabilidades del OWASP Top 10

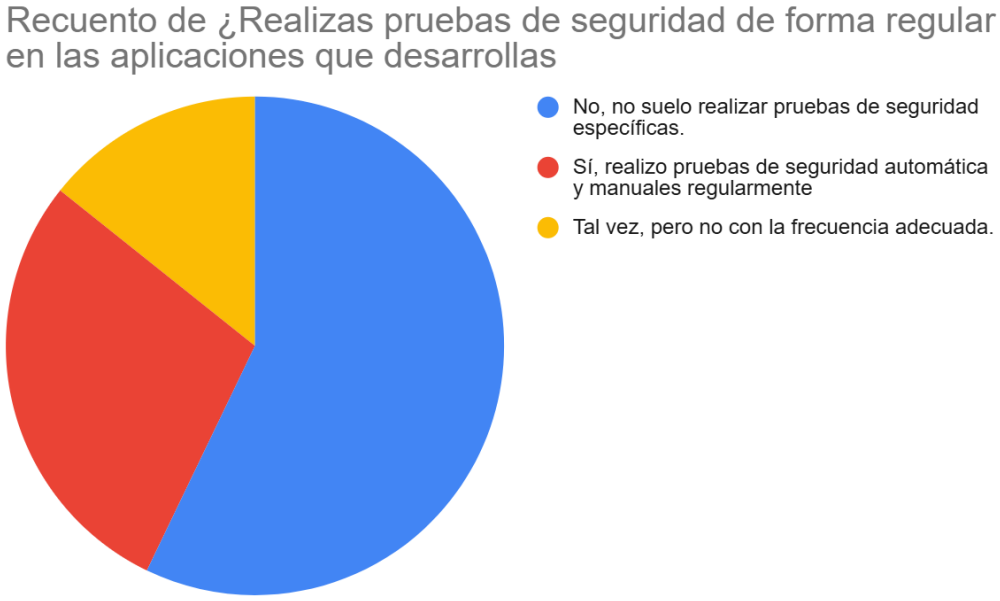


Los resultados muestran que, aunque algunos miembros del equipo identifican a la *OWASP*, otros presentan lagunas en su comprensión o no están completamente familiarizados con ellas. Esta falta de conocimiento puede traducirse en riesgos significativos durante el desarrollo y mantenimiento de aplicaciones web. Esto es un indicativo del desconocimiento de técnicas para detectar vulnerabilidades.

En la *Figura* 2.10, se presentan los resultados de la décima pregunta de la encuesta, la cual evalúa si los encuestados realizan revisiones de código enfocadas en la seguridad antes de desplegar una nueva versión de su aplicación.

Figura 2.10

Uso de Dependencias

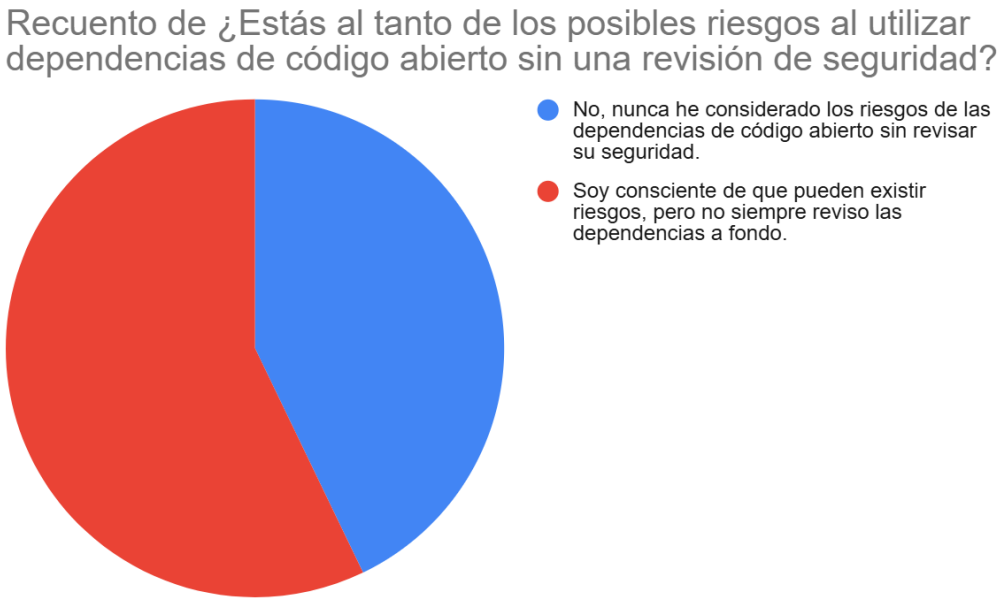


Los resultados reflejan que, la mayoría de los desarrolladores que participan en este proyecto, están cocientes de que debería de incluir revisiones de código como parte de su proceso de desarrollo, una parte menor no aplica estas revisiones de manera sistemática o las realiza de forma limitada. Esta práctica insuficiente podría aumentar el riesgo de desplegar aplicaciones con vulnerabilidades no detectadas y destaca la alta probabilidad de uso de dependencia sin revision.

En la *Figura* 2.11, se presentan los resultados de la undécima pregunta de la encuesta, la cual analiza si los encuestados conocen el concepto de ”seguridad por diseñoz lo aplican en sus procesos de desarrollo.

Figura 2.11

Adopción del principio de seguridad por diseño

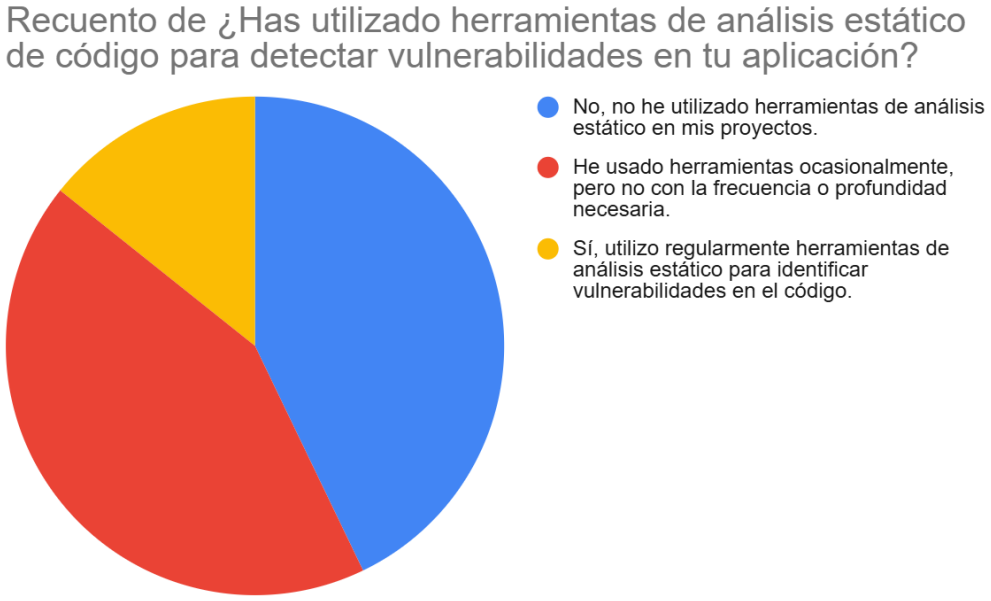


Los resultados muestran que, la mayoría de los desarrolladores que trabajan en este proyecto, no están familiarizados con el principio de seguridad por diseño y lo aplican parcialmente, una proporción menor aún no integra esta práctica de manera consistente en sus procesos. Esto evidencia una oportunidad de mejora en la adopción de estrategias proactivas de seguridad desde las etapas iniciales del desarrollo. Esto resalta la nula aplicación de practicas de seguridad durante el desarrollo.

En la *Figura* 2.12, se presentan los resultados de la duodécima pregunta de la encuesta, la cual evalúa si los encuestados han utilizado herramientas de análisis estático de código (*STATIC APPLICATION SCANNING TESTING* (SAST)) para detectar vulnerabilidades en sus aplicaciones.

Figura 2.12

Uso de herramientas de análisis estático



Los resultados indican que, aunque algunos miembros del equipo han empleado herramientas de análisis estático como parte de su flujo de trabajo, una proporción considerable de encuestados aún no ha integrado esta práctica de manera regular o simplemente no han sido informados de como hacerlo. Esta falta de adopción podría dar lugar a la presencia de vulnerabilidades no detectadas en el código antes de su despliegue y la falta de herramientas de análisis automático.

En la *Figura* 2.13, se presentan los resultados de la decimotercera pregunta de la encuesta, la cual evalúa si los encuestados están familiarizados con las mejores prácticas para proteger una *RESTful API* contra ataques comunes.

Figura 2.13

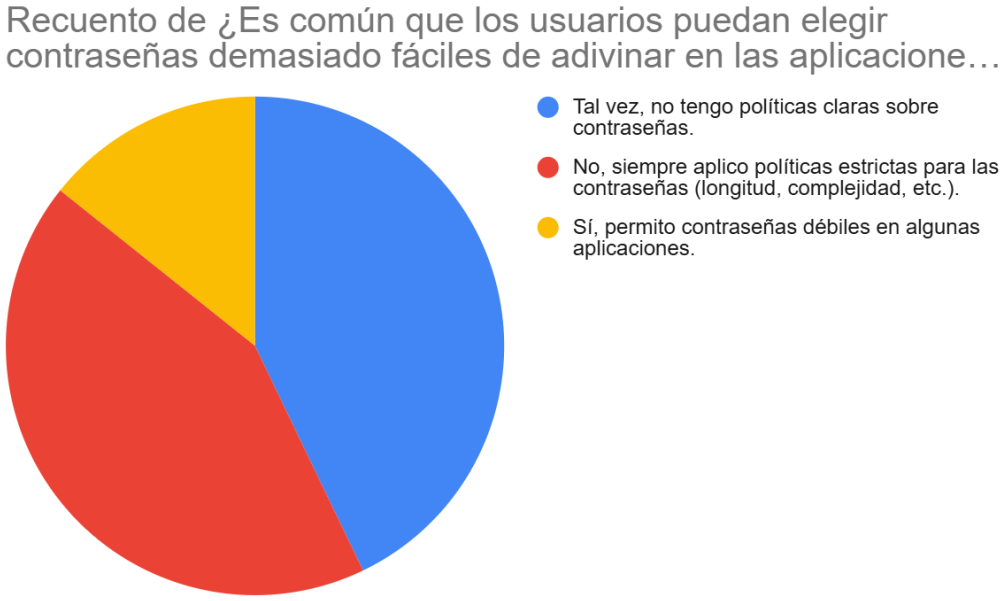
Implementación de medidas para proteger RESTful APIs

Los resultados indican que, la mayoría de los encuestados tiene conocimientos sobre las prácticas recomendadas para proteger APIs RESTful y los aplican en sus proyecto, existe otro grupo que no aplica estas medidas de manera consistente o carece de un entendimiento profundo de los riesgos involucrados. Esto resalta las brechas de conocimiento entre los desarrolladores.

En la *Figura* 2.14, se presentan los resultados de la decimocuarta pregunta de la encuesta, la cual analiza si los encuestados consideran común que los usuarios puedan elegir contraseñas demasiado fáciles de adivinar en las aplicaciones que desarrollan.

Figura 2.14

Contraseñas fáciles



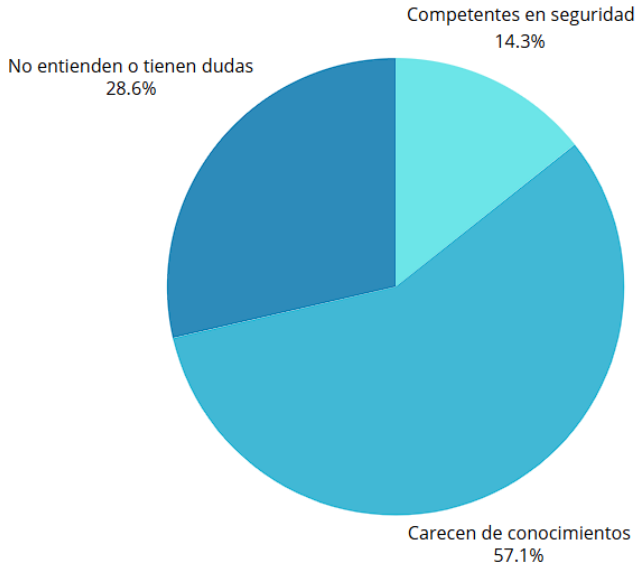
Los resultados indican que una parte significativa de los encuestados reconoce este problema como una práctica habitual, mientras que otros indican que no se presta suficiente atención a las políticas de contraseñas seguras en sus aplicaciones, la falta de implementaciones de mitigación para la parte de autenticación denota un problema bastante posible a existir del mismo.

Las preguntas del cuestionario abordaron temas esenciales de seguridad en software, incluyendo vulnerabilidades comunes como inyección de SQL, Cross-Site Scripting (XSS), autenticación multifactor (MFA), manejo seguro de sesiones y cifrado de datos sensibles, todas basadas en la entrevista *Figura* 8.1 realizada anteriormente y con temas que muy probablemente su sistema deba implementar.

Los resultados de la encuesta *Figura* 2.15 revelaron que la mayoría de los participantes carecían de conocimientos fundamentales en ciberseguridad. Ante esta situación, se diseñó un programa formativo centrado en la OWASP Top 10, el framework SAMM y buenas prácticas en seguridad

Figura 2.15

Resultados de la encuesta



para entornos móviles, web y API.

El objetivo de las encuestas es para identificar que puntos importantes de seguridad carecen los estudiantes, por lo tanto un curso de capacitación es necesario, el cual debe constar e incluir las siguientes partes:

- **Módulos Temáticos:** Cobertura de conceptos clave como principios de seguridad de desarrollo, sobre la lista de OWASP Top 10, fundamentos de seguridad tanto en web como móvil, y el modelo de seguridad *SSDLC* escogido.
- **Evaluaciones Progresivas:** Pruebas para medir el avance en el conocimiento y garantizar la aplicación práctica de lo aprendido.
- **Recursos Complementarios:** Videos explicativos que permitan reforzar los conceptos abordados.

2.3. ¿Es necesario implementar un modelo SSDLC?

Con los resultados de la primera entrevista 8.1.1 y la encuesta 2.15 de conocimientos de seguridad, la necesidad de implementar es evidente y necesaria, al no ser un servicio que espera

una retribución económica, el punto que mas afectaría es la reputación del Instituto, perjudicando posibles productos a entregar y muchos otros puntos que se escapan del alcance de este documento.

Pero otro punto bastante importante, es que los estudiantes aprendan de seguridad y tengan un rol principal que se encargue de gestionar las principales actividades de implementación de seguridad debido a su tiempo limitado y conocimientos.

El modelo seleccionado debe cumplir con la mayoría de características propuesta y debe ser presentado y expuesto periódicamente, cabe recalcar que el enfoque del estudio no es de reducir o mitigar totalmente las amenazas existentes del *Sistema Integrado ANI*, sino implementar todas las pautas necesarias para la realización de la misma, ademas de medir el nivel de adaptabilidad del modelo elegido frente al contexto brindado. **Preguntas de investigación**

1. ¿Cuál es el modelo más adecuado para integrar seguridad en el ciclo de desarrollo de software *SOFTWARE DEVELOPMENT LIFE CYCLE* (SDLC) en ANI y cómo se compara con otros modelos existentes?

2. ¿Cómo pueden las guías OWASP API, Web y Mobile Security Top 10, junto con la metodología STRIDE, ayudar a mitigar las amenazas más comunes en ANI?

3. ¿Cómo la implementación de pruebas de seguridad estática SAST, dinámica *DYNAMIC APPLICATION SCANNING TESTING* (DAST) y evaluaciones de amenazas *Threat Assessment* mejora la identificación y mitigación de riesgos en ANI?

4. ¿Cuáles son las vulnerabilidades de seguridad más críticas en el ciclo de desarrollo actual de ANI y cómo pueden priorizarse para su mitigación?

5. ¿Qué tipo de capacitación se requiere para que los desarrolladores del club de ANI adopten prácticas de seguridad establecidas por el modelo de seguridad seleccionado?

6. ¿Qué documentos referentes a seguridad genera la implementación del modelo de seguridad en la calidad y confiabilidad de las aplicaciones desarrolladas en ANI?

7. ¿Qué lecciones aprendidas pueden extraerse de la aplicación de prácticas de seguridad en ANI y cómo pueden adaptarse a otros contextos académicos o empresariales?

Estas preguntas fueron hechas de manera en que abordan aspectos clave relacionados con las vulnerabilidades actuales, estrategias de integración de seguridad, capacitación del equipo y el uso de herramientas automatizadas, proporcionando así una base sólida para el desarrollo e implementación de soluciones efectivas en la aplicación ANI.

Objetivos de la Investigación

Objetivo General Implementar el framework de seguridad *SOFTWARE ASSURANCE MATURITY MODEL* (SAMM) a la plataforma de aplicaciones ANI para el club de desarrollo del Instituto Sudamericano antes de la finalizacion de su primer ciclo academico 2025, aplicando estándares y buenas prácticas como, OWASP **oasv!** (**oasv!**), OWASP API/Web/Mobile Security Top 10, framework como SAMM y realizando pruebas de seguridad dinámicas (DAST) y estáticas (SAST), evaluaciones de amenazas (*Threat Assessment*) y capacitaciones especializadas. Esto con el fin de mejorar la gestión de la seguridad en el ciclo de vida del desarrollo de software, garantizar la entrega de aplicaciones más seguras y confiables, y fomentar una cultura de seguridad sostenible entre los miembros del club.

Objetivos específicos

- Evaluar la seguridad de las aplicaciones desarrolladas mediante el uso del estándar OWASP *OASV*, y proporcionando recomendaciones para alcanzar niveles L1 Y L2.
- Implementar las guías OWASP API, Web y Mobile Security Top 10 junto con la metodología STRIDE, con el fin de identificar y mitigar amenazas en ANI, documentando los riesgos y las estrategias de mitigación adoptadas.
- Realizar pruebas de seguridad estática SAST y dinámica DAST en las aplicaciones desarrolladas, utilizando herramientas automatizadas para generar informes detallados e integrar sus resultados en el ciclo de desarrollo seguro.
- Aplicar *Threat Assessment* utilizando *Microsoft Threat Modeling Tool (MTMT)* para priorizar los riesgos de seguridad en ANI y definir estrategias efectivas de mitigación.

- Desarrollar un programa de capacitación sobre OWASP *SAMM*, *OASV* y protocolos de seguridad, combinando aprendizaje modular y sesiones presenciales para fortalecer las prácticas de seguridad del equipo de desarrollo.
- Medir el impacto de las estrategias de seguridad implementadas en ANI comparando la cantidad de amenazas detectadas y mitigadas antes y después de su aplicación, identificando avances y áreas de mejora.

CAPÍTULO II: MARCO REFERENCIAL

3.1. Marco Teórico

En el desarrollo de software, la seguridad juega un papel fundamental para garantizar la protección de los datos y la estabilidad de las aplicaciones. Implementar medidas de seguridad desde las primeras fases del ciclo de desarrollo no solo permite identificar y mitigar vulnerabilidades de manera temprana, sino que también reduce los costos y el esfuerzo asociados a la corrección de fallos en etapas avanzadas.

En este contexto, el presente capítulo explora los fundamentos teóricos y conceptuales que sustentan la aplicación de metodologías de desarrollo seguro. Se analizarán marcos de seguridad ampliamente utilizados, como OWASP *SAMM*, *SAFE SOFTWARE DEVELOPMENT LIFE CYCLE* (SSDLC) y *BUILDING SECURITY IN MATURITY MODEL* (BSIMM), así como su impacto en la reducción de riesgos en aplicaciones web. Además, se abordarán las mejores prácticas para evaluar la seguridad en el software, considerando herramientas de análisis estático y dinámico, y la importancia de la gestión de amenazas mediante el modelado de riesgos.

Este marco referencial servirá como base para comprender la relevancia de un enfoque estructurado en seguridad de software y su aplicación en el contexto del sistema ANI, proporcionando un fundamento sólido para la propuesta metodológica presentada en los siguientes capítulos.

3.1.1. Caso de estudio comparativos con OWASP SAMM, SSDLC, BSIMM con Microsoft SDL

La aplicación de un framework de seguridad en el desarrollo de software enfrenta desafíos que varían según el contexto organizacional. Para el proyecto ANI del Instituto Sudamericano, se analizaron varios estudios conocidos que reflejan escenarios críticos, siendo que estos casos ofrecen lecciones clave para evitar errores comunes y fortalecer la implementación de cualquier tipo de framework en un entorno con recursos limitados y alta presión por entregas rápidas.

Security in the Software Security Development Lifecycle

El documento creado por Assal et al. (2018) nos demuestra que la implementación del SSDLC en cualquier proyecto resulta fundamental, no solo por su contribución a la protección de información crítica, sino también porque proporciona una estructura más organizada y eficiente.

A continuación, se presenta un resumen detallado de los hallazgos clave en la *Tabla* 3.1:

Tabla 3.1: Tabla de Análisis SSDLC

Análisis del SSDLC	
Objetivo del Estudio	Explora las prácticas de seguridad de cada etapa del ciclo de vida del desarrollo de software SDLC. El estudio se enfoca en cómo el conocimiento de seguridad influye en el proceso y cómo la seguridad encaja o entra en conflicto con el flujo de trabajo del desarrollo.
Alcance	▪ **Encuesta:** Se entrevistó a varios desarrolladores de 15 equipos de distintas compañías. ▪ **Grupo focal:** 13 participantes de cada equipo. ▪ **Roles clave:** Desarrolladores, revisores de código, especialistas en seguridad.

Metodología

- Se realizó un estudio cualitativo basado en entrevistas semiestructuradas.
- Se evaluaron las actividades de prueba de software, actitudes, conocimientos y procesos de seguridad.
- El análisis de datos se realizó mediante el método de análisis de contenido.

Resultados Clave

- **Se identificaron dos grupos principales:**
  - **Security Adopters:** 6 equipos que integran la seguridad en al menos 4 de las 6 etapas del SDLC.
  - **Security Inattentive:** 9 equipos que apenas consideran la seguridad o no la consideran en absoluto.
- Se encontraron desviaciones significativas entre las prácticas de seguridad reales y las mejores prácticas recomendadas.
- Factores como la cultura de la empresa, la división de trabajo, el conocimiento de seguridad y la presión externa influyen en la integración de seguridad durante el desarrollo.

Niveles de Madurez

- **Baja Madurez:** El grupo *Security Inattentive*, que prioriza el desarrollo y deja de lado la seguridad.
- **Media Madurez:** No hay un grupo intermedio formal, pero algunos equipos de *Security Inattentive* muestran preocupación por la seguridad en algunas etapas.
- **Alta Madurez:** El grupo *Security Adopters*, que revisa e integra medidas de seguridad al código.

Herramientas Usadas

- Herramientas de análisis de código estático (SAST).
- Pruebas de penetración realizadas internamente o por consultores externos.
- Herramientas personalizadas de análisis y pruebas de seguridad.
- Marcos de desarrollo que incluyen funcionalidades de seguridad (aunque algunos equipos los usan incorrectamente).

Desafíos Encontrados

- Falta de conocimiento de seguridad en los desarrolladores.
- Cultura organizacional que no prioriza la seguridad.
- Falta de herramientas de seguridad accesibles y fáciles de usar.
- Falta de integración de seguridad en etapas tempranas del SDLC.
- Resistencia a cambios en la división de tareas dentro del equipo de desarrollo.

Lecciones Aprendidas

- La seguridad debe ser una responsabilidad compartida en todo el equipo de desarrollo.
- La incorporación de seguridad desde la etapa de diseño facilita la mitigación de vulnerabilidades.
- Las herramientas de análisis automático pueden ayudar, pero no deben ser la única línea de defensa.
- Experimentar una brecha de seguridad puede ser un catalizador para mejorar las prácticas de seguridad en un equipo.
- La presión externa, como auditorías y cumplimiento normativo, puede motivar la adopción de mejores prácticas.

Asimismo, la incorporación de un framework adecuado desempeña un papel clave en la integración del SSDLC, ya que ambos se complementan, ofreciendo las herramientas necesarias para

un análisis más sólido en materia de seguridad, entre los modelos de framework que se analizaron para utilizarlo fueron:

Security Champions Without Support: Results from a Case Study with OWASP SAMM in a Large-Scale E-Commerce Enterprise

El primer caso de Gutfleisch et al. (2023) refleja desafíos similares a los identificados en el Instituto Sudamericano, como la falta de capacitación en seguridad y la dependencia de sin apoyo. Por ello, las lecciones de este estudio serán clave para evitar errores en la implementación de *SAMM* en ANI.

La investigación se basó en un enfoque cualitativo, mediante entrevistas con 20 participantes que desempeñaban distintos roles en cinco equipos ágiles de desarrollo.

Los resultados evidenciaron marcadas diferencias en la adopción de prácticas de seguridad entre los equipos, con niveles de madurez desiguales que oscilaron entre 0 y 3. Se identificó que las áreas de **Governance** y **Operations** presentaban los menores niveles de madurez, lo que sugiere la falta de estrategias organizacionales bien definidas y una gestión insuficiente de los procesos de seguridad. En contraste, la función de **Implementation** mostró los niveles más altos de madurez, aunque con una dependencia excesiva de los , quienes, al no contar con un respaldo estructural adecuado, enfrentaban dificultades para sostener las iniciativas de seguridad a largo plazo. Este hallazgo resalta la necesidad de un enfoque más equilibrado y sostenible en la integración de la seguridad dentro del ciclo de vida del desarrollo de software.

En la *Tabla* 3.2 se presentan de manera detallada los datos recopilados, proporcionando una visión más clara del proceso de análisis del sistema de seguridad y permitiendo una mejor comprensión de los resultados obtenidos.

Tabla 3.2: Tabla de Análisis del Modelo SAMM

Análisis del modelo SAMM

Objetivo del Estudio	Evaluar la efectividad del programa de en una gran empresa de comercio electrónico utilizando OWASP *SAMM*. El estudio buscó entender cómo los roles asignados influyen en la adopción de prácticas de seguridad, identificar desafíos organizacionales y analizar la percepción de seguridad entre los desarrolladores, DevOps, Product Owners y Scrum Masters.
Alcance	▪ **Participantes en OWASP SAMM:** 5 desarrolladores senior. ▪ **Entrevistas cualitativas:** 15 participantes, incluyendo desarrolladores (6), DevOps (2), Scrum Masters (3) y Product Owners (4). ▪ **Roles clave:** (asignados en cada equipo), desarrolladores, DevOps, líderes de producto y Scrum Masters.
Metodología	▪ Se utilizó la versión 2.0 de OWASP SAMM. ▪ Se midió la madurez en 5 áreas de negocio mediante entrevistas. ▪ La puntuación mínima de madurez del proyecto fue de 0 a 3 por práctica, basada en 15 prácticas de seguridad y 30 streams.

Resultados Clave

- **Variabilidad en madurez:**
  - Governance y Design tuvieron las puntuaciones más bajas.
  - Operations tuvo la puntuación más alta.
- **Roles y Percepciones:**
  - Los carecían de apoyo y recursos.
  - Los Product Owners carecían de conocimiento técnico en seguridad y delegaban responsabilidades a los .
  - Los equipos seguían su propio enfoque de seguridad ignorando al resto.

Niveles de Madurez

- **Governance:** Baja madurez en Strategy y Metrics.
- **Design:** Prácticas como Threat Assessment y Secure Architecture con alta variabilidad entre equipos.
- **Implementation:** Secure Build y Secure Deployment mostraron avances, pero con dependencias externas mal gestionadas.
- **Verification:** Security Testing fue la práctica más sólida (0.86), pero con baja adopción de pruebas automatizadas.
- **Operations:** Incident Management destacó (1.08), aunque con procesos reactivos en lugar de preventivos.

Herramientas Usadas

- Pruebas de penetración manual y análisis de dependencias.
- Reuniones periódicas de y revisión de requisitos.
- Falta de automatización en pruebas de seguridad.

Desafíos Encontrados

- sin capacitación, recursos o autoridad.
- Priorización de funcionalidad sobre seguridad.
- Estrategia fragmentada y falta de educación en seguridad.
- Ausencia de métricas para medir el progreso en seguridad.

Lecciones Aprendidas

- Necesidad de métricas claras, recursos y estrategia unificada.
- Los Security Champions requieren apoyo activo de la dirección y recursos dedicados.
- La comunicación transversal y la cultura de ”no culpa”son esenciales para mejorar prácticas de seguridad.

Evaluating software security maturity using OWASP SAMM: Different approaches and stakeholders perceptions

El segundo caso desarrollado por Fucci et al. (2024) es similar al primer caso siendo que estos utilizan el framework *SAMM* como su principal modelo.

Este estudio evalúa la madurez de seguridad en el desarrollo de software utilizando el modelo *OPEN WEB APPLICACTION SECURITY PROJECT* (OWASP) *SAMM*, analizando las percepciones de distintos roles en una empresa del sector financiero. Mediante una encuesta en línea (17 participantes) y un grupo focal (siete participantes), se compararon enfoques ligeros y profundos para la evaluación. Los resultados que se mostraron muestran que hay diferencias significativas en cómo los roles perciben la madurez, siendo que algunos desarrolladores y verificadores reportaron mayor madurez en áreas técnicas, mientras operaciones y arquitectos identificaron debilidades en prácticas bajo su responsabilidad.

A continuación, se presenta un resumen detallado de los hallazgos clave en la *Tabla* 3.3:

Tabla 3.3: Tabla de Análisis SAMM

Segundo Análisis del Modelo SAMM	
Objetivo del Estudio	Evaluar la madurez de seguridad de software en una empresa del sector financiero utilizando OWASP *SAMM*, analizando cómo diferentes roles perciben la evaluación y la utilidad del instrumento. El estudio buscó identificar fortalezas, debilidades y áreas de mejora en las prácticas de seguridad.

Alcance

- **Encuesta:** 17 participantes de roles variados: gestión (4), desarrolladores (7), arquitectos (2), operaciones (4) y verificación (3).
- **Grupo focal:** 7 participantes.
- **Roles clave:** gerentes, desarrolladores, arquitectos, personal de operaciones y verificación.

Metodología

- Se utilizó OWASP *SAMM* personalizado con terminología específica de la empresa.
- Los participantes respondieron preguntas sobre la seguridad del proyecto.
- La puntuación mínima de madurez del proyecto fue de 0 a 3 por práctica, basado en respuestas de 4 niveles.

Resultados Clave

- **Diferencias entre roles:**
  - Los roles perciben mayor madurez en áreas relacionadas con sus responsabilidades.
  - Operations reportó menor madurez en prácticas fuera de su ámbito.
- **Comparación de métodos:**
  - La encuesta y el grupo focal mostraron resultados similares, excepto en áreas como Design y Operations.
- **Percepción del cuestionario:**
  - 40 % de los participantes omitieron preguntas sobre Verification.
  - Arquitectos y gerentes encontraron preguntas de Governance más difíciles.

Niveles de Madurez

- **Governance:** Baja madurez en Strategy y Metrics.
- **Design:** Arquitectos percibieron menor madurez en Threat Assessment y Secure Architecture.
- **Implementation:** Desarrolladores reportaron alta madurez en Secure Build y Secure Deployment.
- **Verification:** Máxima madurez en Security Testing.
- **Operations:** Baja madurez en Operational Management.

Herramientas Usadas

- OWASP *SAMM* para evaluación de madurez y planificación de mejoras.
- Análisis de seguridad con herramientas de análisis estático (SAST) y dinámico (DAST).
- Sesiones de análisis de riesgos y revisiones de código.

Desafíos Encontrados

- Fricción entre roles en la evaluación.
- Gerentes subestimaron prácticas como Requirements Testing.
- Preguntas de Governance resultaron complejas para roles no gerenciales.

Lecciones Aprendidas

- Involucrar múltiples roles mejora la precisión de la evaluación.
- Un enfoque ligero (encuesta) es viable, pero requiere validación en áreas críticas.
- Las respuestas con baja confianza revelan prácticas poco conocidas.
- OWASP *SAMM* ayuda a identificar brechas de seguridad en equipos DevOps.
- Adaptar la terminología del cuestionario mejora la comprensión.

Adopting security practices in software development process: Security testing framework for sustainable smart cities.

En el tercer caso, desarrollado por Mothanna et al. (2024), se presenta un enfoque distinto al integrar los frameworks BSIMM y Microsoft SDL, complementando sus respectivas deficiencias para mejorar la seguridad en el desarrollo de software. Este estudio se centra en la protección de aplicaciones para ciudades inteligentes, un entorno donde la interconectividad y la dependencia de dispositivos IoT representan desafíos únicos en términos de seguridad.

En la *Tabla* 3.4 se encuentra mas información sobre el resumen del estudio:

Tabla 3.4: Tabla de Análisis BSIMM con Microsoft SDL

Análisis del BSIMM con Microsoft SDL

Objetivo del Estudio	Evaluar la seguridad en el desarrollo de software para aplicaciones de ciudades inteligentes. Propone un marco de pruebas de seguridad que integra prácticas de seguridad en todas las fases del SDLC, abordando desafíos específicos de las ciudades inteligentes.
Alcance	▪ Desarrollado por un equipo de cinco investigadores de distintas instituciones: • Universidad de Baréin. • Universidad de Mutah, Jordania. • Universidad del Este de Londres, Reino Unido.

Metodología

- Basado en el framework BSIMM, enfocado en la evaluación de seguridad en el desarrollo de software para ciudades inteligentes.
- Se aplicaron cuatro fases principales:
  - **Preparación:** Definir objetivos de pruebas de seguridad y mejorar el conocimiento del equipo.
  - **Requisitos:** Evaluar riesgos y definir directrices de seguridad.
  - **Implementación:** Ejecutar pruebas de seguridad según las directrices establecidas.
  - **Mejora:** Monitoreo continuo, gestión de incidentes y creación de un repositorio de vulnerabilidades.

Resultados Clave

- Las aplicaciones de ciudades inteligentes enfrentan riesgos únicos debido a su interconectividad y dependencia de dispositivos IoT.
- Integrar pruebas de seguridad desde las primeras etapas del desarrollo reduce vulnerabilidades y mejora la seguridad.
- Se propuso la metodología BSIMM para adoptar mejores prácticas de seguridad.

Niveles de Madurez

- **Governance:** Se han definido directrices y normativas (ISO 27001, NIST, GDPR), pero falta una gobernanza centralizada y supervisión ejecutiva.
- **Intelligence:** Se realizan evaluaciones de riesgos y existe un repositorio de vulnerabilidades, pero no hay colaboración activa con otras organizaciones en inteligencia de amenazas.
- **Build:** Se aplican pruebas de seguridad (SAST, DAST) y buenas prácticas de codificación, pero falta automatización avanzada en la revisión de código.
- **Deployment y Operations:** Existen prácticas básicas de monitoreo y gestión de configuración, pero faltan simulaciones de ataques avanzadas (Red Teaming) y pruebas de penetración regulares.

Herramientas Usadas

- Pruebas de seguridad estáticas (SAST) y dinámicas (DAST).
- Análisis de amenazas y modelado de riesgos basado en estándares como ISO 27001 y NIST.
- Evaluación de cumplimiento y gestión de configuraciones para minimizar riesgos de seguridad.

Desafíos Encontrados

- Falta de un marco de seguridad adaptado a las aplicaciones de ciudades inteligentes.
- Dificultades en la integración de seguridad en el ciclo de vida del software debido a la complejidad de los sistemas interconectados.
- Riesgos específicos en entornos IoT y redes urbanas, lo que exige enfoques especializados para la protección de datos y sistemas.

Lecciones Aprendidas

- La seguridad debe ser una responsabilidad compartida en todo el equipo de desarrollo.
- Incorporar seguridad desde la etapa de diseño facilita la mitigación de vulnerabilidades y evita complicaciones si se añade al final.
- Las pruebas de seguridad continuas mejoran la detección de vulnerabilidades y reducen los costos de corrección.
- La formación del equipo de desarrollo en prácticas de seguridad fortalece la protección de las aplicaciones de ciudades inteligentes.

La elección del modelo de seguridad adecuado para un proyecto de desarrollo de software de-

pende de múltiples factores, incluyendo el nivel de madurez en seguridad de la organización, los recursos disponibles y el contexto en el que se aplicará. En este sentido, frameworks como *SSDLC*, OWASP *SAMM*, *Microsoft SDL* y BSIMM ofrecen enfoques distintos, cada uno con ventajas específicas.

OWASP *SAMM* se destaca por su flexibilidad, ya que permite evaluar el nivel de madurez en seguridad sin imponer un esquema rígido de implementación. Esta característica lo convierte en una opción adaptable para equipos ágiles que buscan mejorar su seguridad progresivamente. Sin embargo, su efectividad depende en gran medida de la autoevaluación y del compromiso del equipo en seguir sus lineamientos.

En contraste, *Microsoft SDL* se basa en un marco estructurado con reglas estrictas que lo hacen más adecuado para grandes empresas que requieren un enfoque formalizado en seguridad. No obstante, su implementación demanda un alto nivel de compromiso organizacional, lo que puede dificultar su adopción en entornos ágiles o en empresas con menos recursos.

En este contexto, BSIMM se presenta como un enfoque basado en la observación y medición de prácticas de seguridad en organizaciones reales. A diferencia de OWASP *SAMM*, que proporciona un modelo flexible de autoevaluación, BSIMM se basa en datos empíricos obtenidos de empresas líderes en seguridad, lo que permite a las organizaciones comparar sus estrategias con estándares de la industria. Además, mientras *Microsoft SDL* impone un marco normativo estricto, BSIMM se enfoca en la mejora continua basada en prácticas efectivas comprobadas.

Dado lo anterior, la selección del modelo más adecuado dependerá de los objetivos y necesidades de cada organización. Si se busca flexibilidad y adaptabilidad en equipos ágiles, OWASP *SAMM* es una excelente opción. Para organizaciones que requieren un marco estructurado con reglas estrictas, *Microsoft SDL* es el más adecuado. Finalmente, BSIMM es recomendable para organizaciones que buscan evaluar y mejorar sus prácticas de seguridad con base en datos empíricos obtenidos de la industria.

Si bien la fase de arquitectura de la Aplicación *Sistema Integrado ANI* ya ha sido completada, estos hallazgos resaltan la necesidad de incorporar controles de seguridad retroactivos en el ciclo de desarrollo actual. Para ello, se implementarán revisiones de arquitectura basadas en el modelo

Tabla 3.5: Tabla de comparativa

Comparativa entre BSIMM, Microsoft SDL y SAMM			
Prácticas de seguridad	BSIMM	MS-SDL	SAMM
Definición de requisitos de seguridad	Sí	Sí	Sí
Gestión segura de la configuración	No	No	No
Siguiendo todas las leyes aplicables	Sí	No	Sí
Modelado de amenazas	Sí	Sí	Sí
Análisis de riesgos	Sí	Sí	Sí
Arquitectura de seguridad	Sí	Sí	Sí
Capacitación y concienciación sobre seguridad	Sí	Sí	Sí
Diseño seguro	Sí	Sí	Sí
Análisis del código fuente	Sí	Sí	Sí
Análisis de vulnerabilidad	Sí	Sí	Sí
Verificación de seguridad	Sí	Sí	Sí
Gestión de vulnerabilidades	Sí	Sí	Sí
Técnicas y aplicaciones de desarrollo seguro	Sí	Sí	Sí
Seguridad en un entorno operativo activo	No	Sí	Sí
Integración segura con periféricos	No	Sí	Sí
Entrega segura	No	No	Sí

de amenazas *STRIDE*, *CIA* y respectivamente *OWASP Top 10*,*OWASP API Security Top 10*,*OWASP Mobile Security Top 10* para cada aplicacion que aplique, el objetivo de estas revisiones es el de identificar y mitigar riesgos en componentes críticos ya desarrollados.

Otro cambio que se implementó fue que en vez de usar un programa de rotación, como comúnmente suele usarse, se optó por explorar alternativas más alineadas con la dinámica del equipo y los objetivos del proyecto. Como resultado, se adoptó un enfoque colaborativo, diseñado para distribuir responsabilidades de seguridad de manera eficiente sin generar una carga excesiva en los

.

- **Delegación de tareas específicas**

  - Los desarrolladores asumirán responsabilidades de seguridad en **sprints designados**, participando activamente en revisiones de código mediante el uso de herramientas y documentación provistas por el equipo de seguridad.
  - El *Security Champion* supervisará estas actividades y validará los hallazgos mediante notificaciones automatizadas de herramientas como SonarQube, asegurando un proce-

so eficiente de aceptación de resultados.

- **Historias de usuario**

  - Se fortalecerá la integración de prácticas de seguridad en la definición y desarrollo de historias de usuario.
  - Se destinarán entre 5 y 8 minutos en cada reunión de planificación para discutir alertas de seguridad, avances en la mitigación de riesgos y estrategias de mejora continua.

3.2. Marco Conceptual

3.2.1. ¿Que es la seguridad en aplicaciones?

La seguridad en aplicaciones es un área fundamental en el desarrollo de software, enfocada en proteger sistemas como aplicaciones web, móviles, APIs y bases de datos contra vulnerabilidades que podrían comprometer su funcionamiento e integridad. La presencia de fallos de seguridad puede afectar en accesos no autorizados, robo de datos o manipulación maliciosa, lo que pone en riesgo tanto la información almacenada como la operatividad del sistema.

Logrando así cumplir con el objetivo principal de implementar medidas de seguridad es garantizar los principios de confidencialidad, integridad y disponibilidad de la información. Esto implica prevenir accesos indebidos, asegurar que los datos no sean alterados sin autorización y garantizar que los sistemas permanezcan accesibles para los usuarios legítimos.

Para lograr este tipo de protección efectiva, la seguridad debe incorporarse desde las primeras fases del desarrollo y mantenerse a lo largo de todo el ciclo de vida del software SDLC. Esto se logra mediante estrategias como revisiones de código, pruebas de penetración y auditorías de seguridad, las cuales permiten identificar y mitigar riesgos antes de que sean explotados. Entre las amenazas más comunes que deben abordarse se encuentran las inyecciones de código, autenticación débil, exposición de datos sensibles y configuraciones erróneas en servidores y permisos.

3.2.1.1 ¿Que es SDLC?

El SDLC o mas conocido como **Ciclo de Vida del Desarrollo de Software**, un proceso estructurado que define las etapas necesarias para crear, implementar y mantener un sistema o aplicación de software de manera eficiente y organizada, este sistema se compone de varias fases siendo:

- **Recopilación y análisis de requisitos:** Primero se debe identificar las necesidades y expectativas del cliente o usuario para documentar los hallazgos.
- **Diseño del sistema:** Debe definirse la arquitectura junto con los componentes del software, detallando su implementación y funcionalidades que el cliente requiere.
- **Desarrollo:** Aquí es donde los desarrolladores crean el código fuente según las especificaciones que se definieron anteriormente.
- **Pruebas:** Las pruebas ayudan a verificar y validar el funcionamiento de todos los requisitos.
- **Implementación:** El despliegue del software en el entorno de producción para ser utilizado por los usuarios finales.
- **Mantenimiento:** Dependiendo del contrato se debe realizar actualizaciones, mejoras o mantenimiento.

Todo esto es fundamental ya que ayuda a programar, planificar o estimar proyectos midiendo el tiempo estimado que tardaría en acabarlos, otro tema a abarcar seria que existen variedades de modelos de SDLC, como el modelo en cascada, el incremental y el ágil, cada uno con un enfoque y aplicaciones que se adaptan dependiendo a las características del proyecto.

3.2.1.2 ¿Que es SSDLC?

El *Secure Software Development Life Cycle* es una extensión del *ciclo de vida de desarrollo de software* que integra medidas de seguridad en cada una de las fases del proceso de desarrollo, en

lugar de aplicarlas únicamente al final de este mismo. Esta metodología tiene como objetivo principal garantizar que el software, como producto final, sea resistente a vulnerabilidades y amenazas desde su concepción y diseño hasta su despliegue y mantenimiento.

A diferencia del enfoque tradicional, donde las evaluaciones de seguridad suelen realizarse en las etapas finales, el SSDLC aplica un enfoque preventivo al incorporar controles de seguridad desde el inicio del desarrollo. De esta manera, se facilita la detección temprana de fallos y se reduce el costo asociado a la corrección de vulnerabilidades en fases avanzadas.

Otro uso principal para el SSDLC es que no solo prioriza la seguridad, sino que también mantiene los principios fundamentales del SDLC, como la funcionalidad, la eficiencia y la calidad del software. Al combinar estos aspectos, este enfoque ofrece un marco integral que fortalece tanto la protección contra amenazas como el cumplimiento de los requisitos funcionales y de rendimiento, garantizando así aplicaciones más robustas y confiables.

Actualmente, no hay un modelo global estandarizado con el que se pueda aplicar de manera precisa la implementación del SSDLC. En su lugar, cada empresa adapta varias reglas y prácticas de seguridad según sus necesidades específicas. La implementación del SSDLC debe considerar el modelo de desarrollo adoptado, como ágil, *Waterfall*, entre otros. Además, depende de los recursos disponibles en la organización.

Por ejemplo, en grupos de desarrollo pequeños, puede ser necesario que todos los desarrolladores posean conocimientos previos sobre seguridad, mientras que en empresas más grandes, suelen existir equipos dedicados exclusivamente a esta área, quienes además reciben capacitaciones periódicas y aunque la implementación del SSDLC puede variar significativamente en función del contexto organizacional, existen varios modelos y enfoques que sirven como guía. Estos serán descritos más adelante para ofrecer un marco de referencia útil en la adopción de prácticas seguras durante el desarrollo de software.

Cada fase de desarrollo de SDLC en SSDLC implementa una nueva característica de seguridad, Ahmed et al. (2023) describe las siguientes recomendaciones que se debe tomar en cuenta:

1: Requisitos En la fase de análisis de requisitos, es fundamental establecer metas claras de cumplimiento para garantizar que el equipo de desarrollo pueda abordar las vulnerabilidades de

manera oportuna. Esto incluye la creación de una lista detallada de requisitos de seguridad, como controles de acceso, límites de operación y políticas de privacidad. Además, se deben programar sesiones de entrenamiento para el equipo de desarrollo y realizar evaluaciones de riesgos para identificar posibles amenazas desde las etapas iniciales del proyecto.

2: Diseño Durante la fase de diseño, se deben desarrollar modelos de riesgos que contemplen escenarios potenciales de ataque. Es esencial validar el diseño del documento y cualquier modificación posterior para asegurar su solidez frente a amenazas. Asimismo, se recomienda revisar periódicamente cualquier *THIRD-PARTY SOFTWARE* (T-PS) para identificar vulnerabilidades emergentes y aplicar parches de seguridad de manera inmediata.

3: Implementación En esta etapa, se debe cumplir con los estándares de codificación segura para eliminar errores menores y permitir que el equipo de desarrollo se enfoque en actividades críticas. Es importante ejecutar herramientas de análisis estático de seguridad de aplicaciones (SAST) en el código recientemente escrito, con el fin de identificar vulnerabilidades antes de las compilaciones. Además, se debe realizar una revisión manual del código de manera oportuna para abordar posibles problemas y resolverlos antes de avanzar en el proceso de desarrollo.

4: Testing La fase de pruebas debe integrar herramientas de análisis dinámico de seguridad de aplicaciones (DAST) junto con análisis interactivo (*INTERACTIVE APPLICATION SECURITY TESTING* (IAST)). Esta técnica combina el escaneo en tiempo de ejecución con la monitorización del programa y el análisis del flujo de datos, proporcionando una evaluación más exhaustiva. También se recomienda implementar pruebas de fuzzing (*Fuzz Testing*) para evaluar la capacidad de la aplicación para resistir ataques basados en entradas inesperadas o erróneas.

5: Despliegue y mantenimiento Finalmente, en la etapa de despliegue y mantenimiento, es crucial monitorizar no solo la aplicación, sino también todo el entorno en el que se ejecuta el sistema. Es necesario crear planes de respuesta ante incidentes que detallen las acciones a

seguir en caso de detectar brechas de seguridad. Asimismo, deben realizarse revisiones de seguridad continuas, ya que las vulnerabilidades evolucionan constantemente y requieren un enfoque proactivo para su detección y mitigación.

”No existe software libre de errores, dado que la tecnología es cambiante, lo que provoca que las tendencias en ciberseguridad sean dinámicas e impredecibles”, según Ahmed et al. (2023). En este contexto, el autor propone las siguientes soluciones para abordar los desafíos asociados a la seguridad del software:

Educar y entrenar al equipo de desarrollo sobre practicas de código seguras: Muchos desarrolladores suelen asumir que poseen los conocimientos necesarios para escribir código seguro. Sin embargo, esta percepción puede verse afectada por una comprensión limitada sobre la gravedad de las vulnerabilidades y los riesgos asociados. Esta falta de conocimiento en los fundamentos de seguridad en aplicaciones puede exponer al software a amenazas evitables. Para abordar esta problemática, se recomienda implementar programas de entrenamiento continuos. Programas que permiten a los desarrolladores aprender, comprender e implementar técnicas de codificación segura, contribuyendo así a minimizar los riesgos de seguridad desde las primeras etapas del desarrollo.

Utilizar herramientas automatizadas de testing: El uso de herramientas automatizadas para las pruebas de seguridad ha revolucionado el proceso de detección de vulnerabilidades, simplificando las pruebas manuales y proporcionando informes más completos y detallados. Según Hanna (2018), las pruebas automatizadas pueden reducir en un 68 % el tiempo total dedicado a este proceso, además de acelerar significativamente el lanzamiento de productos al mercado. Esta eficiencia permite a los equipos de desarrollo centrarse en corregir las vulnerabilidades detectadas sin retrasar los ciclos de producción.

Automatización continua del escaneo de amenazas y vulnerabilidades: Con el aumento del tráfico digital y el desarrollo de tecnologías más avanzadas tras la pandemia, los ciberataques también han incrementado en frecuencia y complejidad. Por ello, es fundamental implementar procesos de escaneo continuo para identificar vulnerabilidades en tiempo real. Además,

se recomienda establecer mecanismos automatizados que permitan responder de forma inmediata ante posibles incidentes de seguridad. Estas medidas no solo fortalecen las defensas del sistema, sino que también garantizan una respuesta rápida y eficiente frente a amenazas emergentes.

3.2.1.3 ¿Que es un framework de seguridad?

Un framework de seguridad se podría considerar como un complemento que se puede añadir al SSDLC, este es un conjunto de directrices, procedimientos y políticas que ayuda a las organizaciones en su gestión de ciberseguridad, proporcionando una guía para identificar, proteger y controlar los activos y recursos usados por la organización frente a diferentes amenazas.

Su implementación permite establecer controles y medidas preventivas que ayudan a mitigar los riesgos, a garantizar la integridad junto con las confidencialidad y disponibilidad de la información, esto asegura el cumplimiento de normativas y estándares de seguridad.

3.2.1.4 ¿Cuales frameworks de seguridad existen basados en SSDLC?

Existen diversos frameworks de seguridad que podrían utilizar en conjunto con SSDLC esto depende del tipo de trabajo que se esta realizando ya que cada uno tiene enfoques y aplicaciones especificas, dentro de estos los que mas se destacaron fueron:

- **SAMM**

  SAMM es un modelo de framework abierto que fue creado por la empresa OWASP la cual se especifica en desarrollar herramientas para la seguridad informática, ayudando a formular e implementar estrategias de seguridad en el desarrollo de software, adaptando los riesgos de amenazas cada ciertos años. Su flexibilidad es uno de sus principales beneficios, ya que puede integrarse en cualquier fase del ciclo de desarrollo, ya sea en proyectos en sus primeras etapas o en aquellos que ya han sido desplegados.

  Su adaptabilidad a los distintos tipos de organizaciones, desde pequeñas hasta grandes corpo-

raciones, es una de sus mayores fortalezas, pues su estructura, basada en niveles de madurez, permite una implementación progresiva, facilitando que los equipos mejoren su seguridad de manera escalable sin generar una sobrecarga operativa. Asimismo, *SAMM* no solo fortalece la seguridad del software, sino que también fomenta una cultura de seguridad dentro del equipo de desarrollo, promoviendo la capacitación en buenas prácticas y aumentando la concienciación sobre la importancia de la seguridad en cada fase del proyecto.

La organización de *SAMM* se centra en cinco funciones, cada una subdividida en tres prácticas de seguridad, sumando un total de quince prácticas, donde cada una contiene un conjunto de actividades estructuradas en tres niveles de madurez, siendo que los niveles inferiores son clasificadas como insuficientes en lo que respecta a seguridad.

Las cinco funciones

- **Governance:**
  - **Estrategia y Métricas:** Donde se mide y define la estrategia de seguridad.
  - **Política y Cumplimiento:** Se establece las políticas de seguridad para asegurar el cumplimiento normativo.
  - **Educación y Orientación:** Se debe proporcionar formación al grupo encargado del desarrollo en seguridad.
- **Desing:**
  - **Evaluación de Amenazas:** Con las herramientas o solo con revisión de código se debe identificar y evaluar las amenazas potenciales.
  - **Requisitos de Seguridad:** Se debe definir los requisitos de seguridad para el software.
  - **Arquitectura segura:** Diseñar arquitecturas que incorporen los principios de seguridad.
- **Implementation:**
  - **Construcción Segura:** Aplicar prácticas de códificación seguras al código.

- **Gestión de Defectos:** Gestionar las vulnerabilidades que se encontraron en el código.
- **Gestión de Configuración:** Asegurar que las configuraciones sean seguras y consistentes.

- **Verification:**
  - **Pruebas de Seguridad:** Definir las fechas en las que se deban realizar pruebas para identificar vulnerabilidades.
  - **Revisión de Diseño:** Hacer revisiones diseños para detectar posibles fallos de seguridad.
  - **Revisión de Código:** Analizar el código en profundidad en busca de vulnerabilidades y documentarlas para posteriormente asegurarse en arreglarlas.

- **Operation:**
  - **Gestión de Incidentes:** Saber como actuar ante incidentes de seguridad.
  - **Gestión de Entornos:** Asegurar que los entornos operativos sean seguros para su funcionamiento.
  - **Gestión Operativa:** Supervisar que todas las operaciones se mantengan en funcionamiento.

Sin embargo, su correcta implementación requiere un compromiso por parte del equipo que lo utilice, pues si bien es flexible, su adopción efectiva demanda tiempo, recursos y una planificación adecuada. Esto sin contar que los niveles de madurez más altos pueden ser difíciles de alcanzar sin la formación adecuada, lo que podría representar un desafío para equipos con poca experiencia en seguridad informática.

A pesar de estas limitaciones, *SAMM* sigue siendo una opción viable para mejorar la seguridad en el desarrollo de software, proporcionando un marco estructurado que permite una evolución constante y sostenible en las prácticas de seguridad.

- **BSIM**

  El BSIMM es un modelo de madurez diseñado para evaluar e implementar prácticas de seguridad en el desarrollo de software. Su enfoque se basa en el estudio y análisis de programas de seguridad existentes en diversas organizaciones, proporcionando un marco de referencia detallado para identificar las mejores prácticas adoptadas por empresas líderes en el sector.

  A diferencia de otros modelos más estructurados, BSIMM no impone un conjunto rígido de pasos, sino que funciona como una herramienta comparativa, ayudando a las organizaciones a evaluar su estado actual en términos de seguridad y a definir estrategias de mejora basadas en datos reales.

  Uno de los principales beneficios de BSIMM es su enfoque basado en evidencia, este se construye a partir del análisis de experiencias reales en empresas de diferentes industrias permitiendo a las organizaciones no solo adoptar prácticas de seguridad efectivas, sino también comprender cuáles son las estrategias más utilizadas por otras compañías con un alto nivel de madurez en seguridad.

  Incluyendo, su flexibilidad lo convierte en una herramienta adaptable a distintos entornos, pues no requiere que una empresa siga un camino específico, sino que le permite evolucionar de acuerdo con sus propias necesidades y capacidades, siendo mas útil para organizaciones que buscan un diagnóstico detallado de su seguridad, ya que proporciona métricas comparativas y ayuda a identificar áreas de mejora de manera objetiva.

  Sin embargo, BSIMM también presenta ciertas limitaciones. Al ser un modelo basado en la observación de prácticas existentes, no ofrece una guía específica sobre cómo implementar mejoras, lo que puede dificultar su adopción en empresas con poca experiencia en seguridad informática.

  Otro aspecto a considerar es que BSIMM está diseñado principalmente para empresas grandes y medianas, por lo que las organizaciones más pequeñas o con menos recursos pueden encontrar difícil su implementación debido a la falta de infraestructura y personal especiali-

zado.

El BSIMM se centra en cuatro dominios principales:

- **Gobernanza:** Establecer políticas de seguridad claras y mecanismos de supervisión.
- **Inteligencia de amenazas:** Analizar riesgos emergentes y definir controles de seguridad basados en amenazas específicas.
- **Seguridad del ciclo de desarrollo:** Integrar prácticas como revisiones de código, pruebas de penetración y herramientas automatizadas (SAST y DAST).
- **Capacitación y cultura:** Fomentar el aprendizaje continuo en seguridad y la sensibilización en todo el equipo de desarrollo.

Incluso con estas limitaciones, BSIMM sigue siendo una opción valiosa para empresas que desean evaluar su madurez en seguridad y mejorar sus procesos de manera estructurada. Su enfoque comparativo y basado en evidencia lo convierte en una herramienta útil para organizaciones que ya cuentan con una base en seguridad y buscan evolucionar hacia un modelo más robusto.

- **Microsoft SDL**

Según Shostack (2008) ”la metodología actual de modelado de amenazas en *Microsoft SDL* se basa en un proceso de cuatro pasos diseñado para que los ingenieros, incluso con un conocimiento limitado en seguridad, puedan identificar y evaluar amenazas de manera efectiva”. Este enfoque tiene como objetivo mejorar la seguridad de los diseños, documentar las actividades relacionadas con la seguridad y fomentar el aprendizaje continuo sobre protección de sistemas a medida que los desarrolladores avanzan en el proceso.

El primer paso en el uso de la herramienta de modelado de amenazas en *Microsoft SDL* consiste en definir el diseño del sistema de software. En esta fase, los desarrolladores pueden incorporar diversas entidades en la arquitectura, como procesos, almacenes de datos y

canales de comunicación. Sin embargo, el diagrama resultante no equivale a una especificación de diseño detallada, sino que sirve como una representación estructural que facilita la identificación de posibles amenazas desde un enfoque de seguridad.

Este marco de seguridad se basa en un conjunto de prácticas fundamentales que abarcan distintas fases del ciclo de desarrollo, donde en primer lugar, la capacitación en seguridad nos garantiza que los desarrolladores y evaluadores comprendan las amenazas más comunes y las estrategias para mitigarlas, donde luego, la definición de requisitos de seguridad permite establecer objetivos específicos desde la fase de diseño, asegurando que las decisiones arquitectónicas incluyan criterios de seguridad desde el inicio. Posteriormente, el diseño seguro implica la implementación de revisiones arquitectónicas y modelos de amenazas para prever riesgos antes de que se escriba el código.

Durante la fase de desarrollo, la implementación segura emplea herramientas como el SAST para detectar vulnerabilidades antes del despliegue. A esto le siguen pruebas rigurosas, que incluyen herramientas DAST, pruebas de penetración y técnicas como *fuzzing*, con el fin de evaluar la robustez del sistema en escenarios reales. Finalmente, el marco de seguridad de *Microsoft SDL* incorpora la respuesta ante incidentes, estableciendo protocolos de monitoreo continuo y planes de acción para gestionar posibles brechas de seguridad de manera eficiente.

Los procesos en los que se basa este conjunto de prácticas fundamentales serian:

- **Capacitación en seguridad:** Asegurar que los desarrolladores y evaluadores comprendan las amenazas comunes y las técnicas para mitigarlas.
- **Definición de requisitos de seguridad:** Identificar objetivos específicos y criterios de seguridad desde la fase de diseño.
- **Diseño seguro:** Implementar revisiones arquitectónicas y modelos de amenazas para prever riesgos antes de codificar.
- **Implementación segura:** Utilizar herramientas como análisis estático (SAST) para identificar vulnerabilidades en el código antes de su despliegue.

- **Pruebas rigurosas:** Aplicar análisis dinámico (DAST), pruebas de penetración y *Fuzzing* para evaluar la robustez del sistema.
- **Respuesta ante incidentes:** Establecer protocolos de monitoreo continuo y planes de acción para gestionar posibles brechas de seguridad.

Gracias a su enfoque sistemático y bien estructurado, *Microsoft SDL* ha sido ampliamente adoptado en la industria, destacando por su capacidad de integrarse en diferentes metodologías de desarrollo, incluidas las ágiles. Su implementación permite reducir significativamente los riesgos de seguridad en el software, promoviendo un desarrollo más seguro desde las primeras fases del proyecto.

3.2.1.5 SAMM para el caso de estudio

Como se mencionó anteriormente, la elección de este framework respondió a la necesidad de adaptarse a las condiciones específicas del proyecto. En primer lugar, el desarrollo del software ya había finalizado su fase inicial, lo que limitaba la implementación de ciertos modelos de seguridad desde sus primeras etapas, tomando en cuenta también que el equipo de trabajo era reducido y contaba con poca experiencia en la aplicación de medidas de seguridad en proyectos previos, lo que representaba un desafío adicional al momento de seleccionar un modelo adecuado.

Si bien tanto el *Microsoft SDL* como el BSIMM son marcos de referencia sólidos para la seguridad en el desarrollo de software, presentan limitaciones que los hacían inviables para este caso particular. *Microsoft SDL*, por ejemplo, es un modelo altamente estructurado y poco flexible, diseñado para integrarse desde el inicio del ciclo de vida del desarrollo de software. Su correcta implementación requiere un equipo con experiencia en seguridad informática, un requisito que nuestro equipo no cumplía. Debido a estas razones, su adopción resultaba poco factible.

Por otro lado, BSIMM, a pesar de ser más flexible y permitir la incorporación de prácticas de seguridad en distintas etapas del desarrollo, también demanda un equipo con experiencia en seguridad informática, siendo que este modelo está orientado principalmente a empresas medianas y grandes, lo que lo hace poco adecuado para nuestro contexto, ya que el software ANI es una

aplicación sin fines de lucro desarrollada por un grupo de estudiantes de un instituto sudamericano. Dadas estas consideraciones, BSIMM también fue descartado.

Finalmente, la opción más adecuada resultó ser *SAMM*, ya que cumple con todos los requisitos para su implementación en nuestro proyecto. Su flexibilidad permite aplicarlo en cualquier fase del desarrollo, incluso después de la etapa inicial. Este modelo es utilizado tanto por grandes corporaciones como por pequeñas, lo que demuestra su versatilidad. Lo más importante es que *SAMM* no solo fortalece la seguridad del software, sino que también proporciona un marco educativo para que el equipo adquiera conocimientos en seguridad informática, facilitando así su capacitación a lo largo del proyecto.

Un ejemplo claro de la aplicación del modelo *SAMM* se presenta en el estudio de Fucci et al. (2024), este documento proporciona información valiosa sobre las lecciones aprendidas y las dificultades encontradas durante la documentación e implementación del modelo en el proyecto designado. Aunque por razones de seguridad no se revela el nombre específico del proyecto analizado, el estudio destaca la importancia de involucrar múltiples roles en el proceso, ya que esto mejora la precisión de la evaluación y permite una visión más integral de los riesgos y controles de seguridad. Además, se resalta que *SAMM* facilita la identificación de sistemas de almacenamiento, permitiendo un mejor control sobre la seguridad de los datos. Asimismo, la adaptación de la terminología del cuestionario del modelo contribuye a una mejor comprensión y aplicación de los principios de seguridad dentro de la organización.

3.2.1.6 ¿Que es OWASP?

Opern Web Application Security Porject por sus siglas mas conocido como OWASP es una empresa creada sin fines de lucro, la cual esta dedica a mejorar la seguridad del software dando herramientas como su famosa lista de OWASP Top 10, esta lista sirve como punto de partida para que los equipos de desarrollo comprendan y mitiguen las amenazas más comunes en sus aplicaciones.

Además de OWASP Top 10, la organización ha desarrollado otros marcos de seguridad fundamentales, como ASVS, que establece un conjunto de requisitos para evaluar la seguridad de

aplicaciones, y *SAMM*, un modelo de madurez que permite a las organizaciones medir y mejorar sus prácticas de desarrollo seguro. También mantiene proyectos como OWASP *ZED ATTACK PROXY* (ZAP), una herramienta de análisis DAST utilizada para detectar vulnerabilidades en aplicaciones web en entornos de prueba y producción.

El enfoque abierto y colaborativo de OWASP permite que sus herramientas y metodologías sean utilizadas por empresas, gobiernos y comunidades de desarrolladores en todo el mundo. Su adopción en el SSDLC facilita la integración de prácticas de seguridad desde las primeras etapas del desarrollo, reduciendo el riesgo de ataques y asegurando que las aplicaciones cumplan con estándares de seguridad reconocidos a nivel internacional.

3.2.2. ¿Cómo se analiza la seguridad en aplicaciones?

Si bien el implementar SDLC ya nos ayuda a mitigar vulnerabilidades que podrían se usadas por atacantes, se debe primero definir todos los componentes que se van a utilizar como guías o herramientas desde diferentes perspectivas donde los enfoques principales que usaron para analizar la aplicación a fondo son:

- Threat Analysis
- Marcos y guías de detección
- Stride
- CID
- OWASP Top 10
- Herramientas Automáticas

Estos pasos son fundamentales en este tipo de proyecto, ya que son necesarios para crear una estructura eficiente.

3.2.2.1 Threat Analysis

La Evaluación de Amenazas o *Threat Assessment* es un proceso sistemático para identificar, analizar y evaluar posibles amenazas que pueden comprometer la seguridad de un sistema, aplicación o infraestructura. Su objetivo principal es comprender qué amenazas existen, cuál es su impacto potencial y cómo mitigarlas antes de que se conviertan en vulnerabilidades explotables. Este enfoque es esencial en el desarrollo seguro de software, ya que permite anticipar riesgos y aplicar controles de seguridad adecuados desde las primeras etapas del ciclo de vida del desarrollo.

El proceso de *Threat Assessment* se compone de varias fases clave que garantizan un análisis completo de los riesgos de seguridad:

1. **Identificación de Amenazas:** En esta fase, se analizan las posibles amenazas que pueden afectar al sistema, considerando: actores maliciosos, vectores de ataques y activos críticos.

2. **Análisis de Riesgo:** Una vez identificadas las amenazas, se evalúan en función del impacto potencial junto con la probabilidad de ocurrencia.

3. **Mitigación y Control:** Para reducir el riesgo de las amenazas identificadas, se aplican estrategias de defensa donde se utiliza el parcheo o actualizaciones para mitigar vulnerabilidades reconocidas, implementación de controles de seguridad como la autenticación multifactor o cifrado de datos y por ultimo tener planes de respuesta ante incidentes.

4. **Monitoreo y Actualización Continua:** Las amenazas suelen ir evolucionando de forma continua, es por ello que se recomienda monitorear constantemente las amenazas en busca de vulnerabilidades a mitigar.

Dentro del *SSDLC*, la Evaluación de Amenazas es un componente clave en la fase de diseño y pruebas, ya que permite anticipar riesgos y reducir la superficie de ataque antes de que el software sea implementado.

En el marco de OWASP *SAMM*, *Threat Assessment* forma parte del *Application Risk Profile*, una metodología para clasificar las aplicaciones en función de su nivel de riesgo y asignarles los

controles de seguridad adecuados según su criticidad. Esto permite que los equipos de desarrollo prioricen la mitigación de las amenazas más relevantes y optimicen el uso de los recursos de seguridad.

STRIDE

El modelo *STRIDE* fue desarrollado por Microsoft, es una metodología de modelado de amenazas utilizada para anticipar riesgos de seguridad en aplicaciones y sistemas. Se recomienda utilizarlo en las primeras etapas de desarrollo ya que enfoque permite identificar vulnerabilidades antes de que causen problemas, esto asegura que los riesgos sean mitigados. Su integración en el SSDLC permite fortalecer la seguridad del software a pesar de que la fase de desarrollo ya haya pasado lo cual es nuestro caso.

Este modelo clasifica las amenazas en seis categorías, cada una asociada a un tipo específico de ataque. La *suplantación de identidad* (Spoofing) ocurre cuando un atacante falsifica la identidad de un usuario o sistema para obtener acceso no autorizado, un problema común en ataques de phishing y credenciales comprometidas. La *manipulación de datos* (Tampering) se refiere a la modificación malintencionada de información, lo que puede comprometer la integridad del sistema y facilitar fraudes o alteraciones en bases de datos.

Otro riesgo importante es el *repudio de acciones* (Repudiation), donde un usuario puede negar haber realizado una transacción o una acción dentro del sistema sin que exista evidencia suficiente para demostrar lo contrario. La implementación de registros de auditoría y firmas digitales es una estrategia clave para evitar este tipo de amenazas. Por otro lado, *la divulgación de información* (Information Disclosure) ocurre cuando datos sensibles quedan expuestos de forma indebida, ya sea por configuraciones incorrectas, falta de cifrado o ataques como inyección SQL (SQLi), lo que puede derivar en filtraciones de datos críticos.

Las amenazas también afectan la disponibilidad del sistema, como en el caso de los ataques de *denegación de servicio*, los cuales buscan sobrecargar un sistema o servicio hasta hacerlo inaccesible. Para mitigar estos riesgos, se recomienda utilizar soluciones como balanceo de carga, protección contra bots y mitigación de tráfico malicioso con firewalls de aplicaciones web. Finalmente,

la *elevación de privilegios* permite a un atacante obtener permisos superiores a los asignados, explotando fallos en la autenticación o autorización del sistema.

Es por esto que aplicar *STRIDE* en el SSDLC es algo fundamental para evaluar la seguridad del software en cada fase del desarrollo. Al modelar amenazas desde la fase de diseño, es posible anticipar riesgos y definir estrategias de mitigación efectivas. Además, su integración con metodologías como *Threat Assessment* y OWASP *SAMM*, junto con herramientas de análisis de seguridad como SAST y DAST, permite validar la robustez del software antes de su despliegue en entornos de producción.

CID

En el contexto del SSDLC, la implementación de *CONTINUOUS INTEGRATION AND DEPLOYMENT* (CID) en SSDLC permite detectar y mitigar vulnerabilidades de manera proactiva, asegurando que el software desplegado cumpla con los estándares de seguridad requeridos.

Dentro de este marco, la Integración Continua *CONTINUOUS INTEGRATION* (CI) juega un papel crucial al automatizar la detección de fallos de seguridad en etapas tempranas. Esto se logra mediante la ejecución de pruebas de seguridad automatizadas, como SAST, DAST y *SOFTWARE COMPOSITION ANALYSIS* (SCA), cada vez que se realiza un nuevo commit en el repositorio. Además, la CI incluye escaneo de dependencias para identificar vulnerabilidades en bibliotecas y componentes de terceros, así como validaciones de código seguro antes de fusionarlo con la rama principal del proyecto.

Por otro lado, la Entrega y Despliegue Continuo *CONTINUOUS DEPLOYMENT* (CD) permite que las actualizaciones del software lleguen de manera automatizada a los entornos de prueba y producción sin comprometer la seguridad. En esta fase, se incorporan controles de seguridad previos al despliegue, como la verificación del cumplimiento de políticas de seguridad y el análisis de riesgos de la nueva versión.

La integración de CID en SSDLC es fundamental por varias razones. En primer lugar, reduce el riesgo de vulnerabilidades en entornos de producción al detectar fallos antes de que el software sea desplegado. Además, automatiza auditorías de seguridad sin ralentizar los ciclos de desarrollo,

lo que permite que la seguridad sea un proceso continuo en lugar de una fase aislada. También facilita la detección temprana de problemas de seguridad, evitando costos elevados asociados a la corrección de vulnerabilidades en etapas avanzadas del desarrollo. Finalmente, alinea el proceso de desarrollo con estándares de seguridad reconocidos, como OWASP *SAMM* y OWASP Top 10, garantizando que el software cumpla con las mejores prácticas del sector.

OWASP Top 10

Las vulnerabilidades presentes en las aplicaciones web pueden facilitar diversos tipos de ataques dirigidos contra los usuarios, generando consecuencias potencialmente severas. Por este motivo, resulta esencial identificar y analizar todas las debilidades que posibilitan dichos abusos, ya que, en numerosos casos, una mayor conocimiento sobre los riesgos representa el primer paso hacia la implementación de estrategias de protección más eficaces.

Nedeljković et al. (2020) señala que el marco proporcionado por el OWASP Top 10 se ha consolidado como una herramienta de referencia para evaluar y clasificar los riesgos de seguridad más críticos a los que se enfrentan actualmente las organizaciones. Usar este recurso no solo ofrece un diagnóstico detallado sobre las amenazas más comunes, sino que también actúa como una guía práctica para mitigar vulnerabilidades en entornos de desarrollo contemporáneos.

El OWASP Top 10 contiene una referencia esencial para identificar los diez riesgos de seguridad más utilizados y críticos al momentos en que se desarrollan las aplicaciones web. Esta lista, que se actualiza de manera periódica para abordar las amenazas emergentes, ha sido ampliamente adoptada como estándar cuando se trata sobre la seguridad informática.

Cada uno de los riesgos enumerados en el OWASP Top 10 proporciona un análisis detallado que incluye la descripción de la vulnerabilidad, los posibles impactos asociados, ejemplos concretos de ataques documentados y estrategias prácticas para mitigar dichas amenazas. Este enfoque estructurado convierte al OWASP Top 10 en una herramienta indispensable para fortalecer la seguridad en el desarrollo y mantenimiento de aplicaciones web.

Además de servir como una guía técnica para profesionales especializados en seguridad, el OWASP Top 10 también facilita el acceso a información crítica para desarrolladores en formación.

Al centrarse en los riesgos más comunes, esto ayuda a promover la adopción de buenas prácticas desde las primeras etapas del ciclo de desarrollo, fomentando un enfoque proactivo en la protección de sistemas.

Como ya se ha mencionado antes esta lista de OWASP se modifica periódicamente con el fin de reflejar la evolución de las vulnerabilidades que afectan a las aplicaciones a medida que surgen nuevas amenazas. Esto permite que el OWASP Top 10 se mantenga actualizado y continúe siendo una herramienta relevante para abordar los desafíos de seguridad en un entorno tecnológico en constante transformación ya que con cada actualización de la lista incorpora un análisis exhaustivo de las tendencias emergentes, basado en datos recopilados a nivel global sobre incidentes de seguridad y vulnerabilidades reportadas.

De esta manera, el OWASP Top 10 no solo identifica los riesgos más críticos, sino que también prioriza aquellos que tienen el mayor impacto potencial en la integridad, confidencialidad y disponibilidad de las aplicaciones.

Este proceso de revisión y ajuste garantiza que la lista siga siendo una referencia indispensable para desarrolladores y profesionales de seguridad, quienes pueden emplearla fortaleciendo sus prácticas y mitigar eficazmente las amenazas más recientes.

1. **Broken Access Control (A01:2021)** ha emergido como el riesgo mas crítico, ya que afecta al 3.81 % de las aplicaciones evaluadas y abarca diversas debilidades asociadas al control de accesos.

2. **Cryptographic Failures (A02:2021)**, anteriormente conocida como ”Sensitive Data Exposure”, ahora se concentra en problemas específicos de criptografía, ubicándose en el segundo lugar de la lista.

3. **Inyección (A03:2021)** aunque descendió al tercer puesto, continúa siendo una categoría de vulnerabilidad prevalente y de impacto considerable.

4. **Insecure Design (A04:2021)** es una nueva inclusión que aborda riesgos derivados del diseño de sistemas, los cuales no pueden ser resueltos únicamente mediante una implementación correcta.

5. **Security Misconfiguration (A05:2021)** ha escalado posiciones debido a que afecta aproximadamente al 90 % de las aplicaciones, principalmente por errores de configuración.

6. **Vulnerable and Outdated Components (A06:2021)** subió desde el noveno lugar en 2017, reflejando el uso generalizado de componentes desactualizados o vulnerables.

7. **Identification and Authentication Failures (A07:2021)** se mantiene en la lista, ahora ampliada para incluir fallos tanto en la identificación como en la autenticación de usuarios.

8. **Software and Data Integrity Failures (A08:2021)** es otra nueva categoría, enfocada en la integridad del software y de los datos.

9. **Security Logging and Monitoring Failures (A09:2021)** ha ganado relevancia, subrayando la importancia de contar con registros y monitoreo efectivos para detectar y mitigar incidentes de seguridad.

10. **Server-Side Request Forgery (SSRF) (A10:2021)** fue incluida tras una encuesta en la comunidad, debido a su potencial de explotación y creciente relevancia en el panorama de amenazas.

3.2.2.2 Herramientas automáticas

SAST

El análisis estático de seguridad de aplicaciones, comúnmente conocido como SAST, constituye una herramienta esencial para la detección automatizada de vulnerabilidades en el código fuente. Su integración en los sistemas de distribución continua permite identificar fallos en una etapa temprana del desarrollo, lo que contribuye significativamente a la reducción de riesgos de seguridad.

Según Eterovic et al. (2023), ”la principal ventaja de implementar prácticas de SAST es que ayuda a los equipos de desarrollo a evitar amenazas conocidas, protegiendo aplicaciones web, API y aplicaciones móviles contra vulnerabilidades potenciales”. Esta capacidad de análisis proactivo

no solo mejora la seguridad general del software, sino que también promueve el desarrollo de aplicaciones más robustas y resilientes desde sus primeras fases.

Otra función seria que al centrarse en el análisis del código fuente sin requerir su ejecución, el SAST permite abordar vulnerabilidades antes de que se materialicen en entornos de producción. Esta característica lo convierte en una solución importante para fortalecer las prácticas de seguridad en el ciclo de desarrollo, facilitando la detección temprana de errores de codificación y configuraciones inseguras que podrían ser explotadas por atacantes.

Eterovic et al. (2023) menciona que los análisis pueden variar en complejidad, desde aquellos más rápidos y simples, que revisan solo la estructura básica del código, hasta métodos avanzados que emplean grafos de control y flujo de datos para un análisis semántico profundo en busca de patrones de riesgo. La elección de una herramienta de SAST adecuada dependerá de factores como la velocidad y la profundidad del análisis, el porcentaje de falsos positivos generados y la compatibilidad con el lenguaje de programación en uso.

Las herramientas de SAST permiten a los equipos ahorrar tiempo y recursos, particularmente cuando se comparan con la detección de vulnerabilidades en fases avanzadas del ciclo de vida de desarrollo, donde la incorporación de medidas de seguridad se vuelve más compleja y costosa.Eterovic et al. (2023) destaca algunas ventajas clave de las herramientas SAST:

- **Escalabilidad:** estas herramientas soportan diversos lenguajes de programación y pueden ejecutarse de forma recurrente en sistemas de integración continua, como en compilaciones nocturnas.

- **Identifica ciertas vulnerabilidades bien conocidas, tales como:** son efectivas en la detección de fallos como desbordamientos de búfer e inyección **SQL!** (**SQL!**).

- **Soporte a los desarrolladores:** SAST señala problemas específicos en el código, indicando el archivo, la ubicación exacta, el número de línea e incluso el fragmento problemático, lo cual facilita la corrección.

- **Debilidades:**

- No obstante, Eterovic et al. (2023) también subraya algunas limitaciones de las herramientas SAST:
    - **Dificultad para automatizar ciertos tipos de vulnerabilidades:** tales como problemas de autenticación, control de acceso y criptografía insegura.
    - **Limitación en el alcance de detección:** pueden identificar solo un porcentaje reducido de las fallas de seguridad presentes en las aplicaciones.
    - **Falsos positivos:** estas herramientas suelen generar un volumen considerable de alertas que no corresponden a vulnerabilidades reales.
    - **Problemas de configuración:** no logran identificar problemas de configuración, ya que estos no están reflejados directamente en el código.
    - **Compilación limitada del código:** SAST puede tener problemas al analizar código que no compila correctamente, y el análisis puede verse obstaculizado si los analistas no cuentan con las bibliotecas, instrucciones de compilación y todo el código necesario.

Si bien las herramientas de SAST son un recurso valioso para identificar vulnerabilidades desde las primeras fases del desarrollo, aunque su efectividad depende de un equilibrio entre su capacidad de análisis, el control de falsos positivos y la compatibilidad con el entorno específico de desarrollo.

DAST

”Las pruebas de análisis dinámico de seguridad de aplicaciones, conocido como DAST, se diferencia de las pruebas estáticas SAST al enfocarse en la detección de vulnerabilidades en tiempo real, es decir, mientras la aplicación está en funcionamiento”, según Caño Quintero (2019). Este enfoque permite observar cómo la aplicación maneja entradas y respuestas en un contexto activo, revelando vulnerabilidades que no serían visibles en el código fuente.

Una ventaja clave de DAST es precisamente su capacidad para identificar problemas que no se muestran en el código, lo que la convierte en una herramienta útil para evaluar configuraciones, conexiones y posibles fallos en autenticación o autorización en tiempo real. No obstante, debido

a la necesidad de un entorno activo, el uso de DAST demanda más recursos y una configuración detallada en comparación con otras metodologías.

DAST generalmente se aplica después de que la aplicación ha pasado a producción, ya que su proceso emula ataques reales sobre una aplicación en ejecución. Sin embargo, al incorporar DAST en etapas tempranas del desarrollo, las organizaciones pueden detectar y mitigar vulnerabilidades anticipadamente, lo cual supone un ahorro significativo en tiempo y costos.

Un ejemplo destacado de herramienta DAST es OWASP ZAP, la cual permite identificar vulnerabilidades de seguridad mediante el uso de políticas de escaneo predefinidas que equilibran la rapidez del análisis con el cumplimiento de los requisitos organizacionales. Esta capacidad hace que la herramienta sea altamente eficiente para evaluar la seguridad de aplicaciones web en entornos dinámicos y en constante evolución.

3.2.2.3 Aplicaciones

Para una buena implementación del SSDLC, los principal seria reconocer cuales fueron las herramientas que se utilizaron al momento de desarrollar la aplicación ANI, el club no dio una lista de las herramientas que utilizaron en el desarrollo, tomando en cuenta la escalabilidad y el rendimiento necesarios para este tipo de proyecto.

En el desarrollo de APIs, se optó por NestJS, un framework basado en Node.js que proporciona una arquitectura modular y escalable. Su elección se debe a su compatibilidad con TypeScript, su robusto sistema de inyección de dependencias y su capacidad para estructurar aplicaciones en microservicios, facilitando el mantenimiento del código.

Para complementar la gestión de microservicios, se implementó FastAPI, un framework de alto rendimiento basado en Python. Su elección se fundamenta en su rapidez y facilidad para la creación de APIs RESTful, además de su compatibilidad con validación de datos mediante Pydantic.

En el desarrollo de la interfaz web, se utilizó Vite React, una herramienta que ofrece una experiencia de desarrollo rápida y optimizada en comparación con los entornos tradicionales de React.

Para el desarrollo móvil, se optó por Expo, un entorno basado en React Native que facilita la creación de aplicaciones para múltiples plataformas sin necesidad de configuraciones complejas.

En la siguiente *Tabla* 3.6 se describe el propósito para el cual fueron hechos de esta forma:

Tabla 3.6: Tabla de Herramientas

Lista de Herramientas Utilizadas		
Categoría	**Herramienta**	**Propósito**
API	NestJS	Se utilizó para la autenticación de usuarios.
API-WebSockets	NestJS	Gestiona conversaciones con WebSockets.
API	FastAPI	Fue empleado en la administración de transcripciones y grafos.
Web	Vite React	Mayor control del sistema para administradores y docentes.
Mobile	Expo	Para uso de estudiantes.

3.2.3. Como se mide

Para garantizar un enfoque estructurado en la identificación y documentación de vulnerabilidades dentro del proyecto, se adoptaron diversas metodologías y herramientas alineadas con el Secure Software Development Life Cycle (SSDLC). A pesar de que hasta el momento no se han aplicado medidas de seguridad concretas, la documentación de vulnerabilidades a través de herramientas SAST y DAST ha permitido obtener una visión detallada del estado actual de la seguridad en la aplicación.

3.2.3.1 Como se aplico SAMM

Para garantizar una implementación efectiva de OWASP SAMM, se establecieron varias estrategias enfocadas en la documentación de riesgos en *Google Drive*, la capacitación del equipo y la adopción de herramientas de análisis de seguridad.

Inicialmente, se realizaron evaluaciones para determinar el nivel de madurez en seguridad del equipo de desarrollo, identificando deficiencias en la gestión de vulnerabilidades y en la aplicación de pruebas de seguridad. Posteriormente, se utilizaron herramientas como *Microsoft Threat*

Modeling Tool (MTMT) para estructurar el análisis de amenazas mediante el modelo *STRIDE*, permitiendo una mejor identificación de riesgos y facilitando la priorización de mitigaciones.

También se realizaron revisiones de código automatizadas con SonarQube y pruebas de seguridad dinámica con OWASP ZAP para reforzar la detección de vulnerabilidades en las distintas etapas del ciclo de desarrollo. Con estas medidas, *SAMM* no solo proporcionó un marco para fortalecer la seguridad del software, sino que también estableció una base educativa para el equipo.

3.2.3.2 OASV estandard

El *OWASP APLICACTION SECURITY VERIFICATION* (OASV) es un estándar desarrollado por la entidad OWASP en donde se establece un conjunto de requisitos para evaluar la seguridad de aplicaciones web y móviles. Su propósito es proporcionar una guía estructurada que permita a los desarrolladores, arquitectos y equipos de seguridad verificar que una aplicación cumple con prácticas de seguridad sólidas.

OASV define diferentes niveles de verificación según el grado de seguridad requerido por una aplicación. Estos niveles permiten adaptar el estándar a distintos escenarios:

- Nivel 1 (ASVS L1): Enfocado en aplicaciones de bajo riesgo, asegurando que no existan vulnerabilidades comunes como las identificadas en OWASP Top 10.
- Nivel 2 (ASVS L2): Destinado a aplicaciones que manejan información sensible, incluyendo controles avanzados de autenticación, gestión de sesiones y cifrado.
- Nivel 3 (ASVS L3): Diseñado para aplicaciones críticas en términos de seguridad, como sistemas financieros o gubernamentales, exigiendo un alto grado de protección y verificaciones exhaustivas.

Este estándar abarca múltiples áreas de seguridad, incluyendo control de acceso, protección de datos, seguridad en autenticación, gestión de sesiones, criptografía y seguridad en APIs. Además, su integración con prácticas como Threat Modeling y SSDLC nos permitió identificar las vulnerabilidades desde las primeras fases del desarrollo.

La adopción de OASVS en un proyecto proporciona una base sólida para la evaluación de seguridad, asegurando que la aplicación cumple con los estándares recomendados y minimizando los riesgos de explotación de vulnerabilidades. Su implementación junto con herramientas de análisis estático (SAST), dinámico (DAST) y evaluaciones de amenazas, nos demostró como se puede reforzar la seguridad del software.

3.2.3.3 Documentación

Para gestionar y centralizar la documentación de seguridad, se optó por Google Drive como repositorio principal, esto es debido a que la plataforma permitió almacenar de manera organizada los informes generados durante el análisis de seguridad, asegurando el acceso controlado a la información y permitiendo la colaboración con el club de desarrollo.

El uso de Google Drive también facilitó la gestión de permisos, permitiendo restringir modificaciones y garantizando la integridad de los documentos almacenados. Con esto su compatibilidad con distintos formatos y su capacidad de sincronización facilitaron la actualización de la información sin necesidad de implementar una infraestructura de almacenamiento adicional.

Los documentos principales almacenados en la plataforma incluyeron todas las fases del modelo OWASP *SAMM* junto con sus subdivisiones, permitiendo estructurar y documentar cada etapa del proceso de evaluación de seguridad, asegurando las políticas de seguridad, así facilitando su acceso para el club de desarrollo.

Uno de los documentos clave almacenados fue el de *Threat Modeling*, dado que el club de desarrollo utilizaría esta información para la documentación de vulnerabilidades identificadas en análisis futuros. Este archivo fue estructurado de manera que, al calificar una vulnerabilidad, se pudiera hacer referencia a la lista de OWASP Top 10 o al modelo *STRIDE*, permitiendo clasificar con precisión cada amenaza según su tipo y gravedad.

Microsoft treaht modeling tool

Una herramienta que fue útil para el proyecto es el *MICROSOFT TREAHT MODELING TOOL* (MTMT) *Figura* 5.25, una herramienta útil para identificar, analizar y mitigar riesgos de seguridad

de manera estructurada. Su capacidad de alinearse con los modelos OWASP *SAMM* y *STRIDE* representa una ventaja significativa, ya que facilita la integración de prácticas de seguridad dentro del Secure Software Development Life Cycle (SSDLC).

Su enfoque visual e intuitivo permite la creación de diagramas de flujo de datos, facilitando el análisis automatizado de posibles vulnerabilidades en cada componente del sistema. Al estar basado en el modelo *STRIDE*, *Microsoft Threat Modeling Tool (MTMT)* permite identificar amenazas relacionadas con suplantación de identidad (Spoofing), manipulación de datos (Tampering), repudio de acciones (Repudiation), divulgación de información (Information Disclosure), denegación de servicio (DoS) y elevación de privilegios (Elevation of Privilege), proporcionando una evaluación estructurada de los riesgos.

Una herramienta principal del *Microsoft Threat Modeling Tool (MTMT)* es que ofrece una biblioteca de plantillas y reglas de seguridad, lo que simplifica la evaluación de amenazas en distintos entornos, incluyendo aplicaciones web, servicios en la nube y sistemas embebidos, esto junto con su capacidad de personalización de modelos de amenazas permitió adaptar el análisis a las necesidades específicas del proyecto, optimizando la detección de vulnerabilidades y fortaleciendo la seguridad del sistema desde las primeras fases del desarrollo.

3.2.3.4 Moodle para capacitación

Una parte importante del modelo OWASP *SAMM* enfatiza la importancia de que los equipos de desarrollo adquieran conocimientos en ciberseguridad, algo fundamental para garantizar la implementación de prácticas seguras en el ciclo de vida del desarrollo de software. Tomando en cuenta que el club de desarrollo, compuesto mayoritariamente por estudiantes, tenía un conocimiento limitado en esta área. Aunque algunos integrantes contaban con experiencia previa, la mayoría carecía de una formación sólida en seguridad, esto sumando a la disponibilidad restringida de tiempo de los estudiantes, se optó por una estrategia de aprendizaje en línea utilizando Moodle como plataforma educativa.

Moodle o *Modular Object-Oriented Dynamic Learning Environment* es una plataforma de gestión de aprendizaje de código abierto ampliamente utilizada en entornos educativos y corporativos.

Su diseño flexible permitió personalizar los contenidos de formación de acuerdo con las necesidades específicas del equipo, enfocándose en la enseñanza de conceptos clave de ciberseguridad sin requerir el desarrollo de una plataforma desde cero.

De las características que mas nos llamo la atención, fue su capacidad de evaluación junto con el seguimiento del progreso del estudiantes lo que nos ayudara a documentar su progreso. Junto con su interfaz intuitiva permitió una rápida adopción por parte de los participantes, asegurando una experiencia de formación eficiente sin consumir tiempo excesivo en tareas administrativas o técnicas.

En el contexto del proyecto, Moodle es una excelente opción entre otras herramientas parecidas, ya que nos permitió estructurar un curso basado en estándares de seguridad reconocidos, incluyendo OWASP Top 10, seguridad en aplicaciones web, móviles y APIs, fundamentos de ciberseguridad, OWASP ASVS, *MASVS* y OWASP *SAMM*. Los módulos de aprendizaje se complementaron con material audiovisual, evaluaciones automatizadas y charlas con el equipo, facilitando la retroalimentación.

CAPÍTULO III: METODOLOGÍA DE INVESTIGACIÓN

4.1. Metodología del Estudio

El presente estudio adopta un enfoque metodológico híbrido, combinando técnicas cualitativas y cuantitativas para fortalecer el desarrollo seguro de software y evaluar la seguridad de las aplicaciones en distintos entornos. Para ello, se han tomado como referencia marcos y modelos reconocidos en el ámbito de la ciberseguridad, como el SAMM, así como múltiples recursos de la OWASP, incluyendo *OWASP Top 10*, *OWASP API Security Top 10* y *OWASP Mobile Security Top 10*.

Utilizando *OWASP ZAP*, una herramienta automatizada de análisis de seguridad para aplicaciones web, con el propósito de complementar la evaluación de vulnerabilidades y obtener un diagnóstico más preciso sobre las amenazas presentes en el software en estudio.

Desde el punto de vista metodológico, la investigación integra un enfoque cualitativo y cuantitativo. En el análisis cualitativo, se examina el grado de adopción de prácticas de seguridad dentro del equipo de desarrollo, mientras que en el análisis cuantitativo se mide la incidencia de vulnerabilidades detectadas a lo largo del proceso. Para evaluar la evolución de estos factores, se realizaron mediciones en dos momentos distintos, con un intervalo de un mes entre ellas, permitiendo así una comparación efectiva de los resultados obtenidos.

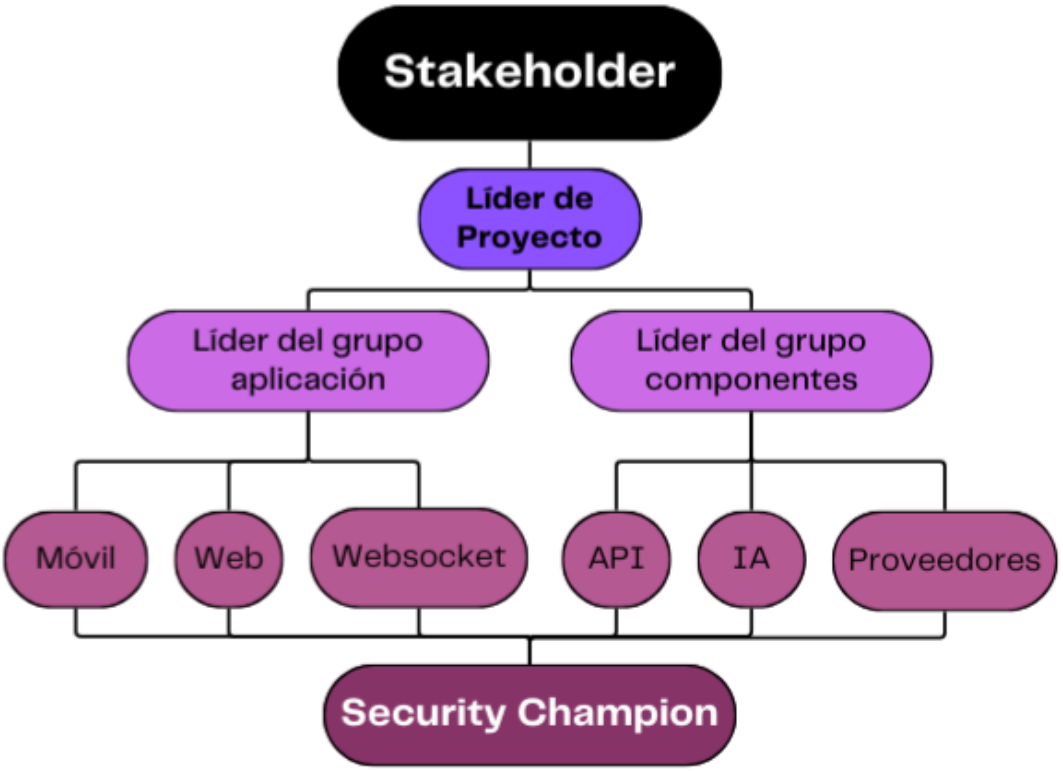
4.2. Recolección de datos

Dado el carácter cuantitativo del estudio, se diseñó una encuesta dirigida al equipo de desarrollo del proyecto, conformado por los siguientes roles:

- **Stakeholder:** Es la persona, grupo o entidad que tiene un interés en un proyecto que puede afectar o ser afectado por el mismo. Estos desempeñan un papel importante, ya que su apoyo, influencia y satisfacción son clave para el éxito del proyecto que se realiza.
- **Líder de Proyecto:** Es el responsable de planificar, ejecutar y supervisar un proyecto para garantizar que se cumplan los objetivos establecidos, dentro de los límites de tiempo, costo y calidad definidos.

Figura 4.1

Organigrama de Roles y Responsabilidades.



- **Líder del grupo aplicación:** El guía al grupo que se encarga de los repositorios del Móvil, Web y Websocket.
- **Líder del grupo componentes:** El guía al grupo que se encarga de los repositorios de la IA, API y Proveedores.
- **Security Champion:** Este miembro es el designado de un equipo de desarrollo que asume la responsabilidad de actuar como enlace entre el equipo de desarrollo y el equipo de seguridad, en este caso esta persona sera el encargado de realizar todos los papeles del equipo de seguridad.
- **Desarrolladores:** Conformado por 6 estudiantes que están repartidos equitativamente entre los dos lideres de grupos.

Esto con el objetivo de evaluar tanto la preparación técnica como las prácticas de seguridad implementadas y planificadas a lo largo del proceso de desarrollo. La encuesta se llevó a cabo de

manera virtual y se estructuró en torno a un conjunto de preguntas específicas que permitieron indagar sobre conocimientos previos en ciberseguridad, utilizando la plataforma de Google que se muestra en el *Cuestionario de conocimientos* **??** como un medio para recolectar la información.

Repasando puntos anteriores, de todos los modelos *SSDLC* que se investigo, se escogió a *SAMM*, debido a que cumple con todos los siguientes puntos establecidos en la justificación 2.1.2:

- **Implementación a SDLC:** Esta diseñado para integrarse directamente al *SDLC* tradicional, ademas brinda practicas y actividades especificas cada cada etapa del ciclo de desarrollo.
- **Niveles Madurez:** Permite comenzar con practicas básicas y avanzas hacía mas sofisticadas y completas, ofrece tres niveles de madurez.
- **Guías Flexibles:** Es un marco abierto y flexible, no impone ningún tipo de herramientas o estándares a seguir, sino proporciona practicas genéricas que pueden adoptarse a distintos contextos organizacionales.
- **Rol de Seguridad:** Promueve la creación de un equipo de seguridad centralizado, este fomenta la colaboración entre el equipo de seguridad y los desarrolladores.

Analizando los puntos anteriores, el modelo *SAMM* cumple con todos los puntos y requerimientos necesarios determinados en 2.1.2, es mas importante a destacar es el nivel de flexibilidad que ofrece, para seleccionar tanto herramientas o estándares a seguir, esto implica que funciona correctamente dentro de Instituciones ya que se puede moldear todas las implementaciones de como que sean alcanzables, ya que requiere cumplir puntos muy complejos o costosos que pueden llegar a ser inalcanzables.

4.2.1. Fases de la Metodología

Para garantizar un análisis integral, la metodología se estructuró en cuatro fases fundamentales, diseñadas para abordar los principales desafíos en materia de seguridad durante el ciclo de desarrollo de software:

- **Evaluación inicial de riesgos y vulnerabilidades**

  En la primera fase, se lleva a cabo un análisis exhaustivo del estado actual de seguridad de la aplicación ANI, con el propósito de identificar vulnerabilidades y puntos críticos susceptibles de explotación. Este proceso se basa en las directrices establecidas por *OWASP SAMM*, tanto para aplicaciones web, las diversas *API*s, la aplicación móvil y el proveedor. Dentro de este modelo se adjuntan otros modelados de amenazas, como es toda la *OWASP* y también el uso de *STRIDE*.

  El diagnóstico contempló una evaluación minuciosa de las vulnerabilidades presentes en la aplicación, así como de los riesgos potenciales asociados, examinando el grado de integración de prácticas de seguridad dentro del ciclo de vida del desarrollo de software (*SDLC*), con el fin de detectar posibles brechas donde no se hayan implementado controles de protección adecuados.

  También se identificaron y priorizaron las áreas críticas que requerían medidas de mitigación, sentando así las bases para las acciones correctivas que se desarrollaron en las siguientes fases de la metodología. Este enfoque permite establecer un panorama claro de los riesgos existentes y orientar las estrategias para fortalecer la seguridad de la aplicación.

- **Implementación de políticas y estrategias de seguridad**

  La fase dos se enfoco más en la integración de prácticas y herramientas diseñadas para fortalecer la seguridad en todas las etapas del desarrollo de la aplicación. Este propósito fue para garantizar que la seguridad no sea un elemento añadido posteriormente, sino un componente inherente al proceso desde su concepción.

  Todo esto siguiendo el enfoque de Security by Design, donde se aplicaron los principios de diseño seguro desde las fases iniciales del desarrollo. Incluyendo actividades como el modelado de amenazas y la revisión de arquitecturas.

  Asimismo, se implementaron dos herramientas de análisis estático (SAST), con el fin de identificar vulnerabilidades tanto en el código fuente como en el entorno de ejecución de

la aplicación. Estas herramientas permiten detectar y corregir errores antes del despliegue, minimizando así los riesgos asociados a fallos de seguridad.

Para ello el equipo de desarrollo tuvo que recibir capacitación específica en técnicas para anticipar posibles vectores de ataque, lo que facilita el diseño de defensas personalizadas contra amenazas potenciales.

Finalmente, se llevaron a cabo varias pruebas manuales junto con automatizadas, orientadas a descubrir vulnerabilidades críticas que podrían pasar inadvertidas en los análisis automatizados. De esta manera, se valida la eficacia de los controles implementados, garantizando un nivel adicional de protección antes de la puesta en producción.

- **Integración de prácticas seguras en el ciclo de desarrollo**

  En esta fase, se estableció un modelo framework SAMM junto con el ciclo de vida de desarrollo seguro (SSDLC) diseñado para guiar al equipo en la adopción de prácticas de seguridad a lo largo de cada etapa del proceso de desarrollo de software, asegurando que la seguridad esté integrada de manera continua desde la planificación hasta las operaciones dando como resultado un entorno resistente frente a amenazas potenciales.

Planificación y Diseño

Durante la etapa inicial, se incorporan revisiones de arquitecturas seguras y modelado de amenazas para identificar riesgos desde las primeras fases del proyecto. Definiendo requisitos de seguridad claros, como los controles de acceso, los mecanismos de autenticación y políticas de autorización, esto dará una base sólida para el desarrollo seguro. Las herramientas usadas para realizar esta etapa:

- **Drive:** Usamos esta plataforma para almacenar toda la documentación necesaria del proyecto. Esto nos permitió tener acceso centralizado, organizado y seguro a la información clave.

- **Documentación OWASP SAMM:** Nos basamos en la documentación de OWASP SAMM para desarrollar modelos que nos permiten analizar las deficiencias de la aplicación.

Desarrollo

En esta etapa, es donde se implemento los estándares de codificación segura minimizando las vulnerabilidades en el código fuente, se logro utilizando principalmente herramientas de análisis estático (SAST) para validar el código, permitiendo la detección temprana de errores antes de la compilación. Las herramientas utilizadas son:

- **SonarQube:** Esta es una plataforma de código abierto que permite identificar problemas con respecto al código, la seguridad y sus vulnerabilidades, lo usamos en su gran mayoría para revisar todos los sistemas que complementan del *Sistema Integrado ANI*.
- ***Mobile Security Framework* (MobSF):** Una herramienta usada principalmente para el análisis en aplicaciones móviles, esta herramienta fue la principal en analizar las vulnerabilidades de la aplicación móvil del *Sistema Integrado ANI*.

Pruebas

Las pruebas de seguridad son parte integral en el proceso de aseguramiento en la implementación de seguridad, con eso en mente se utilizo las herramientas automatizadas que se han comentado en el punto anterior para la evaluación de la seguridad en la aplicación, adicionalmente, se realizaron otros tipos de pruebas, como fuzz testing usado con las herramientas Schemathesis y OWASP ZAP, esto para identificar vulnerabilidades que podrían no ser detectadas por herramientas automatizadas, garantizando así una evaluación más exhaustiva.

- **Schemathesis:** Es una herramienta diseñada principalmente para verificar si una API es segura frente a entradas inesperadas. La utilizamos en conjunto con la API de ANI con el propósito de identificar posibles errores y mejorar su seguridad.

- **OWASP ZAP:** Por ultimo se utilizo la herramienta recomendad por *SAMM*, esta fue usada principalmente para reconocer las vulnerabilidades del websocket, enviando inyecciones SQL.

- **Capacitación continua del equipo de desarrollo**

  En esta etapa final, el objetivo principal fue el de preparar al club de desarrollo a que adquiera las habilidades necesarias para implementar prácticas de seguridad sólidas en cada fase del proyecto.

  En la capacitación se incluyen técnicas avanzadas de programación segura, con un enfoque especial en prevenir vulnerabilidades comunes, como las inyecciones de código, la exposición de datos sensibles y los problemas en la gestión de sesiones.

  Para lograr esto el club recibió formación para reconocer y afrontar amenazas emergentes, mejorando su capacidad de anticipación y respuesta ante nuevas vulnerabilidades y posibles ataques, esta formación se lograr gracias a la plataforma *Moodle*, que permite gestionar los contenidos educativos, ofrecer acceso a materiales de aprendizaje y evaluar el progreso del equipo. Esta herramienta fue indispensable ya que ayudo a organizar los módulos, videos y cuestionarios que implementamos en el curso.

  En conjunto al curso de seguridad se dieron charlas donde se tocaron temas importantes, como:

  - **Desarrollo y funcionalidades:** En la primera reunión 8.1.1, se establecieron las bases del *Sistema Integrado ANI* en sus versiones web y móvil, abordando su funcionamiento, gestión de usuarios y salas de debate. Se identificaron áreas de mejora para optimizar la experiencia y control del docente.
  - **Infraestructura y escalabilidad:** La segunda reunión 8.1.2 se enfocó en la arquitectura técnica, seguridad y crecimiento del sistema. Se discutió los activos tecnológicos, control de accesos, manejo de información institucional y etapas de desarrollo.

- **Seguridad en el desarrollo:** La tercera 8.2.1y cuarta reunión 8.2.2 se profundizo en la seguridad, analizando la metodología *SAMM* la cual fue elegida para integrar en la seguridad de la aplicación, mostrando también documentos que ayudaran en la identificación de amenazas y gestión de dependencias.

- **OWASP Estandard**

  No existe ningún tipo de estándares, por lo cual hemos elegido uno que se adopta de mejor manera al sistema: Para asegurar el entorno de la aplicación, se ha aplicado *OASV*, donde su principal función es el de crear una lista de características a tomar en cuenta antes de desplegar estos proyectos, este se basa en niveles de madurez por lo cual es aplicable para cualquier tipo de sistemas, y se ajusta perfectamente a nuestro contexto por el mismo motivo.

  Las practicas de desarrollo seguro como te conceptos básicos como Autenticación y Autorización están presentes, aunque muchas veces pueden estar mal implementados o no validadas, también existe el factor de que luego de la realización de un *Threat Modeling* puede existir puntos faltantes o que se pasaron por alto, el uso de estándares cubre estas características ya que están realizadas por personal con conocimientos y experiencia.

  A partir de estos resultados y siguiendo el modelo SAMM como referencia, se desarrolló un curso en la plataforma *Moodle*. Este curso tiene como objetivo proporcionar información básica, junto con videos complementarios, para guiar a los desarrolladores en la correcta aplicación de medidas de seguridad dentro del proyecto.

CAPÍTULO IV: PROPUESTA

5.1. Descripcion Aplicaciones

Mediante las primeras entrevistas y encuestas realizadas, se obtiene una clara visión del *Sistema Integrado ANI*, esta fue diseñada para un uso en la institución educativa como una plataforma de debate en tiempo real, permitiendo la interacción estructurada entre estudiantes y docentes. Su funcionamiento se basa en la creación de cursos y salas de debate, donde los docentes actúan como administradores, configurando los espacios de discusión y gestionando la participación de los estudiantes. Un aspecto diferenciador del *Sistema Integrado ANI* es la integración de una *inteligencia artificial* encargada de regular el tiempo de participación de cada estudiante y evaluar su desempeño en función de su intervención en el debate.

El ecosistema de la aplicación está compuesto por múltiples componentes que trabajan de manera integrada. La interfaz web, desarrollada para docentes y administradores, permite gestionar usuarios, cursos y salas de debate. Diversas apis (AniApi 9.0.1, AniWeb 9.0.2, AniMobile 9.0.3, AniWebSockets 9.0.4, AniTransciption 9.0.5), implementadas con NestJS y FastAPI, maneja la autenticación de usuarios, la administración de topics y la gestión de las salas de debate. La arquitectura también incluye WebSockets, que constituye el núcleo de la comunicación en tiempo real, asegurando la correcta asignación de turnos y la grabación del audio durante los debates. Por otro lado, la aplicación móvil es la herramienta principal utilizada por los estudiantes para acceder a los debates y participar en las discusiones. Finalmente, el módulo de transcripción convierte el audio generado en las sesiones en texto, permitiendo la generación de grafos para el análisis de la interacción.

La primera fase de la integración del *SSDLC* en el *Sistema Integrado ANI* consistió en una reunión con el equipo de desarrollo, representado por el club de desarrollo, con el fin de analizar las herramientas tecnológicas utilizadas en la aplicación. Se identificó que, si bien se implementaban medidas de seguridad básicas, estas no eran suficientes, ya que la aplicación fue desarrollada con un enfoque en cumplir con los plazos de entrega, dejando múltiples áreas vulnerables que podrían ser explotadas por atacantes.

Para fortalecer la seguridad de la aplicación, se evaluaron varios frameworks de seguridad compatibles con el SSDLC. BSIMM y Microsoft SDL fueron dos de los modelos considerados, sin embargo, la implementación de Microsoft SDL requería haber sido aplicada desde las primeras fases del desarrollo, mientras que BSIMM estaba diseñado para equipos con experiencia avanzada en ciberseguridad, lo que dificultaba su adopción en un entorno conformado mayoritariamente por estudiantes.

Dado el contexto del proyecto y sus limitaciones, se optó por *SAMM*, un modelo de madurez en seguridad desarrollado por *OWASP*, reconocido por su enfoque práctico en la documentación y mejora progresiva de la seguridad en el desarrollo de software. *SAMM* proporcionó una estructura clara para documentar vulnerabilidades y definir estrategias de mitigación, alineando el proyecto con buenas prácticas en ciberseguridad sin requerir un equipo con conocimientos especializados avanzados.

5.1.1. Modelos de SAMM

1. Governance

Crear y promover (Create and Promote)

Para fortalecer la seguridad de la aplicación ANI, fue esencial comprender los riesgos a los que está expuesta y cómo la organización tolera dichas amenazas. Esta evaluación inicial permitió establecer las prioridades en seguridad del software y definir estrategias para la mejora del ciclo de desarrollo seguro (SSDLC).

Con el objetivo de identificar las principales amenazas, se llevaron a cabo entrevistas con los responsables del negocio y partes interesadas, documentando los factores específicos de la industria y las necesidades particulares de la organización. Durante estas charlas, el equipo de desarrollo expuso los desafíos actuales en seguridad, la ausencia de medidas de protección en la fase inicial del desarrollo y los riesgos asociados con la exposición de datos personales y la dependencia de servicios de terceros.

Como parte del análisis, se documentaron los peores escenarios posibles que podrían afectar

a la organización, así como oportunidades para optimizar el ciclo de desarrollo mediante la adopción de prácticas de seguridad más estrictas. Las siguientes tablas resumen los activos clave y las amenazas operacionales identificadas en este proceso:

La siguiente *Tabla* 5.1 presenta los activos de negocio y técnicos esenciales para el funcionamiento de ANI, los cuales fueron considerados dentro del proceso de modelado de amenazas y evaluación de riesgos.

Tabla 5.1: Clasificación de Herramientas Utilizadas

Categoría	**Descripción**
Herramientas para API	
NestJS	Framework utilizado para la autenticación de usuarios y gestión de WebSockets.
FastAPI	Utilizado para la administración de transcripciones y grafos.
Herramientas para Desarrollo Web y Móvil	
Vite React	Utilizado para la interfaz de administración de docentes y administradores.
Expo	Framework para el desarrollo de la aplicación móvil de estudiantes.
Infraestructura y Gestión	
PostgreSQL	Base de datos estructurada para el almacenamiento de información.
Nginx	Servidor de proxy inverso utilizado en la infraestructura.
Contabo	Plataforma utilizada para el hosteo de los servidores.
GitHub	Control de versiones y repositorios del proyecto.

Continúa en la siguiente página

Categoría	Descripción
Notion	Plataforma de documentación y gestión de información técnica.
Herramientas de Desarrollo	
JetBrains	IDE utilizado por el equipo de desarrollo.
Visual Studio Code	Editor de código ampliamente utilizado en el proyecto.
Python	Lenguaje de programación clave en la implementación de funcionalidades.

Otra representación que se puede tomar en cuenta seria el de la *figura* 5.1 donde se ve mejor como se clasifico las amenazas y sus activos dependiendo su nivel.

Medir y mejorar (Measure and improve)

Para evaluar la efectividad del programa de seguridad implementado en ANI, se establecieron métricas que permiten medir el progreso y la eficiencia de las estrategias aplicadas. La medición de estos resultados es crucial para justificar futuras mejoras, garantizar el respaldo del equipo de desarrollo y facilitar la obtención de recursos para la continuidad del programa.

Dado que los entornos de desarrollo son dinámicos y cambian constantemente, se definieron métricas estructuradas en tres categorías principales:

- **Métricas de esfuerzo:** Miden la cantidad de recursos invertidos en seguridad, como el número de horas de capacitación, tiempo dedicado a la revisión de código y cantidad de aplicaciones analizadas en busca de vulnerabilidades.
- **Métricas de resultados:** Evalúan el impacto de los esfuerzos en seguridad, incluyendo la cantidad de parches aplicados, vulnerabilidades detectadas y mitigadas, así como incidentes de seguridad registrados.
- **Métricas del entorno:** Consideran factores externos que afectan la seguridad, como el

Figura 5.1

Activos y Amenazas

Activo	Amenaza de negocio	Amenaza tecnica	Probabilidad	Impacto	Notas de mitigación
Página web - aplicación móvil	Pérdida de confianza del cliente - Daño a la reputación	Ataque de DDoS	Alta	Medio	Implementar rate limiting.
Datos del Usuario	Pérdida de confianza del cliente - Daño a la reputación - Fuga de datos.	Acceso no autorizado y SQL Injection	Medio	Medio	Implementar control de inputs y RBAC (Role based control access)
Api de terceros	Pérdida de confianza del cliente - Pérdida de la disponibilidad	Baja disponibilidad de Apis de terceros	Baja	Alto	Mostrar mensajes específicos cuando Apis de Chat GPT o Whisper no funcionen.
Reputación	Pérdida de confianza del cliente - Daño de la reputación - Pérdida de disponibilidad	Todas las de arriba	Alta	Alto	Implementar todas las de arriba

número de aplicaciones desarrolladas, la cantidad de líneas de código y la complejidad del sistema.

Los documentos que se han almacenado en *Google Drive* contienen métricas clave, incluyendo el número de vulnerabilidades detectadas y analizadas, fechas de identificación, dependencias de terceros afectadas y correlación con las políticas de seguridad implementadas.

En la figura 5.2 se muestra la estructura utilizada para el análisis de métricas de seguridad en el proyecto.

Esto nos ayudo a establecer en el proyecto ANI, puntos clave de medición en función de su estado de desarrollo y la necesidad de mejorar la seguridad.

Figura 5.2

Métricas de Evaluación de Seguridad

Para threat assessment

Threat assessment

Sistema	Threat Assessment	DAST	SAST	Configuración Review	Mitigation Review	Owasp
AniApi	10	6	2	—		4
AniWebSocket	2	—	2	—		1
AniTranscription	7	—	2	—		2
AniWeb	2	—	2	—		2
AniMobile	3	—	2	—		4
Nginx				3		—
Docker				2		—
Ubuntu Provider				2		—

Curso Capacitación

Total Alumnos:
Total Examenes:
Promedio Horas:

- **Medición del progreso del curso de seguridad:** Se monitores el porcentaje de cumplimiento del curso en *Moodle*, asegurando que los desarrolladores adquieran conocimientos fundamentales en seguridad de aplicaciones.

- **Evaluación de vulnerabilidades identificadas y corregidas:** Se registra la cantidad de fallos de seguridad encontrados en el código mediante análisis SAST y DAST, así como la efectividad de los parches aplicados.

- **Monitoreo de la adopción de políticas de seguridad:** Se revisa el cumplimiento de las directrices establecidas en el OWASP SAMM, garantizando que la seguridad sea un aspecto prioritario en cada fase del desarrollo.

El seguimiento de estas métricas nos permitió analizar la evolución del programa de seguridad en ANI, justificando las futuras mejoras e inversiones en herramientas y capacitación que puedan darse, otro tema es que la estructura puede reutilizarse para diferentes proyectos.

Políticas y Estándares (Policy and Standards)

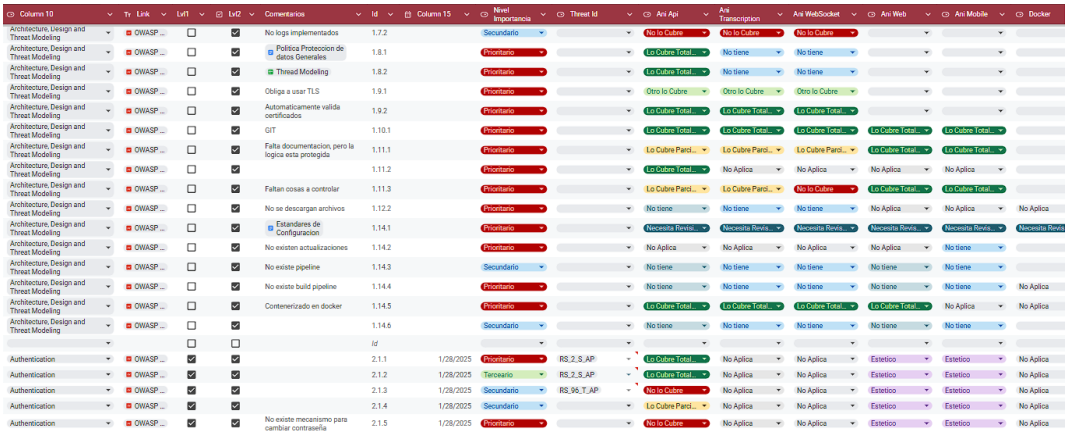
Los estándares de seguridad desarrollados para ANI fueron hechos para cumplir con las políticas requeridas y proporcionar guías específicas en la implementación de tecnologías que son utilizadas. En este sentido, se aseguraron de:

- Incorporar requisitos de seguridad definidos en las políticas.
- Proporcionar lineamientos específicos para los lenguajes y frameworks utilizados.
- Incluir mecanismos de actualización periódica, permitiendo la evolución de los estándares según nuevas amenazas y mejores prácticas.
- Vincular los requisitos individuales con estándares externos, facilitando auditorías.

Para facilitar la gestión y actualización de los estándares, se desarrolló un esquema de referencia en *Google Sheets*, el cual enlaza cada requerimiento con la política correspondiente y con estándares de seguridad de terceros como OWASP OASV tal como se ve en la *figura* 5.3.

Figura 5.3

Thread Modeling del OWASP OASV

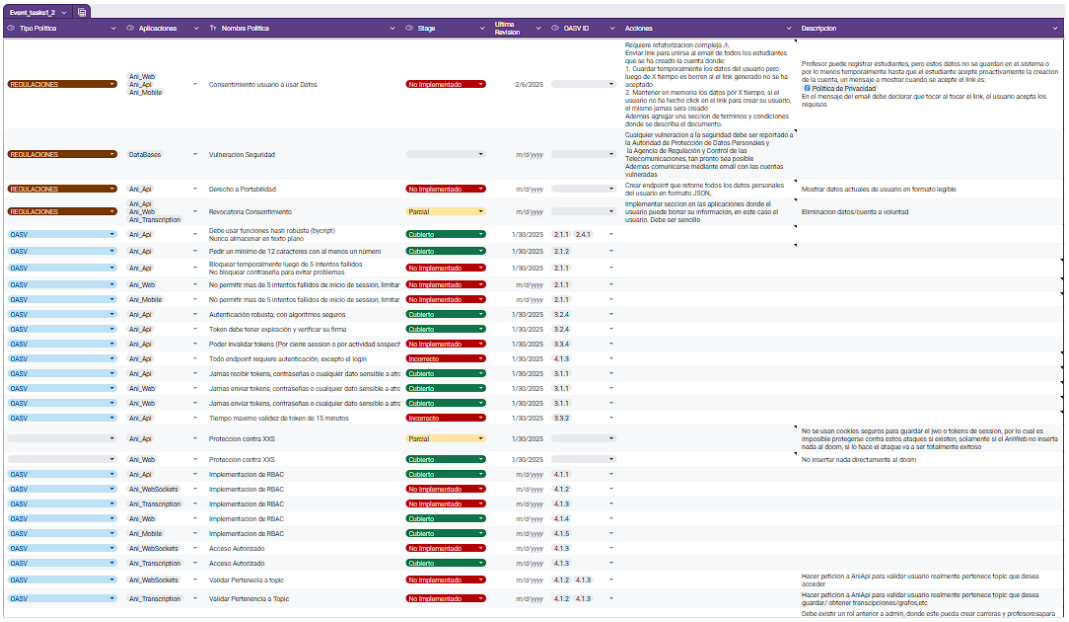


Este enfoque nos permitió facilitar la gestión de seguridad para el equipo, asegurando que cada control de seguridad estuviera alineado con los objetivos estratégicos del proyecto y con las mejores prácticas de la industria.

La aplicación de políticas y estándares de seguridad en ANI ha sido clave para fortalecer su postura de seguridad. La adopción de OWASP Application Security Standard o para abreviar OASV, junto con la definición de políticas internas adaptadas al entorno del proyecto, ha permitido establecer un marco regulador claro que guía el desarrollo seguro.

Figura 5.4

Thread Modeling - Vinculación de Estándares y Políticas



Además, la vinculación de requisitos técnicos con estándares externos ha facilitado las auditorías, el mantenimiento y la evolución del programa de seguridad, asegurando que en el futuro el equipo de desarrollo pueda operar bajo un esquema que garantice la integridad, privacidad y protección de la información en ANI.

Gestión de Cumplimiento (Compliance Management)

En esta parte se llevó a cabo un análisis de los requisitos de cumplimiento, tomando en cuenta regulaciones nacionales e internacionales aplicables, algo con lo que en un principio no contaban. Este proceso permitió identificar los factores que determinan el alcance del cumplimiento, considerando aspectos como la ubicación geográfica, el tipo de datos procesados y las obligaciones contractuales con terceros.

Uno de los principales marcos normativos aplicados fue el *Reglamento General de la Ley*

Orgánica de Protección de Datos Personales 5.5 (LOPDP), el cual rige el manejo de datos dentro de Ecuador. Dado que la aplicación ANI procesa información personal como nombres completos y credenciales de acceso, se consideró fundamental utilizar a esta normativa. Sin embargo, dado que la plataforma no realiza transacciones financieras, almacenamiento de datos sensibles ni procesos relacionados con el sector salud, el alcance de cumplimiento se limitó a la protección de credenciales y privacidad de los usuarios.

Figura 5.5

Política de Protección de Datos

POLÍTICA DE GESTIÓN SECRETOS EN ANI-APP

1. Introducción

1.1. Alcance

- Toda aplicación/proyecto en todo momento

1.2. Objetivo

- Establecer los lineamientos y procedimientos para la gestión de secretos de datos confidenciales.

2. Reglas Generales:

2.1. Seguro

- Todos los secretos se deben almacenar en gestores de secretos aprobados, como Protón Pass
- Queda prohibido almacenar secretos en códigos fuente, archivos de configuración o repositorios.
- Solo el personal autorizado podrá acceder a los secretos.
- Deben implementarse controles de acceso, para la protección de secretos cuando sea posible.
- Todos los secretos deben estar cifrados tanto en reposo como en tránsito.
- Los métodos de cifrado deben cumplir con los estándares de seguridad del Instituto Sudamericano.
- Debe establecerse un monitoreo continuo de accesos a los secretos, así como auditorías periódicas para garantizar el cumplimiento de esta política.

Para facilitar la identificación de requisitos regulatorios aplicables, se elaboró una matriz de cumplimiento la cual se puede apreciar mejor el la *figura* 5.6, aquí se documentaron los factores que podrían afectar la aplicación dentro de un marco legal. Esta matriz permitió establecer requisitos de cumplimiento a nivel organizacional, asegurando que las regulaciones aplicadas sean transversales a toda la infraestructura tecnológica y no dependan únicamente de una aplicación en particular.

Además, se evaluó la relación entre los requisitos de cumplimiento y la biblioteca de políticas y estándares de seguridad definidos en ANI. Encontrando que muchas de las mejores prácticas de seguridad exigidas por normativas internacionales ya estaban contempladas en las políticas establecidas bajo el modelo OWASP SAMM, lo que redujo la necesidad de crear nuevos estándares desde cero.

Una vez revisadas las regulaciones aplicables, se realizó un mapeo de cumplimiento para vincular los requisitos legales con las políticas de seguridad internas. Cuando se identificaron diferencias entre ambas, se actualizaron las políticas existentes para incluir los aspectos necesarios y garantizar la alineación con normativas externas.

Nuestra meta fue desarrollar una matriz de cumplimiento que reflejara qué políticas y estándares internos contienen detalles específicos sobre cada normativa aplicable. Tomando todo esto en cuenta, se garantizó que las políticas organizacionales estuvieran reforzadas con requisitos de cumplimiento, evitando vacíos legales o errores en la implementación de seguridad.

Figura 5.6

Políticas de Threat Modeling y Cumplimiento

Acciones	Descripcion
Requiere refatorizacion compleja ⚠ Enviar link para unirse al email de todos los estudiantes que se ha creado la cuenta donde: 1. Guardar temporalmente los datos del usuario pero luego de X tiempo es borren si el link generado no se ha aceptado 2. Mantener en memoria los datos por X tiempo, si el usuario no ha hecho click en el link para crear su usuario, el mismo jamas sera creado Ademas agregar una seccion de terminos y condiciones donde se describa el documento.	Profesor puede registrar estudiantes, pero estos datos no se guardan en el sistema o por lo menos temporalmente hasta que el estudiante acepte proactivamente la creacion de la cuenta, un mensaje a mostrar cuando se acepte el link es: Politica de Privacidad En el mensaje del email debe declarar que tocar al tocar el link, el usuario acepta los requisos
Cualquier vulneracion a la seguridad debe ser reportado a la Autoridad de Protección de Datos Personales y la Agencia de Regulación y Control de las Telecomunicaciones, tan pronto sea posible Ademas comunicarse mediante email con las cuentas vulneradas	
Crear endpoint que retorne todos los datos personales del usuario en formato JSON,.	Mostrar datos actuales de usuario en formato legible
Implementar seccion en las aplicaciones donde el usuario puede borrar su informacion, en este caso el usuario. Debe ser sencillo	Eliminacion datos/cuenta a voluntad

Este documento es de gran ayuda y puede reutilizarse como referencia si se quiere crear algún proyecto similar o adaptarlo si se ve necesario hacerlo.

Formación y Concientización (Training and Awareness)

Uno de los pasos importantes al momento de implementar OWASP SAMM dentro del desarrollo de ANI fue la formación y concientización en seguridad para todos los miembros del equipo. La capacitación en seguridad es esencial para garantizar que todos los involucrados en el SDLC

comprendan las amenazas más comunes, las mejores prácticas de seguridad y los principios fundamentales de diseño seguro.

Para ello, se creo un curso básico donde se verían los temas necesarios en la plataforma Moodle, aquí los participantes pudieron acceder a cursos sobre seguridad en aplicaciones. Además, se llevaron a cabo charlas presenciales sobre OWASP OASV y análisis de amenazas (Threat Modeling), reforzando los conceptos aprendidos en el entorno virtual.

Por lo tanto el contenido del curso abarca temas clave en seguridad de software, asegurando que incluso aquellos sin conocimientos técnicos avanzados pudieran comprender las mejores prácticas de desarrollo seguro. Los módulos incluyeron:

- **OWASP Top 10:** Explicación detallada de las vulnerabilidades más críticas en aplicaciones web, con ejemplos y estrategias de mitigación.
- **Buenas prácticas de seguridad:** Conceptos fundamentales como privilegio mínimo (Least Privilege), defensa en profundidad (Defense-in-Depth), fail secure, mediación completa (Complete Mediation), gestión de sesiones y diseño abierto (Open Design).
- **OWASP SAMM:** Implementación del modelo en la seguridad del desarrollo, enfocado en su aplicación práctica dentro de ANI.
- **OWASP ASVS y *MASVS*:** Evaluación de seguridad en aplicaciones web y móviles.

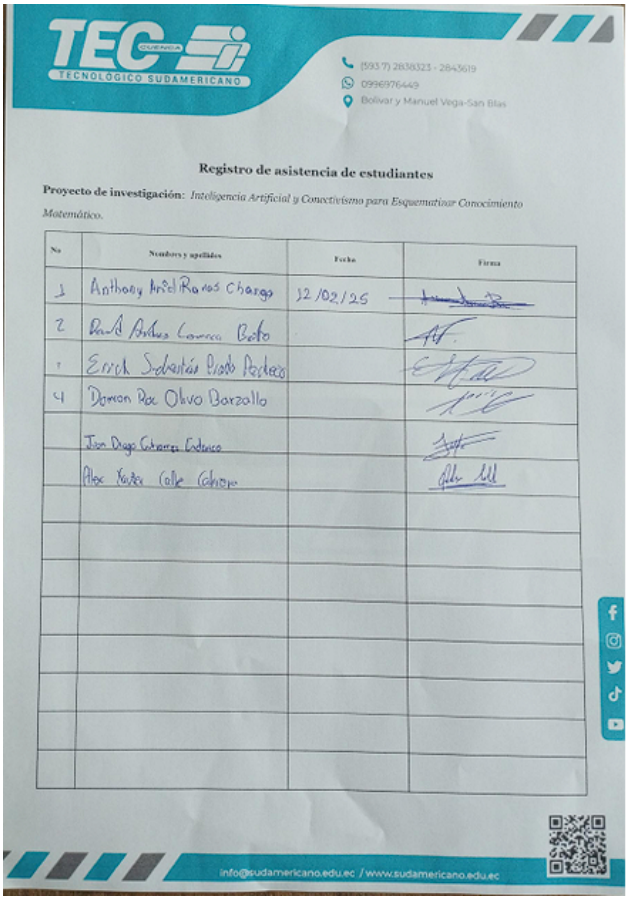
Para garantizar el cumplimiento del curso por parte de los estudiantes, se establecieron mecanismos de seguimiento:

- **Lista de asistencia en reuniones presenciales:** Se tomaron registros en cada charla sobre OWASP ASVS y análisis de amenazas que se dio al equipo, esto se demuestra en la siguiente *figura* 5.7, las otras están guardados en *Google Drive* junto con el resto de los documentos.
- **Actualización periódica:** El contenido del curso se revisó y finalizó en enero de 2025, asegurando que incluya la versión más reciente de los temas que se han tratado.

- **Capacitación para nuevos integrantes:** Aunque no hubo nuevas incorporaciones recientes, cualquier miembro futuro del equipo podrá acceder libremente a los materiales de formación en Moodle.

Figura 5.7

Lista de asistencia



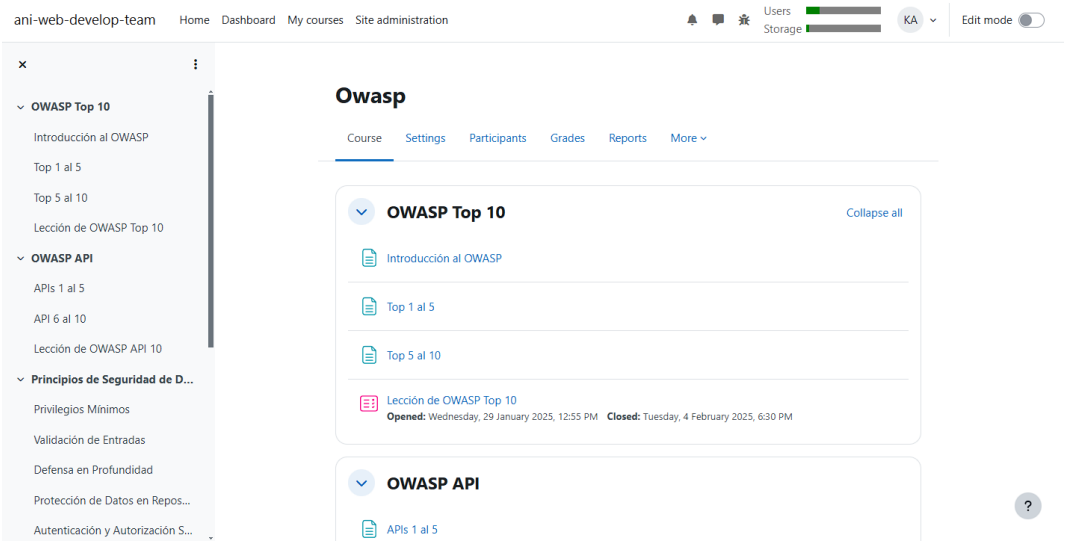
Todos los cursos fueron accesibles a cualquier miembro del equipo, garantizando que la capacitación fuera repetible y consistente. Además, Moodle permitió mantener un registro de acceso, asegurando que cada participante cumpliera con los requisitos del programa.

Organización y Cultura (Organization and Culture)

. Para fortalecer la seguridad dentro del ciclo de desarrollo de ANI, se tuvo que implementó la figura del *Security Champion*, un rol clave que sirve como puente entre el equipo de desarrollo y la gestión de seguridad. Este rol es responsable de identificar amenazas, promover buenas prácticas

Figura 5.8

Curso Moodle



de seguridad y garantizar que la seguridad sea un elemento central en todas las fases del desarrollo.

Dado el tamaño y la estructura del equipo, dos desarrolladores del club asumieron el rol de lideres del equipo, ya que poseen un conocimiento profundo del sistema y su arquitectura. Esto permitió al *Security Champion* establecer un enfoque de seguridad adaptado a las necesidades del proyecto, asegurando que las mejores prácticas de seguridad fueran aplicadas de manera consistente.

El rol del *Security Champion* tiene que cumplir con ciertas características para garantizar su efectividad dentro del equipo, siendo que es necesario que sea un Desarrollador, Tester o Product Manager dentro del equipo de desarrollo, debe de contar con conocimientos básicos o mas avanzados de seguridad y experiencia en OWASP SAMM, Top 10 y el OASV temas importantes en este trabajo.

Normalmente el *Security Champion* desempeña un papel crucial dentro del desarrollo de cualquier tipo de proyecto, siendo el encargado de todo lo relacionado a la seguridad, en el caso de ANI, sus cargos no fueron tan grandes ya que el trabajo fue dividido entre dos desarrolladores, cumpliendo así con una serie de responsabilidades que garantizan la seguridad del software. Estas responsabilidades incluyen:

- **Defensa de la seguridad:** Actuar como un enlace entre el equipo de desarrollo y la seguri-

dad, promoviendo la adopción de prácticas seguras y fomentando una mentalidad donde la seguridad sea prioritaria.

- **Definición de requisitos de seguridad:** Capturar los requisitos del sistema y definir criterios de aceptación para nuevas funcionalidades.
- **Threat Modeling y análisis de riesgos:** Mantener y actualizar el modelo de amenazas utilizando los repositorios de MiguelCriollo/AniThreatAssesment y documentar la priorización de amenazas.
- **Revisión de resultados de pruebas externas:** Evaluar los hallazgos de pruebas de penetración para identificar falsos positivos y amenazas repetitivas, proporcionando retroalimentación al equipo.
- **Revisión de código seguro:** Analizar el código fuente en cada nueva implementación de características para garantizar que cumple con los estándares de seguridad.
- **Implementación de herramientas de seguridad:** Determinar qué pruebas deben ejecutarse en cada entorno y definir las herramientas a utilizar, estableciendo el tiempo y recursos necesarios.
- **Soporte en respuesta ante incidentes:** Tomar medidas inmediatas cuando se detecte un incidente, utilizando la tabla de *Security Incidents* como referencia para la gestión de incidentes.
- **Entrenamiento y difusión de conocimientos:** Organizar sesiones periódicas para discutir el estado del sistema, compartir nuevas implementaciones y mantenerse informado sobre nuevas amenazas emergentes.
- **Monitoreo de avances:** Supervisar los *Logs* del sistema para detectar posibles amenazas en la fase de producción.
- **Reportes y métricas:** Generar informes sobre el estado de la seguridad en ANI, documentando métricas clave para evaluar la madurez del programa de seguridad.

Una parte importante de sus funciones, seria que el *Security Champion* es el encargado de revisar periódicamente todas las cuestiones relacionadas con la seguridad del proyecto para asegurar que el equipo esté al tanto de los problemas actuales y futuros, en este caso los lideres del equipo son un apoyo para gestionar que todo se este aplicando.

Como parte de la gestión del *Security Champion*, se estableció un esquema de trabajo basado en *Threat Modeling* 5.10, donde se documentan y priorizan las amenazas dentro del sistema. Esta información se consolidó en una herramienta de monitoreo visual, asegurando que los *Security Champions* puedan evaluar, mitigar y reportar amenazas en tiempo real.

2. Desing

Perfil de riesgo de la aplicación (Aplication Risk ProFile)

Para evaluar el riesgo de seguridad de la aplicación ANI, se utilizó un método estructurado que permitió estimar el impacto potencial que un ataque podría tener en la organización. La evaluación se basó en los principios de confidencialidad, integridad y disponibilidad de los datos y servicios, analizando qué factores pueden representar un riesgo significativo.

Dado que ANI es una plataforma diseñada para instituciones educativas, su naturaleza abierta a la red implica ciertos desafíos de seguridad. Aunque la aplicación cuenta con un grupo de usuarios cerrado compuesto por docentes y estudiantes, la seguridad no había sido considerada como un elemento clave en su desarrollo inicial y siendo que la aplicación almacena datos sensibles como información personal de usuarios registrados y debates realizados en cursos, esto genera la necesidad de proteger estos registros de accesos no autorizados.

Otro factor crítico en la evaluación fue la dependencia de servicios de terceros. ANI utiliza APIs externas para funciones clave, lo que introduce posibles riesgos asociados a errores en estos servicios o fallos en su disponibilidad. También se sabe que el sistema cuenta con despliegues frecuentes, lo que implica que cualquier actualización mal gestionada podría comprometer la estabilidad o seguridad de la aplicación.

Para comprender mejor el perfil de riesgo de ANI, se formularon una serie de preguntas diri-

gidas a analizar aspectos esenciales de la aplicación, sus funcionalidades y sus vulnerabilidades potenciales.

Se investigó cómo la aplicación maneja los datos personales y los registros de debate, quiénes tienen acceso a estos y de qué manera se gestionan los roles de usuario dentro del sistema. También se analizó si la aplicación está integrada con otros sistemas internos y si existe un control estricto de las versiones de los frameworks y librerías utilizadas.

Otros aspectos considerados incluyeron la necesidad de autenticación multifactor, el impacto de una falla en la disponibilidad del servicio en las instituciones que la utilizan y si ANI ha sido diseñada con buenas prácticas de seguridad, como validación de entradas y manejo adecuado de errores.

Adicionalmente, se evaluó si la aplicación ya cuenta con vulnerabilidades conocidas, presencia de versiones desactualizadas o puertos abiertos que puedan ser explotados por atacantes. También se discutió si existe un plan definido para la aplicación de parches de seguridad después del despliegue.

Con base en los hallazgos de la evaluación, se utilizó un esquema de clasificación de riesgos que permite categorizar la aplicación de forma cualitativa en riesgo alto, medio o bajo. Este esquema facilita la comparación del riesgo de ANI con otras aplicaciones dentro del mismo entorno organizacional.

La *figura* 5.9 siguiente muestra la metodología utilizada para clasificar el riesgo de ANI y representar su impacto en la organización:

Modelado de amenazas (Threat Modeling)

El modelado de amenazas es un documento estructurado que permite identificar, evaluar y gestionar los riesgos de seguridad en una aplicación, analizando tanto las vulnerabilidades del sistema como los errores en el diseño arquitectónico. En el caso de ANI, este proceso se implementó como parte de la evaluación de seguridad.

El modelado de amenazas en ANI se basó en STRIDE, un marco de referencia ampliamente utilizado para la categorización de amenazas. Para su implementación, se realizó un análisis itera-

Figura 5.9

Clasificación de Riesgos

Riesgos

Aplicacion	Datos Sensibles	Impacto Negocio	Dependencia Terceros	Riesgo Confidencialidad	Riesgo Integridad	Riesgo Disponibilidad	Riesgo General
Ani Api	Medio Maneja contraseñas y datos personales no muy sensibles	Alto Principal gestor de creacion y modificacion de propiedades	Medio Depende de BDD externa	Alto	Alto	Alto	Alto
Ani Web	Bajo Posibles claves sensibles	Bajo Poco riesgo a que sea vulnerada	Bajo	Bajo	Nulo	Medio	Bajo
Ani WebSockets	Bajo Información básica participantes	Medio Gestionar orden del debate	Bajo	Medio	Bajo	Medio	Medio
Ani Transcription	Alto Credenciales para servicios IA	Critico Punto principal del servicio, sin generación de grafos la app es inutil	Critico Depende de servicios de IA	Critico	Critico	Critico	Critico
Ani Mobile	Bajo Posibles claves sensibles	Bajo Poco riesgo a ser vulnerada	Bajo	Bajo	Nulo	Bajo	Bajo

tivo que permitió asignar un ID único a cada amenaza, facilitando su rastreo y su vinculación con futuras medidas de seguridad. Además, las amenazas se documentaron de manera estructurada en *Google Sheets - Thread Modeling* (*Figuras* 5.10 y 5.11).

Figura 5.10

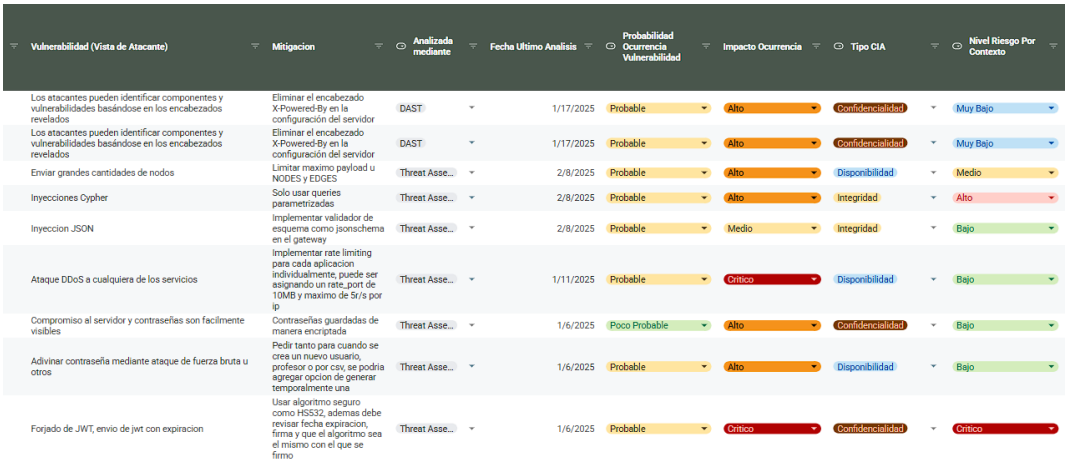
Thread Modeling Parte 1

Realid	Amenaza que cubre	Owasp ID	ID	Tipo Vulnerabilidad	Descripcion en Microsoft Threat Modeling Tool?	Comentarios	Sistemas Prioritarios	STRIDE	Necesidad Implementacion Mitigacion	Caracteristica
RS_4			RS_4_I_TC		☑		TC_TRANSC...	INFORMATION DIS...	NECESARIO	GLOBAL
RS_5			RS_5_I_WS		☑		WS_WEBSO...	INFORMATION DIS...	NECESARIO	GLOBAL
RS_6			RS_6_D_TC	Especifica	☑		TC_TRANSC...	DENIAL OF SERVICE	NECESARIO	GRAFOS
RS_7			RS_7_T_TC	Grupo	☑		TC_TRANSC...	TAMPERING	NECESARIO	GRAFOS
RS_8			RS_8_T_TC	Grupo	☑		TC_TRANSC...	TAMPERING	NECESARIO	GRAFOS
RS_10	TH_12_D		RS_10_D_PV	Global	☑		PV_PROVID...	DENIAL OF SERVICE	NECESARIO	NGINX
RS_13	TH_1_S	OWP_T45_API_08	RS_13_I_AP	Especifica	☑		AP_API	INFORMATION DIS...	NECESARIO	USER
RS_14	TH_1_S	OWP_T18_API_02	RS_14_S_AP	Especifica	☑		AP_API	SPOOFING	NECESARIO	USER GLOBAL
RS_15		OWP_T17_API_02	RS_15_T_AP	Especifica	☑		AP_API	TAMPERING	URGENTE	GLOBAL

El proceso de modelado de amenazas se llevó a cabo en conjunto con desarrolladores, *Security Champions* y *testers* de seguridad, promoviendo una visión compartida sobre los riesgos de la aplicación. Se evitó la elaboración de listas demasiado extensas y se priorizaron las amenazas más relevantes y de alto riesgo.

Figura 5.11

Thread Modeling Parte 2



Vulnerabilidad (Vista de Atacante)	Mitigacion	Analizada mediante	Fecha Ultimo Analisis	Probabilidad Ocurrencia Vulnerabilidad	Impacto Ocurrencia	Tipo CIA	Nivel Riesgo Por Contexto
Los atacantes pueden identificar componentes y vulnerabilidades basándose en los encabezados revelados	Eliminar el encabezado X-Powered-By en la configuración del servidor	DAST	1/17/2025	Probable	Alto	Confidencialidad	Muy Bajo
Los atacantes pueden identificar componentes y vulnerabilidades basándose en los encabezados revelados	Eliminar el encabezado X-Powered-By en la configuración del servidor	DAST	1/17/2025	Probable	Alto	Confidencialidad	Muy Bajo
Enviar grandes cantidades de nodos	Limitar maximo payload u NODES y EDGES	Threat Asse...	2/8/2025	Probable	Alto	Disponibilidad	Medio
Inyecciones Cypher	Solo usar queries parametrizadas	Threat Asse...	2/8/2025	Probable	Alto	Integridad	Alto
Inyeccion JSON	Implementar validador de esquema como jsonschema en el gateway	Threat Asse...	2/8/2025	Probable	Medio	Integridad	Bajo
Ataque DDoS a cualquiera de los servicios	Implementar rate limiting para cada aplicacion individualmente, puede ser asignando un rate_port de 10MB y maximo de 5r/s por ip	Threat Asse...	1/11/2025	Probable	Critico	Disponibilidad	Bajo
Compromiso al servidor y contraseñas son facilmente visibles	Contraseñas guardadas de manera encriptada	Threat Asse...	1/6/2025	Poco Probable	Alto	Confidencialidad	Bajo
Adivinar contraseña mediante ataque de fuerza bruta u otros	Pedir tanto para cuando se crea un nuevo usuario, profesor o por csv, se podria agregar opcion de generar temporalmente una	Threat Asse...	1/6/2025	Probable	Alto	Disponibilidad	Bajo
Forjado de JWT, envio de jwt con expiracion	Usar algoritmo seguro como HS512, ademas debe revisar fecha expiracion, firma y que el algoritmo sea el mismo con el que se firmo	Threat Asse...	1/6/2025	Probable	Critico	Confidencialidad	Critico

A continuación, se presentan las categorías de amenazas identificadas, junto con su descripción y contexto dentro de ANI, todo esto en la *Tabla* 5.2

Para facilitar la documentación y evaluación de amenazas, se utilizó el modelo STRIDE, el cual nos da una estructura que permite clasificar las amenazas en seis categorías principales:

- **Suplantación de identidad (Spoofing)**: Ataques en los que un usuario malintencionado se hace pasar por otro para obtener acceso no autorizado.
- **Manipulación de datos (Tampering):** Modificación malintencionada de datos dentro del sistema, afectando su integridad.
- **Repudio de acciones (Repudiation):** Situaciones en las que un usuario puede negar haber realizado una acción sin evidencia verificable.
- **Divulgación de información (Information Disclosure):** Exposición de datos sensibles a entidades no autorizadas.
- **Denegación de servicio (DoS):** Ataques que afectan la disponibilidad del sistema, impidiendo su uso legítimo.
- Elevación de privilegios **(Elevation of Privilege)**: Casos en los que un usuario sin privilegios logra obtener permisos más altos sin autorización.

Tabla 5.2: Análisis de Amenazas por Categoría

Análisis de Amenazas por Categoría	
Categoría	**Amenazas**
Externo	Hackers Mal uso de los clientes de la aplicación
Interno	Desarrollador vulnere el proyecto Alguien externo del equipo que haya conseguido permiso Suplantación de identidad del desarrollador (mitigación: políticas de desarrolladores)
Entorno	Cortes de luz
Amenazas Comunes	Control de acceso roto Fallas criptográficas Inyección Diseño inseguro Mala configuración de seguridad Componentes vulnerables y obsoletos Fallos de identificación y autenticación Fallas de integridad de datos y software Fallos de registro y monitoreo de seguridad Falsificación de solicitudes del lado del servidor Resultado de atraco Robo de identidades de desarrolladores Pérdida de las API's Repositorio comprometido Escalado de privilegios

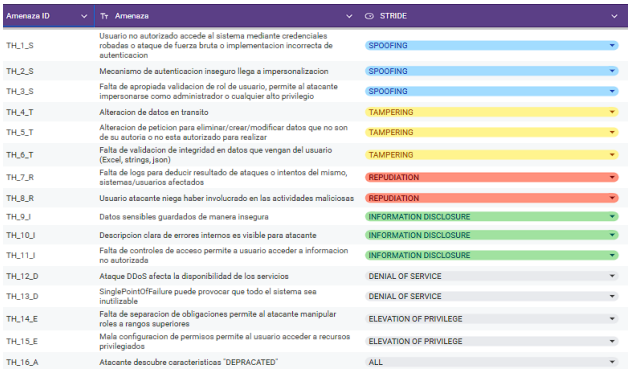
El uso de STRIDE 5.12 permitió estructurar el análisis de seguridad de manera clara para el equipo, asegurando que las amenazas fueran documentadas con un ID único y pudieran ser referenciadas en futuras evaluaciones. Esto estableció una metodología para realizar revisiones iterativas, de modo que las amenazas sean evaluadas en cada actualización significativa de la aplicación.

Requerimientos del software (Software Requirements)

Se necesito una revisión detallada de los requerimientos funcionales y de seguridad de la aplicación ANI, asegurándonos que cada componente del sistema cumpliera con los principios de confidencialidad, integridad y disponibilidad. La identificación de estos requerimientos se basó en la primer entrevista 8.1.1 con el equipo de desarrollo y en el análisis de los objetivos del proyecto, evaluando lo que la aplicación ya tenía implementado y lo que necesitaba para fortalecer su

Figura 5.12

STRIDE - Modelado de Amenazas



Amenaza ID	Amenaza	STRIDE
TH_1_S	Usuario no autorizado accede al sistema mediante credenciales robadas o ataque de fuerza bruta o implementacion incorrecta de autenticacion	SPOOFING
TH_2_S	Mecanismo de autenticacion inseguro llega a impersonalizacion	SPOOFING
TH_3_S	Falta de apropiada validacion de rol de usuario, permite al atacante impersonarse como administrador o cualquier alto privilegio	SPOOFING
TH_4_T	Alteracion de datos en transito	TAMPERING
TH_5_T	Alteracion de peticion para eliminar/crear/modificar datos que no son de su autoria o no esta autorizado para realizar	TAMPERING
TH_6_T	Falta de validacion de integridad en datos que vengan del usuario (Excel, strings, json)	TAMPERING
TH_7_R	Falta de logs para deducir resultado de ataques o intentos del mismo, sistemas/usuarios afectados	REPUDIATION
TH_8_R	Usuario atacante niega haber involucrado en las actividades maliciosas	REPUDIATION
TH_9_I	Datos sensibles guardados de manera insegura	INFORMATION DISCLOSURE
TH_10_I	Descripcion clara de errores internos es visible para atacante	INFORMATION DISCLOSURE
TH_11_I	Falta de controles de acceso permite a usuario acceder a informacion no autorizada	INFORMATION DISCLOSURE
TH_12_D	Ataque DDoS afecta la disponibilidad de los servicios	DENIAL OF SERVICE
TH_13_D	SinglePointOfFailure puede provocar que todo el sistema sea inutilizable	DENIAL OF SERVICE
TH_14_E	Falta de separacion de obligaciones permite al atacante manipular roles a rangos superiores	ELEVATION OF PRIVILEGE
TH_15_E	Mala configuracion de permisos permite al usuario acceder a recursos privilegiados	ELEVATION OF PRIVILEGE
TH_16_A	Atacante descubre caracteristicas "DEPRACATED"	ALL

seguridad.

El proceso de evaluación no solo se centró en los controles de seguridad específicos, sino también en la calidad general y el comportamiento seguro de la aplicación. Por ejemplo, si bien un requerimiento esencial es garantizar la protección de datos personales durante la transmisión y almacenamiento, este objetivo no se limitó a definir una medida técnica específica como el uso de TLS 1.2, sino que se enfocó en establecer expectativas claras de seguridad dentro del sistema.

Para cumplir con los objetivos del proyecto, se seleccionaron herramientas específicas para cada componente del sistema en la *Tabla* 5.3. Estas herramientas fueron elegidas en función de su eficiencia, compatibilidad y capacidad para integrarse con estándares de seguridad modernos.

Tabla 5.3: Lista de herramientas utilizadas

Lista de herramientas utilizadas	
Categoría	**Herramienta**
API	NestJS
API-Web-Sockets	NestJS
API	FastAPI
Web	Vite React
Mobile	Expo

Por otro lado los requerimientos de seguridad del *Sistema Integrado ANI* fueron definidos siguiendo la metodología SMART (Específicos, Medibles, Accionables, Relevantes y con un marco de tiempo definido). Esto aseguró que cada requerimiento fuera claro y aplicable al contexto del proyecto, evitando generalizaciones que no aportaran valor real.

Para estructurar los requerimientos, se establecieron columnas adicionales en la documentación, donde se especificaron las mitigaciones correspondientes para cada caso identificado. La siguiente *Tabla* 5.4 presenta los principales requerimientos levantados y sus respectivas medidas de mitigación.

Tabla 5.4: Requerimientos de Seguridad en ANI

Requerimientos de Seguridad en ANI	
Requerimiento	**Medida de Mitigación**
Garantizar la autenticación del usuario en todo momento	Implementación de JWT con expiración controlada y verificación en cada solicitud.
Asegurar la protección de datos personales en tránsito	Uso de TLS 1.2 o superior en todas las comunicaciones.
Evitar accesos no autorizados a funciones administrativas	Control de acceso basado en roles y permisos mediante autenticación centralizada.
Detectar actividades sospechosas dentro del sistema	Implementación de *Logs* de seguridad con monitoreo y alertas en tiempo real.
Evitar inyección de código malicioso	Uso de ORM seguro y validación estricta de entradas en todas las capas del sistema.
Asegurar la disponibilidad del sistema ante posibles ataques DDoS	Implementación de rate limiting y mitigaciones en el balanceador de carga.
Prevenir la exposición de información sensible en errores	Configuración de manejo seguro de errores y ocultamiento de mensajes en entornos de producción.
Controlar la integridad de los datos almacenados	Uso de hashing seguro en contraseñas y mecanismos de auditoría en la base de datos.

La estructura de esta documentación permitió asociar directamente cada requerimiento con una solución aplicable, facilitando su implementación en el sistema y asegurando que cada aspecto de seguridad tuviera una estrategia de mitigación clara.

Seguridad del proveedor (Supplier Security)

Para comprender las fortalezas y debilidades de la seguridad de Contabo, un proveedor que están utilizando para la aplicación ANI, se realizó una evaluación detallada de sus servicios, incluyendo el nivel de control que ofrecen sobre la seguridad del entorno. A través de esta investigación, se identificaron elementos clave que permitieron determinar si el proveedor cumplía con las expectativas del proyecto y si era necesario implementar medidas adicionales para mitigar riesgos.

Se revisaron aspectos como:

- la gestión de acceso.
- disponibilidad de firewalls.

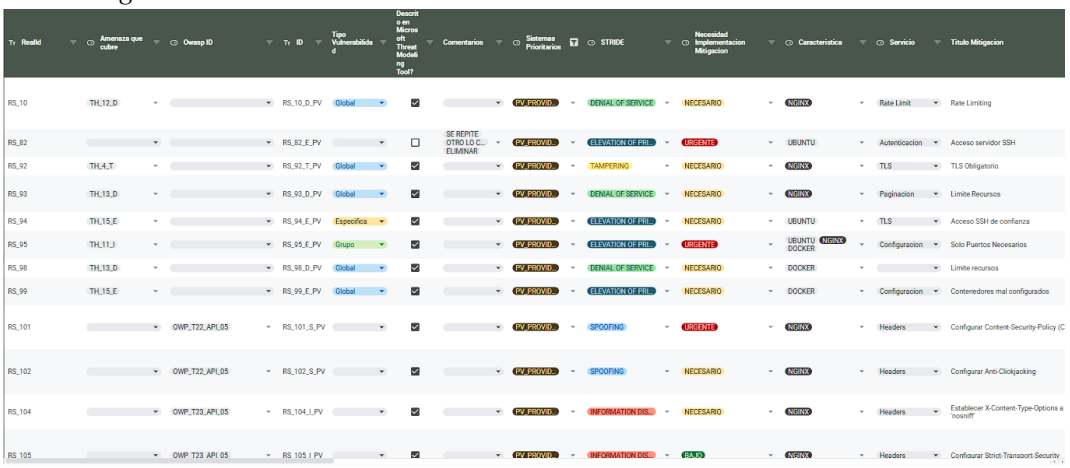
- cifrado de datos.
- medidas contra ataques DDoS.
- controles de seguridad en la infraestructura.

Si bien Contabo ofrece una solución robusta para *hosting*, se identificaron ciertas limitaciones en medidas específicas, como la ausencia de *rate limiting* nativo en sus servicios, lo que obligó al equipo a implementar controles adicionales dentro del propio sistema de ANI.

Los hallazgos que se estructuraron, se documentaron en *Google Sheets - Thread Modeling*, bajo la categoría de ”PROVIDER”5.13, permitiendo visualizar de manera organizada los riesgos asociados al proveedor y las estrategias para mitigarlos.

Figura 5.13

Thread Modeling - PROVIDER



Tr Realid	Amenaza que cubre	Owasp ID	Tr ID	Tipo Vulnerabilidad	Descrito en Microsoft Threat Modeling Tool?	Comentarios	Sistemas Prioritarios	STRIDE	Necesidad Implementacion Mitigacion	Caracteristica	Servicio	Titulo Mitigacion
RS_10	TH_12_D		RS_10_D_PV	Global	☑		PV_PROVEE...	DENIAL OF SERVICE	NECESARIO	NGINX	Rate Limit	Rate Limiting
RS_82			RS_82_E_PV		☐	SE REPITE OTRO LO C... ELIMINAR	PV_PROVEE...	ELEVATION OF PRI...	URGENTE	UBUNTU	Autenticacion	Acceso servidor SSH
RS_92	TH_4_T		RS_92_T_PV	Global	☑		PV_PROVEE...	TAMPERING	NECESARIO	NGINX	TLS	TLS Obligatorio
RS_93	TH_13_D		RS_93_D_PV	Global	☑		PV_PROVEE...	DENIAL OF SERVICE	NECESARIO	NGINX	Paginacion	Limite Recursos
RS_94	TH_15_E		RS_94_E_PV	Especifica	☑		PV_PROVEE...	ELEVATION OF PRI...	NECESARIO	UBUNTU	TLS	Acceso SSH de confianza
RS_95	TH_11_I		RS_95_E_PV	Grupo	☑		PV_PROVEE...	ELEVATION OF PRI...	URGENTE	UBUNTU DOCKER NGINX	Configuracion	Solo Puertos Necesarios
RS_98	TH_13_D		RS_98_D_PV	Global	☑		PV_PROVEE...	DENIAL OF SERVICE	NECESARIO	DOCKER		Limite recursos
RS_99	TH_15_E		RS_99_E_PV	Global	☑		PV_PROVEE...	ELEVATION OF PRI...	NECESARIO	DOCKER	Configuracion	Contenedores mal configurados
RS_101		OWP_T22_API_05	RS_101_S_PV		☑		PV_PROVEE...	SPOOFING	URGENTE	NGINX	Headers	Configurar Content-Security-Policy (C
RS_102		OWP_T22_API_05	RS_102_S_PV		☑		PV_PROVEE...	SPOOFING	NECESARIO	NGINX	Headers	Configurar Anti-Clickjacking
RS_104		OWP_T23_API_05	RS_104_I_PV		☑		PV_PROVEE...	INFORMATION DIS...	NECESARIO	NGINX	Headers	Establecer X-Content-Type-Options a nosniff
RS_105		OWP_T23_API_05	RS_105_I_PV		☑		PV_PROVEE...	INFORMATION DIS...	MUY...	NGINX	Headers	Configurar Strict-Transport-Security

El resultado de esta investigación evidenció que, si bien Contabo proporciona una infraestructura estable y de bajo costo, su enfoque en seguridad requiere refuerzos adicionales por parte del equipo de desarrollo de ANI. Esto significó que varios controles de seguridad, como protección contra accesos no autorizados, monitoreo de tráfico y restricciones de uso de APIs, fueron implementados directamente dentro de la aplicación.

Diseño de Arquitectura (Arquitecture Design)

ANI ya había pasado su fase de desarrollo cuando se trato de implementar el SDLC, esto nos obligo a aplicar un conjunto de principios de seguridad esenciales con el fin de garantizar la protección del sistema en su arquitectura. Estos principios incluyen defensa en profundidad, uso de configuraciones seguras por defecto, diseño simplificado de funciones de seguridad, ejecución con privilegios mínimos y evitación de la seguridad por oscuridad.

Se estableció un documento de Arquitectura de Seguridad, accesible para todos los miembros del equipo, en el cual se documentaron los lineamientos clave para el diseño seguro de ANI. Este documento está almacenado en un repositorio de Google Drive, garantizando su disponibilidad y actualización constante.

El diseño del sistema incorporó medidas de seguridad en puntos críticos de la infraestructura, asegurando que cada componente cumpliera con estándares de protección adecuados. Estas medidas incluyeron:

- **Defensa en profundidad:** Implementación de múltiples capas de seguridad para evitar que una sola falla comprometa todo el sistema.
- **Uso de configuraciones seguras por defecto:** Aplicación de controles de acceso restrictivos desde la primera implementación.
- **Ejecución con privilegios mínimos:** Limitación de los permisos asignados a cada servicio y usuario dentro del sistema.
- **Gestión de fallos segura:** Prevención de fugas de información sensible en mensajes de error.
- **Equilibrio entre seguridad y usabilidad:** Mantenimiento de un sistema accesible sin comprometer su protección.

Estos principios fueron integrados en la documentación de seguridad de arquitectura, la cual se presentó y discutió en sesiones de capacitación con el equipo de desarrollo, .

Figura 5.14

Principios de Seguridad Básicos

Principios Seguridad Básicos

Principios	Descripción	Owner	Status
Defense Depth	Verificar por capas de seguridad, primero firewall, encriptación, etc.	Ani Api Proveedor	Implementar TLS obligatorio Usar algoritmos de encriptación seguros
Securing the weakest link	Fortalecer y revisar las apis más vulnerables	Ani Api Ani Web Sockets Ani Transcription	Proteger links de autorización o borrado de datos.
Seguridad por defecto	Usar configuraciones de seguridad por defecto, firewall y antivirus activado	Todos	Usar configuraciones por defecto de nginx, docker.
Simplicidad en diseño	Evitar hacer sobre complicaciones de seguridad o crear nuevos métodos en vez de usar existentes	Todos	Usar misma lógica de autorización para todos los sistemas, Jwt
Least Privilege	Dar los menos permisos al usuario		Revisar que usuario no puede borrar cuentas que no son de el
Security Through obscurity	Implementar seguridad haciéndola compleja pero no segura	Ani Api Ani Transcription Ani Web Sockets	Solo implementar existentes prácticas de seguridad, no implementar códigos únicos
Rate Limiting	Límite de peticiones/acciones por determinado tiempo	Ani Api Ani Transcription Ani Web Sockets	Límite Peticiones por segundo dependendiendo del sistema

Para fortalecer la seguridad en las interfaces perimetrales del sistema, se evaluaron las posibles vulnerabilidades en cada punto de entrada de la aplicación. Se identificaron mejoras específicas que podían aplicarse sin un impacto significativo en el costo de desarrollo y aquellas que requerían una implementación más profunda en futuras versiones del sistema.

Adicionalmente, antes de comenzar con este proceso de diseño seguro, el equipo recibió capacitación en principios de seguridad en el desarrollo de software 5.15. Este entrenamiento permitió que todos los involucrados comprendieran cómo se debe diseñar aplicaciones seguras desde su base, asegurando que no dejaran la seguridad para el ultimo.

El conocimiento adquirido a través de esta capacitación fue necesario para que el equipo, esto debido a su deficiencia sobre los conocimientos básicos, para poder identificar y mitigar riesgos, evitando vulnerabilidades que podrían haber sido explotadas en su lanzamiento.

Figura 5.15

Principios de Seguridad de Desarrollo

Gestión de Tecnología (Technology Management)

A menudo, los desarrolladores tienden a optar por soluciones que simplifican el proceso de desarrollo y despliegue, sin considerar en profundidad los posibles riesgos de seguridad que pueden introducir nuevas tecnologías. Por esta razón, fue fundamental evaluar cada herramienta en términos de su robustez, vulnerabilidades conocidas y capacidad de integración con medidas de seguridad.

El equipo de ANI llevó a cabo un análisis de las herramientas utilizadas, asegurando que cumplieran con estándares de seguridad básicos que necesitaban.

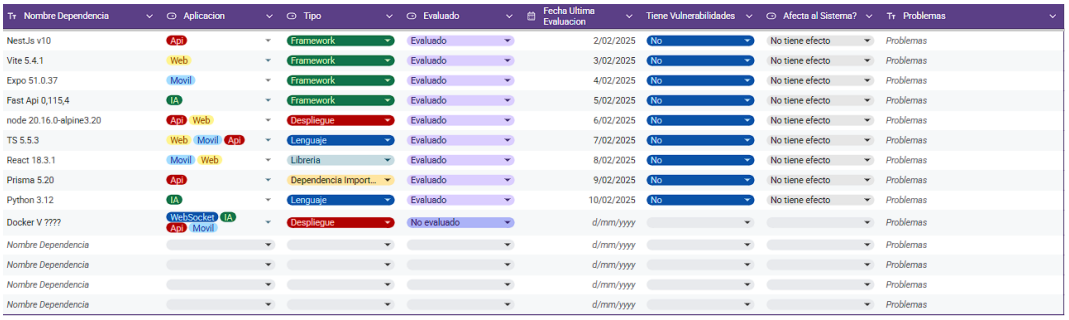
Uno de los principales enfoques fue el análisis de seguridad en cada tecnología implementada, asegurando que se ajustaran a las mejores prácticas de la industria. Se estableció un proceso de revisión continua donde se identificaron posibles dependencias vulnerables, gestionando su actualización o reemplazo cuando fuera necesario.

Para estructurar esta evaluación, se documentaron los hallazgos en *Google Sheets - Lista Tecnologías Usadas* 5.16, proporcionando un registro centralizado de las tecnologías aplicadas en el

proyecto. La siguiente imagen representa esta documentación y su estructura dentro del análisis de seguridad de ANI:

Figura 5.16

Lista de Tecnologías Usadas



Nombre Dependencia	Aplicacion	Tipo	Evaluado	Fecha Ultima Evaluacion	Tiene Vulnerabilidades	Afecta al Sistema?	Problemas
NestJs v10	Api	Framework	Evaluado	2/02/2025	No	No tiene efecto	Problemas
Vite 5.4.1	Web	Framework	Evaluado	3/02/2025	No	No tiene efecto	Problemas
Expo 51.0.37	Movil	Framework	Evaluado	4/02/2025	No	No tiene efecto	Problemas
Fast Api 0,115,4	IA	Framework	Evaluado	5/02/2025	No	No tiene efecto	Problemas
node 20.16.0-alpine3.20	Api Web	Despliegue	Evaluado	6/02/2025	No	No tiene efecto	Problemas
TS 5.5.3	Web Movil Api	Lenguaje	Evaluado	7/02/2025	No	No tiene efecto	Problemas
React 18.3.1	Movil Web	Libreria	Evaluado	8/02/2025	No	No tiene efecto	Problemas
Prisma 5.20	Api	Dependencia Import...	Evaluado	9/02/2025	No	No tiene efecto	Problemas
Python 3.12	IA	Lenguaje	Evaluado	10/02/2025	No	No tiene efecto	Problemas
Docker V ????	WebSocket IA Api Movil	Despliegue	No evaluado	d/mm/yyyy			Problemas
Nombre Dependencia				d/mm/yyyy			Problemas
Nombre Dependencia				d/mm/yyyy			Problemas
Nombre Dependencia				d/mm/yyyy			Problemas
Nombre Dependencia				d/mm/yyyy			Problemas

3. Implementation

Proceso de construcción (Build Process)

Para garantizar un proceso de construcción seguro, estable y repetible, se estableció un flujo de trabajo que sigue las mejores prácticas de OWASP SAMM y cumple con los principios de seguridad del SSDLC.

El proceso de construcción en ANI fue documentado y estructurado para minimizar errores y asegurar la reproducibilidad de cada implementación. Se establecieron las siguientes actividades clave:

- **Definición del proceso:** Se documentaron los pasos detallados de la fase del Build Process, incluyendo preparación, ejecución, generación de artefactos y verificación de integridad.
- **Uso de Docker:** Se utilizaron archivos Dockerfile para cada uno de los servicios, asegurando un entorno controlado y uniforme para todas las aplicaciones.
- **Gestión de dependencias:** Se integró Dependabot de GitHub para detectar vulnerabilidades en librerías y paquetes utilizados en el proyecto.
- **Revisión periódica:** Se estableció una política de actualización y revisión de herramientas

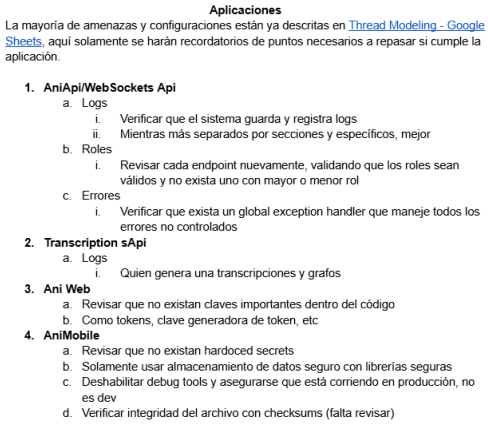
de construcción para garantizar que siempre estén alineadas con los estándares de seguridad más recientes.

- **Evitar exposición de secretos:** Se definieron reglas estrictas para no almacenar credenciales ni claves de acceso dentro de los archivos de configuración del build, siguiendo buenas prácticas de seguridad.

Para evitar problemas de versionado y documentación desactualizada, se creó un documento titulado Estándares de Configuración en Google Docs 5.17. Este documento proporciona una guía detallada para la documentación y verificación de configuraciones, asegurando que no se almacenen secretos en los archivos. Asimismo, permite a los desarrolladores conocer la versión utilizada y garantiza la coherencia en los Dockerfiles almacenados en los repositorios de cada aplicación.

Figura 5.17

Estandares de Configuracion



Aplicaciones

La mayoría de amenazas y configuraciones están ya descritas en Thread Modeling - Google Sheets, aquí solamente se harán recordatorios de puntos necesarios a repasar si cumple la aplicación.

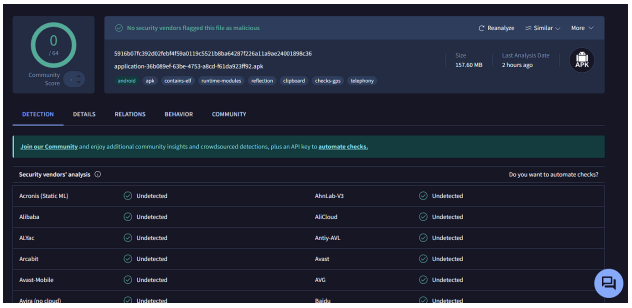
1. **AniApi/WebSockets Api**
   a. Logs
      i. Verificar que el sistema guarda y registra logs
      ii. Mientras más separados por secciones y específicos, mejor
   b. Roles
      i. Revisar cada endpoint nuevamente, validando que los roles sean válidos y no exista uno con mayor o menor rol
   c. Errores
      i. Verificar que exista un global exception handler que maneje todos los errores no controlados
2. **Transcription sApi**
   a. Logs
      i. Quien genera una transcripciones y grafos
3. **Ani Web**
   a. Revisar que no existan claves importantes dentro del código
   b. Como tokens, clave generadora de token, etc
4. **AniMobile**
   a. Revisar que no existan hardoced secrets
   b. Solamente usar almacenamiento de datos seguro con librerías seguras
   c. Deshabilitar debug tools y asegurarse que está corriendo en producción, no es dev
   d. Verificar integridad del archivo con checksums (falta revisar)

Como parte del control de seguridad en el proceso de construcción, se implementó una verificación de integridad en los artefactos generados. Para ello, se definió un procedimiento mediante el cual se genera un hash único del archivo APK una vez construido, garantizando que el producto final no haya sido alterado. Este hash 5.18 se almacena en una ubicación accesible para todos los usuarios, permitiendo que cualquier persona pueda verificar la autenticidad del APK utilizando herramientas de terceros.

Para fortalecer aún más el proceso de construcción en ANI, se recomienda implementar firmas digitales en todos los artefactos generados, no solo en el APK de ANI Mobile. También, se sugiere

Figura 5.18

Hash del APK



automatizar la revisión de seguridad de las herramientas de construcción para garantizar su alineación con las mejores prácticas y establecer un proceso de auditoría del Build Process, verificando el cumplimiento de las configuraciones establecidas.

Dependencias del Software (Software Dependencies)

Se implementó un sistema de gestión y monitoreo de dependencias que permite identificar y mitigar vulnerabilidades en bibliotecas y frameworks utilizados en el desarrollo. La administración de estas dependencias es clave para prevenir ataques derivados de componentes desactualizados o inseguros, asegurando que cada versión utilizada sea compatible con las mejores prácticas de seguridad.

Como parte de esta estrategia, se activó *GitHub Dependabot* en todos los repositorios del proyecto, lo que permite mantener un registro actualizado de las dependencias (Bill of Materials - BOM). Esta herramienta automatiza la detección de vulnerabilidades en paquetes utilizados tanto en el lado servidor (API) como en el lado cliente (frontend y scripts), proporcionando alertas en caso de detectar amenazas críticas.

El registro de dependencias de ANI incluye información detallada sobre cada componente utilizado en la aplicación, permitiendo un análisis exhaustivo de su impacto en la seguridad. Para cada dependencia se recopila:

- **Ubicación en el proyecto:** Identificación del módulo o servicio donde se usa.
- **Versión actual:** Seguimiento de actualizaciones y compatibilidad.

- **Licencia:** Evaluación de restricciones legales y términos de uso.
- **Fuente:** Enlace al repositorio del autor o proveedor del paquete.
- **Estado de mantenimiento:** Verificación de si la dependencia sigue siendo soportada y recibe actualizaciones de seguridad.

GitHub Dependabot automatiza el proceso de revisión y permite detectar rápidamente qué aplicaciones se ven afectadas por vulnerabilidades reportadas en la base de datos Common Vulnerabilities and Exposures (CVE). Esto proporciona una visión clara del estado de seguridad de los paquetes, facilitando la toma de decisiones sobre su actualización o reemplazo.

La siguiente *Tabla* 5.19 resume la estructura utilizada para la gestión de dependencias en ANI:

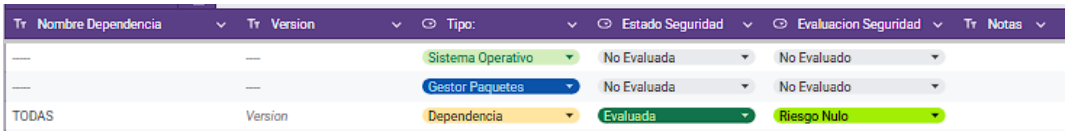
Tabla 5.5: Gestión de Dependencias en ANI

Gestión de Dependencias en ANI			
Dependencia	**Versión**	**Ubicación en ANI**	**Estado de Seguridad**
React	18.2.0	Frontend Web	Actualizado
NestJS	9.0.0	API Backend	Sin vulnerabilidades conocidas
FastAPI	0.95.2	Microservicios	En revisión
Expo	49.0.5	Aplicación Móvil	Actualización pendiente

Además, cada tres meses se debe realizar un análisis completo de todas las dependencias mediante una tabla de seguimiento en *Google Sheets - Lista Dependencias* 5.19, en la cual se documentan la seguridad de las dependencias dependiendo si fue usada para WEB, API o el APK.

Figura 5.19

Lista de Dependencias Usadas

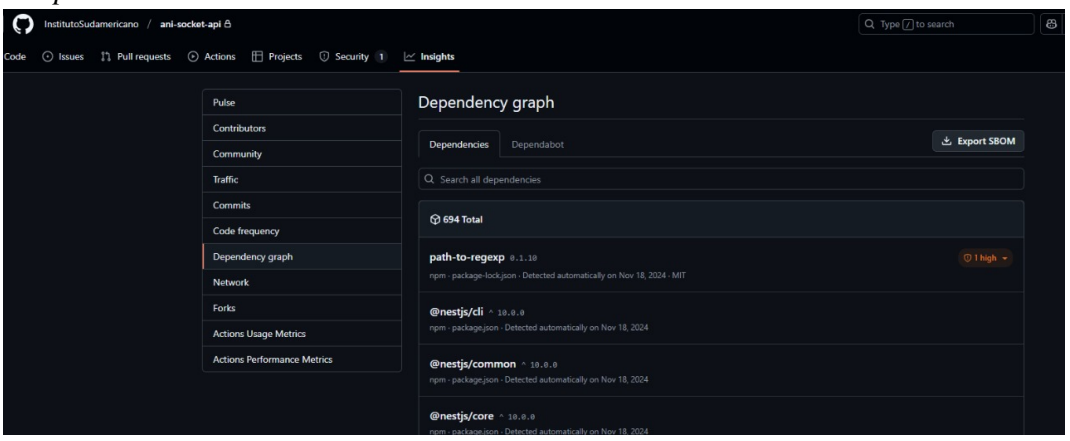


Para estandarizar la seguridad en el manejo de dependencias, se estableció una política de gestión de dependencias, donde se definen los tiempos y acciones a seguir en caso de encontrar amenazas. Esta política establece los siguientes lineamientos:

- Verificación periódica de todas las dependencias mediante *GitHub Dependabot* 5.20.
- Corrección inmediata de vulnerabilidades críticas notificadas por la base de datos CVE.
- Evaluación de compatibilidad antes de actualizar cualquier paquete.
- Documentación de cambios y registros de versiones en la tabla de seguimiento.

Figura 5.20

GitHub Dependabot



Proceso de implementación (Deployment Process)

En un despliegue seguro, se espera que el equipo encargado gestione varios aspectos esenciales, entre ellos:

- Automatización del despliegue mediante herramientas de integración y entrega continua (CI/CD) para evitar errores manuales.
- Configuración segura del entorno para prevenir vulnerabilidades relacionadas con permisos inadecuados, exposición de servicios innecesarios o credenciales mal gestionadas.
- Validaciones post-despliegue, como pruebas funcionales y de seguridad en producción para asegurar que no se introduzcan nuevos riesgos.
- Monitoreo continuo del entorno de producción para detectar actividades anómalas y responder a incidentes de manera oportuna.

- Control de versiones y rollback, permitiendo revertir a una versión estable en caso de detectar fallos críticos.

A pesar de la importancia de estas prácticas dentro de OWASP SAMM, en el caso específico de ANI, no fue posible implementar ni evaluar directamente estos controles, ya que el proceso de despliegue fue realizado por un agente externo sin comunicación directa con el equipo de desarrollo.

Esta falta de visibilidad impidió validar si se cumplieron estándares de seguridad en la fase de implementación, lo que representa una brecha en el control del ciclo de vida del software. Para futuras iteraciones del proyecto, se recomienda establecer una coordinación más estrecha con los responsables del despliegue, asegurando que se sigan las mejores prácticas de OWASP y que el equipo de desarrollo tenga acceso a información clave sobre la implementación en producción.

Gestión de Secretos (Secret Management)

Uno de los objetivos cuando se toco el tema de gestión de secretos, fue asegurar el acceso en varios tipos de funciones que se encuentran dentro de la aplicación, para esto se recomendo seguir el principio de mínimo privilegio, para esto se dividió en tres actividades importantes la implementación de controles.

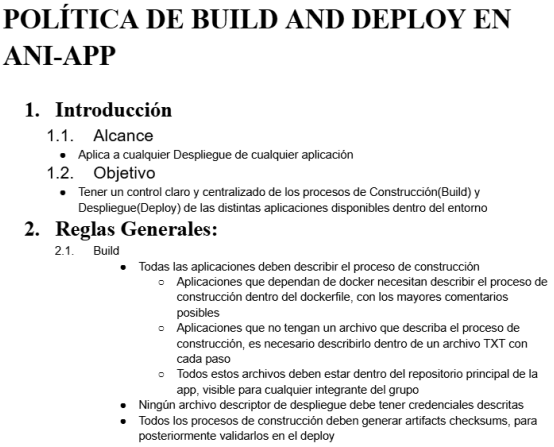
Para tener una guía a seguir se creó una estructura junto con la *Política Gestión de Secretos* 5.21 donde se categoriza los secretos según su aplicación y nivel de acceso.

- Se utilizó ProtonPass como un gestor seguro para almacenar credenciales y claves sensibles.
- Cada secreto se clasificó por aplicación, asegurando una mejor organización y minimizando la exposición innecesaria.
- Se definió una ubicación segura para el almacenamiento, evitando su presencia en repositorios de código o entornos de desarrollo.

Para reducir la posibilidades de accesos no autorizados, se comento la creación de un esquema basado en roles y separaciones de funciones:

Figura 5.21

Política de Gestión de Secretos



POLÍTICA DE BUILD AND DEPLOY EN ANI-APP

1. **Introducción**
   - 1.1. Alcance
     - Aplica a cualquier Despliegue de cualquier aplicación
   - 1.2. Objetivo
     - Tener un control claro y centralizado de los procesos de Construcción(Build) y Despliegue(Deploy) de las distintas aplicaciones disponibles dentro del entorno
2. **Reglas Generales:**
   - 2.1. Build
     - Todas las aplicaciones deben describir el proceso de construcción
       - Aplicaciones que dependan de docker necesitan describir el proceso de construcción dentro del dockerfile, con los mayores comentarios posibles
       - Aplicaciones que no tengan un archivo que describa el proceso de construcción, es necesario describirlo dentro de un archivo TXT con cada paso
       - Todos estos archivos deben estar dentro del repositorio principal de la app, visible para cualquier integrante del grupo
     - Ningún archivo descriptor de despliegue debe tener credenciales descritas
     - Todos los procesos de construcción deben generar artifacts checksums, para posteriormente validarlos en el deploy

- Se establecieron roles específicos para el manejo de secretos, garantizando que solo los usuarios con autorización puedan acceder a información sensible.

- Aplicar restricciones de acceso en entornos productivos, asegurando que los desarrolladores no puedan modificar ni visualizar secretos en producción sin aprobación previa del *Security Champion*.

- Se implementaron controles de auditoría para registrar intentos de acceso y modificaciones en los secretos almacenados.

- A pesar de estas medidas, se llego a un acuerdo de que los desarrolladores puedan aún tener acceso a ciertos secretos de producción, lo cual si bien representa un riesgo que debe es necesario ya que no se cuenta con un equipo completo, para poder ayudar al *Security Champion* con sus deberes.

Si bien se establecieron medidas básicas de gestión de secretos, aún existen riesgos que deben ser mitigados para cumplir con los estándares de OWASP SAMM y SSDLC, para que en un futuro puedan con todos los estándares requeridos, entre estos recomendaciones serian la eliminación al acceso a desarrolladores a secretos de producción, integrar un sistema de rotación de claves y ampliar las herramientas diseñadas para la gestión de secretos.

Seguimiento de amenazas (Defeat Tracking)

El seguimiento de amenazas en ANI fue implementado con el objetivo de identificar, documentar y gestionar defectos de seguridad de manera eficiente, permitiendo priorizar la corrección de vulnerabilidades según su criticidad y probabilidad de explotación.

Para lograr esto, se definió un marco estructurado de detección y seguimiento de defectos, basado en diversas fuentes de identificación de amenazas. Entre los métodos utilizados se incluyeron:

- Evaluaciones de amenazas para detectar posibles riesgos en la arquitectura y funcionamiento del sistema.
- Pruebas de penetración, realizadas en diferentes fases del desarrollo.
- Análisis estático y dinámico de seguridad (SAST y DAST), implementados con herramientas automatizadas para detectar vulnerabilidades en el código fuente y en la aplicación en ejecución.
- Modelado de amenazas, documentado 5.10, donde se registra la criticidad y el impacto de cada vulnerabilidad.

Para garantizar un manejo eficiente de los defectos de seguridad, se estableció una política de reporte y solución de vulnerabilidades documentada en *Google Docs - Política Reporte y Solución Defectos de Seguridad* 5.27. Esta política proporciona directrices claras sobre cómo reportar, priorizar y corregir fallos en el sistema.

Uno de los principios fundamentales del seguimiento de amenazas en ANI fue fomentar una cultura de transparencia, evitando culpar a los desarrolladores o equipos por la introducción de vulnerabilidades. En su lugar, el enfoque se centró en la detección y resolución rápida de problemas, asegurando que todos los defectos fueran documentados en un repositorio accesible únicamente a los miembros autorizados.

Para mitigar el riesgo de filtración o uso indebido de esta información, se definieron reglas de acceso en *Thread Modeling - Google Sheets* 5.10, estableciendo qué miembros del equipo pueden visualizar y modificar los registros de vulnerabilidades.

Cada defecto de seguridad identificado en ANI fue categorizado siguiendo un esquema de clasificación cualitativa, lo que permitió priorizar los esfuerzos de corrección de manera efectiva. La clasificación consideró tres factores principales:

- **Nivel de impacto en el sistema:** Qué tan grave es la vulnerabilidad en términos de seguridad y estabilidad de la aplicación.
- **Probabilidad de explotación:** Qué tan factible es que un atacante pueda explotar la vulnerabilidad en un entorno real.

Esta clasificación permitió diferenciar entre amenazas críticas que requieren atención inmediata y fallos de menor impacto que podían ser programados para corrección en futuras versiones.

Para visualizar y comprender mejor esta clasificación, se utilizó como referencia la siguiente *Tabla* 5.6, siguiendo la documentación en *Google Sheets - Thread Modeling* 5.10:

Tabla 5.6: Clasificación de Defectos de Seguridad en ANI

Clasificación de Defectos de Seguridad en ANI			
Defecto de Seguridad	**Impacto**	**Probabilidad**	**Estado**
Falta de validación de entradas en API	Alto	Alta	En corrección
Manejo inadecuado de tokens de sesión	Medio	Media	Mitigado
Exposición de información en mensajes de error	Bajo	Alta	Pendiente de revisión
Configuración débil en permisos de usuario	Alto	Alta	Mitigado

Para evitar la duplicación de información y minimizar los falsos positivos, se estableció un proceso de revisión y validación de vulnerabilidades. Cualquier defecto identificado es sometido a:

1. Verificación en *Thread Modeling - Google Sheets* 5.10 para comprobar si ya ha sido documentado previamente.

2. Análisis de impacto y priorización, evaluando su urgencia con base en la clasificación de riesgos.

3. Corrección y validación posterior, asegurando que el defecto haya sido efectivamente mitigado.

Métricas y comentarios (Metrics and Feedback)

El seguimiento de métricas en ANI se diseñó para evaluar la eficacia del proceso de seguridad, asegurando que los defectos de seguridad identificados fueran analizados y documentados. Esto funciono estableciendo un proceso periódico de revisión de vulnerabilidades, esto fue para extraer información clave que permita fortalecer la postura de seguridad de la aplicación.

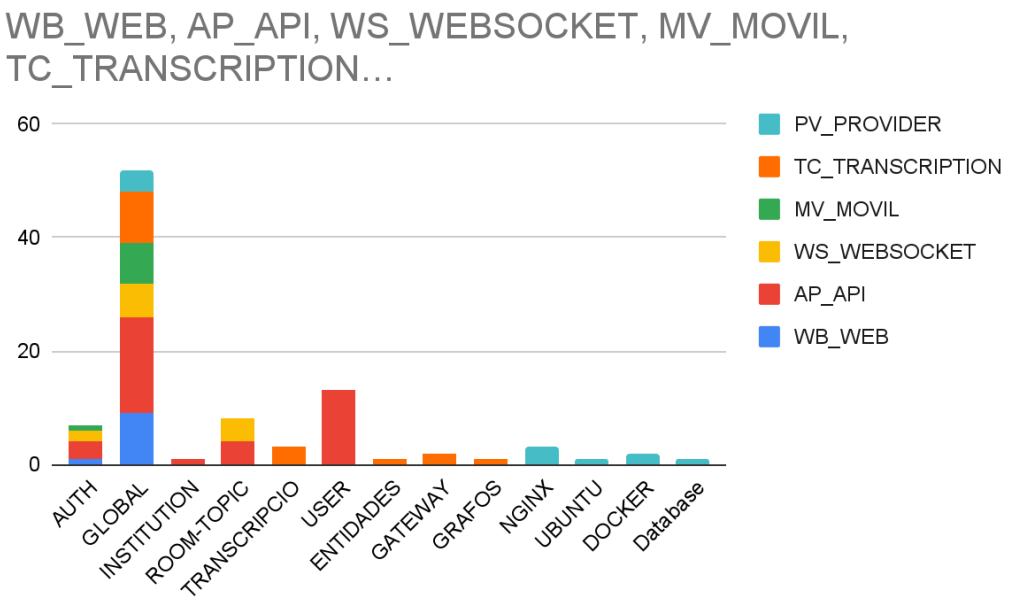
El análisis de métricas incluyó una evaluación de todas las amenazas detectadas y documentadas, permitiendo generar informes detallados sobre la cantidad, ubicación, tipo y gravedad de los defectos de seguridad. Esto proporcionó información clave para enfocar los esfuerzos de mitigación, mejorar la capacitación del equipo y realizar ajustes en la arquitectura de la aplicación.

Uno de los indicadores evaluados fue la cantidad de amenazas identificadas en cada fase del proceso de seguridad. Este análisis permitió comprender si las actividades de verificación eran lo suficientemente efectivas y en qué áreas se necesitaba mejorar la detección de vulnerabilidades.

La siguiente *figura* 5.22 representa la cantidad de amenazas registradas en ANI:

Figura 5.22

Cantidad de Amenazas Identificadas



Además, se evaluó dónde residen las vulnerabilidades dentro de la aplicación, lo que permitió identificar módulos más propensos a fallos de seguridad.

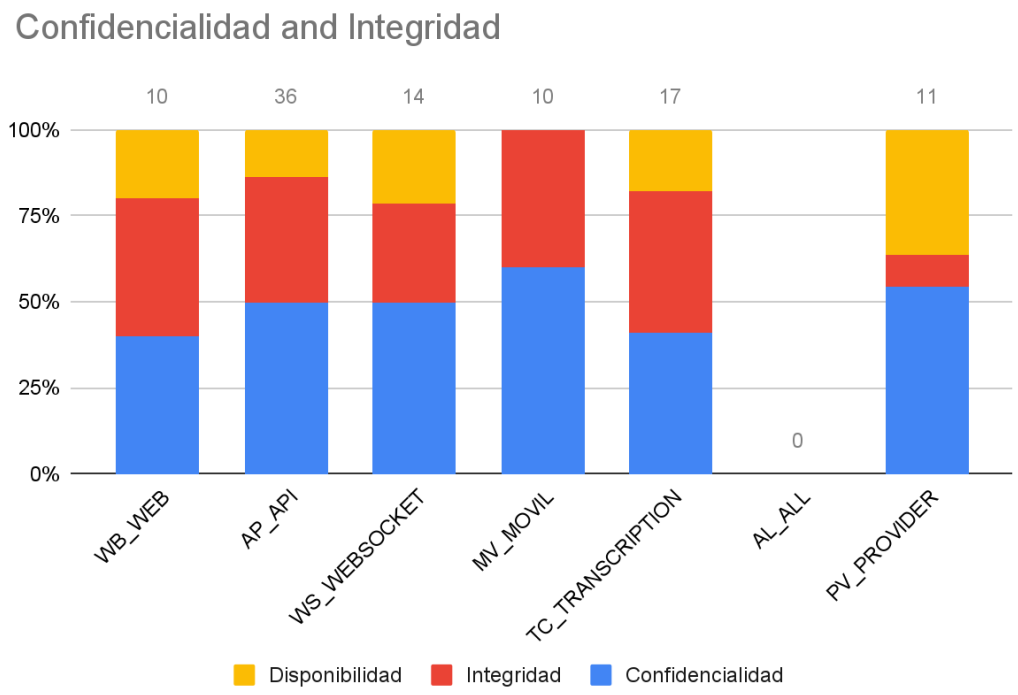
El análisis de métricas incluyó la categorización de los defectos según el tipo de vulnerabilidad detectada. Esto permitió identificar qué áreas requerían mayor capacitación en seguridad, asegurando que el equipo de desarrollo pudiera reforzar su conocimiento en aspectos clave.

A continuación, se muestra la clasificación de vulnerabilidades en términos de confidencialidad e integridad dentro de ANI en la *figura* 5.23:

También se realizó un análisis del nivel de severidad de las amenazas 5.24, permitiendo identificar las amenazas a priorizar para la corrección de fallos en función del riesgo que representan para la aplicación. La siguiente imagen ilustra la clasificación de riesgos de ANI según su nivel de criticidad:

Figura 5.23

Clasificación de Defectos: Confidencialidad e Integridad



Con base en los hallazgos obtenidos de las métricas, se llevaron a cabo acciones correctivas inmediatas y sesiones de formación para el equipo. Se realizaron charlas de seguridad, en las que se explicaron las vulnerabilidades detectadas, su impacto y las estrategias para mitigarlas.

Las sesiones incluyeron:

- Análisis de vulnerabilidades más críticas y su impacto en ANI.
- Explicación detallada de fallos recurrentes en el código y su corrección.
- Uso de herramientas de análisis de amenazas para mejorar la detección de defectos.

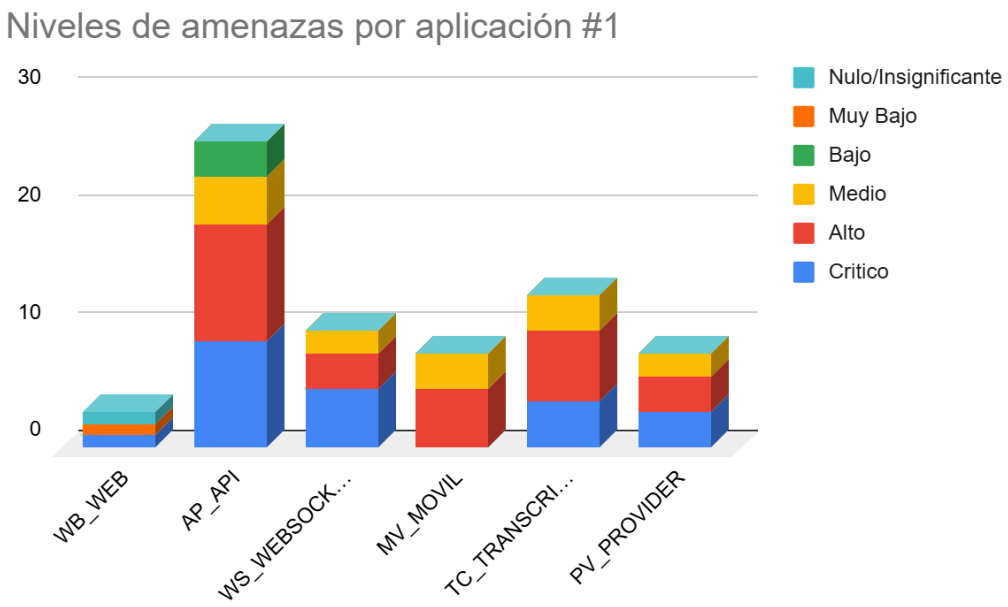
4. Verification

Validación de arquitectura (Architecture Validation)

Se realizó una evaluación de sus módulos, autenticación, roles, comunicaciones y gestión de datos. Esta validación se llevó a cabo utilizando un modelado de amenazas estructurado, donde cada componente del sistema fue analizado en función de los riesgos que podría presentar.

Figura 5.24

Niveles de Amenazas por Aplicación



Uno de los principales enfoques fue la creación del *Threat Modeling*, que permitió graficar la arquitectura general del sistema y detallar los módulos de seguridad implementados. Dentro de esta estructura, se establecieron criterios específicos de validación, asegurando que aspectos clave como autenticación, gestión de roles y comunicación segura estuvieran correctamente configurados.

El proceso de validación de arquitectura se basó en la revisión de cada módulo y su interacción con otros componentes del sistema. Se identificaron amenazas específicas para usuarios anónimos, usuarios autenticados y roles con permisos elevados.

La validación incluyó los siguientes aspectos fundamentales:

- **Autenticación y control de acceso:** Se verificó que la gestión de usuarios y permisos fuera sólida y alineada con las mejores prácticas.
- **Protección de datos:** Se revisó el manejo de información sensible para asegurar su cifrado y almacenamiento seguro.
- **Comunicación segura:** Se validó el uso de protocolos de seguridad en la transferencia de datos.

- **Gestión de claves y *Logs*:** Se analizaron los mecanismos de administración de claves y el registro de eventos de seguridad.

Cada vulnerabilidad detectada en la validación de arquitectura fue documentada en *Google Sheets - Thread Modeling*5.10, asegurando que cualquier falla en los controles de seguridad pudiera ser corregida a su debido tiempo.

Las deficiencias identificadas se categorizaron e integraron en el sistema de gestión de amenazas, con el objetivo de priorizar aquellas que representaban un mayor impacto. Aunque no se documentaron en extremo detalle, se aseguraron las conexiones más relevantes, enfocándose en aquellas que podían representar un riesgo de seguridad significativo.

Mitigación de arquitectura (Architectura Mitigation)

Este proceso fue llevado por nosotros el *Security-savvy staff* y reforzado mediante la aplicación de herramientas automatizadas como SAST, DAST y Fuzz Testing.

El análisis se realizó en tres niveles principales:

1. **Evaluación de la arquitectura contra OWASP Top 10:** Se revisaron vulnerabilidades críticas como inyección de código, errores de autenticación y configuraciones inseguras.
2. **Verificación con OWASP Application Security Verification Standard (OASV):** Se aseguraron controles de acceso, protección de datos y medidas de cifrado.
3. **Aplicación del modelo STRIDE:** Se identificaron amenazas relacionadas con suplantación de identidad, manipulación de datos, repudio de acciones, divulgación de información, denegación de servicio y escalamiento de privilegios.

Para que el equipo pueda abordar las amenazas identificadas, se aplicaron varias estrategias de mitigación en ANI, asegurando que la arquitectura fuera resistente a ataques y fallos de seguridad. Estas estrategias incluyeron:

- **Refuerzo en autenticación y gestión de sesiones:** Implementación de JWT con expiración controlada y uso de *MFA* (Autenticación Multifactor).

- **Protección de datos sensibles:** Cifrado de datos en reposo y en tránsito, asegurando que la información no pueda ser interceptada ni manipulada.

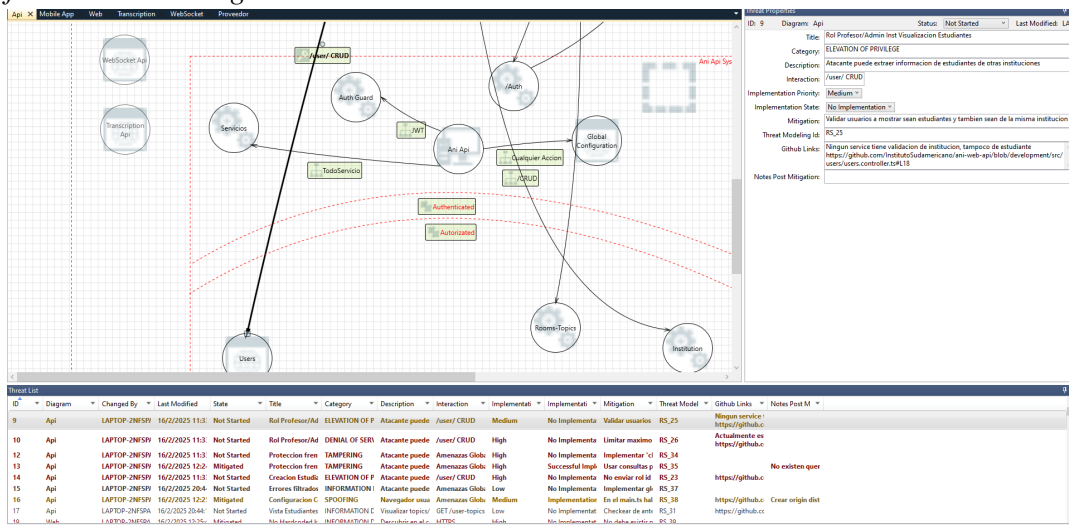
- **Seguridad en la comunicación:** Recomendación para el uso de TLS 1.2+ para encriptar la comunicación entre clientes y servidores.

- **Monitoreo de amenazas internas y externas:** Que implementen *Logs* de seguridad para detectar accesos sospechosos o actividades maliciosas.

Para modelar las amenazas se utilizo *Microsoft Threat Modeling Tool (MTMT)*, nos permite tanto gratificar el sistema, asignar las amenazas a cada conexión y generar fácilmente informes.

Figura 5.25

Microsft Threat Modeling Tool



Verificación de controles (Control Verification)

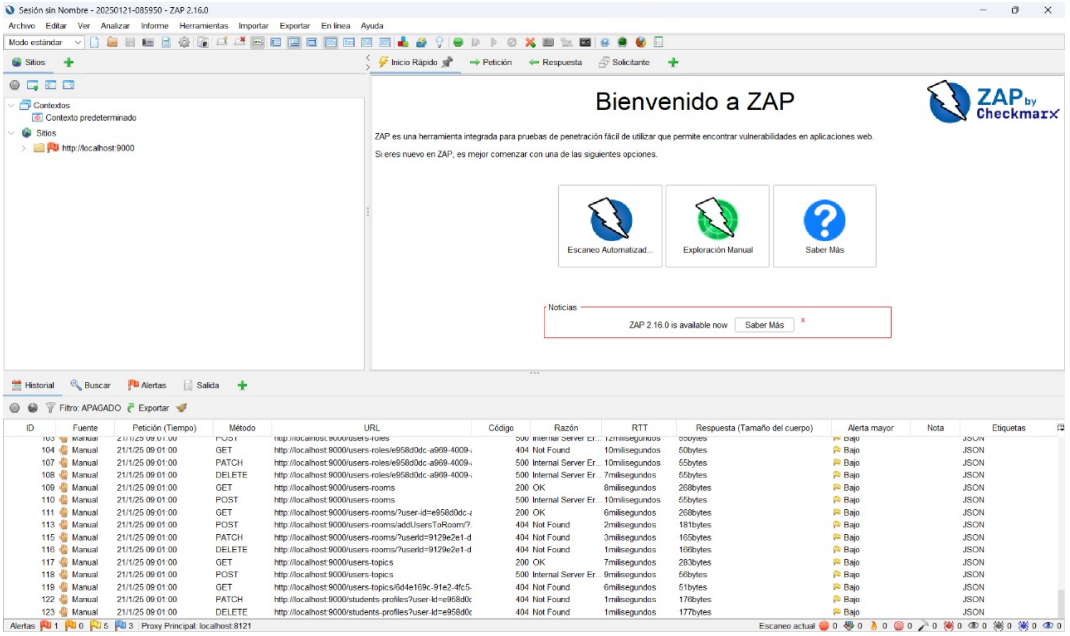
Esta parte se llevó a cabo mediante un enfoque estructurado que combinó pruebas automatizadas y revisiones manuales. Este proceso fue esencial para garantizar que los mecanismos de autenticación, control de acceso, validación de entradas, encriptación y manipulación de datos funcionaran correctamente y protegieran la aplicación contra ataques.

Para ello, se realizaron pruebas de penetración automatizadas con OWASP ZAP *Figura* 5.26, herramienta que permitió identificar vulnerabilidades en autorización, autenticación y manipula-

ción de parámetros. Además, se llevaron a cabo revisiones manuales de los mecanismos de encriptación y almacenamiento de datos, asegurando que tokens, contraseñas y claves estuvieran correctamente protegidos.

Figura 5.26

Uso de Herramienta OWASP ZAP



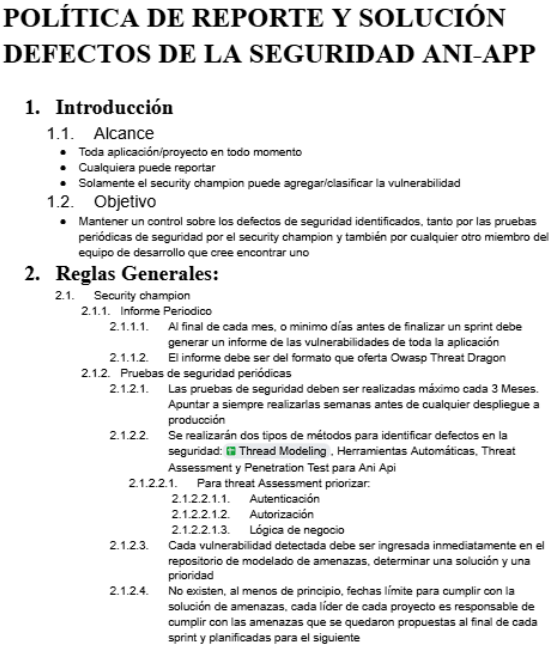
Para garantizar que las pruebas de seguridad sean un proceso continuo y obligatorio, se estableció una política específica dentro de *Google Docs - Política Reporte y Solución Defectos de la Seguridad Figura* 5.27. Esta política define en qué momentos es obligatorio informar y ejecutar pruebas de seguridad.

El proceso de verificación de controles permitió evaluar si ANI cumplía con los estándares de seguridad requeridos. A continuación, se resumen los principales hallazgos:

- Se validó la autenticación y el control de acceso en diferentes roles mediante OWASP ZAP.
- Se detectaron vulnerabilidades relacionadas con inyección SQL y manipulación de parámetros.
- Se reforzaron los mecanismos de encriptación, asegurando que puedan proteger las contraseñas y tokens de manera efectiva.

Figura 5.27

Política Reporte y Solución Defectos de la Seguridad



POLÍTICA DE REPORTE Y SOLUCIÓN DEFECTOS DE LA SEGURIDAD ANI-APP

1. Introducción

1.1. Alcance

- Toda aplicación/proyecto en todo momento
- Cualquiera puede reportar
- Solamente el security champion puede agregar/clasificar la vulnerabilidad

1.2. Objetivo

- Mantener un control sobre los defectos de seguridad identificados, tanto por las pruebas periódicas de seguridad por el security champion y también por cualquier otro miembro del equipo de desarrollo que cree encontrar uno

2. Reglas Generales:

2.1. Security champion

2.1.1. Informe Periodico

2.1.1.1. Al final de cada mes, o minimo días antes de finalizar un sprint debe generar un informe de las vulnerabilidades de toda la aplicación

2.1.1.2. El informe debe ser del formato que oferta Owasp Threat Dragon

2.1.2. Pruebas de seguridad periódicas

2.1.2.1. Las pruebas de seguridad deben ser realizadas máximo cada 3 Meses. Apuntar a siempre realizarlas semanas antes de cualquier despliegue a producción

2.1.2.2. Se realizarán dos tipos de métodos para identificar defectos en la seguridad: Thread Modeling , Herramientas Automáticas, Threat Assessment y Penetration Test para Ani Api

2.1.2.2.1. Para threat Assessment priorizar:

2.1.2.2.1.1. Autenticación

2.1.2.2.1.2. Autorización

2.1.2.2.1.3. Lógica de negocio

2.1.2.3. Cada vulnerabilidad detectada debe ser ingresada inmediatamente en el repositorio de modelado de amenazas, determinar una solución y una prioridad

2.1.2.4. No existen, al menos de principio, fechas límite para cumplir con la solución de amenazas, cada líder de cada proyecto es responsable de cumplir con las amenazas que se quedaron propuestas al final de cada sprint y planificadas para el siguiente

- Se implementó un proceso de pruebas recurrentes, documentado en la política de seguridad, asegurando que cualquier cambio en la aplicación sea sometido a pruebas de seguridad antes de su implementación.

Pruebas de mal uso/abuso (Misuse/Abuse Testing)

Se llevaron a cabo pruebas de *Fuzzing* enfocadas en la API principal de ANI. Estas pruebas buscaron identificar vulnerabilidades explotables mediante la inyección de datos aleatorios o mal-formados, una técnica comúnmente utilizada por atacantes para encontrar fallos en sistemas.

El fuzz testing es una técnica de caja negra que no parte de suposiciones sobre cómo debe comportarse el sistema, sino que busca errores inesperados mediante el envío de datos anómalos a los puntos de entrada de la aplicación. Gracias a esta metodología, es posible detectar fallos que podrían pasar desapercibidos en pruebas estructuradas o revisiones manuales.

Las pruebas de mal uso y abuso se centraron en evaluar las entradas críticas de la API de ANI, ya que es el módulo que procesa y almacena datos relevantes. Para ello, se utilizo la herramienta OWASP ZAP, aquí en las *Figuras* 5.28 y 5.29 están los resultados del *Fuzzing*:

Figura 5.28

Peticiones de OWASP ZAP

Peticiones Exitosas		
Método	Código	Total
GET	200	60

Peticiones Con Errores		
Método	Código	Total
DELETE	404	48
DELETE	500	12
GET	404	93
GET	500	35
PATCH	400	100
PATCH	403	12
PATCH	404	60
PATCH	500	160
POST	201	80
POST	400	148
POST	401	28
POST	404	50
POST	500	104

Figura 5.29

Cantidad de Amenazas y Posibles Ataques

Código	Método	Posible Ataque	Descripción
400	PATCH, POST	Validación de entrada o Inyección SQL	Datos mal formateados son enviados para probar validaciones en el servidor.
401	POST	Elusión de autenticación	Intentos de acceder a recursos sin credenciales o credenciales incorrectas.
403	PATCH	Escalada de privilegios	Intentos de acceder a recursos restringidos manipulando permisos.
404	GET, DELETE, PATCH, POST	Enumeración de recursos	Exploración de rutas o endpoints no existentes.
500	GET, DELETE, PATCH, POST	Explotación de vulnerabilidades del servidor	Ataques que causaron fallos internos, como inyecciones o datos maliciosos.
200	GET	Pruebas de acceso exitosas	Solicitudes que lograron acceder correctamente a recursos disponibles.

Cabe destacar que estas pruebas solo se realizaron en la API de ANI, ya que el resto de las APIs no almacenan datos críticos más allá de identificadores (IDs), reduciendo significativamente su superficie de ataque.

Línea de base escalable (Scalable Baseline)

se implementaron herramientas automatizadas de análisis estático y dinámico. Este enfoque permitió incrementar la cobertura de código en las pruebas y mejorar la capacidad de detección de vulnerabilidades en la aplicación, sin necesidad de realizar inspecciones manuales extensas.

Las pruebas de seguridad aplicadas se basaron en los siguientes métodos:

- **Static Application Security Testing (SAST):** Evaluación del código fuente sin ejecutarlo, detectando vulnerabilidades en su estructura y lógica.
- **Dynamic Application Security Testing (DAST):** Pruebas en tiempo de ejecución para observar el comportamiento de la aplicación frente a distintos tipos de ataques.

Se seleccionaron herramientas especializadas para cada tipo de prueba, asegurando que se ajustaran a la arquitectura de ANI y brindaran resultados en la *Tabla* 5.7:

Tabla 5.7: Herramientas de Pruebas de Seguridad en ANI

Herramientas de Pruebas de Seguridad en ANI		
Herramienta	**Tipo de Prueba**	**Ámbito de Aplicación**
OWASP ZAP	DAST (Dynamic Application Security Testing)	Pruebas de seguridad en API y WebSockets
SonarQube	SAST (Static Application Security Testing)	Análisis de código fuente en todas las aplicaciones de ANI
MobSF	SAST	Análisis de seguridad en la aplicación móvil (APK)

Comprensión profunda (Deep Understanding)

El análisis de seguridad en ANI no se limitó únicamente a herramientas automatizadas, sino que también se implementó un proceso de revisión manual y análisis en profundidad para detectar vulnerabilidades críticas en los módulos de mayor riesgo. Si bien las herramientas SAST y DAST facilitaron la identificación de vulnerabilidades superficiales, el análisis manual fue fundamental para detectar errores lógicos y problemas de seguridad difíciles de identificar mediante escaneo automático.

El enfoque utilizado se basó en una revisión independiente y espontánea del código, en la que se llevaron a cabo inspecciones detalladas de las funciones más sensibles de la aplicación, incluyendo módulos de autenticación, control de acceso, gestión de sesiones y validadores de entrada.

La metodología de Comprensión Profunda en ANI se desarrolló en varias etapas clave:

- **Evaluación de amenazas mediante análisis de código:** Se realizó una inspección manual del código fuente en busca de vulnerabilidades, las amenazas encontradas fueron divididas según la lista OWASP y STRIDE.
- **Análisis detallado de autenticación y control de acceso:** Se validaron las medidas implementadas para garantizar que los usuarios solo pudieran acceder a los recursos autorizados.

- **Validación de lógica de negocio:** Se revisaron posibles fallos en la lógica de la aplicación que pudieran permitir acciones indebidas o escalación de privilegios.
- **Revisión de findings en base a OWASP Top 10 y OASV:** Se establecieron criterios de revisión alineados con estándares de seguridad reconocidos.

A continuación, se presenta una *Tabla* 5.8 con una estructura para guiarse:

Tabla 5.8: Ejemplo Módulos Evaluados y Hallazgos de Seguridad en ANI

Módulos Evaluados y Hallazgos de Seguridad en ANI		
Módulo Evaluado	**Vulnerabilidad Identificada**	**Estado**
Autenticación	Tokens con tiempos de expiración demasiado largos	Mitigado
Autorización	Falta de validación adecuada en endpoints protegidos	En proceso
Gestión de sesiones	Sesiones no terminadas correctamente al cerrar sesión	Mitigado
Validación de entrada	Ausencia de limpieza en ciertos campos de entrada	En proceso
Lógica de negocio	Posible omisión de controles en acciones administrativas críticas	En revisión

Todos los hallazgos detectados fueron registrados y tratados de acuerdo con la *Política de Reporte y Solución de Defectos de Seguridad* **??**, estableciendo criterios de priorización en función del impacto y la probabilidad de explotación.

5. Operation

Detección de incidentes (Incident Detection)

El enfoque de detección se alineó con estándares de seguridad como OWASP Top 10 y OASV, definiendo procedimientos para la identificación de intentos de acceso no autorizado, ataques de fuerza bruta, patrones de errores 500 y otras anomalías en el sistema.

Debido a la naturaleza del proyecto y sus limitaciones de recursos, se optó por un sistema de *Logs* en formato TXT para las aplicaciones WEB y APK, el cual permite un análisis manual eficiente sin requerir agregación centralizada de registros.

En la *Tabla* 5.9 son datos que se deben tomar en cuenta cuando se este analizando los *Logs*.

Tabla 5.9: Criterios de Análisis de *Logs* en ANI

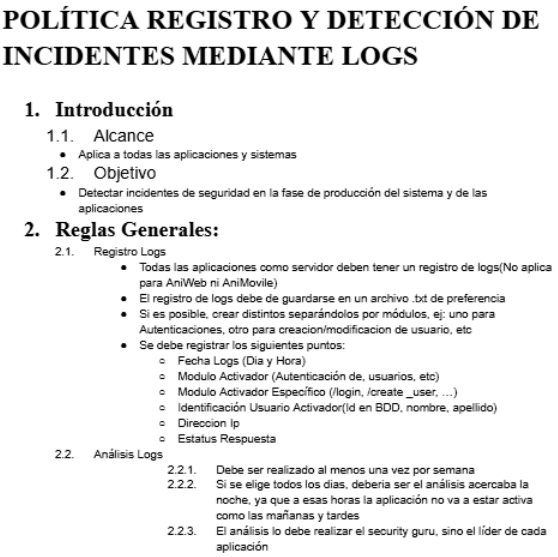
Criterios de Análisis de *Logs* en ANI		
Evento Registrado	**Descripción**	**Acción de Respuesta**
Intentos fallidos de login	Múltiples intentos en un corto período de tiempo	Analizar el origen y bloquear si es necesario
Errores 500 frecuentes	Fallos inesperados en la aplicación	Revisar los *Logs* y corregir la causa raíz
Peticiones excesivas	Tráfico anómalo o intentos de ataque automatizados	Implementar rate limiting si es necesario
Accesos no autorizados	Intentos de acceso sin los permisos adecuados	Notificar al *Security Champion* para revisión

Para garantizar que los incidentes sean identificados y atendidos rápidamente, se estableció un punto de contacto formal. En caso de detección de una amenaza, el *Security Champion* es el responsable de recibir los reportes y tomar las medidas necesarias.

Si el *Security Champion* no está disponible, el jefe de cada grupo de desarrollo asume la responsabilidad de revisar y gestionar los incidentes. Este procedimiento está descrito en *Google Docs - Política Registro y Análisis de Logs* 5.30, también se puede apoyar con la *Tabla* 5.9 que fue creada y documentada a la par, asegurando que la detección de amenazas no se vea afectada por la ausencia de una persona específica.

Figura 5.30

Política Registro y Análisis de Logs



POLÍTICA REGISTRO Y DETECCIÓN DE INCIDENTES MEDIANTE LOGS

1. **Introducción**
   - 1.1. Alcance
     - Aplica a todas las aplicaciones y sistemas
   - 1.2. Objetivo
     - Detectar incidentes de seguridad en la fase de producción del sistema y de las aplicaciones
2. **Reglas Generales:**
   - 2.1. Registro Logs
     - Todas las aplicaciones como servidor deben tener un registro de logs(No aplica para AniWeb ni AniMovile)
     - El registro de logs debe de guardarse en un archivo .txt de preferencia
     - Si es posible, crear distintos separándolos por módulos, ej: uno para Autenticaciones, otro para creacion/modificacion de usuario, etc
     - Se debe registrar los siguientes puntos:
       - Fecha Logs (Dia y Hora)
       - Modulo Activador (Autenticación de, usuarios, etc)
       - Modulo Activador Específico (/login, /create _user, ...)
       - Identificación Usuario Activador(Id en BDD, nombre, apellido)
       - Direccion Ip
       - Estatus Respuesta
   - 2.2. Análisis Logs
     - 2.2.1. Debe ser realizado al menos una vez por semana
     - 2.2.2. Si se elige todos los dias, deberia ser el análisis acercaba la noche, ya que a esas horas la aplicación no va a estar activa como las mañanas y tardes
     - 2.2.3. El análisis lo debe realizar el security guru, sino el líder de cada aplicación

Se determino que la revisión de *Logs* en ANI realizaría una vez por semana, con un mínimo de una vez al mes, dependiendo de la criticidad de cada aplicación. Se determinó que la API de Python IA es el componente más sensible, por lo que se recomendó priorizar su monitoreo debido a la posibilidad de ataques dirigidos.

Respuestas a Incidentes (Incident Response)

Al no tener un SDLC dentro del proyecto, se dejaron muchos temas importantes a lado, esto nos dejo con la tarea de crear un proceso estructurado de respuesta a incidentes en ANI. Este proceso asegura que cualquier evento de seguridad sea registrado, evaluado y atendido de manera oportuna por el equipo, reduciendo el impacto en la aplicación.

El *Security Champion* ha sido designado como la persona responsable de la gestión de incidentes, asegurando la coordinación de todas las acciones necesarias para mitigar riesgos. Además, cada líder de sistema o aplicación tiene la responsabilidad de colaborar en la gestión de incidentes dentro de su área específica, garantizando que los problemas sean abordados.

Otro tema es que se estableció un punto de contacto único para la comunicación de incidentes, permitiendo que todos los miembros del equipo sepan quién contactar y cómo proceder ante un evento de seguridad. Esto facilita la asignación de responsabilidades y asegura que los tiempos de respuesta sean óptimos.

Para garantizar la trazabilidad y análisis de incidentes, se creó un repositorio centralizado con acceso restringido en *Google Drive - UpRoles* 5.31, donde se almacenan todos los reportes de incidentes, incluyendo:

- Reportes
- Threat Analisis
- Security Incidents

Además, la gestión de incidentes está regulada por la *Política de Registro y Análisis de Logs*, la cual define cómo se deben documentar, clasificar y resolver los problemas de seguridad detectados, los temas que se deben tomar en cuenta para realizar esta actividad están en la *Tabla* 5.10.

Figura 5.31

UpRoles

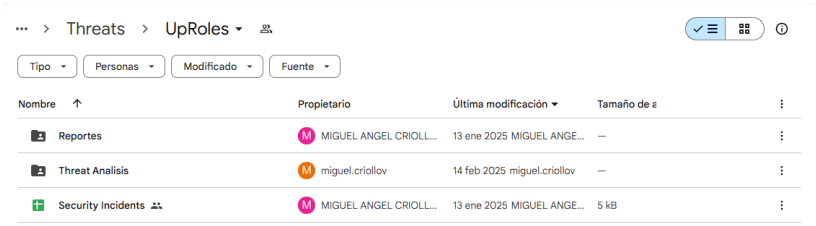


Tabla 5.10: Flujo de Gestión de Incidentes en ANI

Flujo de Gestión de Incidentes en ANI	
Fase	**Descripción**
Detección	Identificación de un incidente de seguridad mediante *Logs*, alertas o reportes manuales.
Registro	Documentación del incidente en el repositorio *UpRoles* con detalles sobre el evento.
Evaluación	Análisis del impacto y urgencia por parte del *Security Champion* y el equipo correspondiente.
Mitigación	Aplicación de medidas correctivas y preventivas según la naturaleza del incidente.
Seguimiento	Verificación de la resolución del problema y documentación de aprendizajes.

Endurecimiento de la configuración (Configuration Hardening)

Esta parte se enfocó en identificar estándares de seguridad para los principales componentes tecnológicos utilizados en la plataforma. A pesar de que inicialmente no existía una guía formal de configuración, se adoptaron prácticas de OWASP Cheat Sheets y otras fuentes reconocidas para garantizar una configuración segura.

Para establecer una configuración segura, se identificaron los siguientes elementos críticos dentro del stack tecnológico de ANI, estos estan clasificados en la *Tabla* 5.11:

Tabla 5.11: Configuraciones de Seguridad Aplicadas en ANI

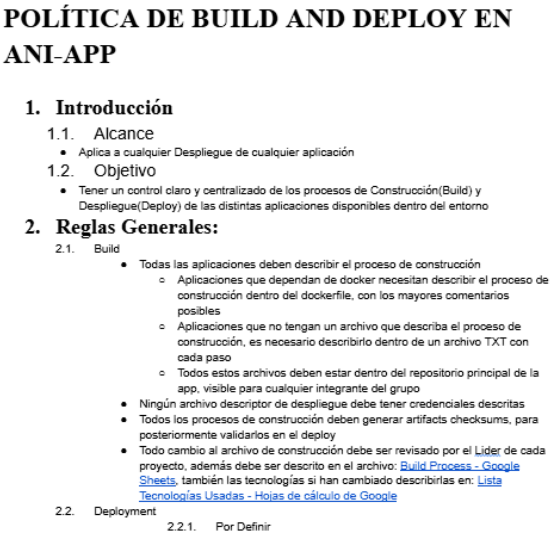
Configuraciones de Seguridad	
Componente	**Configuración Recomendada**
Ubuntu	Eliminación de paquetes innecesarios, firewall UFW, restricción de acceso SSH
Nginx	Seguridad en cabeceras HTTP, restricción de métodos inseguros
Docker	Uso de imágenes mínimas, restricción de privilegios y control de red interno
PostgreSQL	Restricción de acceso, cifrado de datos en tránsito, control de permisos granulares

Actualmente, ANI no cuenta con un proceso formalizado para gestionar configuraciones base de manera centralizada. Esto significa que las configuraciones pueden no aplicarse de manera uniforme en todos los entornos y la monitoreo de cumplimiento es limitado.

Sin embargo, para mitigar estos riesgos se ha creado varias recomendaciones que el equipo puede integrar:

Figura 5.32

Política Build and Deploy



POLÍTICA DE BUILD AND DEPLOY EN ANI-APP

1. **Introducción**
   - 1.1. Alcance
     - Aplica a cualquier Despliegue de cualquier aplicación
   - 1.2. Objetivo
     - Tener un control claro y centralizado de los procesos de Construcción(Build) y Despliegue(Deploy) de las distintas aplicaciones disponibles dentro del entorno
2. **Reglas Generales:**
   - 2.1. Build
     - Todas las aplicaciones deben describir el proceso de construcción
       - Aplicaciones que dependan de docker necesitan describir el proceso de construcción dentro del dockerfile, con los mayores comentarios posibles
       - Aplicaciones que no tengan un archivo que describa el proceso de construcción, es necesario describirlo dentro de un archivo TXT con cada paso
       - Todos estos archivos deben estar dentro del repositorio principal de la app, visible para cualquier integrante del grupo
     - Ningún archivo descriptor de despliegue debe tener credenciales descritas
     - Todos los procesos de construcción deben generar artifacts checksums, para posteriormente validarlos en el deploy
     - Todo cambio al archivo de construcción debe ser revisado por el Líder de cada proyecto, además debe ser descrito en el archivo: Build Process - Google Sheets, también las tecnologías si han cambiado describirlas en: Lista Tecnologías Usadas - Hojas de cálculo de Google
   - 2.2. Deployment
     - 2.2.1. Por Definir

- Uso de políticas definidas en *Google Docs - Política Build and Deploy* 5.32 para estandarizar configuraciones en cada despliegue.

- Aplicación de directrices básicas de seguridad en cada componente tecnológico, asegurando una configuración inicial robusta.
- Documentación de configuraciones efectivas en el equipo de desarrollo, permitiendo compartir aprendizajes y optimizar la seguridad progresivamente.

Parcheo y Actualización (Patching and Updating)

A pesar de haber utilizado los componentes mas óptimos para la creación de ANI, no se tomo en cuenta los parcheos o actualizaciones que podrían beneficiar a la segurida de este mismo, por lo tanto se estableció un proceso estructurado de parcheo y actualización de todos los componentes utilizados en la infraestructura y en el desarrollo del software. Este enfoque permitió mantener versiones actualizadas de librerías, frameworks y sistemas operativos, reduciendo el riesgo de explotación de vulnerabilidades conocidas.

La mayoría de las dependencias del proyecto están gestionadas a través de *GitHub Dependabot* 5.20, el cual proporciona reportes automáticos sobre actualizaciones disponibles y vulnerabilidades detectadas en librerías y paquetes utilizados en el código.

Un claro ejemplo seria como se demuestra en la siguiente *Tabla* 5.12:

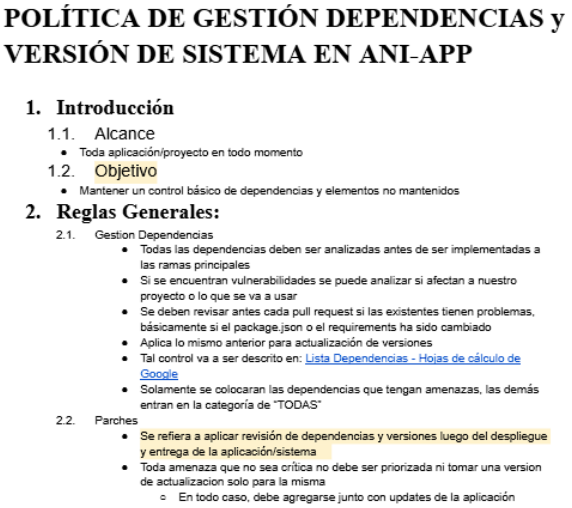
Tabla 5.12: Estado de Dependencias en ANI

Estado de Dependencias en ANI			
Componente	**Versión Actual**	**Última Versión Disponible**	**Estado**
Node.js	16.13.0	18.0.0	Pendiente de actualización
Nginx	1.21.6	1.23.3	Actualizado
Ubuntu	20.04 LTS	22.04 LTS	Evaluando compatibilidad
PostgreSQL	13.3	15.0	Pendiente de actualización
OWASP ZAP	2.10.0	2.11.1	Actualizado

La gestión de dependencias y actualizaciones sigue los lineamientos establecidos en *Google Docs - Política de Gestión de Dependencias y Versiones* 5.33, donde se detalla:

Figura 5.33

Política de Gestión de Dependencias y Versiones



POLÍTICA DE GESTIÓN DEPENDENCIAS y VERSIÓN DE SISTEMA EN ANI-APP

1. **Introducción**
   - 1.1. Alcance
     - Toda aplicación/proyecto en todo momento
   - 1.2. Objetivo
     - Mantener un control básico de dependencias y elementos no mantenidos
2. **Reglas Generales:**
   - 2.1. Gestion Dependencias
     - Todas las dependencias deben ser analizadas antes de ser implementadas a las ramas principales
     - Si se encuentran vulnerabilidades se puede analizar si afectan a nuestro proyecto o lo que se va a usar
     - Se deben revisar antes cada pull request si las existentes tienen problemas, básicamente si el package.json o el requirements ha sido cambiado
     - Aplica lo mismo anterior para actualización de versiones
     - Tal control va a ser descrito en: Lista Dependencias - Hojas de cálculo de Google
     - Solamente se colocaran las dependencias que tengan amenazas, las demás entran en la categoría de "TODAS"
   - 2.2. Parches
     - Se refiera a aplicar revisión de dependencias y versiones luego del despliegue y entrega de la aplicación/sistema
     - Toda amenaza que no sea crítica no debe ser priorizada ni tomar una version de actualizacion solo para la misma
       - En todo caso, debe agregarse junto con updates de la aplicación

- Cómo evaluar el impacto de una actualización antes de aplicarla.
- El procedimiento para reportar y registrar vulnerabilidades encontradas en dependencias de terceros.
- Las acciones necesarias en caso de detectar una vulnerabilidad crítica, priorizando parches de seguridad urgentes.

Para sistemas operativos y servidores como Ubuntu y Nginx, la revisión de actualizaciones tiene que realizarse de manera periódica y manual, asegurando que los cambios no afecten la estabilidad del entorno de producción.

Protección de datos (Data Protection)

Si bien la aplicación no maneja grandes volúmenes de información sensible, se recomienda agregar controles adecuados para garantizar la confidencialidad e integridad de los datos utilizados.

En ANI, la información almacenada y procesada se clasifica en dos categorías principales:

- **Datos no sensibles:** Información de acceso público o de bajo riesgo, como registros de acceso, nombres de usuario sin identificadores únicos y metadatos de debates.

- Da**tos sensibles:** Información personal identificable (PII), como nombres completos y credenciales de usuarios, utilizada exclusivamente dentro del sistema y protegida mediante políticas de seguridad adecuadas.

A continuación, se presenta una *Tabla* 5.13 con la clasificación de los datos en ANI:

Tabla 5.13: Clasificación de Datos en ANI

Clasificación de Datos		
Tipo de Dato	**Ejemplo**	**Nivel de Sensibilidad**
Información pública	ID de debates, fechas de acceso	Bajo
Información personal básica	Nombre de usuario, correo electrónico	Medio
Información sensible	Credenciales encriptadas, nombres completos	Alto

El manejo de la información en ANI sigue los lineamientos establecidos en la *Política de Protección de Datos Generales*, asegurando que los datos sean protegidos de acuerdo con su nivel de sensibilidad.

Algunos de los puntos clave de esta política incluyen que, no tiene que almacenar datos que sean innecesarios, no se debe propagar datos de producción a entornos de desarrollo y cualquier respaldo de los datos sensibles no debe generarse.

Si bien en el documento tiene buenos puntos que se deben a tomar en cuenta también se recomienda, fortalecer la seguridad de los datos en ANI, como:

- Implementar cifrado avanzado para la información sensible, mejorando la protección de credenciales y datos personales.
- Automatizar la detección de accesos inusuales a la base de datos, generando alertas en caso de actividad sospechosa.
- Incluir auditorías periódicas de acceso y uso de datos, asegurando que solo usuarios autorizados puedan interactuar con información crítica.

Gestión heredada (Legacy Management)

Crear una estrategia de gestión de software heredado en ANI fue fundamental, ya que es necesario enfocarse en identificar, evaluar y eliminar aplicaciones, versiones o dependencias en desuso. Esto ayuda a no tener vulnerabilidades que puedan tener las versiones anteriores, eso y que optimiza los recursos de desarrollo.

La detección de componentes obsoletos o en desuso se debe realizar de manera periódica y manual, a través de revisiones dentro del equipo de desarrollo. Cuando se identifica una aplicación, dependencia o funcionalidad obsoleta, esta se evalúa y se gestiona su eliminación conforme a los procedimientos establecidos en la *Política de Gestión de Dependencias y Versiones* 5.33.

Un buen ejemplo de como debería realizare esta actividad seria tomando en cuenta los puntos puestos en la *Tabla* 5.14 donde se muestra los componentes su estado y la acción que deberán realizar.

Tabla 5.14: Estado de Software Heredado en ANI

Estado de Software Heredado		
Componente	**Estado**	**Acción a Realizar**
NestJS versión 7.x	Obsoleto	Migrar a versión 9.x
Ubuntu 18.04 LTS	Obsoleto	Evaluar compatibilidad con 22.04 LTS
PostgreSQL 12	En desuso	Migrado a PostgreSQL 15
Dependencias sin soporte	Identificadas en Dependabot	Eliminar o reemplazar

Para poder proceder con la migración de los usuarios y componentes de software, se tiene que asegurar que las versiones obsoletas sean sustituidas gradualmente sin afectar la estabilidad de la aplicación.

5.2. Resultados Finales

5.2.1. Resultados del primer análisis de amenazas

El primer análisis al proyecto se realizo dentro del marco especificado de *SAMM* permitiendo identificar y clasificar los principales riesgos de seguridad presentes en las aplicaciones evaluadas que conforman el *Sistema Integrado ANI* mediante diversas etapas que logran un acercamiento cada vez mas especifico de las amenazas. El primer análisis que se genero fue se utilizo el modelo *STRIDE* para lograr un acercamiento general de las posibles amenazas que puede tener el sistema en la *tabla* 5.12.

Dado que esta evaluación se realizó en una única fase, los datos obtenidos reflejan únicamente el estado inicial del sistema con una comparativa de un mes después. Por lo tanto, los resultados deben considerarse como un diagnóstico preliminar y no como una evaluación de efectividad de mitigación, se describen mitigaciones para todas las amenazas, ademas de sugerencias, pero cable aclarar que el objetivo del estudio no es el mitigar todas las amenazas.

Para este primer análisis, se realizo mayoritariamente *Threat Assessment*, siguiendo las especificaciones de *STRIDE*, , *OWASP API Security Top 10*, *OWASP Mobile Security Top 10*, *OWASP Web Security top 10*, existen descripciones de amenazas que van desde puntos generales usando *STRIDE*, donde cubre toda la aplicación en puntos generales como ”Modificación de datos en trafico.º .Ejecución de acciones sin rol permitido”, estos nos sirve para futuro para detectar y pensar fácilmente cuales pueden englobar al sistema y definir específicos.

El segundo acercamiento se lo realizo usando los diversos top 10 de *OWASP*, este sirve para clasificar e ignorar amenazas que el sistema no tiene, por ejemplo jamas se usa sistemas criptográficos complejos o transacciones bancarias, por lo tal un análisis de las posibles amenazas siguiendo sugerencias de una organización conocedora es esencial para determinar, con un enfoque mas especifico y tomando en cuenta las tecnológicas y características de cada aplicación, las diversas amenazas posibles.

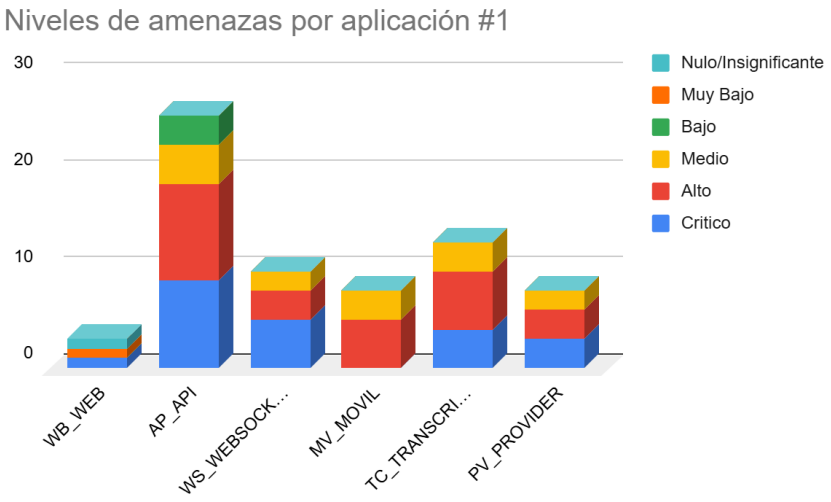
El tercer acercamiento desarrollo las amenazas especificas, con lo puntos anteriores se pudo determinar que y donde buscar amenazas, ademas se realizaron pruebas *SAST* para determinar

otras que se hayan pasado por alto.

La siguiente *figura* 5.34 ilustra los niveles de amenaza detectados en las distintas aplicaciones evaluadas. Estas amenazas se clasifican en diferentes categorías, desde ”nulo”(riesgo mínimo) hasta çrítico”(riesgo elevado con potencial de afectar tanto la aplicación como a sus usuarios). Este análisis proporciona una visión general del impacto y niveles de los riesgos identificados.

Figura 5.34

Primer Análisis de Niveles de Amenazas por Aplicación



Se puede apreciar claramente que *AniApi* (AP-API) es el sistema que tiene la mayor cantidad de amenazas, mas de la mitad de ellas de categoría critica y alta, esto nos demuestra que, siendo el sistema central y principal, contienen la mayor cantidad de amenazas con diferencia, la menor cantidad de amenazas la obtiene *AniWeb* (WB-WEB), la razón principal es que es una aplicación web de lado de cliente, significando que el usuario (significando su navegador web), gestiona todo lo que ve el usuario y accede, no existe forma a evitar que el usuario pueda modificar el código del mismo, por lo tanto aquí jamas se incluyen datos, credenciales ni ninguna dato sensible ya que siempre sera visible para el usuario, la mayoría de amenazas existentes son por ataques como *XSS* y similares, ademas sus vulnerabilidades sin de riesgos bajos a excepciona de una. La infraestructura de desligue *AniProvider* (PV-PROVIDER) tiene pocas amenazas, pero las existentes sin de alto grado, mayoritariamente por errores de mala configuración. La aplicación *AniMovile* (MV-MOVIL) sigue una pauta general a la parte web, al ser del lado del cliente, no

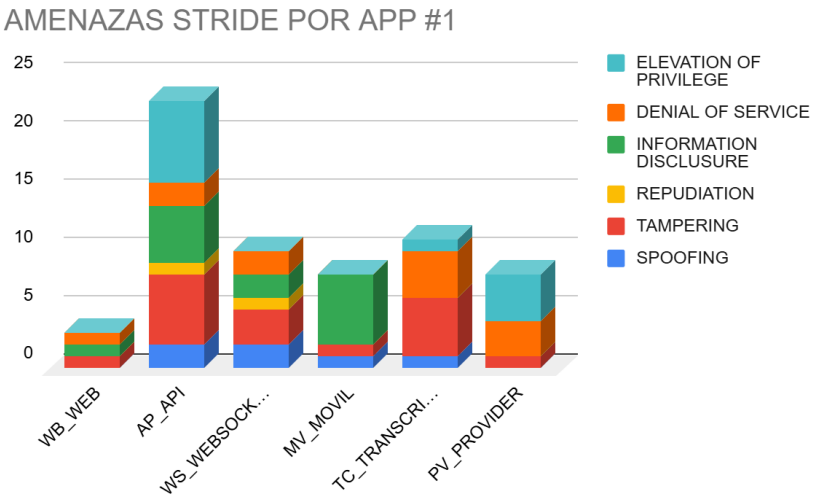
se puede restringir en su totalidad, lo que puede hacer el código del mismo, sabiendo que la app móvil no tiene características de pago u otra que requiera control de acceso dentro del mismo, las amenazas existentes tienen relación con guardado de secretos y autenticación.

La aplicación *AniWebsocket* (WS-WEBSOCKET) no cuenta con un alto numero de amenazas debido a que el sistema es bastante sencillo, la mayor cantidad vienen de lógica no aplicada, como falta de control por autenticación/autenticación que sin niveles críticos. finalmente esta *AniTranscription* (TC-TRANSCRIPTION), las principales amenazas vienen de falta de control de lógica de negocio controlar el orden de archivos de audio que llega u verificar que un usuario puede enviar el audio.

En la siguiente *figura* 5.35 se presentan las amenazas clasificadas bajo este modelo, segmentadas por aplicación analizada:

Figura 5.35

Primer Análisis de Amenazas STRIDE



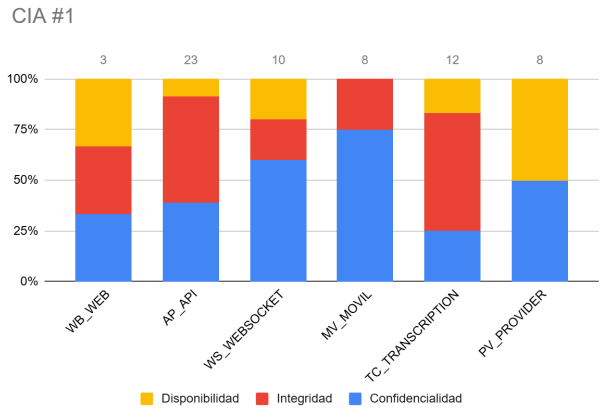
Podemos observar a simple vista que vulnerabilidades de tipo Elevation of Provilege/Tampering/Information Disclosure son las mas comunes esto tiene que ver en parte por el uso no controlado de IDs para modificar/crear datos los cuales pueden generar usuario con roles superiores o modificar salas a usuarios que no les pertenece la acción.

Otro modelo usado además del STRIDE fue el análisis del impacto de las vulnerabilidades en términos de los principios de Confidencialidad, Integridad y Disponibilidad (CIA) *Figura* 5.36.

Este enfoque permite evaluar cómo cada amenaza afecta estos principios. En el proyecto AP-API se identificó el mayor número de riesgos, ya que en esta parte se concentran la mayoría de las acciones críticas para el funcionamiento de la aplicación.

Figura 5.36

Primer Análisis de CIA



Se observa que el tipo de amenaza mas común con la confidencialidad de la aplicación seguido de su integridad, similar a como se comento en el gráfico5.35 de *Spoofing, Tampering, Repudiation, Information disclosure, Denial of service and Elevation of privilege* (STRIDE), por la falta de control de IDs, varios datos pueden ser visibles a personas que no deberían poder observar, como su cuenta, salas a la que pertenecen y demás, el alto indice de integridad indica que lógica no controlada puede afectar al funcionamiento de lógica interno, por ejemplo usuario ingresando a salas que no pertenecen o enviando audio al terminar la discusión pero sin validación.

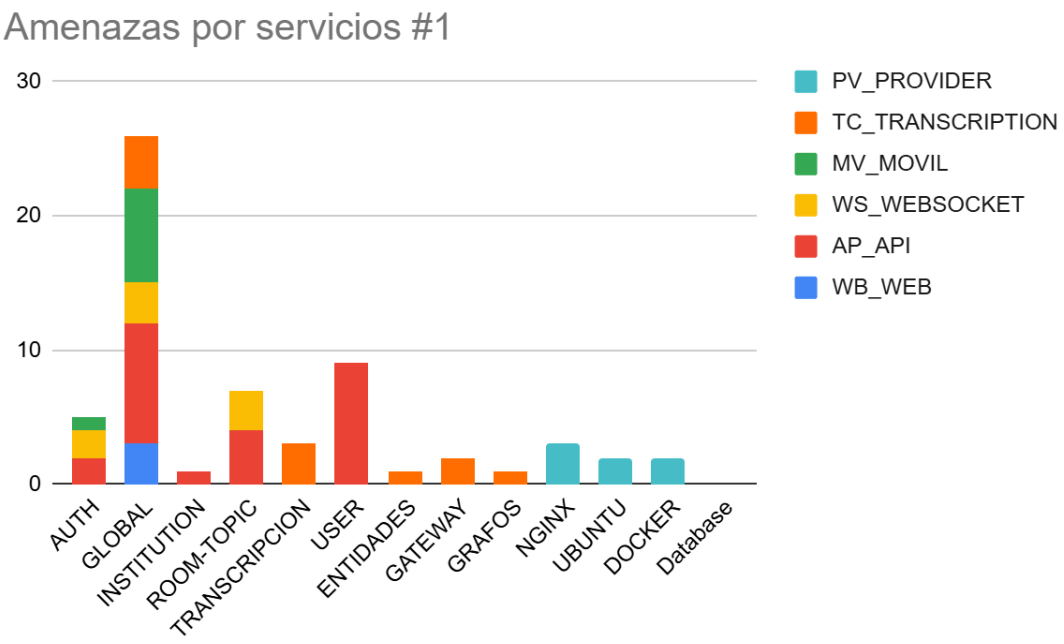
Para complementar la evaluación, se llevó a cabo una segmentación de amenazas por servicio (5.37) y por características (5.38). Este análisis permitió identificar los riesgos específicos asociados a cada componente, evidenciando que los servicios globales y el control de acceso basado en roles *RBAC* presentaban un alto número de amenazas detectadas. Además, se observó una concentración significativa de estas amenazas en el módulo TC-TRANSCRIPTION.

Los resultados5.375.38 obtenidos representan un primer acercamiento al panorama de amenazas dentro del sistema evaluado.

Esta vista es bastante útil, ya que tiene que ver con vulnerabilidades por características del sistema en conjunto, por ejemplo AUTH hace referencia a amenazas de autorización dentro de

Figura 5.37

Primer Análisis de Amenazas por Servicios



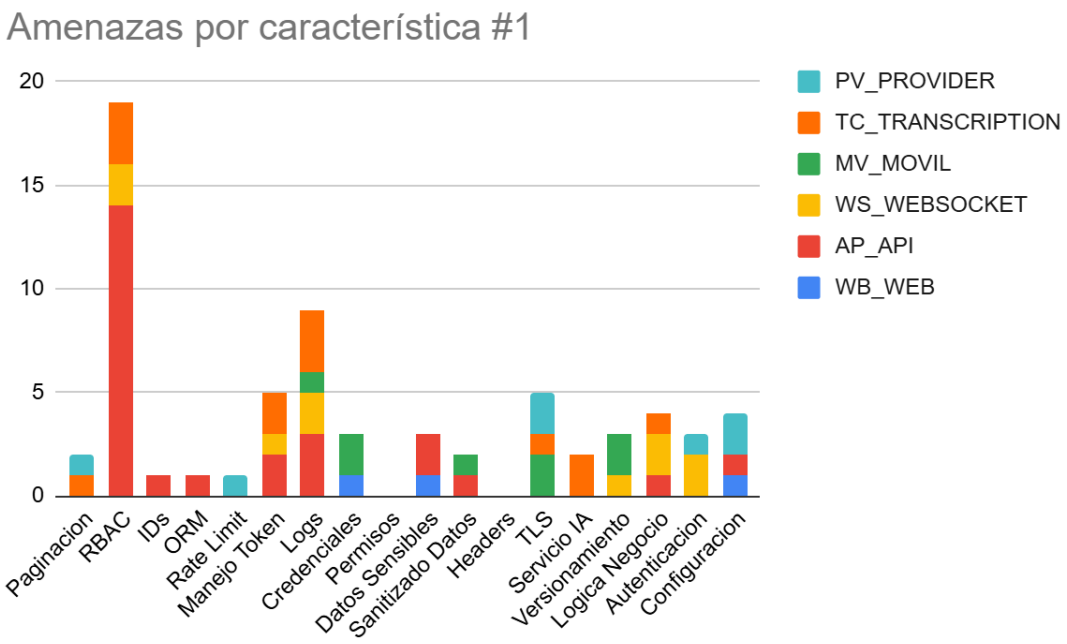
cualquiera de las Apis, cuales afectan al usuario, etc. La que mayor cantidad tiene es GLOBAL, la cual significa que afectan toda la aplicación y sistema, por ejemplo la api AP-API sin limite de peticiones por segundo es susceptible a *Ataque de Fuerza Bruta*, sabiendo que esta api esta desplegada en el mismo servidor que el resto, puede generar un alto consumo de recursos e inutilizar todas los otros servicios, otro puede ser implementar incorrectamente autenticación y dar acceso libre a todos lo que ofrece la api o aplicación sin restricciones.

El segundo gráfico5.38 refiere a características especificas de desarrollo, por ejemplo *RBAC* aplica tanto a la parte de api como a las vistas en la parte web, pero claro tienen su distinto nivel de importancia, de este podemos observar que es, con gran diferencia, la amenaza mas presente en el sistema, en especial AP-API, debido a que existen problemas de IDs y lógica, como eliminar una sala de profesor sin ser el propietario de la misma.

También destacan amenazas de *Logs*, debido a que ningún sistema al momento del primer análisis implementa registro de *Logs* de ninguna forma, se ha descrito en las reuniones con el equipo la necesidad de implementación del mismo, sin el mismo es casi imposible determinar

Figura 5.38

Primer Análisis de Amenazas por Características

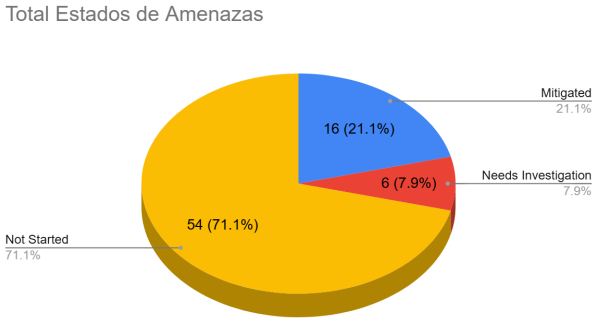


cuando ha existido un ataque y como.

5.2.2. Resultados de Mitigaciones del Primer Análisis

Figura 5.39

Primer Estado de Amenazas

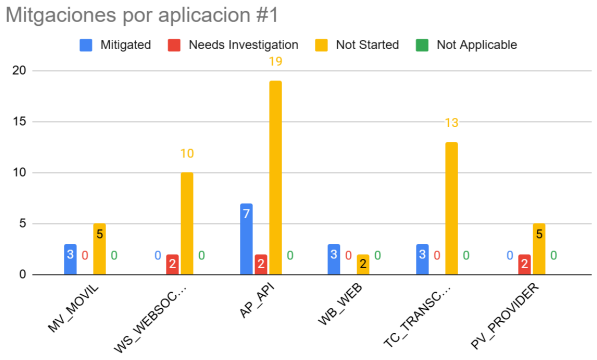


El análisis inicial de amenazas reveló un total de 76 riesgos identificados *Figura* 5.39, con una alta proporción de mitigaciones no iniciadas (vulnerables), conformando el 71.1 %, siendo un porcentaje demasiado alto, luego el 7.9 % están marcadas como ”Needs Investigation”, significando

que son amenazas existen en el sistema, pero es necesaria mas investigación para determinar si están Mitigadas o no, luego las mitigadas representa el 21 %. Estos resultados comprenden el total de los sistemas. Este resultado era esperable ya que no implementan ningún modelo de seguridad y la poca experiencia del equipo de desarrollo, sin embargo un importante porcentaje están cubiertas, esto refleja que la seguridad no es nula, pero no suficiente.

Figura 5.40

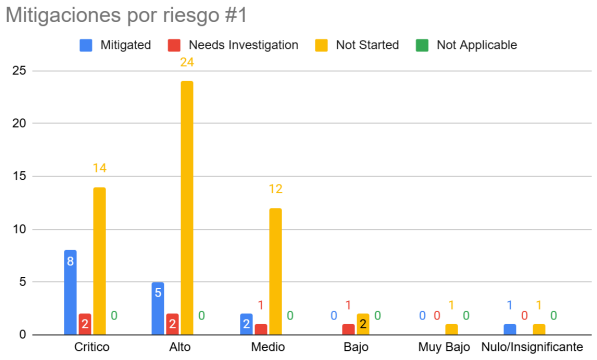
Primera Análisis de amenazas por Aplicación



Similar al gráfico 5.34 de amenazas por aplicación, la mayoría de amenazas sin mitigar se encuentra en AP-API con diferencia, y siguen en posiciones bajas MV-MOVIL y WB-WEB, la relación entre mitigados y no realizados es baja, de hecho la aplicación web tiene un mayor numero de amenazas mitigadas.

Figura 5.41

Primeras Mitigaciones por Riesgo

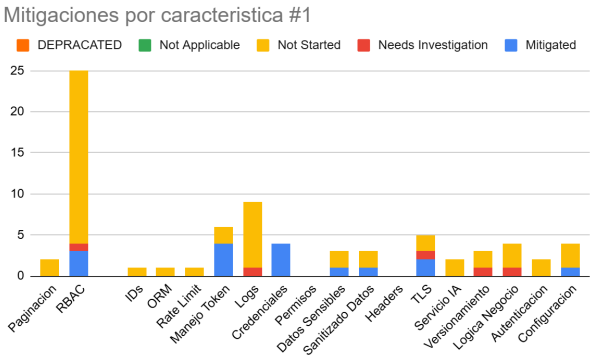


El análisis por riesgo *Figura* 5.40 nos presenta una clara tendencia del nivel de amenazas existentes, donde sobresalen niveles críticos y altos, las ademas la mayor cantidad de mitigaciones son

en de nivel critico, esto nos resalta que las pautas de seguridad aplicadas cubren partes importante del sistema y no resuelven amenazas triviales no poco importantes.

Figura 5.42

Primeras Mitigaciones por Característica

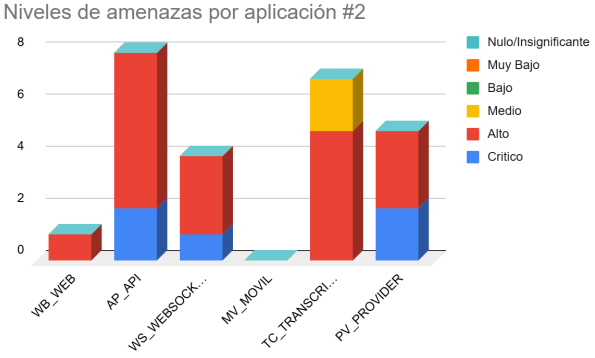


Similar al gráfico de la Figura 5.38, la característica *RBAC* tiene la mayor cantidad de amenazas no resultas, la parte de credenciales tiene todos sus literales debido a la correcta gestión de secretos, la parte de *Logs* tiene un importante numero de amenazas sin mitigar ya que se comento que no existe sistema de *Logs*.

5.2.3. Resultados del Segundo Análisis de las Amenazas

Figura 5.43

Segundo Análisis de Niveles de Amenazas por Aplicación



El segundo análisis se llevó a cabo un mes después de la primera evaluación de seguridad, esta se basa en el uso de las especificaciones de *OASV* del documento5.3 generado, luego se realizaron

análisis con herramientas unicamente a AP-API y nuevamente *Threat Assessment* de todas las aplicaciones, de las cuales se muestran unicamente las nuevas detectadas en este periodo.

La mayor cantidad de amenazas detectadas fueron de niveles altos y críticos, esto debido a las pruebas *Figuras* 5.26, 5.28 y 5.29 realizadas a AP-API, el motivo de esta selección se debe a que es el sistema mas importante y con mayores cantidades de vulnerabilidades y también porque probarlo sencillo de ejecutarlo, WS-WEBSOCKET se ignoro porque el sistema es demasiado sencillo y *Threat Assessment* es mas que suficiente para tener una idea clara se sus problemas, ahora con respecto a TC-TRANSCRIPTION, este es mas complicado de probar ya que requiere credenciales del servicio de IA, ademas el código no es extenso ni complejo por lo cual se lo puede cubrir fácilmente.

Figura 5.44

Segundo Análisis de Amenazas STRIDE

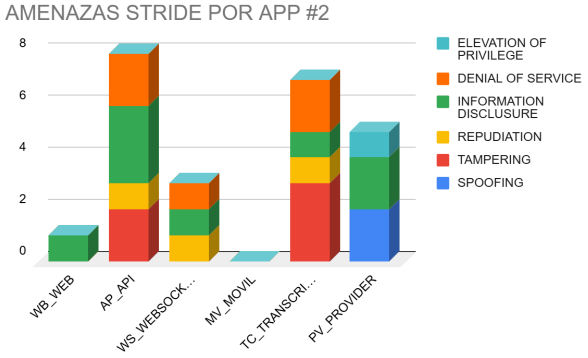


Figura 5.45

Segundo Análisis de CIA

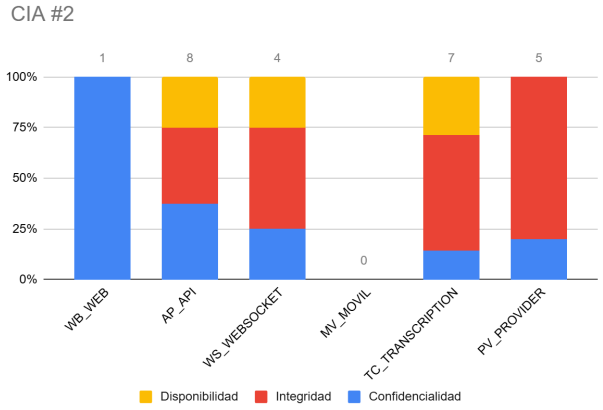


Figura 5.46

Segundo Análisis de Amenazas por Servicios

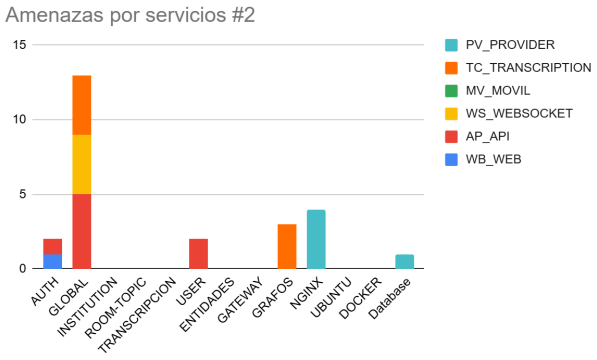
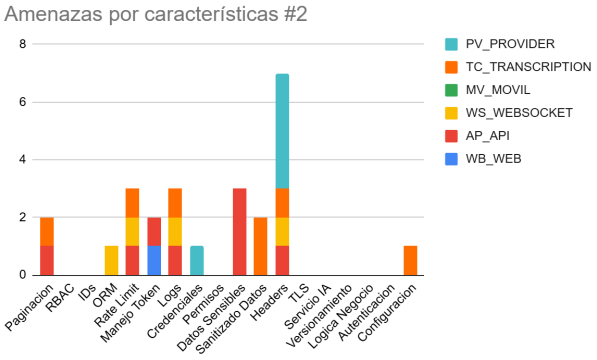


Figura 5.47

Segundo Análisis de Amenazas por Características



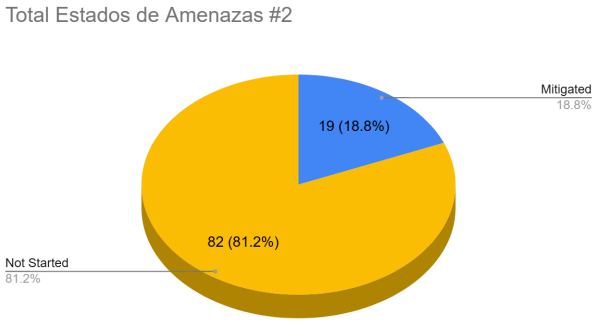
Los errores generados son mucho menores debido a que solo se incluyen nuevas amenazas y se actualizaron las existentes, se puede observar que existe un alto numero de amenazas de niveles críticos, en la segunda figura5.12 se observa mayor cantidad de amenazas ”Denial of Serviceza que se analiza que las tres apis existentes tienen limites personalizados de peticiones por endpoints dependiendo de su lógica.

Gracias a la integración de *OASV* y pruebas *DAST* estas amenazas destacan en la parte de confidencialidad e integridad según la Figura 5.45, bastante similar la Figura 5.36, se destaca también la no detección de nuevas amenazas en la parte móvil MV-MOVIL, ya que no se han hecho cambios significativos del mismo con relación a seguridad.

5.2.4. Resultados de Mitigaciones del Ultimo Análisis

Figura 5.48

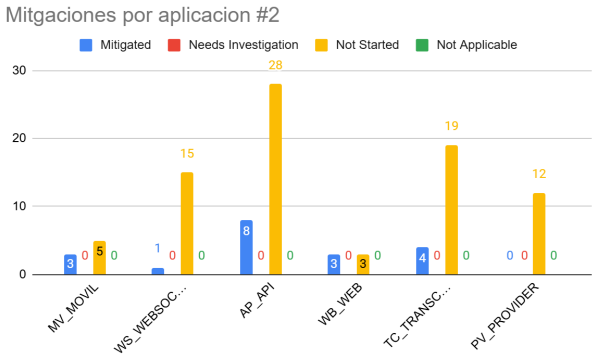
Segundo Estado de Amenazas



A nivel de aplicaciones *Figura* 5.49, las mitigaciones aumentaron a en tres, pero también el numero de amenazas sin mitigar, consolidando un 81 % de el total, la sección ”Needs Investigation”desaparece debido a que toda duda de la correcta implementación de las amenazas ha sido resuelta mediante preguntas individuales, investigación de mitigación de amenazas y revisión mas detallada del código.

Figura 5.49

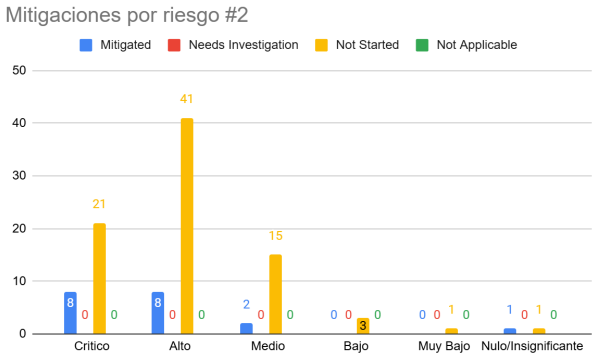
Segundo Análisis de amenazas por Aplicación



Desde una perspectiva de nivel de riesgo *Figura* 5.50, las amenazas críticas fueron el foco principal de mitigación, con un total de 21 mitigadas y 8 clasificadas como *No Aplicables*. Sin embargo, las amenazas de riesgo alto y medio continúan sin una atención suficiente, lo que mantenía latente el peligro de explotación en el sistema.

Figura 5.50

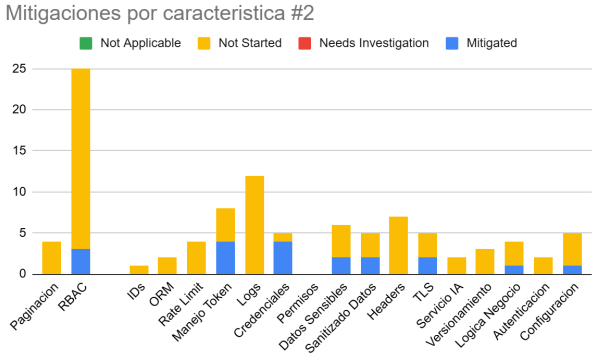
Segundas Mitigaciones por Riesgo



En cuanto a características específicas del sistema *Figura* 5.51, *RBAC* y *Logs* siguieron presentando una gran cantidad de amenazas sin resolver, lo que indicaba que la autenticación y el registro de eventos aún no eran prioridad dentro de la estrategia de seguridad. En contraste, características como Manejo de Tokens, Credenciales y TLS mostraron algunos avances, aunque con mitigaciones parciales.

Figura 5.51

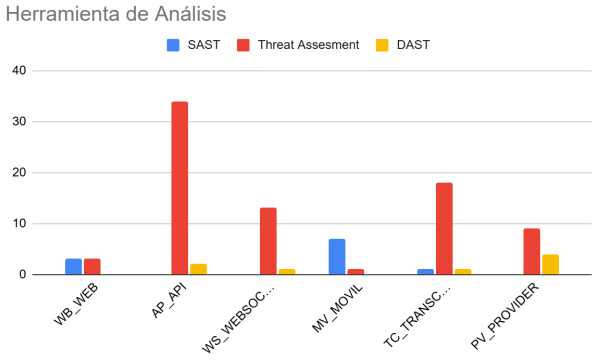
Segundas Mitigaciones por Característica



Los resultados de este primer esfuerzo de mitigación demuestran que, se avanzó en la clasificación y tratamiento de algunas amenazas críticas, persisten otros riesgos sin abordar, especialmente en áreas clave como la autenticación y la seguridad en el almacenamiento de datos. Se requiere un enfoque más estructurado y una estrategia de priorización para garantizar una mitigación efectiva en futuras evaluaciones.

Figura 5.52

Cantidad de Amenazas Detectadas por Tipo de Detección



5.2.5. Resultados del Curso

Para mejorar la comprensión y aplicación de prácticas de seguridad en el desarrollo de software, se diseñó un curso de capacitación basado en los principios de OWASP. La primera parte del curso introdujo los fundamentos de ciberseguridad, utilizando el OWASP Top 10 como referencia para explicar las vulnerabilidades más comunes en aplicaciones web. Se abordaron temas como inyección SQL y fallos en autenticación y control de acceso, con apoyo de material audiovisual para reforzar el aprendizaje. Al final de esta sección, los participantes fueron evaluados para medir su comprensión.

Tabla 5.15: Tabla de Resultados de Lección OWASP Top 10

Resultados de lección	
NOMBRE	LECCIÓN DE OWASP TOP 10
Alex Calle	10
David Correa	7.78
Estrella Garcia	10
Verónica Guerrero	8.89
Juan Gutierrez	8.89
Damian Olivo	8.89
Anthony Ramos	7.78

En la siguiente fase, el curso se enfocó en la seguridad de APIs, utilizando la lista OWASP API Security Top 10 como base para identificar amenazas comunes en entornos basados en servicios web. Se analizaron técnicas de ataque como exposición de datos sensibles, inyecciones y gestión inadecuada de tokens, lo que permitió a los participantes reconocer vulnerabilidades específicas de APIs y comprender estrategias de mitigación.

Tabla 5.16: Tabla de Resultados de Lección OWASP API Top 10

Resultados de lección	
NOMBRE	LECCIÓN DE OWASP API 10
Alex Calle	10
David Correa	10
Estrella Garcia	10
Verónica Guerrero	8.33
Juan Gutierrez	10
Damian Olivo	10
Anthony Ramos	8.33

Otro módulo clave del curso abordó los principios fundamentales de ciberseguridad, incluyendo principio de privilegio mínimo, validación de entradas, defensa en profundidad, protección de datos en reposo y en tránsito, y autenticación segura. Estos conceptos proporcionaron una base sólida para la implementación de medidas de seguridad efectivas en el desarrollo de software.

Tabla 5.17: Tabla de Resultados de Lección Principios de Seguridad

Resultados de lección	
NOMBRE	LECCIÓN DE LOS PRINCIPIOS DE SEGURIDAD
Alex Calle	10
David Correa	10
Estrella Garcia	10
Verónica Guerrero	10
Juan Gutierrez	9.33
Damian Olivo	10
Anthony Ramos	9.33

Además de los módulos obligatorios, se ofreció una sección opcional sobre seguridad en aplicaciones web y móviles, basada en los estándares OWASP ASVS y *MASVS*. Dado que algunos participantes tenían responsabilidades adicionales, se diseñó una versión reducida del contenido, enfocada en los conceptos esenciales sin evaluaciones obligatorias.

Tabla 5.18: Tabla de Resultados de Lección Aplicaciones Web y OWASP ASVS

Resultados de lección	
NOMBRE	LECCIÓN APLICACIONES WEB Y OWASP ASVS
Alex Calle	-
David Correa	-
Estrella Garcia	10
Verónica Guerrero	10
Juan Gutierrez	-
Damian Olivo	10
Anthony Ramos	8.33

Tabla 5.19: Tabla de Resultados de Lección Aplicaciones Móviles y OWASP MASVS

Resultados de lección	
NOMBRE	LECCIÓN APLICACIONES MÓVILES Y OWASP *MASVS*
Alex Calle	-
David Correa	-
Estrella Garcia	10
Verónica Guerrero	10
Juan Gutierrez	10
Damian Olivo	10
Anthony Ramos	10

Finalmente, se impartió una capacitación específica sobre OWASP SAMM, donde se explicó su estructura y aplicación en el contexto del desarrollo seguro. Se ofrecieron sesiones tanto en formato virtual como en una charla presencial, en la que se discutió en mayor profundidad su implementación en ANI. También se abordó el uso de herramientas para documentar vulnerabilidades y definir estrategias de mitigación, asegurando que los participantes pudieran aplicar el modelo en proyectos futuros. 5.20

Tabla 5.20: Tabla de Resultados de Lección OWASP SAMM

Resultados de lección	
NOMBRE	EVALUACIÓN OWASP SAMM
Alex Calle	-
David Correa	8
Estrella Garcia	9.33
Verónica Guerrero	4.57
Juan Gutierrez	9.33
Damian Olivo	-
Anthony Ramos	9.33

5.2.6. Respuestas de la Encuesta Final

La encuesta se aplicó al finalizar el curso desarrollado en *Moodle*, así como las charlas sobre seguridad impartidas al equipo de desarrollo. Su objetivo fue evaluar la percepción de los participantes respecto al curso y determinar si considerarían adecuado utilizarlo como capacitación para futuros interesados en ciberseguridad.

En la *Figura* 5.53 se presenta la evaluación de la satisfacción general con el curso, considerando si los temas abordados fueron interesantes o si quedaron aspectos por mejorar.

Figura 5.53

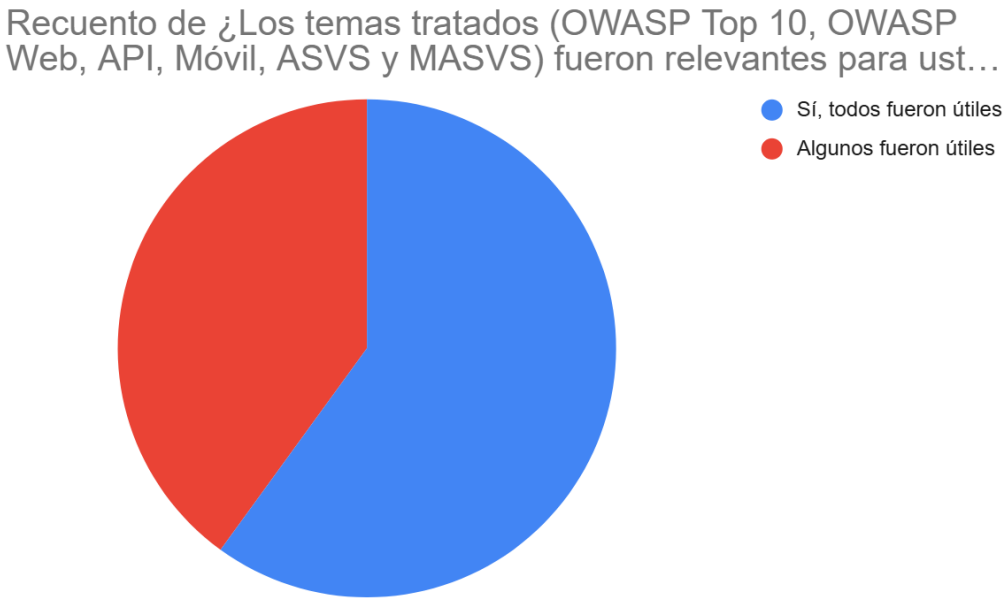
¿Cómo calificaría su satisfacción general con el curso?

Los resultados muestran que el 20 % de los participantes se encuentran muy satisfechos, el 40 % satisfechos y el 40 % restante mantiene una opinión neutral.

En la *Figura* 5.54 se analiza la relevancia de los temas abordados en el curso, específicamente *OWASP Top 10*, *OWASP Web Security top 10*, *OWASP API Security Top 10*, *OWASP Mobile Security Top 10*, ASVS y *MASVS*, en relación con su utilidad para la implementación de seguridad.

Figura 5.54

¿Los temas tratados (OWASP Top 10, OWASP Web Security top 10, OWASP API Security Top 10, OWASP Mobile Security Top 10, ASVS y MASVS) fueron relevantes para usted?

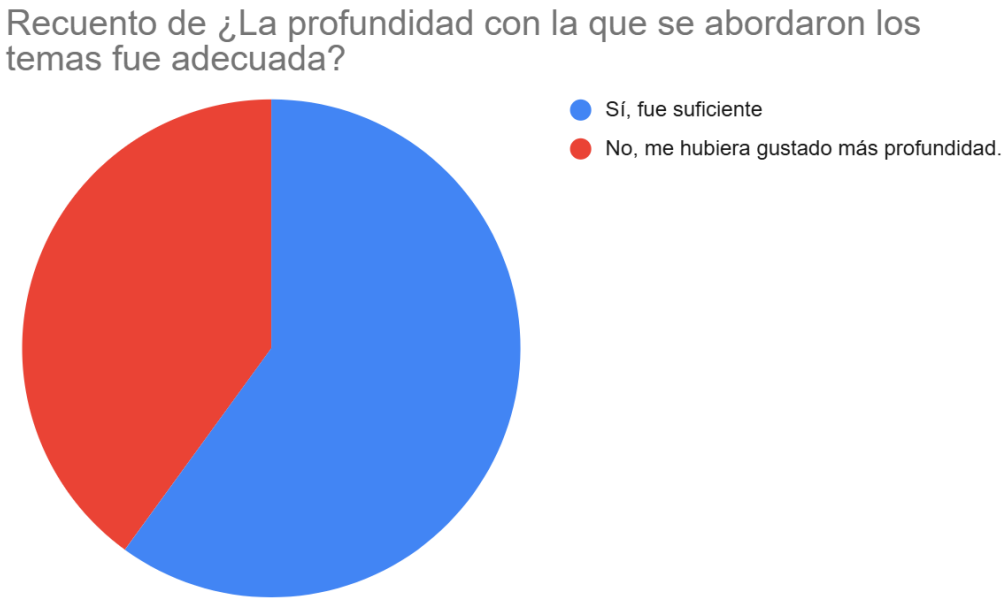


El 60 % de los encuestados considera que los temas fueron útiles, mientras que el 40 % opina que fueron algo útiles para la seguridad en la aplicación ANI.

En la *Figura* 5.55 se consulta si la profundidad con la que se abordaron los temas fue adecuada.

Figura 5.55

¿La profundidad con la que se abordaron los temas fue adecuada?

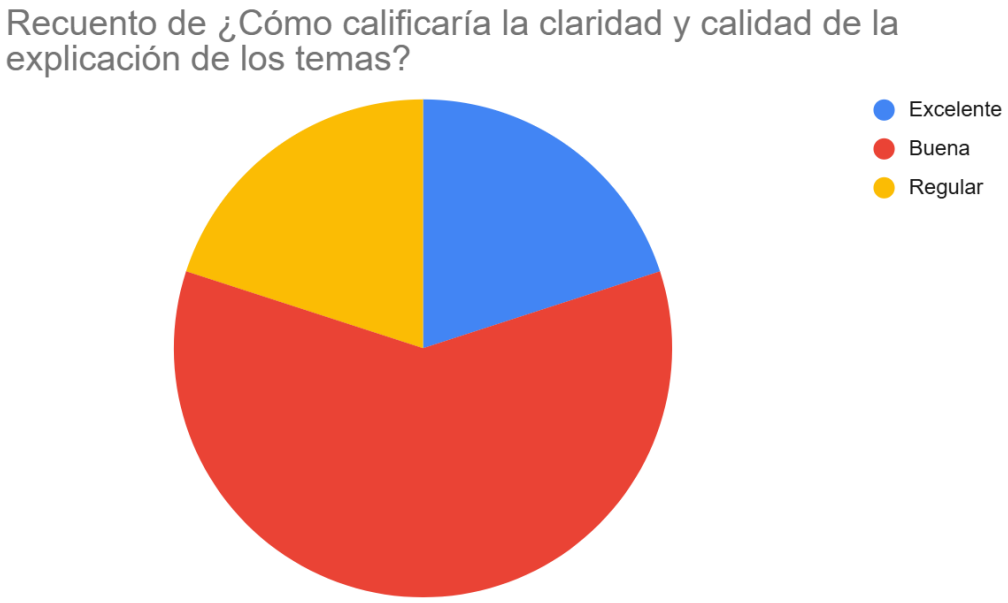


El 60 % de los encuestados considera que la profundidad fue suficiente, mientras que el 40 % preferiría una mayor profundización.

En la *Figura* 5.56 se evalúa la claridad y calidad de la información presentada en el curso.

Figura 5.56

¿Cómo calificaría la claridad y calidad de la explicación de los temas?

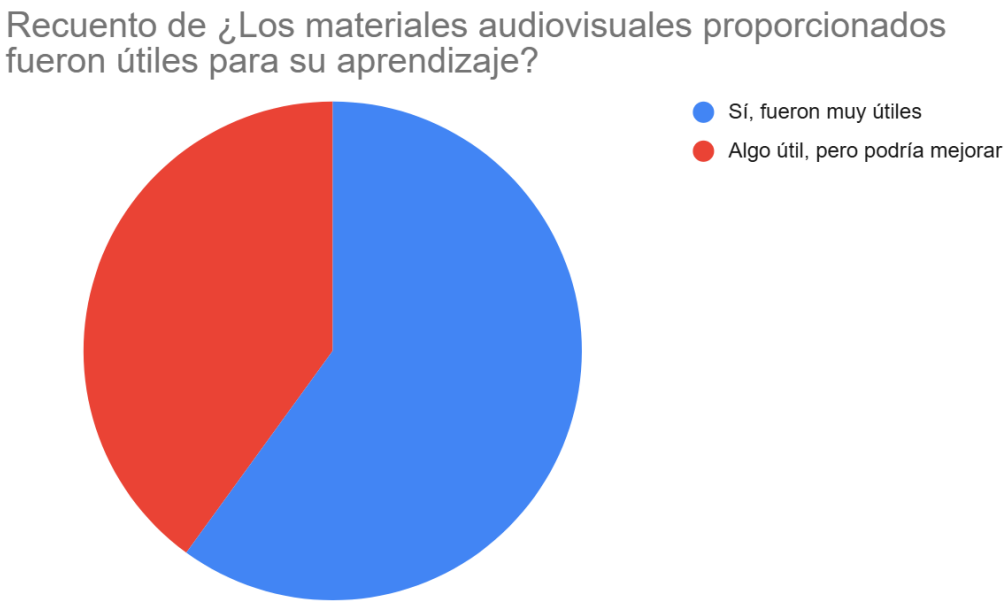


Los resultados muestran que el 20 % califica la explicación como excelente, el 60 % como buena y el 20 % como regular.

En la *Figura* 5.57 se analiza la utilidad de los materiales audiovisuales utilizados en el curso.

Figura 5.57

¿Los materiales audiovisuales proporcionados fueron útiles para su aprendizaje?



El 60 % de los encuestados considera que los videos fueron muy útiles, mientras que el 40 % los encontró útiles pero mejorables.

En la *Figura* 5.58 se consulta sobre la organización y estructura del curso.

Figura 5.58

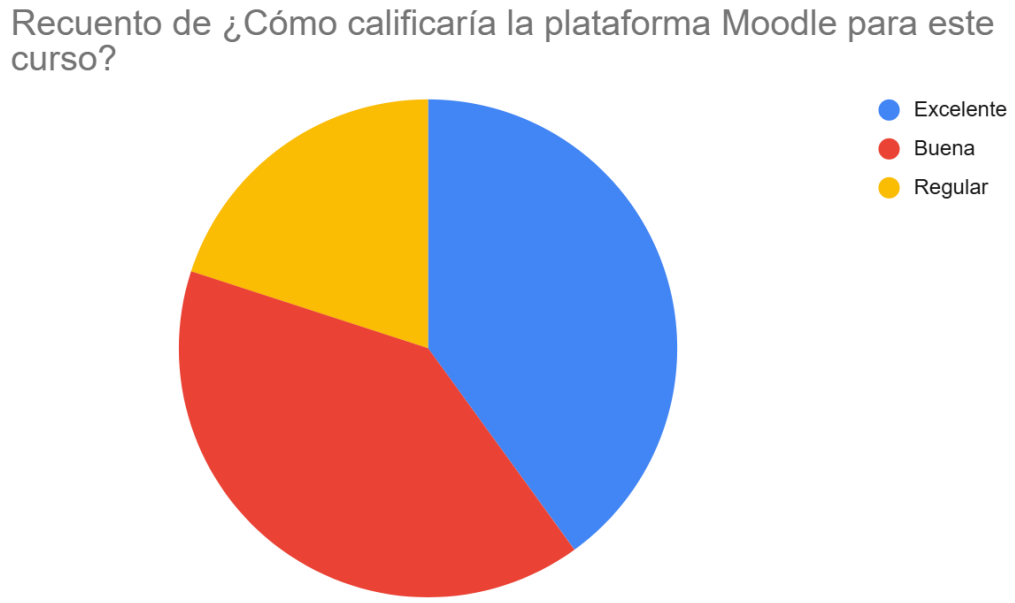
¿Considera que el curso tenía una estructura lógica y organizada?

El 60 % de los encuestados considera que la estructura del curso fue adecuada, aunque mejorable, mientras que el 40 % la califica como bien organizada.

En la *Figura* 5.59 se evalúa la percepción sobre la plataforma Moodle utilizada en el curso.

Figura 5.59

¿Cómo calificaría la plataforma Moodle para este curso?

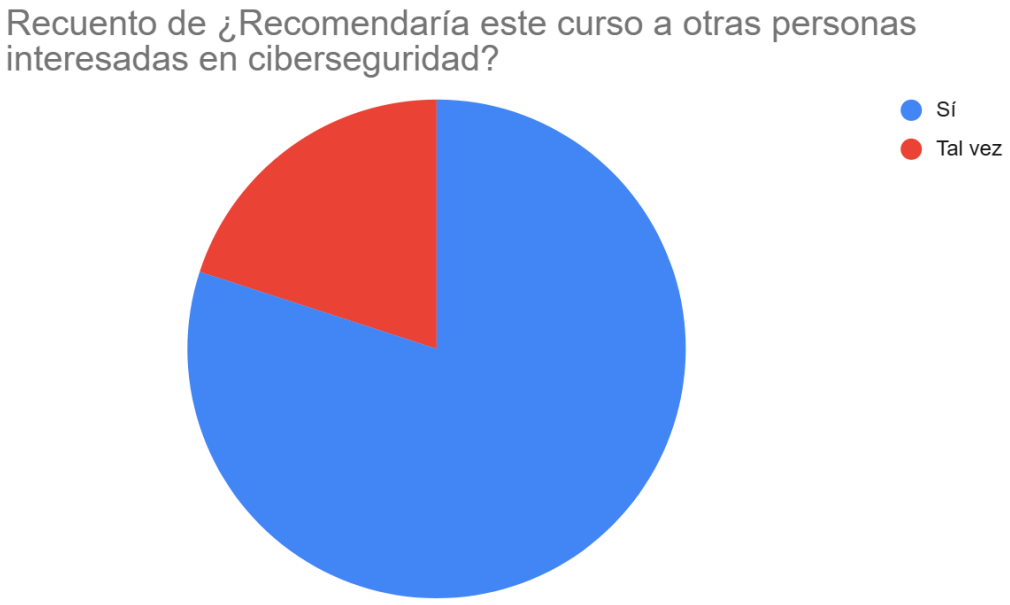


El 40 % de los participantes calificó la plataforma como excelente, el 40 % como buena y el 20 % como regular.

En la *Figura* 5.60 se consulta si recomendarían el curso a otras personas interesadas en ciberseguridad.

Figura 5.60

¿Recomendaría este curso a otras personas interesadas en ciberseguridad?

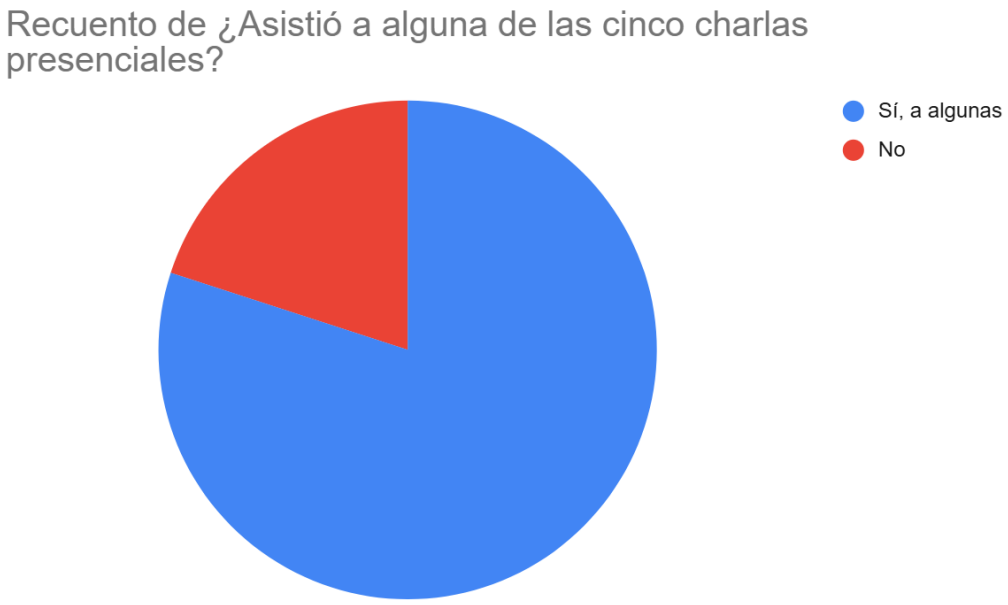


El 80 % de los encuestados recomendaría el curso, mientras que el 20 % respondió ”tal vez”.

En la *Figura* 5.61 se indaga sobre la asistencia a las charlas presenciales complementarias al curso.

Figura 5.61

¿Asistió a alguna de las cinco charlas presenciales?



El 80 % de los encuestados asistió a las charlas presenciales, mientras que el 20 % no lo hizo.

En la *Figura* 5.62 se analiza la calidad de las charlas presenciales.

Figura 5.62

Si asistió, ¿cómo calificaría la calidad de las charlas presenciales?

El 20 % de los participantes calificó las charlas como excelentes, mientras que el 80 % consideró que fueron buenas pero mejorables.

En la *Figura* 5.63 se consulta si las charlas complementaron el contenido del curso.

Figura 5.63

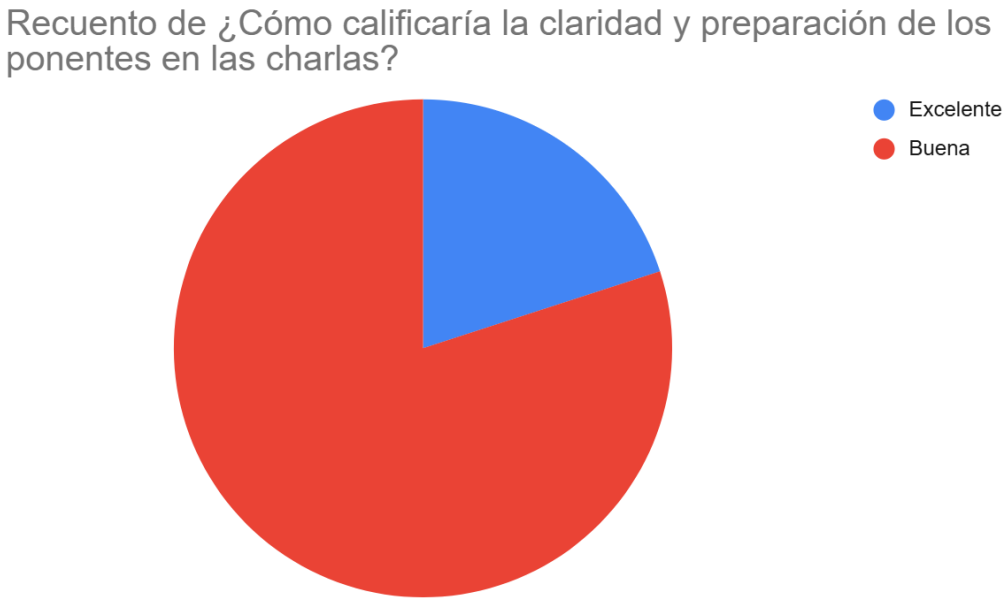
¿Las charlas presenciales complementaron bien el contenido del curso?

El 80 % de los encuestados considera que las charlas complementaron adecuadamente el curso, mientras que el 20 % cree que pudieron haber sido más útiles.

Finalmente, en la *Figura* 5.64 se evalúa la claridad y preparación de los ponentes en las charlas.

Figura 5.64

¿Cómo calificaría la claridad y preparación de los ponentes en las charlas?



El 20 % de los encuestados calificó a los ponentes como excelentes, mientras que el 80 % los consideró buenos.

Los resultados obtenidos pueden explicarse por varios factores. En primer lugar, el equipo de desarrollo es mayoritariamente inexperto en temas de seguridad, lo que puede haber generado una curva de aprendizaje más difícil al tratarse de un nuevo tema para ellos., otro es la falta de tiempo y la posible sobrecarga de tareas también influyeron en la percepción del curso, ya que algunos estudiantes pueden no haberlo realizado de manera consciente o con el nivel de dedicación necesario.

Esto se confirma por el hecho de que una parte de los participantes considere que los temas no se profundizaron lo suficiente puede deberse a que, para algunos, fue su primer acercamiento a la seguridad informática, mientras que los mas preparados esperaban un nivel más avanzado.

En conclusión, la capacitación en seguridad fue bien recibida y resultó útil para la mayoría de los participantes, pero requiere mejoras en la metodología, la profundidad del contenido y la interacción con los materiales.

CONCLUSIONES

Dentro de los modelos *SSDLC* investigados para aplicar al *Sistema Integrado ANI* se selecciono como mejor candidato a *SAMM*, debido a su flexibilidad para implementar estándares y análisis de amenazas junto con su guía detallada y fácil de seguir para personas no expertas para la implementación dentro de un ciclo *SDLC*. A través del uso de *Microsoft Threat Modeling Tool (MTMT)*, se logró documentar de manera efectiva las vulnerabilidades detectadas en las aplicaciones, estableciendo un plan de mitigación.

El uso de *OWASP Top 10*, *OWASP API Security Top 10* y *OWASP Mobile Security Top 10* junto con *STRIDE* se da mediante el *Threat Modeling* descrito y sugerido por *SAMM*, representado mediante la tabla 5.10 5.11 que junta la clasificación de amenazas *STRIDE*5.12 y los diversos top 10 de *OWASP* analizados por cada aplicación, esta unión de modelos permite un acercamiento por etapas al análisis de amenazas cada vez mas detallado hasta conseguir una clasificación efectiva de las mismas.

El modelo seleccionado describe el uso de herramientas *SAST*, *DAST* y de *Threat Modeling* para detectar distintas amenazas, la mayoría se han identificado usando la revisión de código 5.52, debido a que el código no es extenso y la revisión se da en un tiempo plausible, ademas de que este tipo de análisis permite una compresión profunda de las técnicas usadas por el equipo de desarrollo que muchas veces pasan por desapercibidas, el resto de pruebas de seguridad automáticas detectaron pocas amenazas, en el caso de *SAST*, amenazas repetidas o que no tienen la suficiente relevancia en el sistema como para describirlas, a comparación de pruebas *DAST* que a pesar que involucran mayor tiempo para implementarlas con la aplicación AP-API, devuelven resultado relevantes ademas de realizar otro tipo de pruebas como *Fuzz Testing*.

Para garantizar una buena implementación de estas mejoras, se diseñó un curso de capacitación en *Moodle* 5.8, orientado a otorgar o reforzar los conocimientos en seguridad del equipo de desarrollo que necesiten. A lo largo del programa, se abordaron temas clave como OWASP *SAMM*, *OASV* y *Threat Modeling*, proporcionando un marco teórico y práctico para la adopción de medidas de seguridad en el ciclo de desarrollo. Sin embargo, la evaluación final reflejó que, aunque los participantes lograron mejorar su comprensión, aún es necesario profundizar en áreas como

pruebas de penetración y gestión avanzada de vulnerabilidades.

Dentro de la implementación del modelo *SAMM* se generaron diversos documentos referentes a la seguridad, todos ellos cumplen y documentan diversos análisis de amenazas y sugieren practicas a seguir, desde políticas como detección y control de incidentes de seguridad mediante logs5.30, hasta políticas5.5 de protección de datos personales basados en la *Ley Orgánica de Protección de Datos Personales*, ademas de *Threat Modeling* completo de la aplicación, sugerencias de configuraciones, control de dependencias y análisis de estándar de seguridad de *OASV*.

La experiencia adquirida en este estudio fue valiosa, ya que nos permitió comprender la importancia de aplicar un modelo de seguridad desde las primeras fases del desarrollo, en lugar de integrarlo en etapas avanzadas, otra cosa que se evidenció a lo largo de este estudio fue que delegar toda la responsabilidad a una sola persona puede generar complicaciones a largo plazo, lo que resalta la necesidad de una distribución adecuada de las tareas de seguridad. Aquí fue donde quedó claro que la integración de herramientas automatizadas y el monitoreo continuo de los riesgos son fundamentales para garantizar la protección en cada fase del desarrollo de cualquier proyecto.

La reacción del equipo de desarrollo ante la capacitación en seguridad fue mayormente positiva, reflejada en los resultados de la encuesta. La mayoría de los participantes consideró que los temas abordados fueron útiles y relevantes, destacando el valor de aplicar OWASP *SAMM* y otras metodologías en el ciclo de desarrollo. Sin embargo, también se identificaron áreas de mejora. Una parte de los desarrolladores dio a entender que la profundidad de los temas podría haber sido mayor, lo que indica la necesidad de ampliar el contenido en futuras ediciones.

RECOMENDACIONES

La implementación de los modelos SAMM se la puede realizar de diversas maneras, sugerimos hacerlo de la manera más relacional posible. La unión de distintos módulos en un solo documento o en varios facilita la cohesión, el entendimiento y la escalabilidad ante posibles cambios futuros, se recomienda no generar documentos separados para cada sección, sino estructurarlos de manera general. Esto permitirá identificar rápidamente nuevas normas, políticas o amenazas y modificarlas globalmente si es necesario. La existencia de diversos niveles de mitigación permite realizar un primer acercamiento sencillo pero completo, ademas abre la posibilidad de aumentar su complejidad en un futuro.

En proyectos realizados en instituciones universitarias, hemos observado un limitado interés por parte de los estudiantes en implementar nuevas tareas. Esto es comprensible, ya que generar una base de políticas, estándares y otras actividades requiere una inversión de tiempo que el equipo de desarrollo suele no tener disponible, *SAMM* no establece que solo puede ser usado para empresas, pero es evidente que no funciona de la misma forma en los redactados entornos, realizar motivaciones a los estudiantes, así como presentaciones resumidas y cortas es lo esencial para captar su atención. La dependencia de completar los cursos de capacitación para obtener notas académicas nos ayudo de gran manera a que los estudiantes completaran el curso.

El análisis de amenazas que mas resultados brindo fue el *Threat Assessment*, a pesar de que también fue el que mayor tiempo tomo de todos, consideramos que este es el mas importante y que detecta la mayor cantidad amenazas cuando es realizado por un personal que tiene cierto nivel de experiencia en la mayoría de campos posibles, ademas de que el uso de *STRIDE*, *CIA*, y *OASV* ayuda en gran medida para detectar otras amenazas y repasar puntos que probablemente se hayan pasado por alto.

Consideramos que realizar *Threat Assessment* ayuda a entender de la mejor manera como funciona el código, que practicas se realizan, el nivel de conocimientos de los desarrolladores y en que puntos son los mas probables a tener incidencias de seguridad, es evidente la cantidad de tiempo que se necesita y se invierte para el mismo, pero se logra el mayor entendimiento del ecosistema de desarrollo, nuestro caso implica cierto nivel de separación del grupo de desarrollo, significando

que no existe presencia constantes en todas y cada una de las reuniones realizadas ni se toma participación directa del desarrollo, por lo cual este tipo de análisis permite conocer mas de cerca al tipo de código y practicas implementadas y entender en profundidad como funciona el sistema.

Apéndice A: Imágenes adicionales

8.1. Entrevistas

8.1.1. Primera Entrevista - 17/11/2024

Se abordó el diseño funcional de las aplicaciones ANI Móvil y ANI Web, detallando aspectos como el control docente, registro de usuarios, gestión de salas y debates (manejados por IA mediante websockets), y restricciones de perfiles. Se identificaron puntos críticos pendientes, como filtros institucionales, validación de correos, recuperación automatizada de contraseñas, y permisos en la versión web. Propuestas técnicas incluyeron bases de datos segmentadas, APIs de validación de correos, y modelos RBAC para roles (M. Criollo, A. Kevin, comunicación personal, 17 de noviembre del 2024).

Preguntas

Figura 8.1

Preguntas primera entrevista

- **¿La aplicación tiene acceso a información institucional o del cliente?**
- **¿Cómo la aplicación guarda y procesa datos personales y del debate, quienes tienen acceso a todos ellos?**
- **¿Cómo se manejan los roles?**
- **¿Qué roles tienen acceso a qué actividades y cuáles no?**
- **¿Cuál es el alcance de la aplicación(Libre a todo el internet, requiere una paga), es para un gran número de usuarios?**
- **¿Esta aplicación usa servicios de terceros, que importancia tiene estos servicios y que tanta posibilidad tiene de tener errores?**
- **¿La aplicación significa un servicio crítico a las instituciones?**
- **¿La aplicación interactúa con datos de otras entidades o industrias?**
- **¿La seguridad ha sido evaluada en la aplicación alguna vez?**
- **¿Está la aplicación integrada con otros sistemas internos(frontend, backend, API)?**
- **¿La aplicación depende de frameworks o librerías, que tan seguro es el control de sus versiones?**
- **¿La aplicación tiene un fuerte control de acceso(multifactor)?**
- **¿La aplicación funciona a tiempo real o afecta operaciones institucionales urgentes?**
- **¿La aplicación ha sido diseñada con prácticas de código seguro(validación de inputs, manejo de errores)?**
- **¿La aplicación tiene actualmente conocidas vulnerabilidades, versiones antiguas, puertos libres?**
- **¿Tiene planificado realizar parches de seguridad una vez se despliegue la aplicación?**

8.1.2. Segunda Entrevista - 12/11/2024

Se abordó el diseño funcional de las aplicaciones ANI Móvil y ANI Web, detallando aspectos como el control docente, registro de usuarios, gestión de salas y debates (manejados por IA mediante websockets), y restricciones de perfiles. Se identificaron puntos críticos pendientes, como filtros institucionales, validación de correos, recuperación automatizada de contraseñas, y permisos en la versión web. Propuestas técnicas incluyeron bases de datos segmentadas, APIs de validación de correos, y modelos RBAC para roles (M. Criollo, A. Kevin, comunicación personal, 17 de noviembre del 2024).

8.2. Reuniones Capacitación

8.2.1. Primera Reunión 07/01/2025

Se explico STRIDE, OWASP, Herramientas Enfocada en seguridad, se adoptó el modelo OWASP SAMM y el enfoque STRIDE para clasificar amenazas (suplantación, denegación de servicio, etc.). Se priorizó la integración de seguridad en todas las fases del desarrollo, uso de herramientas como SonarQube y Dependabot, y la creación de un rol ”Security Champion”. Se destacó la importancia de logs y la propuesta de un rol ”Super Admin”para limitar permisos excesivo(M. Criollo, A. Kevin, comunicación personal, 7 de enero del 2025).

8.2.2. Segunda Reunión 14/01/2025

Profundizó en el modelado de amenazas mediante OWASP SAMM y STRIDE, utilizando diagramas para visualizar vulnerabilidades en módulos clave (API, WebSocket, móvil). Se presento un reporte con 80 amenazas clasificadas por prioridad y estado, vinculadas a repositorios de GitHub. Se destacó la autenticación con JWT y el uso combinado de herramientas estáticas (SonarQube) y dinámicas (pruebas de fuzzing) para análisis de código (M. Criollo, A. Kevin, comunicación personal, 14 de enero del 2025).

9.0.1. AniApi

Desarrollado con NestJS, es la api gestionadora de la creación, modificación y eliminación de estudiantes, instituciones, salas y topics, su uso principal es la utilización de profesores y administradores, los estudiantes mayoritariamente pueden ver información de los mismos.

9.0.2. AniWeb

Aplicación web del lado del cliente desarrollada con Vite React, su uso es exclusivo para profesores y administradores, donde con la interfaz pueden crear, eliminar, modificar estudiantes, salas, y revisar los grafos generados de la discusion.

9.0.3. AniMobile

Aplicacion Movil desarrollada con React Native, su uso es exclusivo de estudiantes, donde pueden asistir a topics agregados por profesores, modificar su perfil y participar en discusiones por turnos y grabación de audio.

9.0.4. AniApiWebSocket

Api desarrollada con NestJs, gestiona la lógica de negocio para la gestión de discusiones, a quien dar la palabra y los tiempos, ademas de brindar información centralizada de los participantes.

9.0.5. AniApiTranscription

Api desarrollada con FastApi, gestiona la generación de transcripciones y grafos, ademas de los datos necesarios para la visualización, ademas se pueden subir los audios generados de los debates.

9.0.6. Proveedor

Infraestructura de desligue conformada por Ubuntu, nginx y docker para brindar el servicio completo de aplicación.

Bibliografía

Ahmed, F. M., Hassan, Das, S. R., & Hussain, M. (2023). Importance of Secure Software Development for the Software Development at Different SDLC Phases. *null*. https://doi.org/10.31124/advance.23947392.v1

Assal, H., Assal, H., Chiasson, S., & Chiasson, S. (2018). Security in the Software Development Lifecycle. *SOUPS @ USENIX Security Symposium*. https://doi.org/null

Bhaharin, S. H., Bhaharin, S. H., Mokhtar, U. A., Mokhtar, U. A., Sulaiman, R., Sulaiman, R., Sulaiman, R., Yusof, M. M., & Yusof, M. M. (2019). Issues and Trends in Information Security Policy Compliance. *International Conference on Research and Innovation in Information Systems*. https://doi.org/10.1109/icriis48246.2019.9073645

Caño Quintero, J. J. (2019). DevSecOps: integración de herramientas SAST, DAST y de análisis de Dockers en un sistema de integración continua. *null*.

de Vicente Mohino, J., de Vicente Mohino, J., Dagdeviren, Z. A., Higuera, J. B., Higuera, J. B., Higuera, J. R. B., Higuera, J. R. B., Higuera, J. R. B., Sicilia, J. A., Sicilia, J. A., & Montalvo, J. A. S. (2019). The application of a new secure software development life cycle (S-SDLC) with agile methodologies. *Electronics*. https://doi.org/10.3390/electronics8111218

Duggineni, S. (2023). Impact of Controls on Data Integrity and Information Systems, 29-35. https://doi.org/10.5923/j.scit.20231302.04

Eterovic, J., Silvestari, V., Zeballos, M., & Vera, A. F. (2023). Características de las herramientas de pruebas estáticas de seguridad de las aplicaciones. *ReDDI: Revista Digital del Departamento de Ingeniería*. https://doi.org/10.54789/reddi.7.2.3

Fucci, D., Alégroth, E., Felderer, M., & Johannesson, C. (2024). Evaluating software security maturity using OWASP SAMM: Different approaches and stakeholders perceptions. *Journal of Systems and Software*. https://doi.org/10.1016/j.jss.2024.112062

Gutfleisch, M., Schöps, M., Horstmann, S., Wichmann, D., & Sasse, M. A. (2023). Security Champions Without Support: Results from a Case Study with OWASP SAMM in a Large-Scale E-Commerce Enterprise. *European Symposium on Usable Security*. https://doi.org/10.1145/3617072.3617115

Hanna, M. (2018). Automated Software Testing Framework for Web Applications. *null*.

Mothanna, Y., Elmedany, W., Hammad, M., Ksantini, R., & Sharif, M. S. (2024). Adopting security practices in software development process: Security testing framework for sustainable smart cities. *Computers security*. https://doi.org/10.1016/j.cose.2024.103985

Shostack, A. (2008). Experiences Threat Modeling at Microsoft. *MODSEC@ MoDELS*.

Zambrano, A., Guarda, T., Valenzuela, E. V. H., & Quiña, G. N. (2019). Técnicas de mitigación para principales vulnerabilidades de seguridad en aplicaciones web. *Revista Ibérica de Sistemas e Tecnologias de Informação*, (E17), 299-308.