
Aplicación móvil para la gestión de residuos y reducción de la huella de carbono

Seguridad

Yongbum Park



UNIVERSIDAD DEL VALLE DE GUATEMALA

Facultad de Ingeniería



**Aplicación móvil para la gestión de residuos y reducción
de la huella de carbono**

Seguridad

Trabajo de graduación en modalidad de Megaproyecto Tecnológico
presentado por

Yongbum Park

Para optar al grado académico de Licenciado en Ingeniería en Ciencia de la
Computación y Tecnologías de la Información

Guatemala, agosto del 2024

Vo.Bo.:

(f) _____
Ing. Juan Cacerez

Tribunal Examinador:

(f) _____
Ing. Juan Cacerez

(f) _____
Ing. Eddy Castro

(f) _____
Ing. Jorge Yass

Fecha de aprobación: Guatemala, noviembre de 2024.

Prefacio

Desde el cuarto año de mi carrera, cuando elegí la especialidad en seguridad, descubrí el campo de la ciberseguridad, un área que rápidamente capturó mi interés por sus desafíos técnicos y su impacto en la protección de la información. A medida que me sumergía en sus fundamentos y prácticas, me di cuenta de la responsabilidad y el conocimiento profundo que exige esta disciplina, y esta revelación me motivó a contribuir con herramientas que fortalezcan la seguridad en el ámbito digital.

Este proyecto de graduación ha sido el resultado de largas horas de trabajo en equipo y colaboración constante, en donde el apoyo y guía del Ing. Juan Pedro Cáceres López fueron invaluable. Su experiencia y paciencia han sido una fuente de conocimiento fundamental, dándome las herramientas necesarias para afrontar cada desafío que surgió a lo largo del desarrollo del proyecto.

Quiero agradecer también a mis compañeros de equipo, en especial a Santiago, cuya dedicación y compromiso con el proyecto marcaron cada etapa de nuestro trabajo. La colaboración y creatividad compartidas me han dejado grandes aprendizajes y el privilegio de trabajar con amigos cuyo apoyo ha sido un pilar en este proceso.

Deseo expresar mi profundo agradecimiento a mi familia y amigos por su apoyo incondicional. Este trabajo es una muestra de los valores de perseverancia y dedicación que siempre me han inculcado, y su respaldo me impulsó a alcanzar cada objetivo.

Finalmente, quiero agradecer de manera especial a mi pareja, Helen Nataly, por su apoyo emocional durante todo este transcurso. Su confianza en mí, sus palabras de ánimo y su fe en que puedo lograr mis metas han sido un motor invaluable en este camino. Gracias por estar a mi lado y ser una fuente constante de fortaleza y motivación.

Prefacio	III
Lista de Figuras	IX
Lista de Cuadros	X
Resumen	XI
1. Introducción	1
2. Objetivos	3
2.1. Objetivo General	3
2.2. Objetivos Específicos	3
3. Justificación	5
4. Marco Teórico	7
4.1. gestión de residuos y Reducción de Huella de Carbono	7
4.1.1. Importancia de la gestión de residuos	7
4.1.2. Estrategias para la reducción de la huella de carbono	8
4.1.3. Herramientas tecnológicas en la gestión de residuos	10
4.2. Seguridad Cibernética en el Desarrollo de Aplicaciones	12
4.2.1. Importancia de la seguridad cibernética	12
4.2.2. OWASP Top 10	14
4.2.3. Buenas prácticas de seguridad en el desarrollo de software	16
4.3. Contexto Local: Guatemala	17
4.3.1. Situación actual de la gestión de residuos en Guatemala	17
4.3.2. Políticas y regulaciones ambientales	18
4.3.3. Desafíos y oportunidades en la seguridad cibernética en Guatemala	19
4.4. Metodologías de Desarrollo Seguro	20

4.4.1.	Introducción al desarrollo seguro de software	20
4.4.2.	Ciclo de Vida del Desarrollo de Software Seguro (SDLC)	21
4.4.3.	Principios de diseño seguro	22
4.5.	OWASP SAMM (Software Assurance Maturity Model)	24
4.5.1.	Introducción a OWASP SAMM	24
4.5.2.	Objetivos de OWASP SAMM	25
4.5.3.	Componentes y prácticas de SAMM	25
4.6.	Herramientas de Análisis de Seguridad	27
4.6.1.	SonarQube	27
4.6.2.	Appsweep	30
4.7.	Reglamento General de Protección de Datos (GDPR)	33
4.7.1.	GDPR y Guatemala	34
5.	Metodología	35
5.1.	Fase 1: Selección de Tecnologías	35
5.1.1.	Herramientas de Análisis Estático de Código	35
5.1.2.	Herramientas de Análisis Dinámico de Código para Aplicaciones Móviles	36
5.2.	Fase 2: Preparación e Instalación de Herramientas	39
5.2.1.	Reuniones Semanales de Seguimiento y Planificación	42
5.3.	Fase 3: Definición de la Estrategia de Seguridad	42
5.3.1.	Gobernanza: Educación y Orientación	42
5.3.2.	Diseño: Arquitectura Segura	43
5.3.3.	Implementación: Gestión de Defectos	43
5.3.4.	Verificación: Pruebas de Seguridad	43
5.4.	Fase 4: Evaluaciones de Seguridad y Mejora Continua	43
5.4.1.	Capacitación del Equipo en Clean Code y OWASP Top 10	43
5.4.2.	Reuniones Semanales de Seguimiento y Planificación	44
5.4.3.	Prueba de Phishing	45
5.4.4.	Análisis Estático del Código	46
5.4.5.	Migración a SonarCloud y Resultados de Cobertura	47
5.4.6.	Configuración de CI/CD en SonarCloud	51
5.4.7.	Revisión de Seguridad: Detección y Mitigación de Vulnerabilidades	52
5.4.8.	Evaluación de Seguridad en Modelos de IA y Chatbot	53
5.5.	Fase 5: Escaneo y Resultados de la Aplicación Móvil	55
5.5.1.	Escaneo Estático con MobSF	55
5.5.2.	Escaneo Dinámico con AppSweep	58
5.6.	Fase 6: Revisión y Optimización de Permisos en AWS IAM Identity Center	59
5.6.1.	Revisión de Grupos y Permisos	59
5.6.2.	Proceso de Validación y Ajuste de Permisos	60
6.	Resultados	61
6.1.	Educación y Orientación	61
6.1.1.	Resultados en la Prueba de Phishing	61
6.2.	Calidad del Código y Mejora Continua	62

6.2.1. Resultados en la Gráfica de Vulnerabilidades	62
6.2.2. Cobertura de Código	64
6.2.3. Code Smells	65
6.2.4. Código Duplicado	66
6.2.5. Complejidad Ciclométrica	67
6.2.6. Densidad de Comentarios	68
6.3. Escaneo de Vulnerabilidades en Aplicación Móvil	69
6.4. Análisis del Código de los Modelos de IA	70
6.4.1. Análisis de Seguridad y Cumplimiento de Estándares	71
6.4.2. Prácticas de Seguridad en el Chatbot	71
6.5. Análisis de Permisos en AWS	71
6.5.1. Oscar López: Permisos de DatabaseAdministrator	71
6.5.2. Pedro Pablo: Permisos de SystemAdministrator	72
6.5.3. Santiago Taracena y Yongbum Park: Permisos de ReadOnlyAccess	73
7. Análisis de resultados	75
7.1. Cultura de Seguridad y Capacitación en el Equipo	75
7.2. Reducción de Vulnerabilidades mediante SonarQube y SonarCloud	76
7.3. Exclusiones de Cobertura y Configuración de Calidad de Código	76
7.4. Code Smells y Resolución	77
7.5. Código Duplicado y Principio DRY	77
7.6. Complejidad Ciclométrica	78
7.6.1. Complejidad por Línea de Código (Ratio)	78
7.7. Densidad de Comentarios	78
7.8. Integración de Seguridad en Componentes de Inteligencia Artificial y Chatbot	79
7.9. Análisis de Vulnerabilidades en la Aplicación Móvil con AppSweep y MobSF	79
7.10. Configuraciones de Permisos en AWS	80
7.10.1. Oscar López: Revisión de Permisos de DatabaseAdministrator	80
7.10.2. Pedro Pablo: Revisión de Permisos de SystemAdministrator	81
7.10.3. Santiago Taracena y Yongbum Park: Revisión de Permisos de ReadOnlyAccess	81
8. Conclusiones	83
9. Recomendaciones	85
Referencias	87
Bibliografía	89
Anexos	90
A. Prácticas de Clean Code	90
A.1. Uso de Nombres que Revelan la Intención	90
A.2. Evitar la Desinformación	90
A.3. Crear Funciones Pequeñas	91
A.4. Evitar la Duplicación de Código	91

A.5. Nombres Descriptivos y Consistentes	91
A.6. Usar Nombres Pronunciables	92
A.7. Comentarios No Sustituyen el Código Limpio	92
A.8. Usar Excepciones en Lugar de Códigos de Retorno	92
A.9. Mantener Pruebas Unitarias Limpias	92
A.10. Organización de Clases y Principio de Responsabilidad Única	92
A.11. Evitar Comentarios Redundantes o Engañosos	93
B. OWASP Top 10	94
B.1. A01:2021 - Control de Acceso Quebrantado (Broken Access Control)	94
B.2. A02:2021 - Fallos Criptográficos (Cryptographic Failures)	95
B.3. A03:2021 - Inyección (Injection)	95
B.4. A04:2021 - Diseño Inseguro (Insecure Design)	95
B.5. A05:2021 - Configuración de Seguridad Incorrecta (Security Misconfiguration)	96
B.6. A06:2021 - Componentes Vulnerables y Desactualizados (Vulnerable and Outdated Components)	96
B.7. A07:2021 - Fallos en la Identificación y Autenticación (Identification and Authentication Failures)	96
B.8. A08:2021 - Fallos en la Integridad de Software y Datos (Software and Data Integrity Failures)	97
B.9. A09:2021 - Fallos en el Registro y Monitoreo de Seguridad (Security Logging and Monitoring Failures)	97
B.10. A10:2021 - Falsificación de Solicitudes del Lado del Servidor (Server-Side Request Forgery, SSRF)	97
C. OWASP Software Assurance Maturity Model (SAMM)	98
C.1. Prácticas de Seguridad en SAMM	98
C.1.1. Governance	99
C.1.2. Design	99
C.1.3. Implementation	100
C.1.4. Verification	100
C.1.5. Operations	101
C.2. Evolución y Futuro del OWASP SAMM	101
D. Análisis con Mobile Security Framework (MobSF)	102
E. Google Test de Phishing	106

Lista de Figuras

4.1. La huella de carbono de una organización a lo largo de su cadena de valor.	9
4.2. Ejemplo de línea de tratamiento de gases de combustión secos con captura de gases ácidos por bicarbonato de sodio y SCR	11
5.1. Registro de Scan del 5 de Agosto.	39
5.2. Resultado de Scan del 5 de Agosto.	39
5.3. Página de inicio de AppSweep.	40
5.4. Página de carga de archivos en MobSF.	41
5.5. Resultados de phishing del equipo.	46
5.6. Resultado de Scan del 14 de Agosto.	47
5.7. Primer escaneo en SonarCloud.	48
5.8. Configuración de exclusiones en SonarCloud.	49
5.9. Configuración de calidad en SonarCloud.	50
5.10. Cobertura del código en SonarCloud.	51
5.11. Grafica de cobertura en soanrcloude.	51
5.12. Configuración de SonarCloud.	52
5.13. Gráfica de vulnerabilidades detectadas en SonarQube.	52
5.14. Gráfica de vulnerabilidades detectadas en SonarCloud.	53
5.15. Resultado del escaneo del 20 de octubre para el segundo modelo.	54
5.16. Primer escaneo del chatbot en SonarCloud.	54
5.17. Último escaneo del chatbot en SonarCloud.	55
5.18. Repositorio de MobSF.	56
5.19. MobSF ejecutándose localmente.	57
5.20. Interfaz gráfica de MobSF	57
5.21. Resultado del escaneo inicial en MobSF.	58
5.22. Resultado del escaneo en AppSweep.	58
5.23. AWS IAM Identity Center: Cuentas y Grupos Predeterminados.	59
5.24. AWS IAM Identity Center: Permisos de Usuario Revisados.	59

6.1. Gráfica de vulnerabilidades detectadas en SonarQube.	63
6.2. Gráfica de vulnerabilidades detectadas en SonarCloud.	64
6.3. Cobertura del código en SonarCloud.	64
6.4. Gráfica de cobertura en SonarCloud.	65
6.5. Gráfica de code smell en SonarCloud.	66
6.6. Gráfica de líneas de código duplicadas en SonarCloud.	67
6.7. Gráfica de complejidad en SonarCloud.	68
6.8. Gráfica de densidad de comentarios en SonarCloud.	69
6.9. Vulnerabilidades detectadas en AppSweep.	70
6.10. Permisos del rol DatabaseAdministrator asignados a Oscar López.	72
6.11. Permisos del rol SystemAdministrator asignados a Pedro Pablo.	73
6.12. Permisos del rol ReadOnlyAccess asignados a Santiago Taracena y Yongbum Park.	74
7.1. Diagrama de Roles y Responsabilidades del Equipo	80
D.1. Certificado de firma de MobSF.	102
D.2. Permisos de aplicaciones de MobSF.	102
D.3. APIs de Android en MobSF.	103
D.4. Actividades navegables en MobSF.	103
D.5. Análisis de manifiesto en MobSF.	103
D.6. Análisis de código en MobSF.	103
D.7. Análisis de archivos en MobSF.	104
D.8. Análisis de APKid en MobSF.	104
D.9. Análisis de malware de dominio en MobSF.	104
D.10. Listado de URLs en MobSF.	105
D.11. Strings en MobSF.	105
E.1. Pregunta 1 del test de phishing de google.	107
E.2. Pregunta 2 del test de phishing de google.	108
E.3. Pregunta 3 del test de phishing de google.	109
E.4. Pregunta 4 del test de phishing de google.	110
E.5. Pregunta 5 del test de phishing de google.	110
E.6. Pregunta 6 del test de phishing de google.	111
E.7. Pregunta 7 del test de phishing de google.	111
E.8. Pregunta 8 del test de phishing de google.	112
E.9. Pregunta 9 del test de phishing de google.	113
E.10. Pregunta 10 del test de phishing de google.	114

Lista de Cuadros

4.1. Etapas del SDLC y medidas de seguridad.	23
6.1. Resultados de los tests de phishing.	62

Este proyecto se centra en la implementación de buenas prácticas de seguridad en el desarrollo de software, abordando temas críticos como la gestión de residuos y la reducción de la huella de carbono en la gestión de datos. Para alcanzar estos objetivos, se utilizó el marco *OWASP SAMM*, aplicando prácticas en Gobernanza, Diseño, Implementación y Verificación. Las herramientas *SonarQube*, *SonarCloud*, *AppSweep* y *MobSF* se seleccionaron tras un análisis exhaustivo por sus capacidades de análisis estático y dinámico, permitiendo identificar y mitigar vulnerabilidades en el código del *backend* y la aplicación móvil.

La metodología consistió en varias fases, comenzando con la selección e instalación de herramientas, seguida de un análisis detallado de vulnerabilidades. Las reuniones de seguimiento semanales permitieron ajustes continuos y capacitación en seguridad. Los análisis en *SonarQube* y *SonarCloud* revelaron una mejora significativa en la cobertura del código y en la reducción de vulnerabilidades, logrando un 82 % de cobertura total.

Los resultados demostraron una mejora continua en la seguridad y calidad del código, cumpliendo con los estándares de *OWASP* y reduciendo el impacto de posibles amenazas. Este enfoque no solo refuerza la seguridad del proyecto, sino que también establece una base robusta para la mejora continua y la respuesta a futuras amenazas.

CAPÍTULO 1

Introducción

En un entorno cada vez más interconectado y consciente de los desafíos ambientales, el presente trabajo aborda dos problemas clave: la gestión de residuos y la reducción de la huella de carbono, junto con la integración de la seguridad cibernética en el desarrollo de *software*. La relevancia de estos temas se ha intensificado en el contexto global actual, donde la sostenibilidad ambiental y la protección de datos personales son esenciales para el desarrollo sostenible de las organizaciones y la confianza de los usuarios.

El objetivo principal de este trabajo es demostrar cómo la implementación de prácticas adecuadas en ambas áreas puede no solo optimizar el manejo de recursos y la protección ambiental, sino también fortalecer la postura de seguridad de las aplicaciones de *software*. Se plantea la hipótesis de que una gestión eficiente de residuos y una seguridad cibernética proactiva mejoran la resiliencia de las organizaciones y reducen su impacto negativo en el entorno físico y digital.

El primer tema, la gestión de residuos, se centra en técnicas como el compostaje, el reciclaje y tecnologías avanzadas, como la pirólisis y la gasificación, destacando su aplicación en Guatemala y la importancia de las políticas y regulaciones locales. Se argumenta que la adopción de estas técnicas contribuye significativamente a la sostenibilidad ambiental, mejorando la calidad de vida de las comunidades al reducir la contaminación y promover el uso eficiente de los recursos. La revisión de casos exitosos y la aplicación de políticas internacionales sirven de base para evaluar su efectividad y adaptabilidad en el contexto guatemalteco.

En la segunda parte, el trabajo explora la importancia de la seguridad cibernética en el desarrollo de *software*, un aspecto crítico dado el aumento de amenazas y la sofisticación de los ciberataques. Se enfatiza que la incorporación de medidas de seguridad desde las primeras etapas del desarrollo, como se establece en el marco *OWASP SAMM*, puede prevenir vulnerabilidades críticas y proteger tanto a las organizaciones como a los usuarios finales. Herramientas como *SonarQube* y *AppSweep* se utilizaron para realizar análisis estáticos y dinámicos del código, evidenciando mejoras en la calidad y seguridad del *software*.

La metodología de este estudio incluyó la aplicación del marco de *OWASP SAMM*, enfocándose en prác-

ticas de Gobernanza, Diseño, Implementación y Verificación para integrar la seguridad de manera integral. Se llevaron a cabo análisis utilizando herramientas de evaluación de seguridad, se realizaron pruebas de *phishing* para evaluar la conciencia de seguridad del equipo y se implementaron reuniones de capacitación regulares. Además, se configuraron y ajustaron permisos en *AWS* para asegurar el cumplimiento del principio de mínimo privilegio, fortaleciendo así la seguridad de la infraestructura en la nube.

Entre las conclusiones más importantes, se destaca que las medidas implementadas lograron reducir vulnerabilidades en el código y mejorar la postura de seguridad del equipo. La cobertura de código en *Sonar-Cloud* superó el 80 %, validando la efectividad de las pruebas realizadas. Las sesiones de capacitación en seguridad cibernética no solo incrementaron el conocimiento del equipo, sino que también redujeron el riesgo de incidentes humanos, como los ataques de *phishing*. Por otro lado, las recomendaciones obtenidas de los análisis de la aplicación móvil se consideran un área de mejora continua, que el equipo seguirá desarrollando.

Este trabajo proporciona una base sólida para futuras aplicaciones seguras y sostenibles, destacando la importancia de un marco regulatorio robusto en Guatemala y la necesidad de fomentar una cultura de seguridad y sostenibilidad en las organizaciones.

2.1. Objetivo General

Asegurar que la aplicación cumpla con los más altos estándares de seguridad a lo largo de todo el ciclo de vida del software, desde el diseño hasta la implementación y el mantenimiento, protegiendo la privacidad y seguridad de los datos de los usuarios.

2.2. Objetivos Específicos

- Realizar una evaluación exhaustiva de posibles vulnerabilidades en el diseño y código de la aplicación utilizando herramientas como *SonarQube* y *AppSweep*, complementadas con pruebas manuales, siguiendo el *OWASP Top 10*, para generar un informe detallado con vulnerabilidades detectadas y recomendaciones de mitigación.
- Auditar y apoyar la implementación de prácticas de desarrollo seguro en todas las etapas del ciclo de vida del *software* integrando políticas de seguridad desde la planificación hasta el despliegue, siguiendo una metodología *SAMM*, asegurando que todas las versiones cumplan con los estándares de seguridad establecidos y reduzcan vulnerabilidades en producción.
- Auditar y apoyar la implementación de las mejores prácticas de autenticación, autorización y protección de datos mediante las implementaciones como el de autenticación multifactor (*MFA*), controles de acceso basados en roles (*RBAC*), y cifrado avanzado para datos en tránsito y en reposo, siguiendo las guías de *OWASP*, mejorando la protección de la integridad, confidencialidad y disponibilidad de los datos de los usuarios.
- Colaborar estrechamente con el equipo para mitigar riesgos y resolver problemas de seguridad durante el desarrollo mediante reuniones semanales de revisión de seguridad, fomentando una cultura de

seguridad proactiva para reducir riesgos y mejorar la capacidad de respuesta del equipo.

- Auditar y apoyar la realización de análisis de seguridad exhaustivos para identificar y mitigar vulnerabilidades en la aplicación móvil de la organización. Esto incluirá la realización de diversos tipos de evaluaciones de seguridad, como análisis de vulnerabilidades y revisiones de configuración, para garantizar la protección de los datos de los usuarios, asegurando así la integridad, confidencialidad y disponibilidad de la información.

El desarrollo del módulo de seguridad para una aplicación de gestión de residuos y reducción de la huella de carbono es crucial debido a la creciente amenaza de ciberataques y la importancia de proteger los datos sensibles de los usuarios. En este caso, la aplicación almacenará información personal de los usuarios de la población de Guatemala, como su nombre, correo electrónico, contraseña, ubicación y datos relacionados con sus hábitos de manejo de residuos, tales como la cantidad y tipo de basura generada. Esta información será utilizada para ayudar a los ciudadanos en el proceso de identificación y reciclaje de residuos, en colaboración con la Municipalidad de Guatemala. Además, la aplicación permitirá a los usuarios interactuar con funciones que incluyen el registro de compras, el análisis de huella de carbono, y la interacción con recomendaciones para el reciclaje.

En un contexto global donde los ciberdelincuentes buscan constantemente vulnerabilidades para explotar, asegurar la integridad, confidencialidad y disponibilidad de la información es vital. Según el informe de *IBM* sobre el costo de una brecha de datos, el costo promedio de una brecha de seguridad en 2023 fue de 4.45 millones de dólares, lo que resalta la importancia de la ciberseguridad para prevenir pérdidas económicas significativas y daños a la reputación de la organización.

Además, la aplicación maneja datos personales y ambientales que deben ser protegidos contra accesos no autorizados y posibles manipulaciones. En este caso, se utiliza una base de datos relacional, *PostgreSQL*, para almacenar la información de manera segura y garantizar su integridad. La protección de estos datos no solo es una cuestión de seguridad, sino también de cumplimiento legal. Por ejemplo, el Reglamento General de Protección de Datos (*GDPR*) de la Unión Europea establece estrictos requisitos sobre cómo deben protegerse los datos personales, con multas significativas para las organizaciones que no cumplan.

La seguridad cibernética no sólo protege la información de los usuarios, sino que también garantiza el correcto funcionamiento de la aplicación, evitando interrupciones que podrían afectar la experiencia del usuario y la efectividad de la gestión de residuos. La implementación de medidas de seguridad desde el inicio del desarrollo permite identificar y mitigar vulnerabilidades en etapas tempranas, lo que no solo aumenta la

seguridad, sino que también reduce costos y tiempo en correcciones posteriores. Un estudio de la Universidad de *Cambridge* indica que la identificación de vulnerabilidades en las primeras etapas del ciclo de desarrollo de *software* puede reducir el costo de corrección en un 70

En el contexto de Guatemala, un país en vías de digitalización, la seguridad cibernética adquiere una relevancia especial. La falta de educación y conciencia sobre prácticas seguras en el uso de aplicaciones digitales puede aumentar la vulnerabilidad de los usuarios y, por ende, de la aplicación misma. Según un informe del Banco Mundial, los países en desarrollo enfrentan desafíos significativos en la implementación de medidas de ciberseguridad debido a la falta de infraestructura y recursos técnicos. Por lo tanto, es esencial desarrollar un módulo de seguridad que no sólo proteja los datos, sino que también sea intuitivo y fomente buenas prácticas entre los usuarios.

La integración de la seguridad en cada etapa del ciclo de vida del *software*, desde el diseño hasta el mantenimiento, asegura que se cumplan las normativas locales e internacionales de protección de datos, reforzando la confianza en la aplicación y promoviendo un entorno digital seguro y fiable. La adopción de marcos de seguridad reconocidos como la *OWASP Software Assurance Maturity Model (SAMM)* puede proporcionar una guía estructurada para integrar prácticas de seguridad en el desarrollo de *software*.

4.1. gestión de residuos y Reducción de Huella de Carbono

4.1.1. Importancia de la gestión de residuos

Según GADISA(GADISA, 2024), los residuos pueden clasificarse en diversas categorías, siendo los más comunes los orgánicos, inorgánicos, peligrosos y esenciales. Las empresas deben implementar medidas de gestión de residuos para evitar un impacto negativo en el medio ambiente.

Los residuos orgánicos son aquellos que provienen de la actividad de la empresa, como restos de alimentos, cartón o papel; mientras que los residuos inorgánicos, como el vidrio, metal o plástico, no tienen un origen biológico. Estos dos tipos son los más comunes. Por otro lado, los residuos peligrosos son aquellos que pueden resultar dañinos para la salud humana, como productos químicos, pilas o baterías. Finalmente, los residuos esenciales son materiales clave que pueden ser reutilizados o reciclados de manera efectiva, como vidrio borosilicatado, restos de cobre, aluminio u oro(GADISA, 2024).

El término "gestión de residuos" se refiere al proceso de planificar, implementar, operar y controlar todas las actividades relacionadas con el manejo de residuos de manera segura y eficiente. Este proceso incluye desde la recolección y transporte de los residuos, hasta su tratamiento y disposición final, abarcando además los riesgos asociados a estas operaciones(GADISA, 2024).

Este proceso es fundamental porque contribuye a la protección del medio ambiente al reducir la cantidad de residuos que llegan a los vertederos, promoviendo el reciclaje. Asimismo, protege la salud humana al minimizar la exposición a los riesgos asociados a los residuos peligrosos(GADISA, 2024).

Actualmente, existen varios tipos de gestión de residuos, entre los más habituales destacan:

- **Gestión de residuos sólidos:** Consiste en el manejo de los residuos sólidos urbanos y comerciales, tales como papel, vidrio, plástico y metal. Este tipo de gestión abarca la recolección, transporte, tratamiento y disposición final de los residuos(GADISA, 2024).
- **Gestión de residuos peligrosos:** Hace referencia al manejo de los residuos considerados tóxicos o perjudiciales para el medio ambiente, como productos químicos, pilas o baterías. Estos residuos requieren un manejo especializado y una disposición adecuada para evitar daños(GADISA, 2024).
- **Gestión de residuos sanitarios:** Implica la manipulación de los residuos procedentes de centros hospitalarios, que presentan un alto riesgo biológico. Estos deben ser clasificados, ordenados y almacenados adecuadamente para su correcta eliminación(GADISA, 2024).
- **Gestión de residuos industriales:** Los procesos de fabricación y producción industrial generan grandes cantidades de residuos. Para eliminarlos, se emplean procedimientos físicos, químicos, biológicos o térmicos(GADISA, 2024).

Existen también técnicas de gestión de residuos, siendo la más común la **recogida selectiva**, que consiste en separar los residuos en diferentes categorías (orgánicos, inorgánicos, peligrosos, etc.) para facilitar su tratamiento. Además de esta técnica, destacan las siguientes:

- **Compostaje:** Proceso biológico controlado que transforma los residuos orgánicos en abono(GADISA, 2024).
- **Reciclaje:** Conversión de residuos en materiales nuevos mediante una serie de procesos industriales(GADISA, 2024).
- **Digestión anaerobia:** Tratamiento de residuos, especialmente de origen animal, en ausencia de oxígeno, generando biogás y fertilizantes como productos(GADISA, 2024).
- **Técnica de vertedero:** Consiste en depositar los residuos en un lugar específico y sellarlos para evitar la contaminación del suelo y del agua(GADISA, 2024).

4.1.2. Estrategias para la reducción de la huella de carbono

El término "huella de carbono" se refiere a la cantidad de gases de efecto invernadero (**GEI**) emitidos como consecuencia de las actividades humanas, los cuales se acumulan en la atmósfera y contribuyen al calentamiento global, acelerando el cambio climático (Iberdrola, 2024).

La huella de carbono no solo mide las emisiones directas realizadas por el ser humano, sino también las indirectas producidas por compuestos como el metano (CH_4), el óxido de nitrógeno (N_2O), los hidrofluorocarburos (**HFCs**), los perfluorocarburos (**PFCs**), el hexafluoruro de azufre (SF_6) y, sobre todo, el dióxido de carbono (CO_2), que es el compuesto más abundante y ha sido el mayor contribuyente al calentamiento global desde 1990 (Iberdrola, 2024).

Además de las emisiones humanas, las empresas también generan una huella de carbono significativa a través de actividades que producen **GEI**, tales como la fabricación, el transporte y el consumo energético.

Las empresas pueden tomar diversas acciones para reducir su huella de carbono, como mejorar su eficiencia energética, consumir energía proveniente de fuentes 100 % renovables, llevar a cabo campañas de sensibilización, invertir en proyectos medioambientales, pagar impuestos verdes o adquirir toneladas de CO₂ en el mercado internacional de emisiones (Iberdrola, 2024).

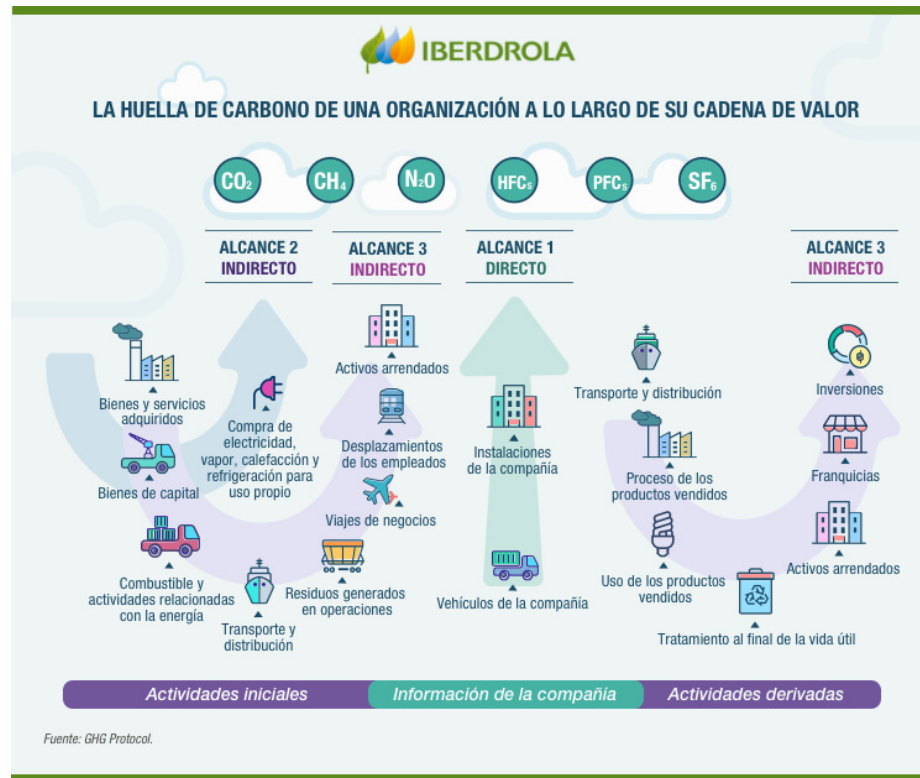


Figura 4.1: La huella de carbono de una organización a lo largo de su cadena de valor.

Según el portal de la Unión Europea, se pueden seguir los siguientes consejos para reducir la huella de carbono en distintas áreas de la vida cotidiana (Europa, 2024):

Alimentos

- Consumir productos locales y de temporada, evitando, por ejemplo, el consumo de frutas tropicales durante el invierno.
- Limitar el consumo de carne, especialmente la carne de res.
- Consumir pescado obtenido mediante la pesca natural.
- Comprar solo lo necesario para evitar el desperdicio de alimentos.

Ropa

- Cuidar adecuadamente la ropa para evitar desecharla en un corto período de tiempo.

- Comprar ropa fabricada por empresas que utilicen materiales reciclados o que cuenten con etiqueta ecológica.

Transporte

- Utilizar la bicicleta o el transporte público para reducir el uso de vehículos propios.

Energía y residuos

- Reducir la temperatura de la calefacción en 1°C, lo cual puede marcar una gran diferencia.
- Acortar el tiempo en la ducha.
- Desenchufar aparatos electrónicos cuando no se estén utilizando.

4.1.3. Herramientas tecnológicas en la gestión de residuos

El propósito del uso de la tecnología en la gestión de residuos es procesar los desechos de manera más eficiente, respetuosa con el medio ambiente y sostenible. El uso de tecnologías avanzadas permite optimizar el manejo de los residuos, reduciendo su impacto y promoviendo un enfoque más responsable hacia el medio ambiente.

Según Romuno, diversas empresas innovadoras están desarrollando tecnologías de gestión de residuos que abordan distintos problemas del sector, con un enfoque en la sostenibilidad. Estas tecnologías adoptan formas variadas y comparten el objetivo de reducir la contaminación y optimizar los sistemas de recogida de residuos, contribuyendo así a la protección del medio ambiente (Romuno, 2024).

De las tecnologías más destacadas para el tratamiento de residuos por empresas, se pueden mencionar las siguientes:

Incineración: La incineración es un proceso en el que los residuos son quemados a altas temperaturas para reducir su volumen y peso. Aunque es una tecnología ampliamente comercializada en el manejo de residuos sólidos urbanos (RSU), presenta ciertas desventajas desde el punto de vista medioambiental.

- Reduce hasta el 90 % del volumen de los residuos.
- Es la tecnología más utilizada a nivel comercial para el tratamiento de RSU.
- No es considerada una tecnología muy amigable con el medio ambiente debido a la emisión de contaminantes durante el proceso (Rodríguez, 2024).

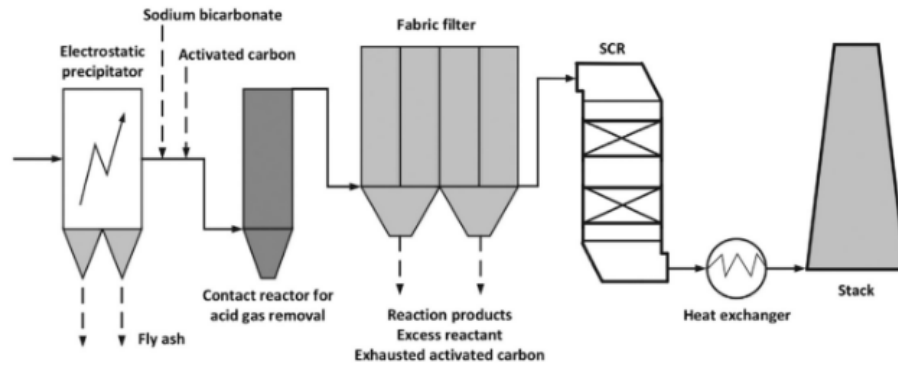


Figura 4.2: Ejemplo de línea de tratamiento de gases de combustión secos con captura de gases ácidos por bicarbonato de sodio y SCR

Pirólisis: La pirólisis es una tecnología que descompone los residuos en ausencia de oxígeno, generando productos como gas de síntesis y materiales que pueden ser reutilizados para generar energía o ser transformados en otros productos.

- Los materiales se someten a procesos en los que se libera gas de síntesis, el cual es tratado para eliminar sustancias nocivas.
- La pirólisis puede generar hasta 571 kWh por tonelada de RSU.
- Los residuos resultantes, como cenizas, carbón, metales y aceites, pueden ser utilizados para producir combustible sólido para plantas de energía, bioaceite o relleno de hormigón (Rodríguez, 2024).

Gasificación: La gasificación es un proceso termoquímico que convierte los residuos en un gas de alta energía, que puede ser utilizado para generar electricidad o como base para la producción de otros productos químicos.

- Puede generar hasta 685 kWh por tonelada de RSU.
- Los sólidos restantes pueden ser utilizados como agregados para hormigón y asfalto (Rodríguez, 2024).

Ahora, en lo que respecta a algunas herramientas que utilizan tecnologías innovadoras para la gestión de residuos, cabe destacar las siguientes que se consideran particularmente relevantes para 2024:

Pello: Pello es una tecnología desarrollada para ayudar a las empresas a reducir su impacto ambiental y gestionar la recogida de residuos de manera más eficiente. Esta herramienta permite a las empresas alcanzar estos objetivos de varias maneras, como controlar el nivel de llenado de los contenedores de basura y proporcionar información en tiempo real sobre su ubicación. Además, Pello informa si el contenedor ha sido contaminado y envía alertas cuando se aproxima la fecha de recogida.

Contar con esta información al alcance permite a las empresas optimizar sus prácticas de gestión de residuos y tener un mayor control sobre la eliminación de los mismos. Asimismo, al enviar los camiones de

basura únicamente cuando es necesario, se contribuye a reducir las emisiones de gases de efecto invernadero y el tráfico asociado a la recolección (Romuno, 2024).

Tubos de desagüe neumáticos: A medida que las áreas urbanas crecen, los sistemas tradicionales de recogida y eliminación de residuos se vuelven más complejos. Una solución innovadora a este problema es el uso de tuberías neumáticas para la gestión de residuos, en las que los contenedores de basura se conectan a un sistema que transporta los residuos directamente a los centros de tratamiento, sin necesidad de recogida mediante camiones.

Este sistema presenta dos ventajas clave: en primer lugar, reduce significativamente la necesidad de camiones de basura y, por ende, las emisiones de gases contaminantes; en segundo lugar, es una solución más estética, ya que se evita el desbordamiento visible de los contenedores de basura el cual puede llegar a ser un riesgo para la salud ambiental (Romuno, 2024).

4.2. Seguridad Cibernética en el Desarrollo de Aplicaciones

4.2.1. Importancia de la seguridad cibernética

La seguridad cibernética es el conjunto de procesos y herramientas utilizados para proteger de forma anticipada y defender dispositivos, sistemas electrónicos, servidores, redes y programas frente a posibles ciberataques. Con el mundo cada vez más digitalizado, la ciberseguridad se ha vuelto indispensable para la protección de la información personal y empresarial ante amenazas cibernéticas. La creciente interconexión de dispositivos y sistemas incrementa la vulnerabilidad frente a ataques como el robo de datos, *hacking* y *ransomware* (Colombia, 2024).

Existen principalmente tres tipos de seguridad informática:

1. **Seguridad de *hardware*:** Se refiere a la protección de los dispositivos que se utilizan para proteger sistemas, redes, aplicaciones y programas. Entre los métodos más empleados se encuentran los sistemas de alimentación ininterrumpida (SAI), servidores *proxy*, cortafuegos (*firewalls*), módulos de seguridad de *hardware* (HSM) y los sistemas de prevención de pérdida de datos (DLP). Esta protección no solo incluye el entorno virtual, sino también la seguridad física de los equipos frente a posibles daños.
2. **Seguridad de *software*:** Está enfocada en defender los sistemas contra ataques cibernéticos que puedan aprovechar vulnerabilidades en los programas o configuraciones incorrectas de los mismos.
3. **Seguridad de red:** Su objetivo es proteger los datos accesibles a través de la red, evitando que sean modificados o robados. Entre las principales amenazas se encuentran los virus, troyanos y programas espía (de Cataluña, 2024).

En la seguridad cibernética, existen tres pilares fundamentales:

- **Confidencialidad:** Garantiza que únicamente los usuarios autorizados puedan acceder a los recursos, datos e información.

- **Integridad:** Se enfoca en asegurar que los datos no sean alterados sin autorización y que solo los usuarios con permisos puedan realizar dichas modificaciones.
- **Disponibilidad:** Asegura que los datos siempre estén disponibles para su uso cuando se necesiten (de Cataluña, 2024).

Los ciberdelincuentes emplean diversos métodos de ataque, entre los cuales destacan los siguientes:

Malware: Este es el tipo de ataque más común y hace referencia a un software malicioso (*malicious software*), que suele provenir de archivos descargados. El *malware* permite a los delincuentes acceder a redes y dañar, bloquear o robar información de los equipos de los usuarios.

Phishing: Es un ataque donde se suplanta la identidad de una persona o entidad. Consiste en crear copias de sitios web o enviar correos electrónicos que solicitan información confidencial, como contraseñas o datos de tarjetas de crédito, haciéndose pasar por instituciones legítimas.

Man in the middle: En este ataque, el delincuente intercepta la comunicación entre dos personas y roba la información que están intercambiando.

Ataques de denegación de servicio distribuido (DDoS): Este tipo de ataque busca saturar o colapsar un servidor o página *web* mediante el envío masivo de falsas solicitudes, lo que impide su funcionamiento normal.

Amenaza interna: Este tipo de ataque suele ser llevado a cabo por ex empleados que aún tienen acceso a los sistemas de la empresa y utilizan esa ventaja para obtener datos confidenciales o infectarlos con malware (Colombia, 2024).

Para mitigar estos riesgos, las empresas pueden seguir las siguientes recomendaciones:

- **Educar a los empleados sobre los riesgos cibernéticos:** La formación del personal es fundamental para que los empleados conozcan los riesgos a los que están expuestos y puedan identificar posibles amenazas, como correos electrónicos de *phishing* o enlaces sospechosos.
- **Implementar políticas de seguridad adecuadas:** Es necesario establecer políticas claras que definan cómo debe gestionarse la seguridad en la empresa, desde el uso de contraseñas seguras hasta la correcta manipulación de la información confidencial.
- **Utilizar software de seguridad actualizado:** Mantener el *software* de seguridad actualizado es crucial para protegerse contra las amenazas más recientes. Los sistemas obsoletos pueden tener vulnerabilidades que los ciberdelincuentes pueden aprovechar.
- **Cifrar los datos sensibles:** El cifrado de datos asegura que, aunque la información sea interceptada por terceros, no pueda ser leída sin la clave adecuada. Esto es especialmente importante para proteger datos sensibles como información financiera o personal.
- **Implementar un control de acceso y autenticación multifactor:** Limitar el acceso a los sistemas y datos solo a los usuarios autorizados es esencial. Además, el uso de autenticación multifactor agrega una capa adicional de seguridad al requerir más de un método de verificación.

- **Realizar auditorías y evaluaciones de riesgos periódicas:** Es importante llevar a cabo revisiones regulares para identificar posibles vulnerabilidades en los sistemas y evaluar si las medidas de seguridad actuales son adecuadas o necesitan ser mejoradas.
- **Tener un plan de respuesta ante incidentes:** Contar con un plan de acción en caso de un ataque cibernético es fundamental para minimizar el impacto. Esto incluye procedimientos para la recuperación de datos y la contención del incidente.
- **Mantener copias de seguridad de los datos:** Las copias de seguridad periódicas garantizan que, en caso de pérdida o robo de información, los datos puedan ser recuperados sin mayores problemas.
- **Cumplir con las normativas de seguridad:** Las empresas deben seguir las regulaciones locales e internacionales en materia de protección de datos y ciberseguridad para evitar sanciones legales y garantizar la seguridad de la información.
- **Colaborar con expertos en ciberseguridad:** Trabajar con profesionales del sector permite a las empresas tener acceso a las mejores prácticas y soluciones especializadas para enfrentar las amenazas más sofisticadas.
- **Monitorear continuamente los sistemas:** La supervisión constante de los sistemas permite detectar actividades sospechosas en tiempo real y tomar medidas inmediatas para evitar daños (Colombia, 2024).

4.2.2. OWASP Top 10

El desarrollo de aplicaciones, en un entorno tecnológico cada vez más complejo y dinámico, enfrenta una serie de amenazas y vulnerabilidades que pueden comprometer la seguridad, integridad y disponibilidad de las aplicaciones y los datos que gestionan. Estas amenazas afectan tanto a las aplicaciones web como a las aplicaciones móviles, y pueden ser identificadas mediante el *OWASP Top 10*, que proporciona un listado de las vulnerabilidades más críticas en cada caso (Foundation, 2024b).

Según **OWASP**, las 10 amenazas más comunes en aplicaciones web son las siguientes:

1. **Control de acceso interrumpido:** La falta de control adecuado puede provocar la divulgación no autorizada de información, la modificación o destrucción de datos, o la ejecución de funciones comerciales sin los permisos correspondientes.
2. **Fallos criptográficos:** La falta de cifrado adecuado en los datos sensibles, como contraseñas, números de tarjetas de crédito o registros médicos, puede exponer información confidencial y violar normativas como el Reglamento General de Protección de Datos (**GDPR**) o el Estándar de Seguridad de Datos PCI (**PCI DSS**).
3. **Inyección:** Ocurre cuando no se validan adecuadamente los datos de entrada, permitiendo a los atacantes ejecutar comandos no autorizados y obtener acceso a información confidencial.
4. **Diseño inseguro:** Se refiere a la falta de un diseño seguro desde el inicio del desarrollo de la aplicación, lo que provoca la ausencia de controles de seguridad necesarios para proteger frente a ataques específicos.

5. **Configuración incorrecta de seguridad:** Errores en la configuración de la aplicación, el servidor o la infraestructura pueden dejar vulnerabilidades abiertas que los atacantes pueden aprovechar para comprometer la seguridad.
6. **Componentes vulnerables y obsoletos:** Utilizar componentes desactualizados o con vulnerabilidades conocidas permite a los atacantes explotar esas debilidades.
7. **Errores de identificación y autenticación:** Fallos en la identificación y autenticación de usuarios permiten a los atacantes suplantar la identidad de otros usuarios y acceder a funciones restringidas.
8. **Fallas en la integridad de datos y software:** La falta de validación en las actualizaciones de *software* puede permitir a los atacantes inyectar código malicioso o alterar datos críticos.
9. **Errores de registro y monitoreo de seguridad:** La falta de implementación adecuada de mecanismos de registro y monitoreo impide detectar y responder a amenazas de manera efectiva.
10. **Falsificación de solicitudes del lado del servidor (SSRF):** Los atacantes pueden manipular el servidor para realizar solicitudes a otros servicios sin autorización, accediendo a datos sensibles sin necesidad de autenticación (Foundation, 2024b).

Además, *OWASP* también ha publicado un Top 10 de riesgos específicos para aplicaciones móviles:

1. **Uso inadecuado de credenciales:** Esta vulnerabilidad se refiere a la explotación de credenciales codificadas de manera inadecuada, lo que facilita el robo de datos.
2. **Seguridad inadecuada de la cadena de suministro:** Los atacantes pueden aprovechar las vulnerabilidades en la cadena de suministro para manipular funcionalidades de la aplicación, acceder a aplicaciones móviles de terceros o al *backend* de los servidores, e incluso ejecutar ataques de denegación de servicio (**DoS**).
3. **Autenticación y autorización inseguras:** Los atacantes pueden enviar solicitudes directamente al servidor *backend*, sin interactuar con la aplicación, obteniendo información o actuando con privilegios excesivos.
4. **Validación insuficiente de la entrada y salida de datos:** La falta de validación adecuada puede permitir inyecciones *SQL*, lo que resulta en el robo o manipulación de datos, o en la ejecución de código que interrumpa el funcionamiento de la aplicación.
5. **Comunicación insegura:** Las fallas en los protocolos criptográficos permiten a los atacantes interceptar datos, suplantar identidades o robar información confidencial.
6. **Controles de privacidad inadecuados:** La gestión incorrecta de la información de los usuarios puede permitir a los atacantes realizar extorsiones, fraudes financieros o dañar la reputación de las víctimas.
7. **Protecciones insuficientes de los binarios:** Los binarios pueden contener claves de acceso a *APIs* comerciales o ser utilizados para eludir controles de seguridad de la aplicación.
8. **Mala configuración de seguridad:** Las configuraciones incorrectas de seguridad y permisos pueden ser explotadas por los atacantes para acceder a información confidencial o realizar acciones maliciosas.

9. **Almacenamiento de datos inseguro:** Cuando los desarrolladores no almacenan correctamente los datos, los atacantes pueden comprometer cuentas, manipular información o acceder a archivos de configuración y claves criptográficas.
10. **Criptografía insuficiente:** El uso de cifrado débil o la falta total de cifrado puede resultar en la exfiltración de información confidencial, pérdidas financieras o problemas legales por incumplimiento de normativas (Security, 2024b).

4.2.3. Buenas prácticas de seguridad en el desarrollo de software

El término "buenas prácticas de seguridad" se refiere al conjunto de técnicas y metodologías que deben implementarse para que el desarrollo de software sea más eficiente, rápido y seguro. Estas prácticas ayudan a minimizar las vulnerabilidades desde el inicio del ciclo de vida del software, garantizando su calidad y protección frente a posibles ataques.

Existen varias formas de implementar buenas prácticas de seguridad, entre las que destacan:

COBIT (Control Objectives for Information and Related Technology): *COBIT* es un marco que permite evaluar y controlar todos los aspectos de la tecnología de la información (*TI*) dentro de una organización, abarcando desde los sistemas de información hasta el personal que los gestiona. Su objetivo principal es proporcionar a las empresas una herramienta que facilite el cumplimiento de los objetivos del negocio a través de la automatización y optimización de los recursos *TI*.

ITIL (Information Technology Infrastructure Library): *ITIL* es un conjunto de buenas prácticas para la gestión de servicios y recursos *TI*, cuyo objetivo es garantizar la calidad y eficiencia de los procesos productivos de una organización. *ITIL* divide el ciclo de vida de los servicios en cinco etapas:

- **Estrategia del servicio:** Identifica los activos estratégicos de la empresa que generan valor o la diferencian en el mercado.
- **Diseño del servicio:** Explora el portafolio de servicios para identificar debilidades y fortalezas, y establecer políticas y estándares que mejoren la calidad.
- **Transición del servicio:** Cubre el proceso de cambio entre la mejora de servicios existentes o la creación de nuevos servicios, identificados durante la fase de diseño.
- **Operación del servicio:** Gestiona y controla los procesos necesarios para la puesta en marcha del servicio en producción.
- **Mejora continua del servicio:** Optimiza la calidad del servicio mediante la retroalimentación constante de las fases anteriores, facilitando la adaptación al entorno cambiante (Bernal, 2024).

Clean Code: "*Clean Code*" se refiere a código que es fácil de leer, entender y mantener. Este concepto promueve principios como el uso de nombres claros, evitar la duplicación de código, escribir funciones cortas y utilizar comentarios solo cuando sean realmente útiles. El objetivo es que el código no solo funcione bien, sino que sea comprensible para otros desarrolladores, lo que facilita su mantenimiento y mejora la calidad del software a largo plazo (Blog, 2024).

4.3. Contexto Local: Guatemala

4.3.1. Situación actual de la gestión de residuos en Guatemala

En los últimos años, la población de Guatemala ha experimentado un crecimiento sostenido. Según datos del Banco Mundial, entre 1960 y 2021, el 20% de la población rural migró hacia las zonas urbanas. En Guatemala, actualmente solo existen dos sitios principales para la gestión de residuos sólidos: el relleno sanitario de la zona 3 y el vertedero de AMSA en Villa Nueva. Estos son los encargados de procesar los residuos generados por toda la población urbana (Pineda, 2023).

Este escenario es preocupante, ya que, de acuerdo con el Banco Mundial, cada guatemalteco genera en promedio 0.47 kg de desechos sólidos diarios. Con una población aproximada de 17 millones de habitantes, esto equivale a la generación de 7,990 toneladas de desechos sólidos al día, acumulando anualmente un total de 2,916,350 toneladas de basura que deben ser gestionadas en vertederos municipales autorizados (de Problemas Nacionales de la USAC (IPN), 2023).

En 2021, el Ministerio de Ambiente y Recursos Naturales (MARN) emitió el Acuerdo Gubernativo 164-2021: **Reglamento para la gestión integral de los residuos y desechos sólidos comunes**. Este reglamento establece normas sanitarias y ambientales que buscan prevenir el deterioro ambiental, reducir la contaminación y mejorar la salud de los guatemaltecos. Durante los primeros dos años desde la implementación del reglamento, es obligatoria la clasificación primaria de los desechos en orgánicos e inorgánicos. Además, se contempla una clasificación secundaria, que incluye categorías como reciclables (papel, cartón, vidrio, plástico, metal, multicapa) y no reciclables, cuyo cumplimiento será obligatorio a partir del tercer año de vigencia del acuerdo. Sin embargo, un cambio reciente, realizado el 9 de agosto de 2024, postergó la obligatoriedad de esta clasificación secundaria hasta el 11 de febrero de 2025, permitiendo por ahora el cumplimiento únicamente de la clasificación primaria (de Ambiente y Recursos Naturales (MARN), 2024b).

Otra modificación significativa del reglamento se refiere al Artículo 17, que regula la recolección y transporte de residuos. Anteriormente, se exigía que los vehículos utilizados en estas tareas fueran construidos con materiales sólidos y resistentes a la corrosión, pero ahora el requisito se ha simplificado, solicitando únicamente protección contra la corrosión. Además, se exige que los vehículos sean herméticos para evitar el derrame de líquidos contaminantes durante el transporte (Libre, 2024).

El Artículo 28 del reglamento establece que las tecnologías empleadas para el tratamiento de los desechos sólidos deben seleccionarse en función de la naturaleza de los residuos y los objetivos de gestión. No es obligatorio aplicar un tratamiento previo antes de la disposición final si no es necesario. Por otro lado, el Artículo 29 especifica que el tratamiento de residuos puede incluir procesos como la incineración o la reducción mecánica de su tamaño mediante técnicas de fragmentación, molienda o trituración, lo que facilita su disposición final (de Ambiente y Recursos Naturales (MARN), 2024b).

Por último, el Artículo 54 estipula que se impondrán multas a las entidades que, tras una verificación, no cumplan con los compromisos ambientales establecidos. Las sanciones pueden variar entre uno (1) y cuarenta (40) salarios mínimos mensuales por cada incumplimiento, dependiendo de la gravedad de la infracción, la cual será determinada por el MARN (de Ambiente y Recursos Naturales (MARN), 2024b).

Es alentador que Guatemala esté tomando medidas concretas para abordar el tema de la gestión de residuos. A lo largo de los últimos años, se han implementado y mejorado reglamentos clave que buscan reducir la contaminación, proteger la salud pública y garantizar la sostenibilidad ambiental. Con la aplicación gradual de estos reglamentos, se espera que el país logre un manejo más eficiente y responsable de sus desechos, promoviendo una cultura de reciclaje y reducción de residuos. La postergación de ciertas medidas muestra flexibilidad en la adaptación de las normativas, pero también resalta la necesidad de compromiso por parte de la ciudadanía y las instituciones para lograr una gestión integral y efectiva en el largo plazo.

4.3.2. Políticas y regulaciones ambientales

En Guatemala, diversas entidades administrativas a nivel nacional son responsables de la gestión ambiental en diferentes áreas. La Comisión Nacional del Medio Ambiente (CONAMA) se encarga de asesorar, coordinar y aplicar la política nacional ambiental. Su estructura organizacional y funciones están reguladas por la Ley de Protección y Mejoramiento del Medio Ambiente.

El Artículo 97 de la Constitución de Guatemala establece que ^{.El} Estado, las municipalidades y los habitantes del territorio nacional están obligados a propiciar el desarrollo social, económico y tecnológico que prevenga la contaminación del ambiente y mantenga el equilibrio ecológico. Se dictarán todas las normas necesarias para garantizar que la utilización y el aprovechamiento de la fauna, de la flora, de la tierra y del agua se realicen racionalmente, evitando su depredación"(de Derechos Humanos (Corte IDH), 2024).

Por su parte, el Artículo 64 establece que "Se declara de interés nacional la conservación, protección y mejoramiento del patrimonio natural de la Nación. El Estado fomentará la creación de parques nacionales, reservas y refugios naturales, los cuales son inalienables. Una ley garantizará su protección y la de la fauna y la flora que en ellos exista"(de Derechos Humanos (Corte IDH), 2024).

En cuanto a las políticas nacionales de protección del medio ambiente y desarrollo sostenible, a pesar de los esfuerzos realizados en sectores como el agropecuario y forestal, Guatemala aún carece de una política nacional del medio ambiente sancionada por las más altas instancias del Ejecutivo, tal como lo exige el Decreto 68-86. En el sector agropecuario, se han definido políticas orientadas al desarrollo sostenible con la participación de los actores involucrados. En el ámbito forestal, se busca maximizar los beneficios socioeconómicos de los recursos mediante la ampliación de la cobertura forestal y el manejo sostenible de los bosques. En cuanto a la protección de la diversidad biológica, aunque no existe una política explícita, se han desarrollado estrategias institucionales para la conservación y el uso sostenible de los recursos genéticos (de Derechos Humanos (Corte IDH), 2024).

En cuanto a las regulaciones más recientes, el Acuerdo Gubernativo 189-2017 establece que las políticas públicas de Guatemala están orientadas a la protección, conservación y mejoramiento de los recursos naturales. La Política Nacional de Desarrollo Rural Integral (PNDRI) impulsa políticas sectoriales dirigidas al desarrollo rural, buscando mejorar la calidad de vida de las poblaciones en situación de pobreza y exclusión. Por su parte, la Política Marco de Gestión Ambiental se enfoca en mejorar la calidad ambiental y conservar el patrimonio natural, garantizando el equilibrio ecológico necesario para la vida y promoviendo el bienestar social, económico y cultural de las generaciones actuales y futuras (de Ambiente y Recursos Naturales (MARN), 2024a).

Otra política importante es la Política Nacional de Cambio Climático, que tiene como objetivo reducir las vulnerabilidades, promover la adaptación al cambio climático y mitigar las emisiones de gases de efecto invernadero. Esta política también integra la educación sobre cambio climático en los sistemas educativos formales e informales (de Ambiente y Recursos Naturales (MARN), 2024a).

Asimismo, la Política Nacional para el Manejo Integral de los Residuos y Desechos Sólidos establece lineamientos claros para garantizar la salud pública a través del manejo adecuado de los residuos, fomentando prácticas de producción más limpia. De igual manera, la Política Nacional para la Gestión Ambientalmente Racional de Productos Químicos y Desechos Peligrosos busca reducir los riesgos ambientales y de salud derivados de la gestión inadecuada de productos químicos (de Ambiente y Recursos Naturales (MARN), 2024a).

Por otro lado, la Política para el Manejo Integral de las Zonas Marino Costeras se centra en la mejora de la calidad de vida de las comunidades costeras, mientras que la Política Ambiental de Género y su Plan de Acción promueven la equidad de género y la inclusión de hombres y mujeres en la protección de los bienes y servicios ambientales (de Ambiente y Recursos Naturales (MARN), 2024a).

4.3.3. Desafíos y oportunidades en la seguridad cibernética en Guatemala

El aumento significativo de los ciberataques en Guatemala ha puesto de relieve la necesidad de implementar soluciones integrales de seguridad cibernética, especialmente en un contexto de rápida transformación digital. Con un promedio de 1,727 ataques por semana, las organizaciones guatemaltecas, particularmente en el sector financiero, se han convertido en objetivos frecuentes de estos ataques. Según Eli Faskha, CEO de Soluciones Seguras, la digitalización no gestionada adecuadamente puede aumentar significativamente los riesgos de ciberseguridad. En este contexto, es crucial que las empresas identifiquen sus vulnerabilidades y aprendan a detectar, protegerse y responder a las amenazas cibernéticas para garantizar la continuidad de sus operaciones (Seguras, 2022).

En el marco legal, Guatemala cuenta con disposiciones específicas para sancionar delitos informáticos a través de su Código Penal. Entre los artículos más relevantes se encuentran:

- **Artículo 274 A:** Sanciona con prisión de seis meses a cuatro años y una multa de doscientos a dos mil quetzales a quien destruya, borre o inutilice registros informáticos.
- **Artículo 274 B:** Aplica la misma pena a quien altere, borre o inutilice instrucciones o programas utilizados por computadoras.
- **Artículo 274 C:** Impone prisión de seis meses a cuatro años y una multa de quinientos a dos mil quinientos quetzales a quien, sin autorización, copie o reproduzca instrucciones o programas de computación.
- **Artículo 274 D:** Establece una pena de seis meses a cuatro años de prisión y una multa de doscientos a mil quetzales para quien cree un banco de datos o registro informático que afecte la intimidad de las personas.

- **Artículo 274 E:** Sanciona con prisión de uno a cinco años y una multa de quinientos a tres mil quetzales a quien utilice registros informáticos para ocultar, alterar o distorsionar información en actividades comerciales.
- **Artículo 274 F:** Impone prisión de seis meses a dos años y una multa de doscientos a mil quetzales a quien acceda sin permiso a bancos de datos o archivos electrónicos ajenos (Guatemala, 2021).

A pesar de las sanciones contempladas en el Código Penal, estas leyes resultan insuficientes frente al creciente número de ataques cibernéticos. La digitalización en el país ha evidenciado la necesidad urgente de actualizar y fortalecer las normativas legales en esta área. Aunque existen penalizaciones, no disuaden de manera efectiva a los ciberdelincuentes ni protegen adecuadamente a las empresas y usuarios guatemaltecos. El desarrollo de leyes más robustas y modernas es fundamental para hacer frente a las amenazas cibernéticas actuales y emergentes.

En 2022, se presentó el Decreto 39-2022, que proponía la Ley de prevención y protección contra la ciberdelincuencia. Sin embargo, esta normativa fue finalmente archivada debido a la controversia que generaron algunos de sus artículos. En particular, el Artículo 9, que penalizaba el acceso no autorizado a información confidencial, fue criticado por contravenir el Artículo 35 de la Constitución, que garantiza el acceso a fuentes de información. Asimismo, el Artículo 19, que sancionaba el ciberacoso y la divulgación de información, fue considerado ambiguo, lo que planteaba el riesgo de que pudiera utilizarse para criminalizar críticas a funcionarios públicos o restringir la libertad de expresión (López, 2022).

Estas disposiciones planteaban el peligro de que la ley fuera utilizada para silenciar a los medios de comunicación, periodistas y ciudadanos que ejercen su derecho a la libre expresión. La posibilidad de que investigaciones periodísticas o críticas en redes sociales, como memes o sátira política, fueran tratadas como delitos, despertó preocupación sobre su impacto en el ejercicio periodístico y la participación cívica. A pesar de que la ley buscaba mejorar la ciberseguridad en Guatemala, fue percibida como una amenaza a los derechos fundamentales, lo que finalmente llevó a su archivo el 24 de agosto de 2022 (López, 2022).

Es evidente que el fortalecimiento de la ciberseguridad en Guatemala presenta desafíos significativos, no solo en términos tecnológicos, sino también en el marco legal y regulatorio. Sin embargo, esta situación también representa una oportunidad para que el país desarrolle normativas más sólidas y protectoras, que equilibren la seguridad con la preservación de los derechos fundamentales de sus ciudadanos.

4.4. Metodologías de Desarrollo Seguro

4.4.1. Introducción al desarrollo seguro de software

El desarrollo seguro de software coloca la seguridad como un requisito en todas las fases del Ciclo de Vida del Desarrollo de Software (*SDLC*, por sus siglas en inglés). Desde la fase de diseño hasta la implementación y el mantenimiento, el objetivo principal es identificar, mitigar y prevenir vulnerabilidades que puedan afectar la integridad del software y la seguridad de los usuarios.

Uno de los aspectos más críticos de un modelo de desarrollo seguro es que, según *IBM Security*, el 60 %

de los ataques cibernéticos provienen de vulnerabilidades presentes en el código fuente (Caro, 2023). La falta de un enfoque de seguridad desde el inicio del proyecto puede resultar en una integración deficiente de las medidas de protección durante el proceso de desarrollo. Esto no solo hace que sea más complejo y costoso aplicar correcciones de seguridad al final de las fases, sino que también puede generar pérdidas económicas significativas y la pérdida de confianza por parte de los clientes debido a incidentes de seguridad.

Existen varias metodologías y enfoques que complementan el Desarrollo Seguro de *Software* (SSD), cada una de ellas con un enfoque específico para abordar la seguridad en diferentes momentos del ciclo de desarrollo:

- **Ciclo de Vida de Desarrollo de Seguridad (SecDevOps):** Esta metodología incorpora herramientas y prácticas de seguridad en el ciclo de desarrollo continuo. *SecDevOps* incluye la automatización de pruebas de seguridad y proporciona retroalimentación rápida para identificar y corregir vulnerabilidades a lo largo del proceso de desarrollo.
- **Desarrollo Seguro por Diseño (SSDLC):** El *SSDLC* se enfoca en integrar la seguridad desde la etapa de diseño, incorporando controles de seguridad en los requisitos iniciales y en la arquitectura del *software*, lo que ayuda a prevenir vulnerabilidades desde las primeras fases del proyecto.
- **Modelo de Madurez de Seguridad (SMM):** Esta metodología evalúa y mejora la madurez de la seguridad en el entorno de desarrollo. Proporciona un marco para identificar el estado actual de seguridad, analizar los puntos débiles y establecer metas claras de mejora en la seguridad del *software*.
- **Metodologías Ágiles y Seguridad (Agile Security):** A través de revisiones continuas y en cada iteración del ciclo de desarrollo ágil, se incorporan análisis de seguridad de código, pruebas de penetración y ajustes necesarios para abordar nuevas amenazas de manera proactiva.
- **Análisis de Amenazas y Modelado (TAM):** El *TAM* identifica amenazas y riesgos desde las etapas iniciales del desarrollo mediante el análisis de riesgos y el modelado de amenazas, lo que permite a los equipos de desarrollo anticipar posibles vulnerabilidades y diseñar estrategias de mitigación tempranas (Micucci, 2023).

Cada una de estas metodologías tiene como objetivo garantizar que el *software* no solo sea funcional, sino que también esté protegido frente a amenazas y vulnerabilidades. Al implementar estas prácticas desde las primeras fases del *SDLC*, se logra un entorno de desarrollo más seguro, con menos errores críticos que podrían afectar la continuidad y seguridad de las operaciones.

4.4.2. Ciclo de Vida del Desarrollo de Software Seguro (SDLC)

El Ciclo de Vida del Desarrollo de Software (*SDLC*, por sus siglas en inglés) es un proceso estructurado que permite a los equipos de desarrollo diseñar y crear *software* de alta calidad de manera rentable y eficiente. El objetivo principal del *SDLC* es reducir los riesgos del proyecto a través de una planificación anticipada, garantizando que el *software* cumpla con las expectativas del cliente durante su fase de producción y a lo largo de su vida útil. Este enfoque divide el desarrollo de *software* en una serie de fases bien definidas, que pueden ser asignadas, completadas y medidas de manera efectiva ((AWS), 2024).

Generalmente, el *SDLC* se compone de seis a ocho fases:

- **Planificación:** En esta fase se recopilan los requisitos de expertos, directivos y clientes para crear un documento que especifique los objetivos del *software*. También se realizan análisis de costos, estimación de recursos y beneficios, y se asignan roles dentro del equipo de desarrollo.
- **Análisis de los requisitos:** Se definen las funciones que debe ejecutar el *software*, asegurando que estas se alineen con las necesidades del cliente.
- **Diseño:** Se analizan las especificaciones técnicas del *software* y se define la mejor forma de integrarlo en la infraestructura de tecnología de la información (**TI**) existente en la empresa.
- **Desarrollo:** Esta fase incluye la creación y codificación del *software* conforme a los lineamientos establecidos en las fases previas.
- **Documentación:** Se produce toda la información necesaria para que los usuarios y las partes interesadas puedan utilizar el sistema de manera efectiva.
- **Pruebas:** Se llevan a cabo pruebas automáticas y manuales para verificar que el *software* cumple con los requisitos establecidos y no presenta fallos de seguridad. A menudo, las pruebas se realizan simultáneamente con el desarrollo.
- **Implementación:** El *software* es desplegado en el entorno de producción, haciéndolo accesible a los usuarios finales.
- **Mantenimiento:** Después de la implementación, se ofrece soporte continuo para resolver problemas, supervisar el rendimiento, mejorar la experiencia del usuario y reforzar la ciberseguridad (Chávez, 2024).

Para garantizar la seguridad del software a lo largo de todas las fases del *SDLC*, se debe seguir el enfoque del Ciclo de Vida de Desarrollo de Software Seguro (*SSDLC*), que incorpora medidas de seguridad específicas en cada etapa del proceso. Estas medidas ayudan a identificar y mitigar vulnerabilidades antes de que afecten el rendimiento o la seguridad del software.

La integración de estas medidas de seguridad a lo largo del *SDLC* asegura que el *software* no solo cumpla con sus requisitos funcionales, sino que también esté protegido contra amenazas emergentes y riesgos cibernéticos (Hat, 2024).

4.4.3. Principios de diseño seguro

Los principios de diseño seguro son esenciales para asegurar que las aplicaciones y los sistemas sean resistentes frente a vulnerabilidades y amenazas, protegiendo la confidencialidad, integridad y disponibilidad de los datos. Estos principios ofrecen un marco claro para que los desarrolladores construyan aplicaciones más seguras y confiables, alineando las mejores prácticas de seguridad desde el diseño inicial hasta la implementación final.

Etapas del SDLC	Medidas de seguridad
Planificación	Evaluar los riesgos y el panorama de amenazas a la seguridad. Evaluar el impacto potencial de los incidentes de seguridad, como los riesgos para la reputación de la empresa.
Análisis de los requisitos	Incluir requisitos de seguridad. Asegurarse de cumplir con las normativas aplicables.
Diseño	Elaborar modelos de amenazas. Incorporar consideraciones de seguridad en el plan de arquitectura. Evaluar cómo las decisiones de diseño afectan la seguridad, como la elección de la plataforma y la interfaz de usuario.
Desarrollo	Capacitar a los desarrolladores en prácticas de codificación seguras. Utilizar herramientas de pruebas de seguridad a lo largo del proceso de desarrollo. Evaluar las dependencias del <i>software</i> y mitigar los riesgos de seguridad asociados.
Documentación	Documentar los procesos y controles de seguridad. Prepararse para auditorías, controles de cumplimiento normativo y revisiones de seguridad.
Pruebas	Implementar revisiones de código. Realizar pruebas estáticas y dinámicas de seguridad de las aplicaciones.
Implementación	Evaluar la seguridad del entorno de producción. Revisar las configuraciones de seguridad.
Mantenimiento	Supervisar el sistema en busca de amenazas. Estar preparado para corregir vulnerabilidades y gestionar intrusiones.

Tabla 4.1: Etapas del SDLC y medidas de seguridad.

Uno de los principios más relevantes es el de menor privilegio, que establece que un usuario o proceso solo debe tener los permisos mínimos necesarios para realizar sus tareas, lo que minimiza el riesgo de uso indebido de esos privilegios. La reducción de permisos disminuye la exposición a posibles ataques y garantiza que, en caso de comprometerse un usuario o sistema, el impacto sea limitado (Electro, 2011).

Otro principio clave es la economía y simplificación de los mecanismos de seguridad. Este principio sugiere que los mecanismos de seguridad deben ser lo más simples posible para reducir la posibilidad de errores y, a su vez, garantizar una mayor eficacia. Al reducir la complejidad del sistema, se mejora la capacidad de detectar y corregir errores de seguridad. Además, las configuraciones por defecto seguras garantizan que un sistema esté protegido desde el principio, asegurando que, si un acceso no está explícitamente permitido, esté prohibido por defecto. Esto refuerza la idea de limitar accesos no autorizados (Electro, 2011).

El principio de diseño abierto es igualmente fundamental. Establece que la seguridad de un sistema no debe depender del secreto de su diseño, sino de la solidez de su implementación. Esto promueve la transparencia y permite que el diseño de seguridad sea revisado por la comunidad, lo que ayuda a identificar debilidades que podrían ser pasadas por alto en un entorno cerrado. La seguridad no debe basarse en el secretismo, sino en la robustez del diseño (Laotshi, 2020).

La mediación completa es otro principio clave que implica que todos los accesos a recursos deben ser verificados, independientemente de los permisos previamente concedidos o almacenados en caché. Esto asegura que cualquier cambio en los privilegios o en el contexto del usuario sea aplicado de manera inmediata y consistente, evitando accesos indebidos.

El enfoque de *MICROSOFT WELL-ARCHITECTED* se alinea con estos principios, promoviendo la filosofía de confianza cero (*zero trust*), en la que cada interacción dentro del sistema debe ser verificada explícitamente, sin suposiciones sobre la confiabilidad de los usuarios o dispositivos. Este enfoque ayuda a limitar el impacto de un ataque al restringir los permisos y minimizando el radio de explosión.^{en} caso de una violación de seguridad. Además, el enfoque de *Microsoft* refuerza que la seguridad no es un proceso único, sino continuo, y debe ser revisado y mejorado a lo largo del tiempo, adaptándose a nuevos desafíos y ataques (Microsoft, 2021).

Finalmente, la aceptación psicológica también es crucial para el diseño seguro. Los mecanismos de seguridad deben ser fáciles de usar y no interferir de manera significativa con la usabilidad del sistema. Esto asegura que los usuarios no busquen eludir las medidas de seguridad, sino que las adopten de forma natural. Los sistemas deben ser tan accesibles con medidas de seguridad como lo serían sin ellas, promoviendo así su aceptación entre los usuarios finales (Laotshi, 2020).

4.5. OWASP SAMM (Software Assurance Maturity Model)

4.5.1. Introducción a OWASP SAMM

El Modelo de Madurez de Aseguramiento de *Software* (*OWASP SAMM*, por sus siglas en inglés) es un marco desarrollado por *OWASP* diseñado para ayudar a las organizaciones a evaluar, formular e implementar una estrategia de seguridad en el desarrollo de software que pueda integrarse de manera efectiva en su ciclo de vida del desarrollo de *software* (*SDLC*). Este modelo es flexible y se adapta a diversos contextos organizacionales, independientemente de si el *software* es desarrollado internamente, externalizado o adquirido. Además, *OWASP SAMM* es aplicable a distintas metodologías, ya sean tradicionales, ágiles o basadas en *DevOps*.

El principal objetivo de *OWASP SAMM* es proporcionar una guía clara que permita a las organizaciones evaluar su postura actual en cuanto a seguridad del *software*, definir una estrategia de mejora y establecer una hoja de ruta para la implementación de prácticas seguras. Esto facilita que las organizaciones puedan progresar de forma iterativa en la adopción de mejores prácticas, logrando un enfoque más maduro y robusto para el aseguramiento del *software*.

Una de las características más destacadas de *OWASP SAMM* es su flexibilidad, lo que permite que organizaciones de cualquier tamaño personalicen y adopten el modelo según sus necesidades específicas. *OWASP SAMM* organiza sus principios en 15 prácticas de seguridad, distribuidas en cinco funciones empresariales clave. Estas funciones cubren desde la gobernanza hasta la implementación y operación de *software*. Cada una de las prácticas de seguridad está estructurada en tres niveles de madurez, que representan un progreso iterativo y constante hacia metas más sofisticadas y métricas de éxito más estrictas.

A medida que las organizaciones avanzan en los niveles de madurez, pueden medir su progreso y ajustar sus estrategias de acuerdo con sus capacidades y recursos. Es importante destacar que *OWASP SAMM* no exige alcanzar el nivel máximo en todas las categorías, sino que permite a las organizaciones establecer sus propios objetivos de acuerdo con sus necesidades y prioridades. Esto asegura que cada entidad pueda avanzar

en su proceso de aseguramiento de *software* de una manera controlada y adaptada a su contexto operativo (Foundation, 2024a).

4.5.2. Objetivos de OWASP SAMM

El *OWASP Software Assurance Maturity Model (SAMM)* es un marco desarrollado por *OWASP* que permite a las organizaciones mejorar la seguridad de su *software* de manera estructurada y controlada. A través de *OWASP SAMM*, las empresas pueden evaluar sus prácticas actuales, identificar puntos débiles en sus procesos de desarrollo de *software* y establecer estrategias para avanzar en su madurez de seguridad. Este modelo integra la seguridad en cada fase del ciclo de vida del desarrollo de *software* (*SDLC*), lo que es esencial en un entorno donde las amenazas cibernéticas evolucionan constantemente y donde la ciberseguridad debe ser una prioridad tanto para el desarrollo de *software* como para la adquisición de soluciones de terceros (Security, 2024a).

OWASP SAMM tiene varios objetivos clave, entre los que destacan:

- **Evaluar la postura de seguridad actual:** Mediante la evaluación de las prácticas de seguridad vigentes, las organizaciones pueden obtener una visión clara de su nivel de madurez en cuanto a seguridad del *software*. Esto les permite identificar en qué áreas deben mejorar para fortalecer su seguridad.
- **Definir una estrategia de seguridad:** El modelo proporciona una guía detallada para que las empresas puedan establecer una estrategia de seguridad alineada con sus objetivos y necesidades específicas. Esta estrategia considera los recursos disponibles, el nivel de riesgo de la organización y los requisitos normativos que deben cumplir.
- **Establecer un plan de acción y una hoja de ruta:** *OWASP SAMM* facilita la creación de una hoja de ruta con acciones específicas, divididas en fases y plazos, para implementar mejoras en las prácticas de seguridad. Estas actividades se estructuran en tres niveles de madurez, lo que permite a las organizaciones avanzar de manera progresiva y medible en la seguridad de su *software*.

Una de las principales ventajas de *OWASP SAMM* es su flexibilidad. El modelo está diseñado para adaptarse a cualquier tipo de organización, ya sea pequeña, mediana o grande, y puede implementarse independientemente del enfoque de desarrollo utilizado, ya sea tradicional, ágil o *DevOps*. Además, *OWASP SAMM* permite la personalización, lo que facilita que cada organización defina su propio nivel de madurez deseado para cada práctica de seguridad. Esta capacidad de ajuste convierte a *OWASP SAMM* en una herramienta poderosa para la optimización continua de la seguridad del *software*, especialmente en un entorno en el que los ciberataques son cada vez más sofisticados y perjudiciales (Security, 2024a).

4.5.3. Componentes y prácticas de SAMM

El *OWASP Software Assurance Maturity Model (SAMM)* está estructurado en torno a cinco funciones de negocio clave, cada una de las cuales abarca un conjunto de prácticas de seguridad. Estas funciones y prácticas forman el núcleo del modelo y están diseñadas para guiar a las organizaciones en la implementación

de medidas de seguridad a lo largo del ciclo de vida del software (*SDLC*). Cada función tiene asociadas tres prácticas de seguridad específicas, que a su vez están organizadas en diferentes niveles de madurez. Este enfoque modular permite a las organizaciones avanzar progresivamente en su estrategia de seguridad, adaptándose a sus necesidades y capacidades (Security, 2024a).

Funciones de negocio en SAMM

Las funciones de negocio de *SAMM* cubren las principales áreas de seguridad que deben ser consideradas en cualquier organización:

- **Gobernanza (G):** Se enfoca en la gestión de la seguridad a nivel organizativo, incluyendo la estrategia de seguridad, el cumplimiento normativo y la formación. Esta función asegura que la seguridad esté integrada en los procesos empresariales desde el nivel más alto.
- **Diseño (D):** Aborda la seguridad durante el diseño del *software*, desde los requisitos hasta la arquitectura. El objetivo es asegurar que el *software* se construya con medidas de seguridad adecuadas desde sus cimientos.
- **Implementación (I):** Se enfoca en la construcción y despliegue del *software*, asegurando que el código sea seguro y que se minimicen los defectos antes de su lanzamiento.
- **Verificación (V):** Cubre las pruebas de seguridad a lo largo del ciclo de vida del *software*, incluyendo análisis de seguridad y pruebas basadas en requisitos para detectar vulnerabilidades antes de que lleguen a producción.
- **Operaciones (O):** Asegura que la seguridad esté garantizada durante la operación del *software* en producción, preservando la confidencialidad, integridad y disponibilidad de los datos.

Prácticas de seguridad en SAMM

Cada función de negocio contiene tres prácticas de seguridad específicas, lo que da un total de 15 prácticas de seguridad dentro del marco *SAMM*. A continuación, se mencionan algunos ejemplos clave:

- **Gobernanza (G):** Estrategia y métricas; Política y cumplimiento; Formación y orientación.
- **Diseño (D):** Evaluación de amenazas; Requisitos de seguridad; Arquitectura de seguridad.
- **Implementación (I):** Construcción segura; Implantación segura; Gestión de fallos.
- **Verificación (V):** Pruebas de seguridad; Revisión de código; Evaluación de vulnerabilidades.
- **Operaciones (O):** Gestión de incidentes; Monitoreo de seguridad; Respuesta a incidentes.

Flujos y niveles de madurez

Cada práctica de seguridad dentro de *SAMM* se divide en dos flujos que representan diferentes aspectos de la práctica. Por ejemplo, en la práctica de "Implantación segura", uno de los flujos es la gestión de secretos, que incluye medidas para proteger las credenciales y otros secretos críticos.

Además, las actividades dentro de cada flujo están estructuradas en tres niveles de madurez, lo que permite a las organizaciones avanzar de manera progresiva en la implementación de medidas de seguridad:

- **Nivel 1:** Actividades más básicas y fáciles de implementar, como la formalización de procesos.
- **Nivel 2:** Introducción de automatizaciones y pruebas de seguridad más avanzadas.
- **Nivel 3:** Verificación automatizada de la seguridad en todos los componentes, tanto internos como externos.

Actividades y progresión

OWASP SAMM propone hasta 90 actividades distribuidas a lo largo de las prácticas y niveles de madurez, lo que permite a las organizaciones avanzar de manera iterativa en su seguridad. Las actividades están diseñadas para mejorar la seguridad de forma estructurada y controlada. Por ejemplo, en el flujo de "Implantación segura", el Nivel 1 incluye la formalización de procesos de despliegue, el Nivel 2 automatiza las verificaciones de seguridad durante el despliegue y el Nivel 3 asegura la integridad total del software desplegado, independientemente de su origen (interno o externo).

Con estos componentes y prácticas, *OWASP SAMM* proporciona una estructura completa que permite a las organizaciones evaluar, planificar e implementar medidas de seguridad ajustadas a sus necesidades y nivel de madurez. La modularidad del modelo asegura que cada empresa pueda enfocarse en las prácticas más relevantes para su contexto y mejorar gradualmente su postura de seguridad (Security, 2024a).

4.6. Herramientas de Análisis de Seguridad

4.6.1. SonarQube

Funcionalidades principales

¿Qué es SonarQube?

SonarQube es una herramienta de análisis estático de código diseñada para evaluar y mejorar la calidad del código fuente de un proyecto. Su principal objetivo es identificar problemas comunes que puedan afectar la eficiencia y la seguridad del código, tales como *bugs*, vulnerabilidades, malas prácticas de programación y *code smells* (malos olores en el código). Gracias a su capacidad para realizar análisis exhaustivos, SonarQube se ha convertido en una herramienta esencial para garantizar que el código fuente sea seguro y de alta calidad (Formación, 2024).

Características de SonarQube

SonarQube cuenta con una serie de características que lo convierten en una herramienta indispensable para los desarrolladores de *software*. A continuación, se detallan las más importantes:

- **Análisis de calidad del código:** *SonarQube* realiza un análisis profundo del código fuente y proporciona métricas claras sobre su calidad. Permite detectar problemas relacionados con la complejidad ciclomática, la duplicación de código y el incumplimiento de los estándares de codificación. Esto ayuda a los desarrolladores a comprender mejor las debilidades de su código y a mejorar su estructura.
- **Detección de bugs y vulnerabilidades:** *SonarQube* identifica posibles errores y vulnerabilidades que podrían afectar el rendimiento y la seguridad de la aplicación, tales como riesgos del *OWASP Top 10*, incluyendo inyecciones y problemas de autenticación. Esta capacidad es fundamental para asegurar que el software esté libre de problemas graves antes de su implementación, contribuyendo a la estabilidad y seguridad del proyecto (Gupta, 2024).
- **Security Hotspot Detection:** Permite detectar áreas del código que requieren revisión adicional sin ser vulnerabilidades inmediatas, tales como el manejo de permisos y el uso de datos sensibles. *SonarQube* destaca estas secciones para una revisión manual, garantizando que se sigan las mejores prácticas de seguridad (Gupta, 2024).
- **Detección de secretos codificados:** *SonarQube* escanea el código en busca de información sensible, como claves *API*, contraseñas o certificados, y proporciona sugerencias para su almacenamiento seguro, evitando exposiciones accidentales de información (Gupta, 2024).
- **Detección de Cross-Site Scripting (XSS):** Detecta y previene vulnerabilidades XSS en aplicaciones web, escaneando entradas no validadas y sugiriendo prácticas de codificación segura para mitigar riesgos (Gupta, 2024).
- **Prevención de SQL Injection:** Detecta y marca áreas donde las consultas *SQL* pueden ser vulnerables a inyecciones, recomendando consultas parametrizadas o el uso de *ORM* para reducir riesgos (Gupta, 2024).
- **Detección de duplicación de código:** Identifica y destaca secciones duplicadas en la base de código, promoviendo la refactorización para mejorar la consistencia y reducir las vulnerabilidades asociadas (Gupta, 2024).
- **Detección de vulnerabilidades en librerías de código abierto:** *SonarQube* analiza componentes de código abierto y dependencias en el proyecto, cruzando las versiones usadas con vulnerabilidades conocidas y sugiriendo actualizaciones o parches (Gupta, 2024).
- **Seguridad de APIs:** Analiza los *endpoints de API* para detectar problemas de autenticación, entradas no validadas y exposición de datos sensibles, generando informes centrados en vulnerabilidades de *APIs* (Gupta, 2024).
- **Protección contra Buffer Overflow:** Detecta operaciones de manejo de memoria inseguras, como límites de arreglo no verificados o copias excesivas de datos, proporcionando recomendaciones para evitar *exploits* de desbordamiento de búfer (Gupta, 2024).
- **Prevención de Ejecución Remota de Código (RCE):** Escanea el código para detectar deserialización insegura y problemas de validación de entrada que puedan llevar a la ejecución de código remoto, destacando el código que requiere una revisión cuidadosa (Gupta, 2024).

- **Cobertura de código para pruebas de seguridad:** Permite hacer un seguimiento de métricas de cobertura de código para pruebas de seguridad, resaltando las secciones no cubiertas que pueden representar riesgos y proporcionando información para mejorar la cobertura de las pruebas (Gupta, 2024).
- **Detección de Code Smells:** *SonarQube* identifica y recomienda la refactorización de código que presenta "code smells", prácticas deficientes que pueden llevar a problemas de seguridad y mantenimiento a largo plazo (Gupta, 2024).

Estas características no solo permiten que *SonarQube* detecte problemas, sino que también proporcionan soluciones y recomendaciones claras para mejorar el código y asegurar que cumpla con los estándares más altos de calidad y seguridad (Sonar, 2022).

¿Por qué usar SonarQube?

El uso de *SonarQube* ofrece una serie de beneficios clave para los desarrolladores:

- **Mejorar la calidad del código:** *SonarQube* proporciona informes detallados sobre la calidad del código, lo que permite a los desarrolladores identificar y corregir problemas de calidad y seguridad antes de que lleguen a producción.
- **Ahorro de tiempo y esfuerzo:** Al automatizar el proceso de análisis del código, *SonarQube* ayuda a ahorrar tiempo y esfuerzo en la identificación de problemas de calidad. Los desarrolladores pueden centrarse en la corrección de errores y la mejora continua sin necesidad de realizar análisis manuales exhaustivos.
- **Garantizar la seguridad y estabilidad de la aplicación:** *SonarQube* es fundamental para asegurar que la aplicación sea segura y estable, lo que a su vez garantiza la satisfacción del usuario final y la protección de la reputación de la empresa que desarrolla el software.

Cómo implementar SonarQube en tu proyecto

Implementar *SonarQube* en un proyecto es un proceso sencillo que puede integrarse de manera efectiva en los flujos de trabajo existentes. A continuación, se describen los pasos básicos para configurar *SonarQube*:

1. **Instalación:** Descarga e instala la versión más reciente de *SonarQube*. Se recomienda revisar los requisitos de sistema para asegurarse de que la infraestructura del proyecto pueda soportar la herramienta.
2. **Integración en CI/CD:** Configura la integración de *SonarQube* con tu plataforma de CI/CD para asegurar que el análisis de calidad del código se realice automáticamente en cada *commit* o despliegue.
3. **Análisis de código:** Ejecuta los análisis de código y revisa los informes generados. Estos informes identificarán problemas clave y proporcionarán recomendaciones específicas para mejorar el código.
4. **Corrección de problemas:** Usa los resultados del análisis para guiar el proceso de corrección de errores y mejoras en la calidad del código.

La implementación de *SonarQube* desde las primeras fases del proyecto asegura que el código cumpla con los más altos estándares de calidad y seguridad desde el inicio. Esto, a su vez, permite a los desarrolladores detectar y corregir problemas de manera temprana, evitando posibles complicaciones en etapas avanzadas del desarrollo (Sonar, 2022; Gupta, 2024).

Beneficios y limitaciones

Beneficios

El uso de *SonarQube* en este proyecto ofrece varios beneficios clave:

- **Identificación temprana de problemas:** La capacidad de detectar *bugs*, vulnerabilidades y malos olores en el código (*code smells*) en las primeras etapas del desarrollo permite a los desarrolladores realizar correcciones antes de que estos problemas afecten el proyecto en fases más avanzadas.
- **Mejora continua del código:** Gracias a los informes detallados y las recomendaciones proporcionadas por *SonarQube*, se fomenta una cultura de mejora continua dentro del equipo, lo que resulta en un código más limpio y eficiente.
- **Ahorro de tiempo y recursos:** Al automatizar los análisis de código, *SonarQube* ahorra tiempo en las revisiones manuales y reduce la posibilidad de errores humanos, lo que a su vez mejora la eficiencia del equipo de desarrollo.

Limitaciones

Sin embargo, el uso de *SonarQube* también presenta algunas limitaciones:

- **Configuración inicial compleja:** Configurar *SonarQube* y *Sonar Scanner* puede ser un proceso complicado, especialmente para aquellos que no están familiarizados con la herramienta. Esto puede requerir tiempo adicional para aprender a usarlo correctamente.
- **Limitaciones en la personalización:** Aunque *SonarQube* ofrece muchas reglas y configuraciones pre-determinadas, personalizar reglas específicas para proyectos particulares puede ser limitado sin la edición de pago de la herramienta.
- **Dependencia del análisis estático:** *SonarQube* se basa principalmente en el análisis estático de código, lo que significa que no detecta problemas de seguridad que solo son visibles durante la ejecución del software (Sonar, 2022; Gupta, 2024).

4.6.2. Appswep

Funcionalidades principales

¿Qué es AppSweep?

AppSweep es una herramienta gratuita de análisis de seguridad para aplicaciones móviles desarrollada por *Guardsquare*. Está diseñada específicamente para analizar aplicaciones *Android*, evaluando posibles vulnerabilidades y ofreciendo recomendaciones para corregirlas. Su principal objetivo es identificar riesgos de seguridad que podrían comprometer la integridad de la aplicación o la información del usuario, proporcionando a los desarrolladores un informe detallado sobre los problemas encontrados y las soluciones recomendadas.

Características de AppSweep

AppSweep ofrece una serie de características que lo convierten en una herramienta eficaz para la detección de vulnerabilidades en aplicaciones móviles:

- **Análisis de seguridad detallado:** *AppSweep* realiza un análisis exhaustivo de las aplicaciones *Android*, detectando vulnerabilidades que podrían poner en riesgo la seguridad del usuario, como fugas de datos, configuraciones inseguras y posibles accesos no autorizados.
- **Detección de permisos innecesarios:** La herramienta identifica si la aplicación está solicitando permisos innecesarios que puedan aumentar el riesgo de exposición a ataques o a fugas de datos.
- **Compatibilidad con archivos APK:** *AppSweep* permite cargar directamente el archivo *APK* de la aplicación para analizar su seguridad. Esto facilita la integración en el flujo de trabajo de desarrollo, ya que el análisis puede realizarse antes de la publicación de la aplicación en tiendas como *Google Play*.
- **Recomendaciones específicas:** *AppSweep* no solo detecta vulnerabilidades, sino que también proporciona recomendaciones claras y específicas para solucionarlas, ayudando a los desarrolladores a mitigar los riesgos de seguridad de manera eficiente.
- **Interfaz web y local:** *AppSweep* puede ser utilizado en línea mediante su plataforma web o de manera local, descargando la herramienta, lo que ofrece flexibilidad en el modo de implementación dependiendo de las necesidades del proyecto.

Estas características convierten a *AppSweep* en una solución eficiente para garantizar la seguridad en aplicaciones *Android* durante todo el ciclo de desarrollo, asegurando que el producto final esté libre de vulnerabilidades críticas.

Configuración y uso en el proyecto

En este proyecto, *AppSweep* se utilizará para analizar la seguridad de la aplicación móvil que estamos desarrollando. Para ello, el primer paso será descargar *AppSweep* o utilizar su versión en línea, según se considere más adecuado para el flujo de trabajo del equipo.

El proceso de configuración y uso se llevará a cabo de la siguiente manera:

1. **Descarga e instalación:** Se descargará *AppSweep* en el entorno local o se utilizará la versión *web* si se prefiere no instalar la herramienta localmente. En cualquier caso, la herramienta permitirá analizar archivos *APK* de la aplicación *Android*.

2. **Cargar el archivo APK:** Una vez que se haya desarrollado la versión de la aplicación móvil, se generará el archivo *APK* correspondiente, que será cargado en *AppSweep* para su análisis.
3. **Análisis de vulnerabilidades:** *AppSweep* analizará el *APK* para identificar posibles vulnerabilidades, fugas de datos o configuraciones inseguras. El análisis incluirá una revisión de los permisos solicitados por la aplicación, verificando si son adecuados o si exponen a la aplicación a riesgos innecesarios.
4. **Revisión del informe:** Al finalizar el análisis, *AppSweep* generará un informe detallado con todas las vulnerabilidades encontradas. Este informe será revisado por el equipo de desarrollo para corregir los problemas y asegurar que la aplicación cumpla con los estándares de seguridad requeridos antes de su lanzamiento.
5. **Corrección de problemas:** Los desarrolladores realizarán los ajustes y correcciones basados en las recomendaciones proporcionadas por *AppSweep*, asegurando que la aplicación sea lo más segura posible antes de su despliegue.

Este ciclo de análisis y corrección se repetirá de manera iterativa para asegurar que cada nueva versión de la aplicación esté libre de vulnerabilidades y cumpla con los más altos estándares de seguridad.

Beneficios y limitaciones

Beneficios

El uso de *AppSweep* ofrece varias ventajas clave para la seguridad del desarrollo de aplicaciones móviles en este proyecto:

- **Detección exhaustiva de vulnerabilidades:** *AppSweep* realiza un análisis exhaustivo de los archivos *APK*, detectando una amplia gama de vulnerabilidades que podrían poner en riesgo la seguridad de la aplicación y la privacidad de los usuarios.
- **Facilidad de uso:** La herramienta es fácil de usar, con una interfaz intuitiva tanto en su versión *web* como en la local. Esto permite que los desarrolladores puedan identificar problemas de seguridad rápidamente y sin complicaciones.
- **Recomendaciones detalladas:** *AppSweep* no solo detecta problemas, sino que también proporciona soluciones específicas y claras para que los desarrolladores puedan corregir las vulnerabilidades encontradas de manera eficiente.
- **Versatilidad:** La posibilidad de usar *AppSweep* tanto en línea como de manera local ofrece flexibilidad según las necesidades del proyecto.

Limitaciones

Sin embargo, *AppSweep* también presenta algunas limitaciones que deben tenerse en cuenta:

- **Análisis limitado a aplicaciones Android:** *AppSweep* está diseñado específicamente para analizar aplicaciones Android, lo que significa que no es útil para proyectos que incluyan aplicaciones *iOS* u otras plataformas.
- **Limitaciones en la versión gratuita:** Aunque *AppSweep* es gratuito, algunas funciones avanzadas de análisis podrían estar disponibles solo en versiones pagas o complementarias, lo que puede limitar su alcance para algunos proyectos.
- **Dependencia del análisis estático:** Al igual que otras herramientas de análisis estático, *AppSweep* puede no detectar todos los problemas de seguridad que solo se manifiestan durante la ejecución, lo que puede requerir pruebas adicionales con herramientas de análisis dinámico.

4.7. Reglamento General de Protección de Datos (GDPR)

El Reglamento General de Protección de Datos (**GDPR**), implementado desde mayo de 2018, se refiere a una serie de reglas creadas por el Parlamento Europeo, el Consejo de la Unión Europea y la Comisión Europea para regular el tratamiento de datos personales de los ciudadanos de la Unión Europea (UE). Este reglamento tiene como objetivo principal proteger la privacidad de los ciudadanos y asegurar que las organizaciones que recopilan, almacenan y procesan estos datos lo hagan de manera transparente, legal y justa. Las disposiciones del **GDPR** cubren tanto los datos personales, como las fotos, direcciones de correo electrónico, y direcciones *IP*, hasta información más sensible, como los datos financieros o de salud.

Uno de los principios fundamentales establecidos en el artículo 5 del **GDPR** es la licitud, equidad y transparencia en el tratamiento de los datos personales (Consulting, 2022a). Esto significa que los datos deben ser recolectados solo con el consentimiento explícito del usuario, para fines legítimos, y las organizaciones deben garantizar la transparencia al explicar cómo usarán los datos personales de las personas afectadas. Además, el **GDPR** establece el principio de minimización de datos, lo que implica que solo se deben recopilar los datos estrictamente necesarios para el propósito específico para el cual se solicitan. De igual manera, los datos recolectados deben ser exactos y actualizados, y el controlador debe asegurarse de que cualquier dato incorrecto sea corregido o eliminado.

Otro aspecto importante es la limitación de almacenamiento, que exige que los datos no sean conservados más allá del tiempo necesario para cumplir con los fines originales de su recolección. Además, el **GDPR** establece que las empresas deben implementar medidas técnicas y organizativas adecuadas para garantizar la seguridad, confidencialidad e integridad de los datos personales, protegiéndolos contra accesos no autorizados, pérdidas accidentales o destrucción.

El artículo 6 del **GDPR** destaca los casos en los que el tratamiento de datos es lícito (Consulting, 2022b). Para que el procesamiento sea legal, al menos uno de los siguientes puntos debe cumplirse: el consentimiento del usuario, el cumplimiento de un contrato, la necesidad de proteger intereses vitales, el cumplimiento de una obligación legal, la realización de tareas de interés público o la búsqueda de un interés legítimo. En todos los casos, las organizaciones deben demostrar su responsabilidad y capacidad para cumplir con estos requisitos, conforme lo establece el artículo 5.

4.7.1. GDPR y Guatemala

Aunque el Reglamento General de Protección de Datos (**GDPR**) se aplica principalmente dentro de la Unión Europea, su alcance ha tenido efectos más allá de las fronteras europeas, afectando también a empresas y organizaciones fuera de la UE que procesan datos de ciudadanos europeos. Para las organizaciones que tienen clientes o relaciones comerciales con residentes de la UE, cumplir con el **GDPR** es obligatorio, independientemente de la ubicación geográfica de la organización.

En Guatemala, aunque no existe una legislación específica que regule la protección de datos de manera tan detallada como el **GDPR**, se utiliza la Ley de Libre Acceso a la Información Pública (Hernández, 2021), que regula el manejo de los datos personales y los datos sensibles. Esta ley establece ciertos mecanismos de protección y tipifica delitos relacionados con la violación de la privacidad de datos. Sin embargo, su aplicación no es tan estricta ni extensa como el **GDPR**, lo que deja un vacío legal en la protección de datos en comparación con otros países de la región. Esto representa un reto para las organizaciones que buscan operar en Guatemala o en otros países de América Latina con normativas menos estrictas, pero que interactúan con datos de ciudadanos europeos.

En Latinoamérica, varios países han avanzado en la adopción de leyes de protección de datos. Chile fue pionero en la región al promulgar su ley en 1999, seguido por Argentina en 2000, y otros países como Uruguay (2008), México (2010), y Brasil (2018). A pesar de la ausencia de un marco regulatorio uniforme en toda la región, el impacto del **GDPR** ha incentivado el desarrollo de legislaciones locales enfocadas en la protección de datos personales, ya que muchas empresas latinoamericanas realizan transacciones o tienen clientes en Europa, lo que las obliga a adaptarse a los estándares internacionales.

El desarrollo del proyecto se llevó a cabo en varias fases, cada una enfocada en la implementación de buenas prácticas de seguridad y la mejora continua en el análisis de la calidad del código. La metodología adoptada se basó en el marco de trabajo *OWASP SAMM*, abarcando prácticas en Gobernanza, Diseño, Implementación y Verificación para garantizar la seguridad del software. A continuación, se detalla el proceso en cada una de las fases:

5.1. Fase 1: Selección de Tecnologías

El primer paso fundamental del proyecto consistió en realizar una investigación exhaustiva sobre las herramientas disponibles para el análisis estático y dinámico del código, lo cual permitió identificar las tecnologías que mejor se ajustaban a los objetivos de seguridad y calidad establecidos. Las herramientas seleccionadas se centraron en dos áreas clave: análisis estático (*SAST*) y análisis dinámico (*DAST*), cada uno con enfoques específicos para aplicaciones de *backend* y *frontend* móvil.

5.1.1. Herramientas de Análisis Estático de Código

Para el análisis estático, se investigaron varias opciones reconocidas en la industria, tales como *SonarQube*, *Collaborator* y *Embold*. Cada una de estas herramientas ofrece funcionalidades avanzadas para detectar vulnerabilidades y mejorar la calidad del código, aunque presentan diferencias significativas en términos de integración, facilidad de uso y cobertura de estándares.

- **SonarQube:** *SonarQube* fue evaluada como la herramienta con mayores ventajas para el análisis estático de código en este proyecto. Su facilidad de configuración e integración con repositorios como

GitHub permite la exportación y análisis directo del código fuente. *SonarQube* también incluye de forma nativa reglas y estándares de calidad como el *Clean Code* de Robert C. Martin, *OWASP Top 10*, *SANS Top 25* y *CWE (Common Weakness Enumeration)*, lo que proporciona una cobertura completa de los aspectos más críticos de seguridad. Además, *SonarQube* identifica diversas categorías de problemas, tales como *bugs*, *code smells*, y vulnerabilidades, detallando el origen de cada problema, su impacto en la seguridad y brindando recomendaciones sobre cómo resolverlo. Esta capacidad de análisis detallado facilita una gestión efectiva de las vulnerabilidades, razón por la cual se seleccionó *SonarQube* como la herramienta principal para el análisis estático en el *backend*.

SonarQube ha sido utilizado por grandes empresas como *eBay* y *NASA*, quienes lo emplean para garantizar la seguridad y la calidad del código en sus desarrollos. En el caso de *eBay*, por ejemplo, *SonarQube* contribuyó a mantener altos estándares de seguridad en su plataforma de comercio electrónico, una aplicación que procesa millones de transacciones diarias y debe cumplir con estrictos controles de seguridad para proteger la información de los usuarios. La robustez de *SonarQube* y su capacidad de integración con herramientas de *CI/CD* lo convierten en una solución ideal para proyectos de gran escala.

- **Collaborator:** Esta herramienta está diseñada para facilitar revisiones de código colaborativas, destacándose en entornos en los que el análisis manual y el trabajo en equipo son prioritarios. Aunque *Collaborator* ofrece características útiles para la revisión y documentación de problemas, no cuenta con las capacidades integradas de detección de vulnerabilidades y estándares de seguridad que ofrece *SonarQube*. Debido a esta limitación en la cobertura automática de estándares y su menor facilidad de integración, *Collaborator* fue descartada para el análisis estático de seguridad automatizado.

Ejemplos de uso de *Collaborator* incluyen a *Micro Focus* y *Volkswagen*, quienes lo emplean en entornos de desarrollo donde las revisiones de código son colaborativas y requieren un enfoque personalizado. *Volkswagen*, por ejemplo, lo utiliza para revisar componentes de *software* en sus sistemas de vehículos, asegurando una calidad y coherencia en el código mediante revisiones manuales detalladas.

- **Embold:** *Embold* es una plataforma orientada a la mejora de la calidad del código, enfocada principalmente en la detección de *code smells* y problemas de mantenibilidad. Aunque es útil para optimizar el código en términos de rendimiento y legibilidad, *Embold* no cuenta con la misma capacidad de detección de vulnerabilidades de seguridad que *SonarQube*. Por esta razón, *Embold* no fue seleccionada como la herramienta principal de *SAST*, aunque sigue siendo una opción viable para equipos que buscan optimizar el código desde una perspectiva de calidad en lugar de seguridad.

Embold ha sido implementado en empresas como *Siemens* y *Nokia*, quienes buscan mejorar la mantenibilidad de su código y reducir el costo de desarrollo a largo plazo. *Nokia*, por ejemplo, lo utiliza para mejorar la calidad del código en sus soluciones de telecomunicaciones, optimizando el rendimiento y la eficiencia del desarrollo.

5.1.2. Herramientas de Análisis Dinámico de Código para Aplicaciones Móviles

Para el análisis dinámico (*DAST*) de la aplicación móvil, se exploraron herramientas como *AppSweep*, *MobSF*, *Arachni*, *OWASP ZAP* y *Burp Suite*. Las opciones seleccionadas se eligieron con base en su efectividad para analizar aplicaciones móviles y su capacidad para detectar vulnerabilidades durante la ejecución de

la aplicación.

- **AppSweep:** *AppSweep* se destacó como la herramienta ideal para el análisis dinámico de aplicaciones móviles. Desarrollada específicamente para entornos móviles, *AppSweep* permite la detección de vulnerabilidades durante la ejecución de la aplicación, identificando problemas relacionados con permisos, almacenamiento inseguro y configuraciones deficientes en tiempo de ejecución. Además, *AppSweep* ofrece una interfaz intuitiva que facilita la interpretación de los resultados y permite integrar las pruebas en el ciclo de desarrollo. Su capacidad para identificar vulnerabilidades específicas de aplicaciones móviles la hace especialmente útil para el análisis *DAST*, lo cual justifica su elección como la herramienta primaria en este ámbito.

Empresas como **Samsung** y **Facebook** han optado por soluciones similares en seguridad de aplicaciones móviles, aprovechando *AppSweep* para analizar vulnerabilidades en aplicaciones de uso masivo. La capacidad de *AppSweep* para integrarse en *pipelines* de *CI/CD* y su adaptabilidad para aplicaciones móviles de alto rendimiento permiten un ciclo de revisión de seguridad constante y efectivo.

- **MobSF (Mobile Security Framework):** Aunque *MobSF* también ofrece análisis dinámico, fue seleccionado principalmente como la herramienta de análisis estático para la aplicación móvil, debido a su enfoque integral en la seguridad de aplicaciones móviles y su capacidad para detectar una amplia variedad de problemas de seguridad antes de la ejecución. *MobSF* permite identificar vulnerabilidades de configuración, análisis de permisos, y detecta prácticas de desarrollo inseguras en el código. Este enfoque preventivo complementa el análisis *DAST* de *AppSweep*, permitiendo una cobertura más completa de los riesgos de seguridad en la aplicación móvil.

Empresas como **Paytm** y **Grab** han implementado *MobSF* en sus procesos de desarrollo de aplicaciones móviles para garantizar que los elementos de seguridad, desde permisos hasta configuraciones, se revisen exhaustivamente antes de lanzarlas al mercado. Esto asegura un nivel de protección robusto frente a amenazas específicas de entornos móviles.

- **Arachni:** Esta herramienta es útil para la detección de vulnerabilidades en aplicaciones web, aunque carece de características específicas para entornos móviles. Dado que el proyecto requiere una cobertura robusta de aplicaciones móviles, *Arachni* fue descartada en favor de opciones más especializadas como *AppSweep* y *MobSF*.

Arachni ha sido adoptado por empresas como **Adobe** y **Oracle**, que lo utilizan en sus *pipelines* de desarrollo web para identificar y mitigar vulnerabilidades en entornos de producción. *Adobe*, por ejemplo, lo utiliza en sus productos de software en la nube para identificar amenazas de seguridad y mejorar sus defensas ante posibles ataques en sus aplicaciones web.

- **OWASP ZAP:** *OWASP ZAP* es una herramienta poderosa para realizar pruebas de seguridad en aplicaciones web, con un enfoque en la identificación de vulnerabilidades como inyección *SQL* y *XSS*. Sin embargo, al no estar optimizada para el análisis dinámico de aplicaciones móviles, su uso fue considerado menos adecuado en comparación con *AppSweep*.

OWASP ZAP es utilizado ampliamente en organizaciones como **Mozilla** y **GitLab**. *Mozilla* lo utiliza como una herramienta esencial para pruebas de penetración en sus servicios web, y *GitLab* lo ha integrado en su *CI/CD* para realizar escaneos de seguridad en sus aplicaciones web, contribuyendo así a la seguridad de sus proyectos de código abierto y de sus clientes.

- **Burp Suite:** *Burp Suite* es ampliamente utilizada para pruebas de seguridad *web* y permite analizar aplicaciones móviles en ciertos contextos, aunque su enfoque principal no es el entorno móvil. Dado que *AppSweep* y *MobSF* proporcionan una mejor adaptación para los requerimientos específicos del proyecto en cuanto a aplicaciones móviles, *Burp Suite* fue descartada como herramienta principal para el análisis *DAST* en móviles.

Burp Suite es ampliamente adoptada por empresas de seguridad como **Trustwave** y **Verizon**. *Trustwave* la emplea en auditorías de seguridad de redes y aplicaciones, mientras que *Verizon* utiliza *Burp Suite* en sus servicios de consultoría de seguridad para realizar análisis de vulnerabilidades *web* y asegurar los sistemas de sus clientes frente a posibles ataques.

De los cuales se escogieron las siguiente herramientas:

SonarQube, con su capacidad para detectar *bugs*, *code smells* y vulnerabilidades a partir de una configuración que incluye *OWASP Top 10* y otros estándares relevantes, facilita una evaluación exhaustiva de posibles vulnerabilidades en el código y brinda recomendaciones claras que apoyan los esfuerzos de mitigación. Esta herramienta permite además generar informes detallados sobre las vulnerabilidades y riesgos detectados, promoviendo la corrección temprana de los problemas en el ciclo de vida del software, lo cual contribuye a reducir riesgos en las versiones finales de producción y fomenta el cumplimiento de los estándares de seguridad establecidos.

Por su parte, *AppSweep* realiza evaluaciones dinámicas en tiempo de ejecución de la aplicación móvil, detectando problemas de permisos, almacenamiento inseguro y configuraciones deficientes que podrían afectar la seguridad de los datos de los usuarios. Esta herramienta asegura que la aplicación cumpla con las prácticas de seguridad móvil más estrictas. *MobSF* complementa estos análisis proporcionando una revisión estática exhaustiva de permisos, configuraciones y el uso de prácticas seguras en el desarrollo móvil, generando un entorno de confianza y robustez desde la fase de planificación hasta el despliegue, y siguiendo las recomendaciones de seguridad de *OWASP*.

La elección de estas herramientas también apoya la creación de una cultura de seguridad proactiva dentro del equipo de desarrollo, ya que los reportes específicos y detallados permiten revisiones semanales de los riesgos y vulnerabilidades, facilitando una capacidad de respuesta rápida y efectiva. En conjunto, *SonarQube*, *AppSweep* y *MobSF* no solo ofrecen una cobertura completa de las vulnerabilidades críticas, sino que también proporcionan una base sólida para implementar prácticas avanzadas de autenticación, autorización y protección de datos (como la autenticación multifactor y el control de acceso basado en roles), contribuyendo así a garantizar la integridad, confidencialidad y disponibilidad de la información de los usuarios.

Con esta combinación de herramientas, el proyecto se encuentra en una posición óptima para alcanzar el objetivo de cumplir con los más altos estándares de seguridad a lo largo de todo el ciclo de vida del software, logrando una plataforma segura y confiable que protege y prioriza la privacidad de los datos del usuario.

5.2. Fase 2: Preparación e Instalación de Herramientas

El primer paso del proyecto consistió en la instalación y configuración de las herramientas necesarias para el análisis estático del código. Se implementó la versión *Community* de *SonarQube* debido a que es gratuita y ofrece funcionalidades esenciales para realizar un análisis *SAST* (*Static Application Security Testing*). Esta elección se debió a la limitación presupuestaria y a que las funcionalidades adicionales de la versión *Developer* no eran críticas para los objetivos del proyecto.

Posteriormente, se configuró *SonarScanner* en las variables de entorno del sistema para su uso con *SonarQube*. Una vez finalizada la configuración, se creó un nuevo proyecto en *SonarQube* para servir como el espacio destinado a cargar y analizar el código fuente a lo largo del desarrollo del proyecto.

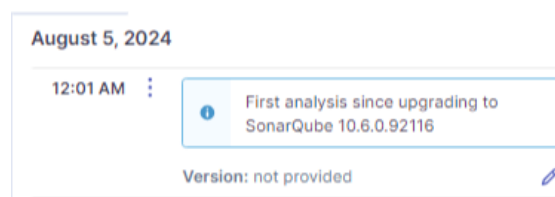


Figura 5.1: Registro de Scan del 5 de Agosto.

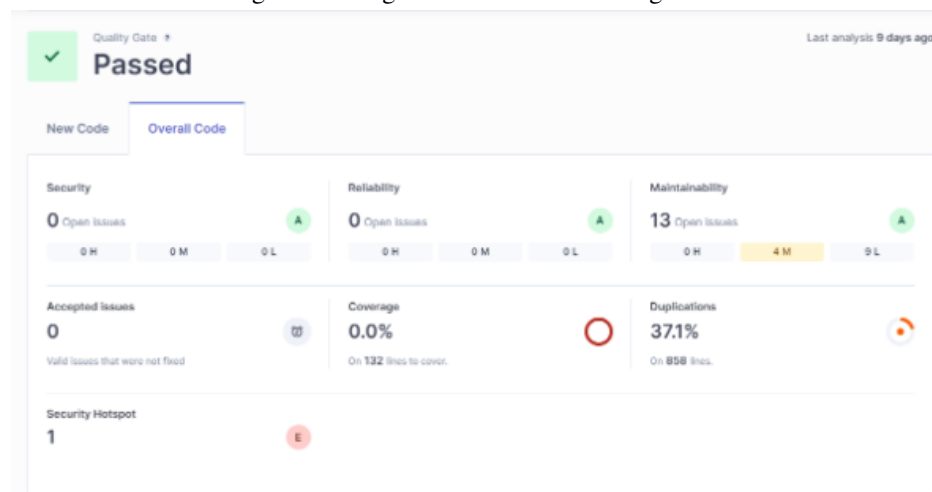


Figura 5.2: Resultado de Scan del 5 de Agosto.

Para el análisis dinámico de la aplicación móvil, se incorporaron las herramientas *AppSweep* y *MobSF*.

En el caso de *AppSweep*, la configuración inicial fue rápida y sencilla. Solo fue necesario ingresar con una cuenta de usuario en la plataforma, la cual proporciona una opción gratuita para escanear aplicaciones móviles, tanto en *Android* como en *iOS*. Esto permitió realizar evaluaciones de seguridad completas sin costos adicionales. La facilidad de uso de *AppSweep* para subir y analizar aplicaciones directamente desde la plataforma facilita la incorporación de pruebas de seguridad dentro del ciclo de desarrollo de forma ágil, como se muestra en la siguiente imagen:

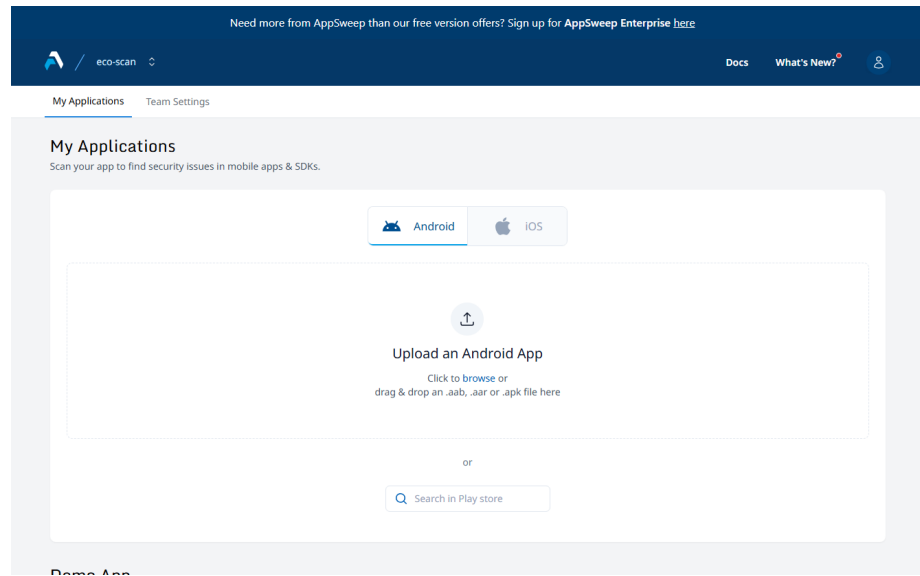


Figura 5.3: Página de inicio de AppSweep.

Por otro lado, *MobSF* también ofrece un portal en el que es posible cargar aplicaciones en formato *.apk* o *.ipa*, correspondientes a los sistemas *Android* e *iOS* respectivamente. Esta herramienta permite analizar la seguridad de los archivos de instalación directamente desde una interfaz *web*, proporcionando un análisis detallado de los permisos y configuraciones que podrían poner en riesgo la seguridad del usuario. La opción de *MobSF* de analizar aplicaciones sin necesidad de integración compleja agiliza su uso para evaluaciones de seguridad en entornos móviles.

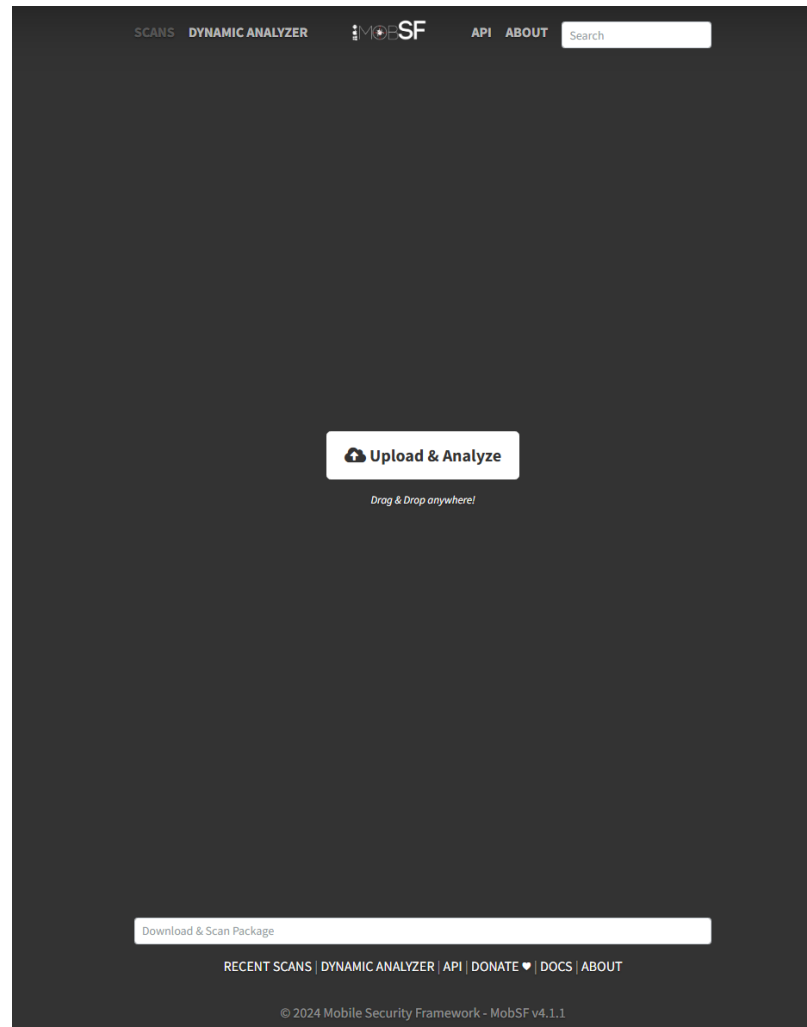


Figura 5.4: Página de carga de archivos en MobSF.

Además, se planificaron las tecnologías fundamentales para el proyecto, incluyendo tanto el entorno de infraestructura en la nube como los lenguajes y *frameworks* específicos para el desarrollo *frontend* y *backend*. Las decisiones se basaron en consideraciones de seguridad, compatibilidad y rendimiento, alineadas con los requisitos de la aplicación.

Infraestructura en la Nube: AWS

El equipo encargado de la infraestructura optó por utilizar AWS en lugar de Azure, debido a la robustez de los servicios de seguridad y la facilidad de integración con herramientas de *DevOps*. Me informaron que AWS proporciona opciones avanzadas de seguridad, incluyendo un control detallado sobre los permisos y el cifrado de datos sensibles. Además, mencionaron que las capacidades de personalización de políticas de seguridad de AWS son esenciales para configurar un entorno seguro, acorde con las mejores prácticas.

Backend: Node.js con Express.js

El equipo de desarrollo *backend* seleccionó Node.js junto con el *framework* Express.js por sus ventajas en términos de escalabilidad y soporte para manejar múltiples solicitudes simultáneas. Me explicaron que

estas características son cruciales para aplicaciones que podrían recibir un alto volumen de tráfico en tiempo real. También discutieron que *Node.js* y *Express.js* cuentan con módulos y herramientas diseñadas para seguir prácticas de codificación segura, lo cual contribuye a la protección contra ataques comunes como inyecciones *SQL* y *Cross-Site Scripting (XSS)*.

Cifrado: Bcrypt

El encargado del *backend* me consultó sobre la elección de un algoritmo para gestionar contraseñas y datos sensibles, considerando varias opciones. Después de investigar, recomendé *bcrypt* debido a sus ventajas de seguridad, especialmente su uso de *salting* para prevenir la reutilización de *hashes* y hacer que los ataques de fuerza bruta sean más difíciles. Esta recomendación se basó en las mejores prácticas de seguridad de *OWASP* para el manejo de contraseñas. El algoritmo fue implementado en *JavaScript*.

Frontend: React Native

El equipo de desarrollo *frontend* decidió utilizar *React Native* por su capacidad de optimizar el rendimiento en dispositivos móviles y su soporte para actualizaciones de seguridad en tiempo real. Me explicaron que *React Native* permite una fácil integración de medidas de seguridad, como el manejo seguro de la comunicación con el *backend* mediante *HTTPS* y el cifrado de datos en dispositivos. Además, destacaron su eficiencia al trabajar con bibliotecas de autenticación segura, lo cual es fundamental para proteger la información del usuario.

Con esta configuración de infraestructura y tecnologías de desarrollo, se estableció una base sólida y segura para el proyecto, optimizando tanto el rendimiento como la seguridad desde el inicio.

5.2.1. Reuniones Semanales de Seguimiento y Planificación

Con el objetivo de asegurar el progreso continuo y la calidad del proyecto, se establecieron reuniones semanales de seguimiento y planificación, las cuales se llevaron a cabo cada viernes. Durante estas reuniones, se abordaron varios aspectos clave para el desarrollo seguro de la aplicación:

5.3. Fase 3: Definición de la Estrategia de Seguridad

Para garantizar la seguridad y calidad del proyecto, se adoptó una estrategia basada en *OWASP SAMM*, enfocándose en las siguientes áreas:

5.3.1. Gobernanza: Educación y Orientación

Se priorizó la capacitación en seguridad para el equipo, proporcionando orientación en la adopción de prácticas seguras y asegurando que todos comprendieran las implicaciones de las decisiones tomadas. Esto incluyó la organización de sesiones formativas y el apoyo continuo durante el desarrollo.

5.3.2. Diseño: Arquitectura Segura

En colaboración con el encargado de la infraestructura, se realizaron revisiones exhaustivas de la arquitectura del sistema para asegurar que las configuraciones cumplieran con los estándares de seguridad necesarios. Se puso un énfasis particular en la implementación de controles de acceso basados en roles (*RBAC*), que permitieron asignar permisos específicos a los usuarios en función de sus responsabilidades dentro del proyecto. Este enfoque garantizó que cada usuario tuviera acceso únicamente a los recursos y funciones necesarios para llevar a cabo sus tareas, minimizando el riesgo de acceso no autorizado a información crítica o funciones sensibles.

Además, se establecieron políticas claras para la asignación y revisión periódica de los roles, asegurando que cualquier cambio en las responsabilidades de los usuarios fuera reflejado de manera oportuna en el sistema. Esto ayudó a mantener una gestión adecuada de los permisos y reforzó la protección de la infraestructura al limitar la exposición de los recursos a los privilegios mínimos necesarios.

5.3.3. Implementación: Gestión de Defectos

Para la gestión de defectos, se emplearon varias herramientas que permitieron un enfoque integral en la detección y corrección de vulnerabilidades. Se utilizaron *SonarQube* para el análisis *SAST* (*Static Application Security Testing*) y *AppSweep* para el análisis *DAST* (*Dynamic Application Security Testing*), lo que permitió la detección continua de problemas de seguridad en el código fuente.

Adicionalmente, se integró *MobSF* (*Mobile Security Framework*) para el análisis estático de aplicaciones móviles.

5.3.4. Verificación: Pruebas de Seguridad

Se estableció un proceso de verificación continua, que incluyó la ejecución de pruebas unitarias e integrales para garantizar el correcto funcionamiento del código. Las auditorías periódicas permitieron revisar la seguridad de los entornos de despliegue y la implementación de configuraciones seguras.

5.4. Fase 4: Evaluaciones de Seguridad y Mejora Continua

5.4.1. Capacitación del Equipo en Clean Code y OWASP Top 10

Como parte del esfuerzo para crear una cultura de seguridad y mejorar la calidad del desarrollo, se implementó un programa de capacitación semanal enfocado en las mejores prácticas de codificación y seguridad. Esta capacitación abarcó dos temas principales: *Clean Code* y los *OWASP Top 10*, los cuales fueron introducidos de manera gradual para asegurar una asimilación efectiva por parte del equipo.

- **Principios de Clean Code:** La primera fase de la capacitación se centró en los principios de *Clean*

Code, abordando aspectos clave como la simplicidad y claridad del código, con el objetivo de promover un desarrollo más estructurado y fácil de mantener. Se discutieron conceptos como:

- Mantener el código lo más simple posible sin sacrificar la funcionalidad.
- Uso adecuado de comentarios en áreas críticas del código para clarificar la funcionalidad y el propósito de los bloques de código.
- Nombres descriptivos para variables, funciones y clases para mejorar la legibilidad.
- Evitar duplicaciones y asegurar que cada fragmento de código tenga una sola responsabilidad.

Estos principios fueron presentados durante varias sesiones semanales, permitiendo que el equipo los asimilara y aplicara de manera gradual. La introducción progresiva de estos conceptos no solo permitió reforzar su importancia en la práctica diaria, sino que también promovió una cultura de calidad en el desarrollo, alineada con los estándares de *OWASP SAMM* en términos de Implementación y Gobernanza.

- **Capacitación en OWASP Top 10:** Además de los principios de *CLEAN CODE*, se dedicaron sesiones a la revisión de los *OWASP Top 10*, un conjunto de las vulnerabilidades de seguridad más comunes en aplicaciones web. Cada semana se presentó uno de los riesgos enumerados en *OWASP*, junto con ejemplos específicos proporcionados por la organización. Los temas abordados incluyeron:
 - Descripción de cada vulnerabilidad, su impacto en el sistema y ejemplos reales.
 - Buenas prácticas para evitar estos riesgos, adaptadas a los módulos del proyecto en desarrollo.
 - Casos prácticos para ilustrar cómo una vulnerabilidad podría comprometer la seguridad de la aplicación si no se gestiona correctamente.

Esta capacitación semanal permitió al equipo absorber los conceptos clave sin sobrecargarlo con demasiada información a la vez. Esta metodología aseguró una comprensión sólida de los riesgos y cómo mitigarlos en sus tareas cotidianas, promoviendo una cultura de seguridad proactiva y alineada con los objetivos específicos del proyecto.

5.4.2. Reuniones Semanales de Seguimiento y Planificación

Con el objetivo de asegurar el progreso continuo y la calidad del proyecto, se establecieron reuniones semanales de seguimiento y planificación, las cuales se llevaron a cabo cada viernes. Durante estas reuniones, se abordaron varios aspectos clave para el desarrollo seguro de la aplicación:

- **Revisión de Avances y Dificultades:** Cada miembro del equipo compartía los avances alcanzados durante la semana, identificando logros, desafíos y obstáculos que requerían apoyo adicional. Esta práctica fomentó un ambiente colaborativo donde se discutían soluciones y se distribuían tareas de acuerdo a las necesidades actuales del proyecto.
- **Planificación de Tareas y Tiempos:** En función de los puntos tratados, se establecían metas semanales con plazos definidos, lo cual facilitaba la organización del trabajo y aseguraba que todos los integrantes tuvieran claridad sobre los próximos pasos y sus responsabilidades específicas.

- **Actualización de Estado de la Seguridad y Calidad del Código:** Como líder del proyecto, presentaba un resumen del estado general del código, incluyendo vulnerabilidades detectadas y aspectos de cobertura de pruebas. También se detallaban áreas que necesitaban atención en términos de seguridad y buenas prácticas, destacando los puntos críticos para garantizar el cumplimiento de los estándares establecidos en *OWASP SAMM*.
- **Retroalimentación y Resolución de Dudas:** Estas sesiones se aprovecharon para aclarar dudas del equipo sobre las recomendaciones y configuraciones de seguridad necesarias. Se brindaba retroalimentación específica sobre las vulnerabilidades detectadas y la cobertura, proporcionando una guía para resolver los problemas detectados y mejorar las prácticas de codificación.

Las reuniones semanales no solo fomentaron una cultura de seguridad proactiva, sino que también facilitaron la alineación continua del equipo con los objetivos del proyecto. Esta práctica contribuyó directamente al área de Gobernanza dentro de *OWASP SAMM*, asegurando una comunicación clara y una gestión eficiente de la seguridad en cada fase del desarrollo.

5.4.3. Prueba de Phishing

Como parte de la evaluación inicial, se llevó a cabo una prueba de *phishing* para medir la capacidad del equipo en la identificación de correos electrónicos maliciosos. La prueba mostró un desempeño inicial con un promedio de 4.5 respuestas correctas, indicando la necesidad de mejorar en esta área. Durante esta etapa, se organizaron sesiones de capacitación específicas para el equipo, enfocadas en identificar señales comunes de *phishing* y reforzar prácticas de seguridad en el manejo del correo electrónico.

La razón para incluir esta prueba de *phishing* es que, dentro de las prácticas de seguridad, no solo me enfoco en la protección de la aplicación y el desarrollo, sino también en la seguridad del propio equipo. En el contexto de ciberseguridad, los miembros del equipo pueden ser un punto de vulnerabilidad, ya que existe la posibilidad de que caigan en un ataque de *phishing*. Esto podría resultar en la divulgación de credenciales sensibles, como las necesarias para acceder a AWS u otros servicios críticos. La capacitación y las pruebas de este tipo ayudan a reducir el riesgo de accesos no autorizados derivados de errores humanos y fortalecen la postura de seguridad del equipo en general.

Después de las sesiones de capacitación, se llevó a cabo una segunda prueba el 14 de septiembre, en la que se observó una mejora significativa con un promedio de 8.25 respuestas correctas. Esto reflejó el impacto positivo de las capacitaciones, ya que el equipo demostró una mayor habilidad para reconocer correos electrónicos maliciosos, mejorando la postura de seguridad general del proyecto.

Finalmente, se realizó una evaluación sorpresa el 19 de octubre para asegurar que el equipo aún recordaba los consejos y técnicas impartidos en las sesiones de capacitación para diferenciar entre correos electrónicos legítimos y maliciosos. En esta evaluación, el equipo obtuvo un promedio de 9 respuestas correctas, lo cual reflejó que las capacitaciones y los tips compartidos fueron efectivamente retenidos y aplicados.

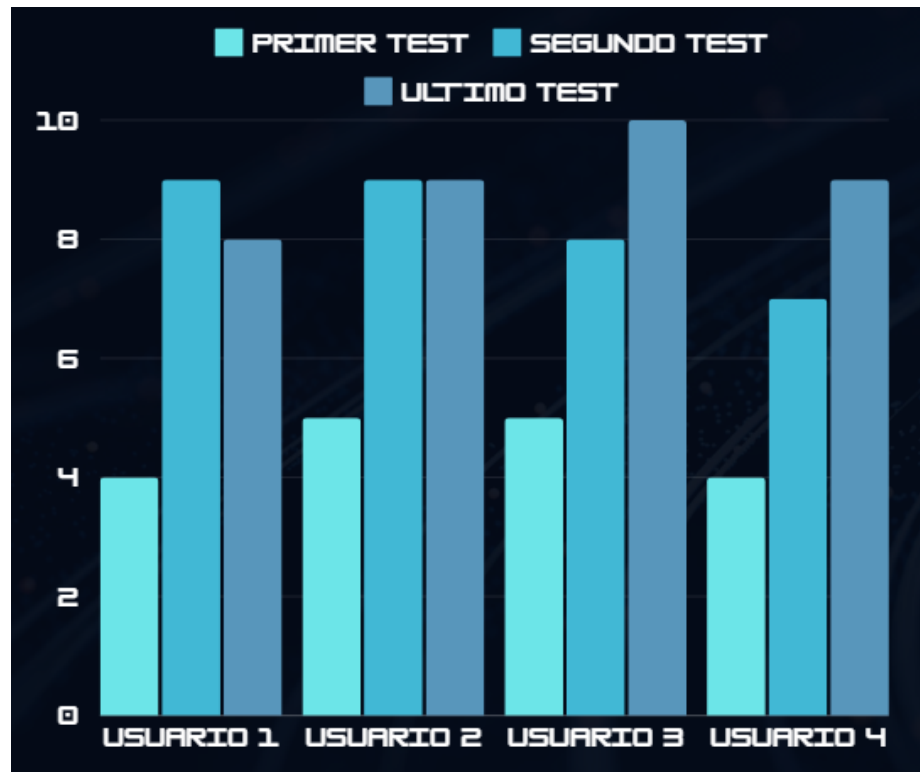


Figura 5.5: Resultados de phishing del equipo.

5.4.4. Análisis Estático del Código

Se llevaron a cabo análisis estáticos en diferentes momentos clave, comenzando el 5 de agosto, lo que permitió detectar problemas iniciales de mantenibilidad y seguridad. Tras varias rondas de corrección y reescaneos, el código mostró mejoras significativas en su calidad. Durante el proyecto, se realizaron capacitaciones regulares para asegurar que el equipo aplicara las mejores prácticas de desarrollo seguro, mejorando continuamente la calidad del código y asegurando la implementación correcta de las recomendaciones derivadas de los análisis.

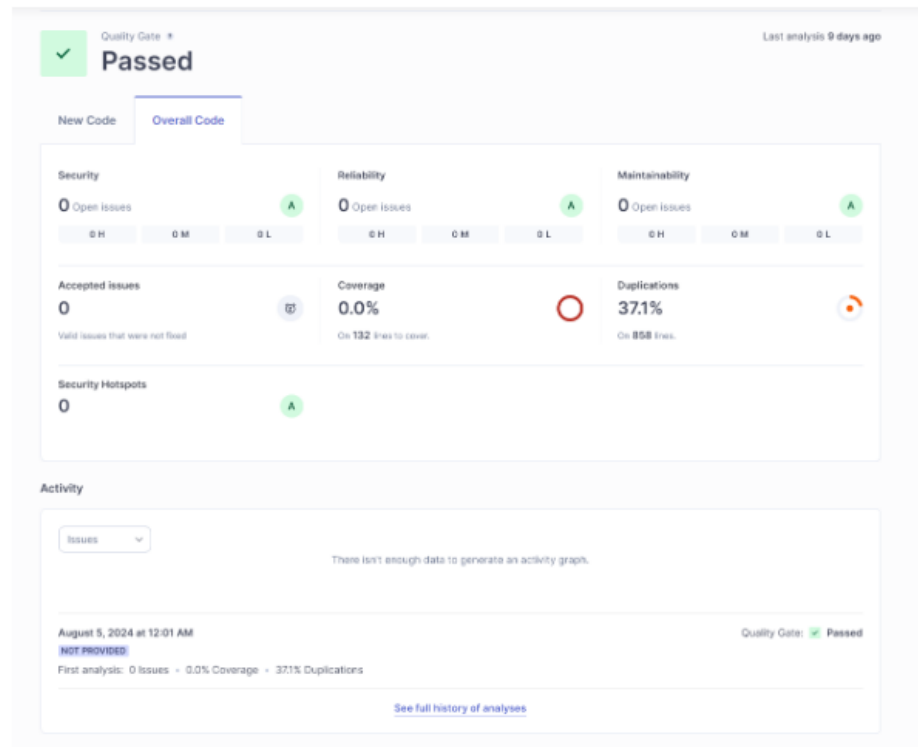


Figura 5.6: Resultado de Scan del 14 de Agosto.

En el análisis del 22 de agosto, se identificó que *SonarQube* requería un porcentaje de cobertura de pruebas mayor al 80%. Para cumplir con este criterio, se configuró el proyecto para excluir ciertas carpetas que no contribuían a la cobertura relevante. La revisión constante de los resultados permitió ajustar las estrategias de prueba y mejorar la precisión de los informes.

5.4.5. Migración a SonarCloud y Resultados de Cobertura

Para mejorar la automatización del análisis estático, se migró el proceso a *SonarCloud*, lo que permitió integrar la plataforma con *GitHub* y utilizar un dominio público para los análisis. Una de las principales razones de esta migración fue que *SonarCloud* proporcionaba un dominio público, lo cual facilitó la automatización de los escaneos mediante *CI/CD* utilizando *GitHub Actions*. Esto permitió configurar flujos de trabajo automáticos para que los análisis de calidad del código se ejecutaran cada vez que se realizaban cambios en la rama principal del proyecto.

Este cambio no solo mejoró la visibilidad de los resultados del análisis, sino que también facilitó una mayor colaboración en el equipo, ya que todos los desarrolladores podían ver los resultados del análisis y recibir capacitaciones sobre cómo interpretar los hallazgos y actuar sobre ellos.

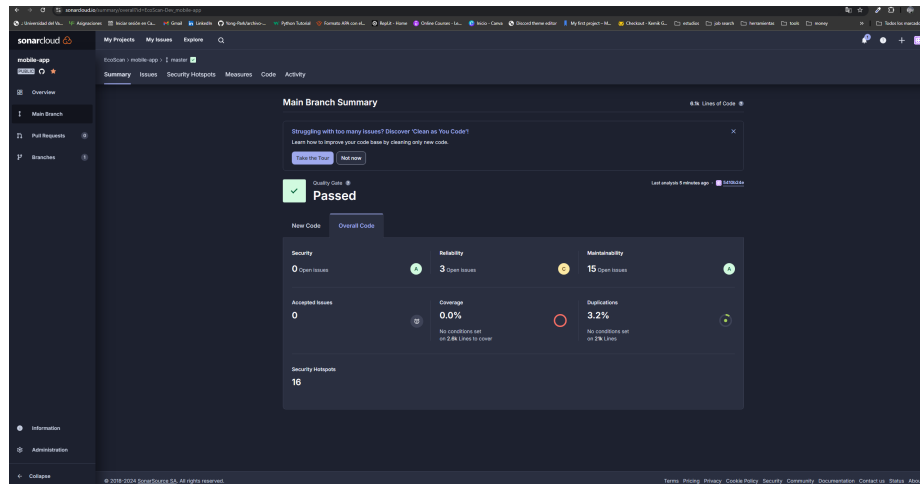


Figura 5.7: Primer escaneo en SonarCloud.

Para evitar que se escanearan los archivos de pruebas unitarias y aparecieran como vulnerabilidades, se configuró *SonarCloud* para excluir estas carpetas del análisis. Además, algunos archivos se excluyeron de la cobertura de código, ya que su lógica no requería cobertura, decisión tomada en consulta con cada especialista del área. Esta configuración específica se realizó dentro de los ajustes de *SonarCloud*, como se muestra en la imagen a continuación:

Code Coverage

Configure the files that should be ignored by code coverage calculations.

Coverage Exclusions

Patterns used to exclude some files from coverage report.

Key: sonar.coverage.exclusions
Default: <no value> (SonarCloud's default)

Reset

Duplications

Configure the files that should be ignored by duplication detection.

Duplication Exclusions

Patterns used to exclude some source files from the duplication detection mechanism. See below to know how to use wildcards to specify this property.

Key: sonar.cpd.exclusions
Default: <no value> (SonarCloud's default)

Files

Configure the files that should be completely ignored by the analysis.

Source File Exclusions

Patterns used to exclude some source files from analysis.

Key: sonar.exclusions
Default: <no value> (SonarCloud's default)

Reset

Figura 5.8: Configuración de exclusiones en SonarCloud.

Además, se creó un *quality check* manual en *SonarCloud* para asegurar el cumplimiento de los estándares definidos para el proyecto. La mayoría de los criterios de calidad proporcionados por *SonarCloud* fueron mantenidos, y se añadieron dos requisitos adicionales: primero, que la cobertura del código completo (no solo del nuevo) supere el 80 %; y segundo, que no existan *security hotspots* en ninguna parte del código, ya que es esencial identificar y resolver estos puntos críticos de seguridad lo más pronto posible.

new quality

You are a few conditions away from Clean as You Code

This quality gate does not have all the conditions needed on new code to follow the Clean as You Code methodology.

Review and update this Quality Gate

Conditions ?

Add Condition

Conditions on New Code

Conditions on New Code apply to all branches and to Pull Requests.

Metric	Operator	Value		
Duplicated Lines (%)	is greater than	3.0%		
Maintainability Rating	is worse than	A		
Reliability Rating	is worse than	A		
Security Hotspots Reviewed	is less than	100%		
Security Rating	is worse than	A		

Conditions on Overall Code

Conditions on Overall Code apply to long-lived branches only.

Metric	Operator	Value		
Coverage	is less than	80.0%		
Security Hotspots Reviewed	is less than	100%		

Projects ?

With

Without

All

Search for projects

☒ mobile-app EcoScan-Dev_mobile-app

Figura 5.9: Configuración de calidad en SonarCloud.

Después de ajustar la configuración, la cobertura del *backend* alcanzó un 82.3 %, mientras que el *frontend* logró un 80.4 %, resultando en una cobertura total del 82 % del proyecto. Este ajuste permitió reflejar de manera más precisa la calidad y el alcance de las pruebas realizadas en el código relevante, excluyendo elementos que no aportan al funcionamiento directo de la aplicación.

	Lines of Code	Security	Reliability	Maintainability	Security Hotspots	Coverage	Duplications
mobile-app							
eco-scan-backend	1,431	0	0	1	0	82.3%	0.0%
eco-scan-frontend	1,025	0	0	7	1	80.4%	0.0%

2 of 2 shown

Figura 5.10: Cobertura del código en SonarCloud.



Figura 5.11: Grafica de cobertura en soanrcloude.

A lo largo de esta fase, se continuaron realizando capacitaciones periódicas para asegurar que el equipo adoptara un enfoque proactivo hacia la mejora continua y la resolución de problemas, garantizando que se mantuviera un ciclo de desarrollo seguro y de alta calidad.

5.4.6. Configuración de CI/CD en SonarCloud

Para implementar el proceso de *CI/CD* en *SonarCloud* junto con *GitHub*, fue necesario configurar el archivo `sonar-project.properties` para definir parámetros específicos de análisis y asegurar la sincronización entre ambas plataformas. En este archivo se configuraron aspectos clave como el *projectKey*,

organization, y las rutas de cobertura. La siguiente imagen muestra la configuración final realizada en el archivo.

```
1 sonar.projectKey=EcoScan-Dev_mobile-app
2 sonar.organization=ecoscan-dev
3
4 # This is the name and version displayed in the SonarCloud UI.
5 sonar.projectName=mobile-app
6 #sonar.projectVersion=1.0
7
8 # Path is relative to the sonar-project.properties file. Replace "\" by "/" on Windows.
9 sonar.sources=.
10
11 # Path to the lcov coverage report
12 sonar.javascript.lcov.reportPaths=eco-scan-backend/test/reports/coverage/lcov.info,eco-scan-frontend/coverage/lcov.info
```

Figura 5.12: Configuración de SonarCloud.

5.4.7. Revisión de Seguridad: Detección y Mitigación de Vulnerabilidades

Como parte del compromiso con la seguridad, se monitorizó constantemente la cantidad de vulnerabilidades detectadas a lo largo del tiempo, tanto en *SonarQube* como en *SonarCloud*. Desde los primeros análisis en *SonarQube* hasta los escaneos automatizados en *SonarCloud*, la cantidad de vulnerabilidades ha disminuido progresivamente, lo cual demuestra la implementación efectiva de las buenas prácticas de seguridad y la gestión de defectos, en línea con el marco *SAMM*.



Figura 5.13: Gráfica de vulnerabilidades detectadas en SonarQube.

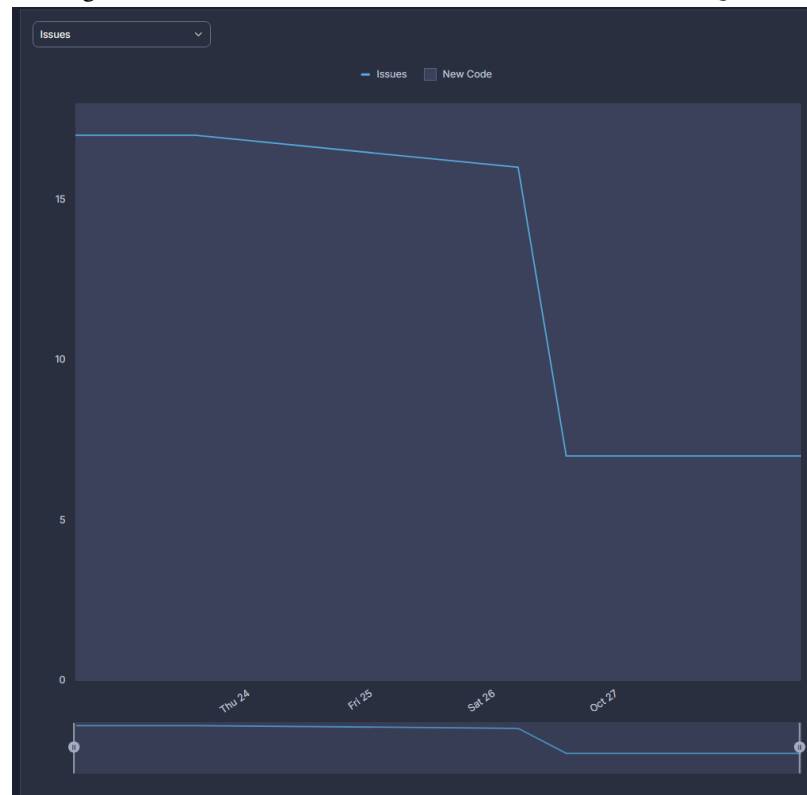


Figura 5.14: Gráfica de vulnerabilidades detectadas en SonarCloud.

Estas gráficas muestran que las vulnerabilidades detectadas en análisis iniciales fueron efectivamente corregidas, y que no reaparecieron los mismos problemas. Esto valida el éxito en la implementación de la gestión de defectos de *SAMM*, asegurando una mejora continua en la seguridad del código a lo largo del proyecto.

5.4.8. Evaluación de Seguridad en Modelos de IA y Chatbot

Se revisaron los modelos de *IA*, identificando algunos falsos positivos en el uso de bibliotecas y funciones. La revisión confirmó que no había problemas críticos, lo cual fue validado mediante un escaneo sin alertas.

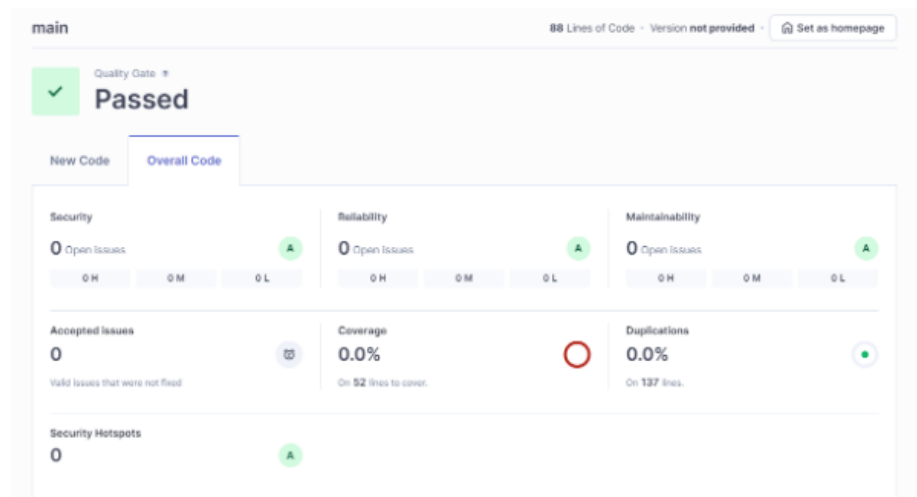


Figura 5.15: Resultado del escaneo del 20 de octubre para el segundo modelo.

Como parte de esta fase, se incluyó el código del modelo de *chatbot* desarrollado por el compañero Pedro Pablo. En el escaneo inicial del código del modelo a utilizar, no se detectaron vulnerabilidades en *SonarCloud*. Debido a que este código corresponde a un modelo de *IA*, no se requiere cobertura de pruebas; sin embargo, se realizó el análisis debido al uso de *endpoints* y *APIs*, cuya correcta implementación era necesaria para asegurar la seguridad y efectividad de la comunicación con servicios externos.

A continuación, se realizó un primer escaneo en *SonarCloud* específicamente para el código del *chatbot*. Durante esta inspección inicial, se identificó un error mínimo junto con una vulnerabilidad de seguridad menor que necesitaba ser atendida.

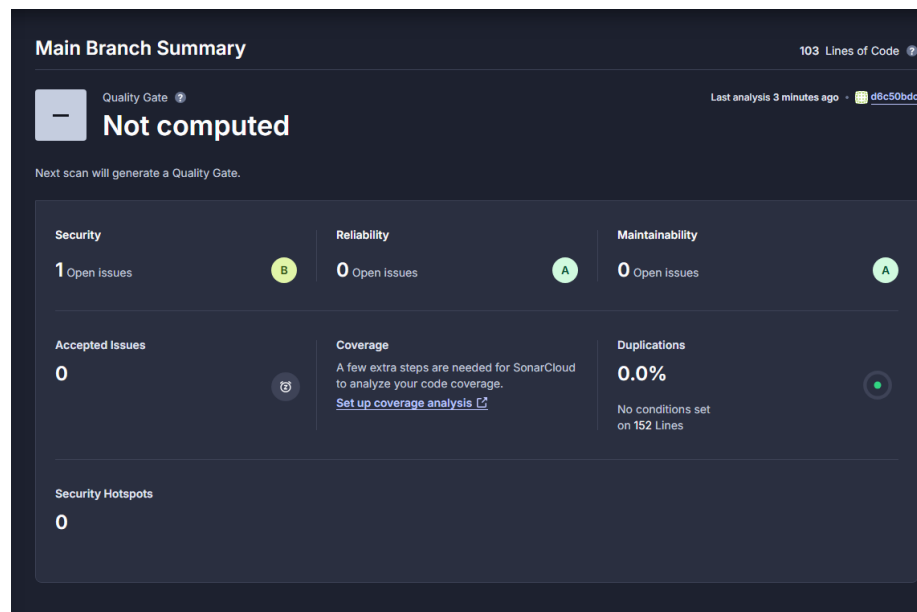


Figura 5.16: Primer escaneo del chatbot en SonarCloud.

Pedro Pablo procedió a realizar los ajustes necesarios en el código del *chatbot* para corregir la vulnerabi-

lidad detectada. Como se observa en el siguiente escaneo, tras los cambios aplicados, el sistema no reportó ninguna vulnerabilidad adicional, asegurando así que el código cumpliera con los estándares de seguridad y calidad establecidos para el proyecto. Dado que este código corresponde a la lógica de un modelo, no se requirió cobertura adicional.

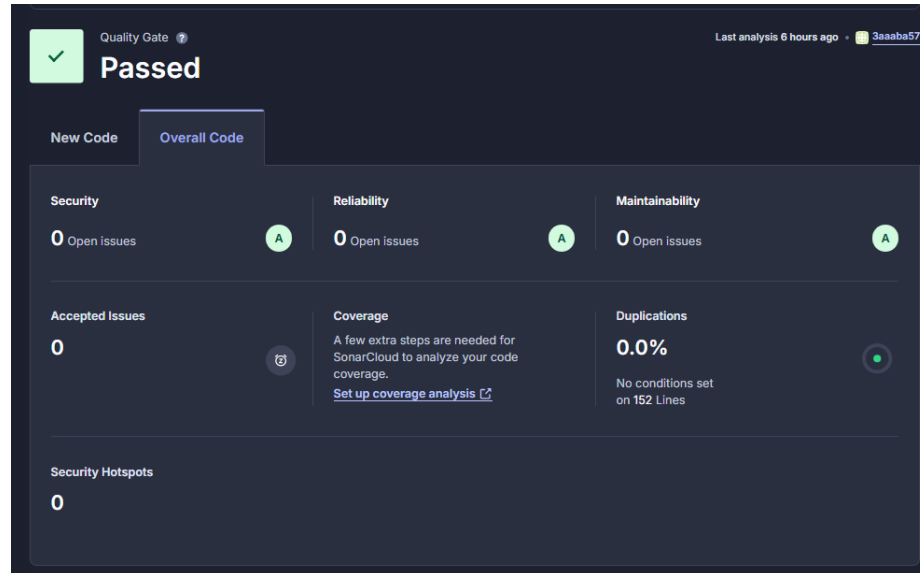


Figura 5.17: Último escaneo del chatbot en SonarCloud.

Con esta última revisión y sin vulnerabilidades reportadas, se dio por completada la fase de verificación de seguridad del código del *chatbot*, finalizando así la contribución del compañero Pedro Pablo en esta parte del proyecto.

5.5. Fase 5: Escaneo y Resultados de la Aplicación Móvil

Con el objetivo de asegurar la seguridad de la aplicación móvil, se realizaron escaneos tanto estáticos como dinámicos utilizando *MobSF* y *AppSweep*, respectivamente. A continuación, se describen los procedimientos y resultados obtenidos en esta fase.

5.5.1. Escaneo Estático con MobSF

Para el análisis estático de la aplicación móvil, se utilizó *Mobile Security Framework (MobSF)*. Inicialmente, se intentó realizar el escaneo del archivo *.aab (Android App Bundle)* mediante la instancia web de *MobSF*. Sin embargo, se encontró un problema recurrente de pérdida de conexión, lo cual interrumpía el proceso de carga y análisis del archivo.

Para solventar esta limitación, se optó por descargar el repositorio de *MobSF* y ejecutarlo localmente. Los pasos seguidos para la configuración local fueron los siguientes:

- `setup.bat`: Este *script* verifica los requisitos previos y, en caso de que falten dependencias, solicita su instalación. Una vez cumplidos los requisitos, procede a instalar todas las librerías necesarias para el funcionamiento de *MobSF*.
- `run.bat`: Una vez que las dependencias están instaladas, este comando inicia *MobSF* para que se ejecute localmente, permitiendo el análisis sin interrupciones de conexión.

En este caso, para ejecutar correctamente `setup.bat`, fue necesario contar con las siguientes herramientas preinstaladas en el sistema: **Git**, **Python 3.8/3.9** y **OpenSSL**. Una vez que `setup.bat` completó la instalación de las librerías necesarias, al ejecutar `run.bat`, *MobSF* se activó localmente, siendo accesible a través de `localhost:8000`. Las credenciales de acceso configuradas de forma predeterminada eran *mobsf* tanto para el usuario como para la contraseña.

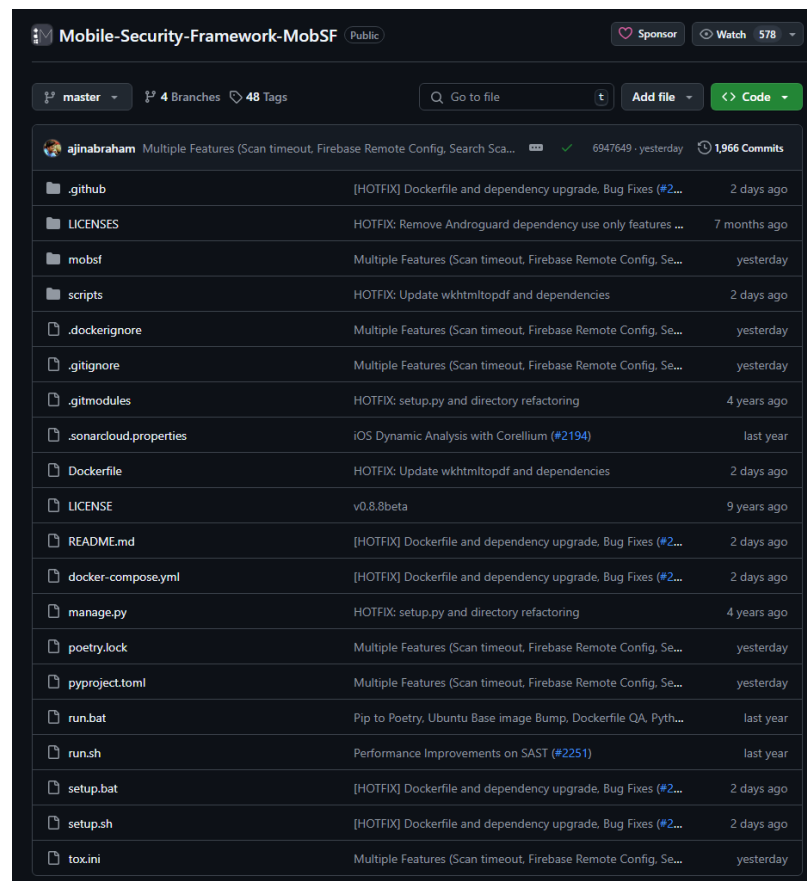


Figura 5.18: Repositorio de MobSF.

```
PS C:\Users\PARK_JONGHYUN\Desktop\Mobile-Security-Framework-MobSF-master> .\run.bat
Running MobSF on "0.0.0.0:8080 [::]:8080"
[INFO] 30/Oct/2024 22:45:30 -
[INFO] 30/Oct/2024 22:45:30 - Author: Ajin Abraham | opensecurity.in
[INFO] 30/Oct/2024 22:45:30 - Mobile Security Framework v4.1.1
REST API Key: 48f6e28bb5b9d5f311e68a6b94d7c0382a9090fd8078391d54d332e53b8796e
Default Credentials: mobsf/mobsf
[INFO] 30/Oct/2024 22:45:30 - OS Environment: Windows Windows-11-10.0.22631-SP0
[INFO] 30/Oct/2024 22:45:30 - MobSF Basic Environment Check
[INFO] 30/Oct/2024 22:45:30 - Checking for Update.
[INFO] 30/Oct/2024 22:45:31 - No updates available.
[WARNING] 30/Oct/2024 22:45:31 - Dynamic Analysis related functions will not work.
Make sure a Genymotion Android VM/Android Studio Emulator is running before performing Dynamic Analysis.
[INFO] 30/Oct/2024 22:45:42 - Scan Hash: 19355342902cc359abf35fda744e108
[INFO] 30/Oct/2024 22:45:42 - Starting Analysis on: application-0f63a7dd-e09e-421a-8737-bcca5b440ee1.aab
[INFO] 30/Oct/2024 22:45:42 - Analysis is already Done. Fetching data from the DB...
¿Deseará terminar el trabajo por lotes (S/N)? s
PS C:\Users\PARK_JONGHYUN\Desktop\Mobile-Security-Framework-MobSF-master>
```

Figura 5.19: MobSF ejecutándose localmente.

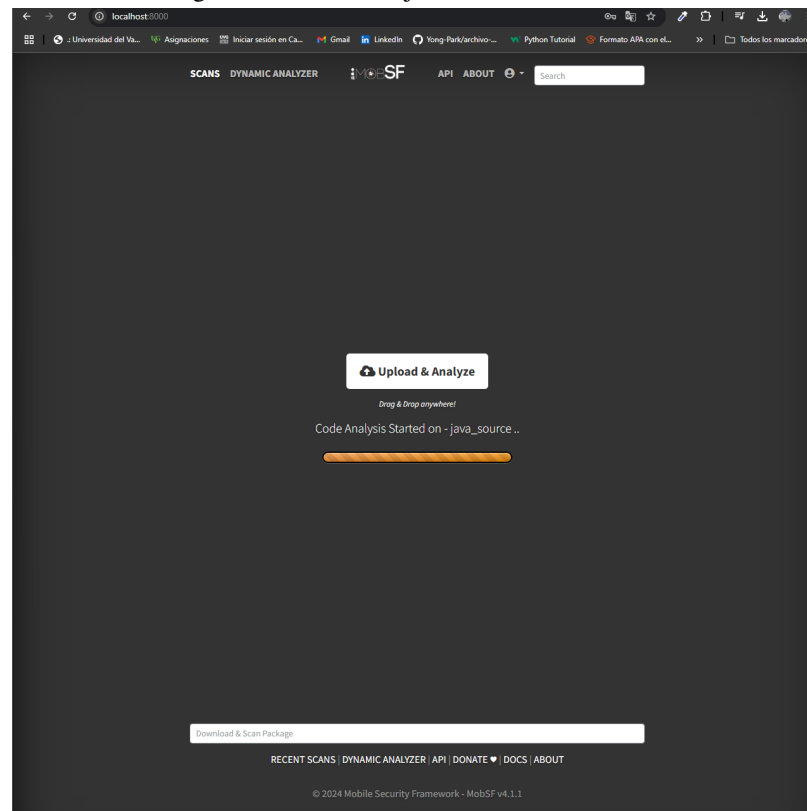


Figura 5.20: Interfaz gráfica de MobSF

Después de la configuración y ejecución de *MobSF* en un entorno local, se procedió a cargar el archivo .aab de la aplicación móvil para su análisis. Los resultados iniciales del escaneo estático revelaron diversos aspectos de seguridad en el código de la aplicación, que fueron posteriormente evaluados y optimizados según las recomendaciones de *MobSF*.

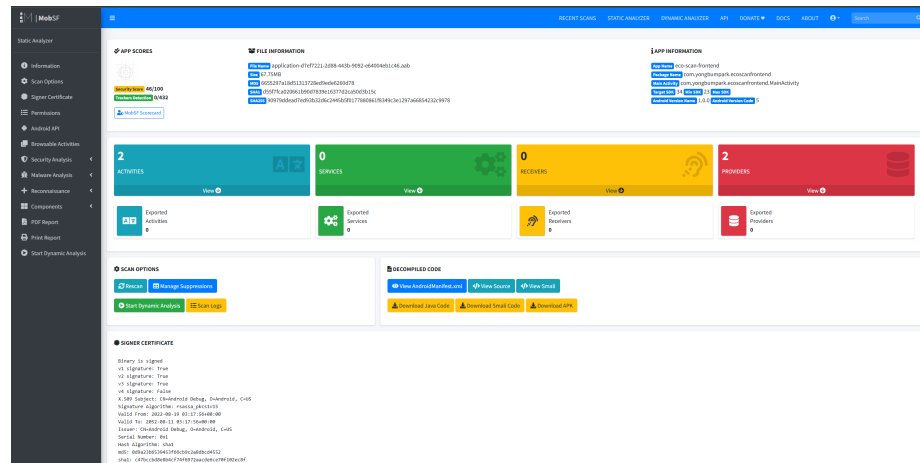


Figura 5.21: Resultado del escaneo inicial en MobSF.

5.5.2. Escaneo Dinámico con AppSweep

Para el análisis dinámico de la aplicación móvil, se seleccionó **AppSweep** debido a su enfoque específico en la detección de vulnerabilidades en aplicaciones móviles en tiempo de ejecución. Este escaneo permitió identificar problemas relacionados con permisos, configuraciones deficientes y almacenamiento inseguro.

El análisis *DAST* en **AppSweep** fue exitoso y se ejecutó sin problemas, validando aspectos críticos de seguridad durante la ejecución de la aplicación móvil.

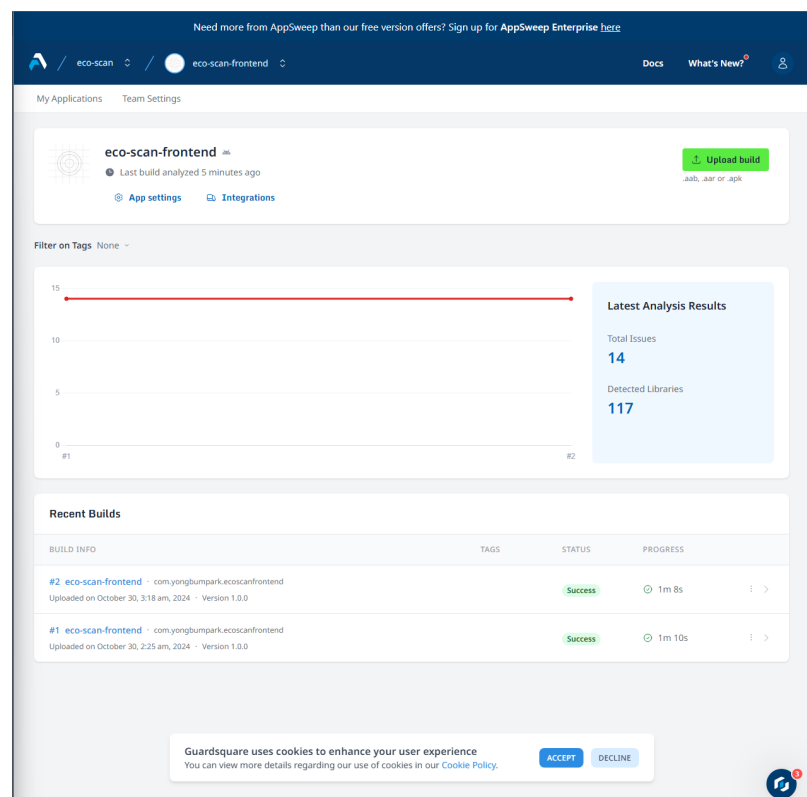


Figura 5.22: Resultado del escaneo en AppSweep.

5.6. Fase 6: Revisión y Optimización de Permisos en AWS IAM Identity Center

Como parte de las medidas de seguridad y buenas prácticas en la gestión de acceso, se llevó a cabo una revisión de los permisos configurados en el *AWS IAM Identity Center*. Este proceso fue esencial para asegurar que los accesos a la infraestructura de AWS estuvieran debidamente controlados y alineados con los principios de privilegios mínimos.

5.6.1. Revisión de Grupos y Permisos

La revisión comenzó con una inspección detallada de los grupos predeterminados generados por AWS en el *IAM Identity Center*. Estos grupos contienen configuraciones estándar proporcionadas por AWS, las cuales fueron evaluadas para determinar su adecuación y relevancia en el contexto del proyecto.

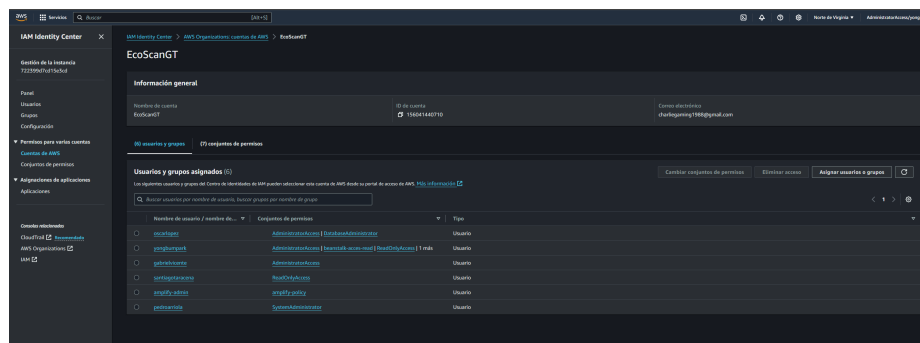


Figura 5.23: AWS IAM Identity Center: Cuentas y Grupos Predeterminados.

Cada grupo de permisos fue revisado cuidadosamente, y se documentaron las asignaciones de permisos para cada usuario. Este análisis permitió identificar permisos que podrían no ser necesarios para el rol asignado, lo cual podría representar un riesgo potencial de seguridad si no se gestionaba adecuadamente.

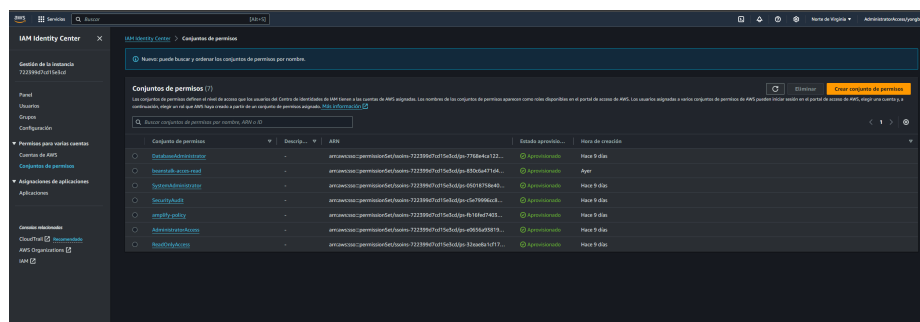


Figura 5.24: AWS IAM Identity Center: Permisos de Usuario Revisados.

5.6.2. Proceso de Validación y Ajuste de Permisos

Durante la revisión, cuando se identificaba un permiso que no era crítico o que representaba un posible exceso de privilegios, se realizaba una consulta directa con el encargado de la infraestructura, Gabriel. Las decisiones sobre cada permiso se tomaban en colaboración, considerando el impacto en las operaciones y la seguridad general del sistema. Esto incluyó las siguientes acciones:

- **Eliminación de Permisos No Necesarios:** En casos donde se determinó que ciertos permisos eran superfluos, se procedió a su eliminación para fortalecer la seguridad.
- **Restricción de Permisos Excesivos:** Algunos permisos fueron restringidos aún más mediante la aplicación de políticas de acceso más limitadas, alineadas con el principio de privilegios mínimos.
- **Mantenimiento de Permisos Esenciales:** Los permisos críticos para la operación se mantuvieron, pero se documentaron adecuadamente para futuras auditorías y revisiones.

Este proceso contribuyó a la seguridad de la infraestructura al garantizar que solo se mantuvieran los permisos absolutamente necesarios y que cualquier exceso de privilegios fuera gestionado de manera proactiva. La colaboración continua con Gabriel aseguró que las decisiones fueran bien informadas y equilibraran la seguridad con la funcionalidad requerida.

6.1. Educación y Orientación

Durante el proyecto, se logró un avance significativo en el área de Educación y Orientación, pasando del Nivel 1 al Nivel 2 en el marco de *OWASP SAMM*. Al inicio, se implementaron prácticas básicas de sensibilización en seguridad, y mi rol como Campeón de Seguridad fue fundamental para alcanzar este progreso.

Como parte de mis responsabilidades, apoyé al equipo en la adopción de prácticas seguras en el ciclo de vida del software. Esto incluyó la creación de guías específicas adaptadas a los roles de cada miembro y a las tecnologías empleadas. El avance al Nivel 2 implicó la educación de todo el personal mediante materiales y guías personalizadas, según sus funciones y las tecnologías utilizadas. A lo largo de este proceso, los miembros del equipo me consultaban sobre diversas tecnologías y buenas prácticas en el desarrollo seguro. Para asegurar respuestas adecuadas, investigué las opciones disponibles y recomendé las mejores soluciones con argumentos basados en seguridad y eficiencia.

Este enfoque de Educación y Orientación demostró su efectividad mediante los resultados de las pruebas de *phishing* y el cumplimiento del estándar de cobertura de código. Ambas actividades reflejan la importancia de la gobernanza y el establecimiento de prácticas seguras, conforme a las directrices de *OWASP SAMM*.

6.1.1. Resultados en la Prueba de Phishing

Como parte de las actividades de capacitación, se llevaron a cabo pruebas de *phishing* para evaluar la capacidad del equipo en la identificación de correos maliciosos. La primera prueba mostró un promedio de 4.5 respuestas correctas, lo que indicó la necesidad de mejorar en esta área. Tras una sesión de formación enfocada en la identificación de *phishing*, se realizó una segunda prueba el 14 de septiembre, en la cual se observó una mejora significativa, alcanzando un promedio de 8.25 respuestas correctas. Finalmente se

realizo una prueba sorpresa para asegurar que el equipo recordara los consejos y técnicas impartidas del cuál se observo un promedio de 9 respuestas correctas.

Nombre	Primer Test de Phishing	Segundo Test de Phishing	Último Test de Phishing
Pedro	4	9	8
Oscar	4	8	10
Santiago	5	9	9
Gabriel	4	7	9

Tabla 6.1: Resultados de los tests de phishing.

La mejora en los resultados de las pruebas de *phishing* evidenció el impacto positivo de las actividades de educación y orientación en la capacidad del equipo para identificar amenazas, reduciendo así el riesgo de caídas en ataques de *phishing*. Este logro contribuye al cumplimiento del área de Gobernanza en Educación y Orientación de *OWASP SAMM*.

6.2. Calidad del Código y Mejora Continua

Además de la capacitación en seguridad, se realizaron mejoras significativas en la calidad del código y en las prácticas de desarrollo. A medida que se proporcionaban recomendaciones sobre mejores prácticas y mantenibilidad del código, utilizando herramientas como *SonarQube* para el análisis de calidad, el equipo adoptó consistentemente técnicas para reducir la deuda técnica.

La reducción de vulnerabilidades detectadas a lo largo del proyecto muestra el cumplimiento en la Gestión de Defectos de la categoría de Implementación de *OWASP SAMM*. Esta práctica implicó que el equipo no solo resolviera vulnerabilidades detectadas sino que implementara mejoras preventivas para evitar futuros problemas de seguridad. Estas mejoras incluyeron la implementación de pruebas unitarias para asegurar que el código funcionara según lo esperado, lo cual no solo incrementó la cobertura de pruebas, sino que también permitió identificar y resolver problemas de manera proactiva.

6.2.1. Resultados en la Gráfica de Vulnerabilidades

La progresión de vulnerabilidades detectadas en *SonarQube* y *SonarCloud* mostró una disminución significativa en los problemas de seguridad, evidenciando el enfoque proactivo del equipo en la resolución de problemas y en la adopción de mejores prácticas de codificación segura. La siguiente gráfica muestra la reducción de vulnerabilidades a lo largo del proyecto.



Figura 6.1: Gráfica de vulnerabilidades detectadas en SonarQube.

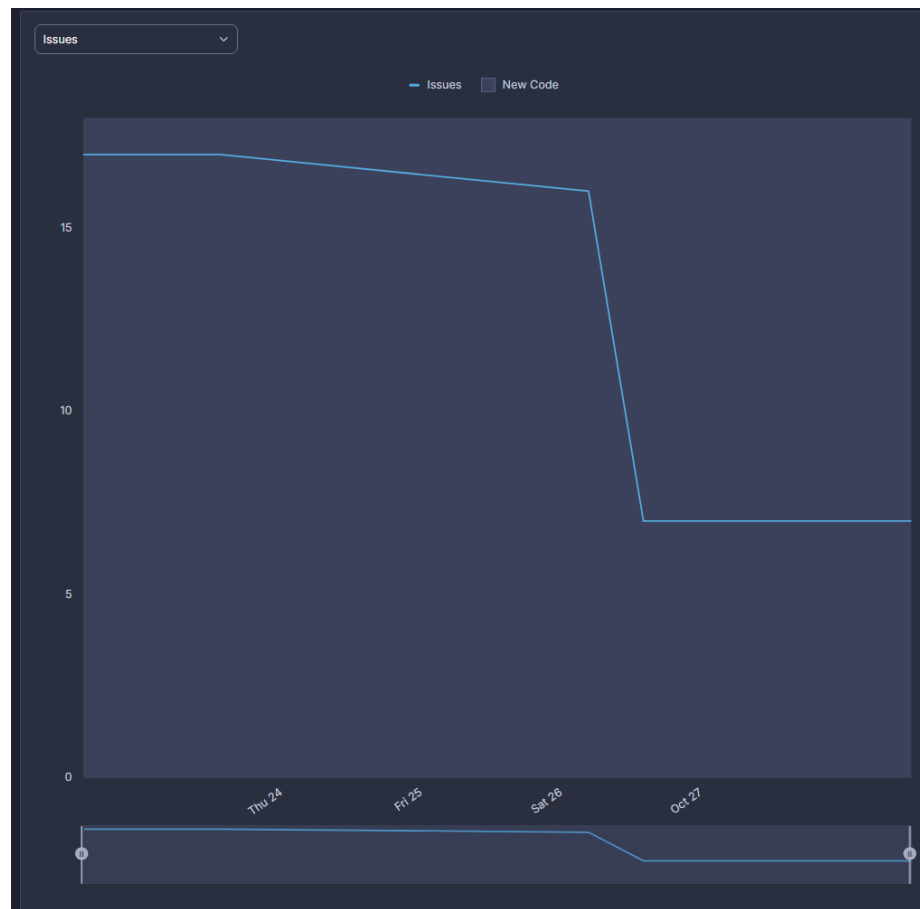


Figura 6.2: Gráfica de vulnerabilidades detectadas en SonarCloud.

La disminución en la cantidad de vulnerabilidades evidencia el éxito en la gestión de defectos, cumpliendo con los principios de *OWASP SAMM* en la categoría de Implementación.

6.2.2. Cobertura de Código

El proyecto logró superar el 80 % de cobertura de código en *SonarCloud*, cumpliendo con los estándares de la herramienta y con los objetivos planteados en el área de Verificación de *OWASP SAMM*. La cobertura del *backend* alcanzó un 82.3 % mientras que el *frontend* logró un 80.4 %, logrando una cobertura general de 82 % en el proyecto.

	Lines of Code	Security	Reliability	Maintainability	Security Hotspots	Coverage	Duplications
mobile-app							
eco-scan-backend	1,431	0	0	1	0	82.3%	0.0%
eco-scan-frontend	1,025	0	0	7	1	80.4%	0.0%

2 of 2 shown

Figura 6.3: Cobertura del código en SonarCloud.



Figura 6.4: Gráfica de cobertura en SonarCloud.

Este logro no solo muestra la adopción de buenas prácticas de desarrollo seguro, sino que también cumple con el área de Pruebas de Seguridad en Verificación de *OWASP SAMM*, garantizando una adecuada validación y cobertura en el proyecto.

6.2.3. Code Smells

Los *Code Smells* son problemas de diseño o prácticas de programación que pueden afectar la calidad del código pero no necesariamente son errores de funcionamiento. A lo largo del proyecto, se realizó un monitoreo constante de estos *Code Smells*, lo cual permitió al equipo mejorar la estructura y diseño del código para mantenerlo limpio y eficiente.

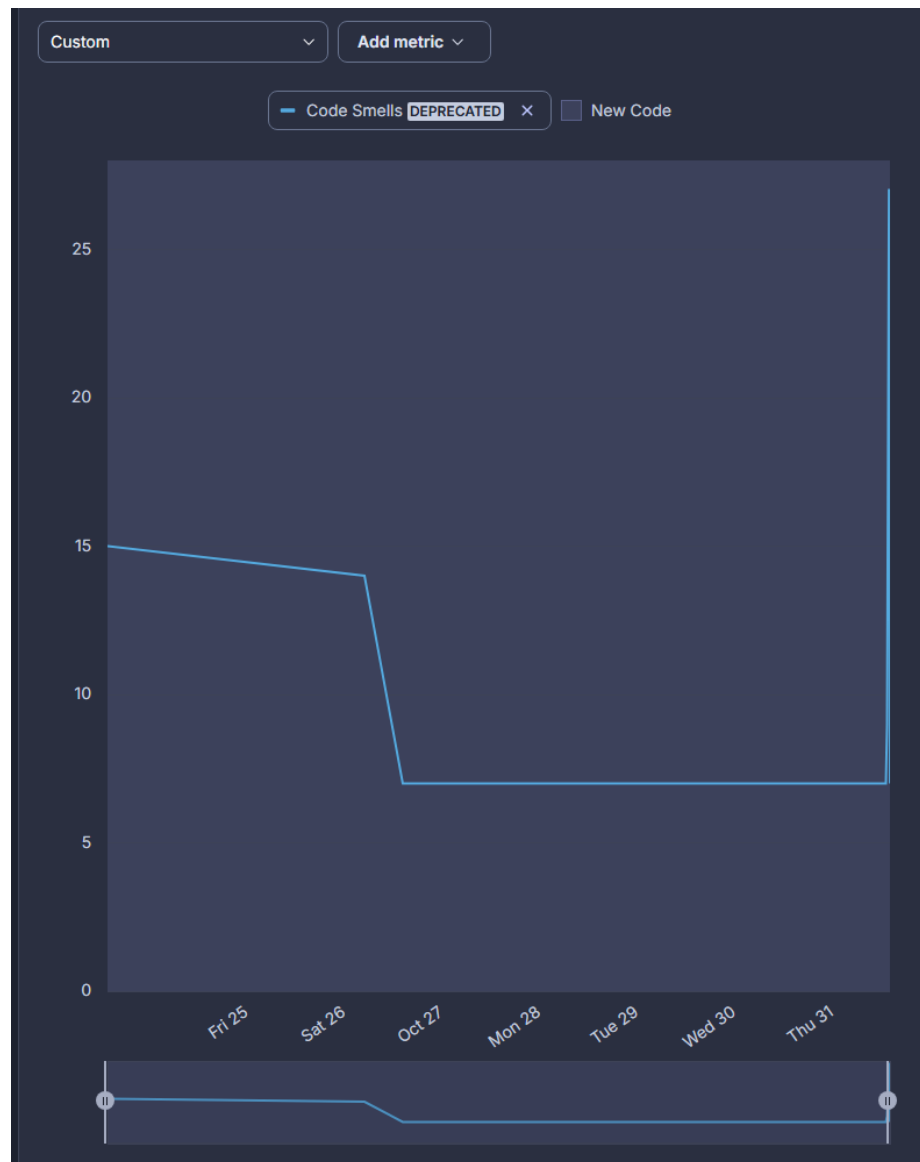


Figura 6.5: Gráfica de code smell en SonarCloud.

6.2.4. Código Duplicado

El código duplicado mide el porcentaje de líneas de código que están repetidas en la base de código. Durante el análisis, se identificaron y redujeron las duplicaciones, lo cual contribuyó a una mejor mantenibilidad y eficiencia del proyecto. Este proceso fue fundamental para optimizar el rendimiento y la claridad del código.

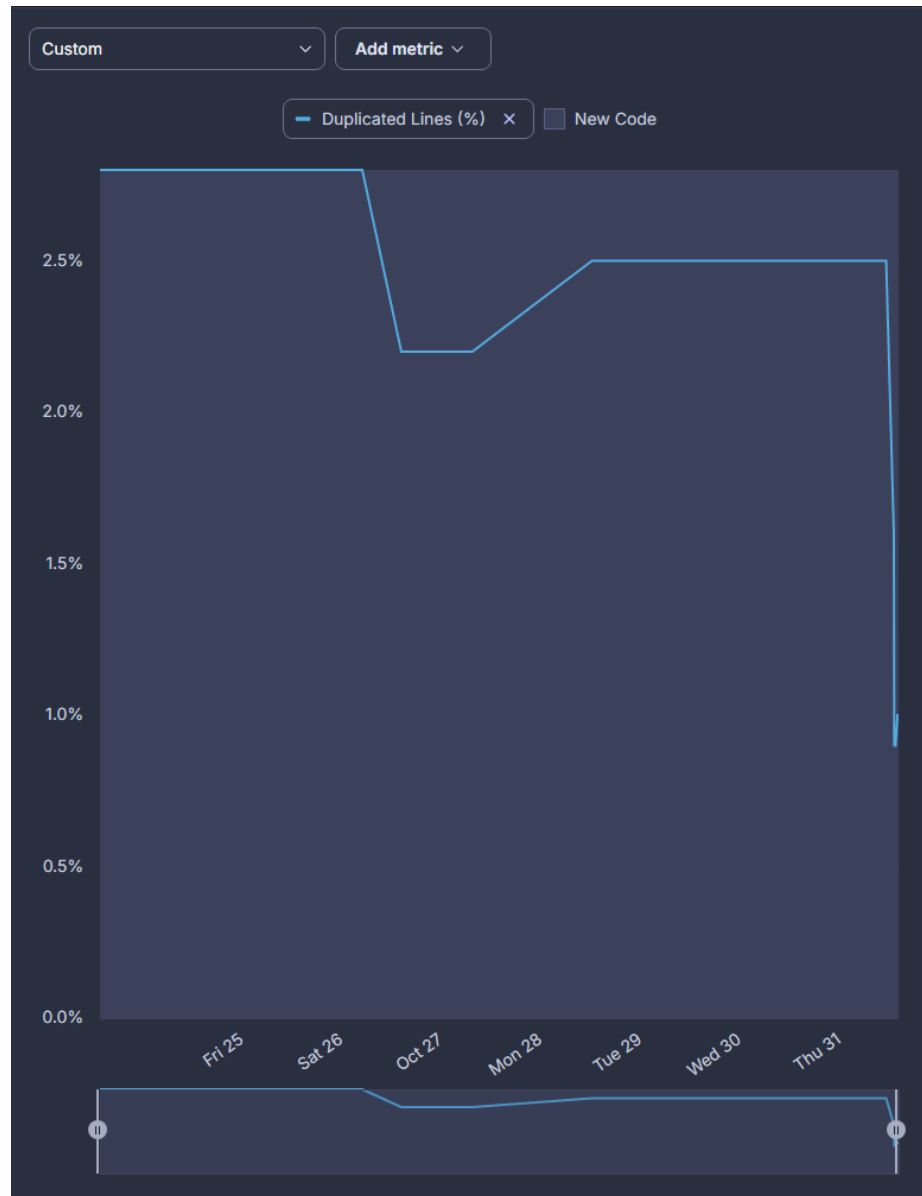


Figura 6.6: Gráfica de líneas de código duplicadas en SonarCloud.

6.2.5. Complejidad Ciclomática

La complejidad ciclomática mide la complejidad de los métodos, clases o módulos del proyecto, calculando el número de caminos distintos que se pueden recorrer en el código. Mantener la complejidad bajo control es esencial para asegurar que el código sea fácil de entender, probar y mantener. Se realizaron esfuerzos para simplificar los métodos y módulos más complejos.

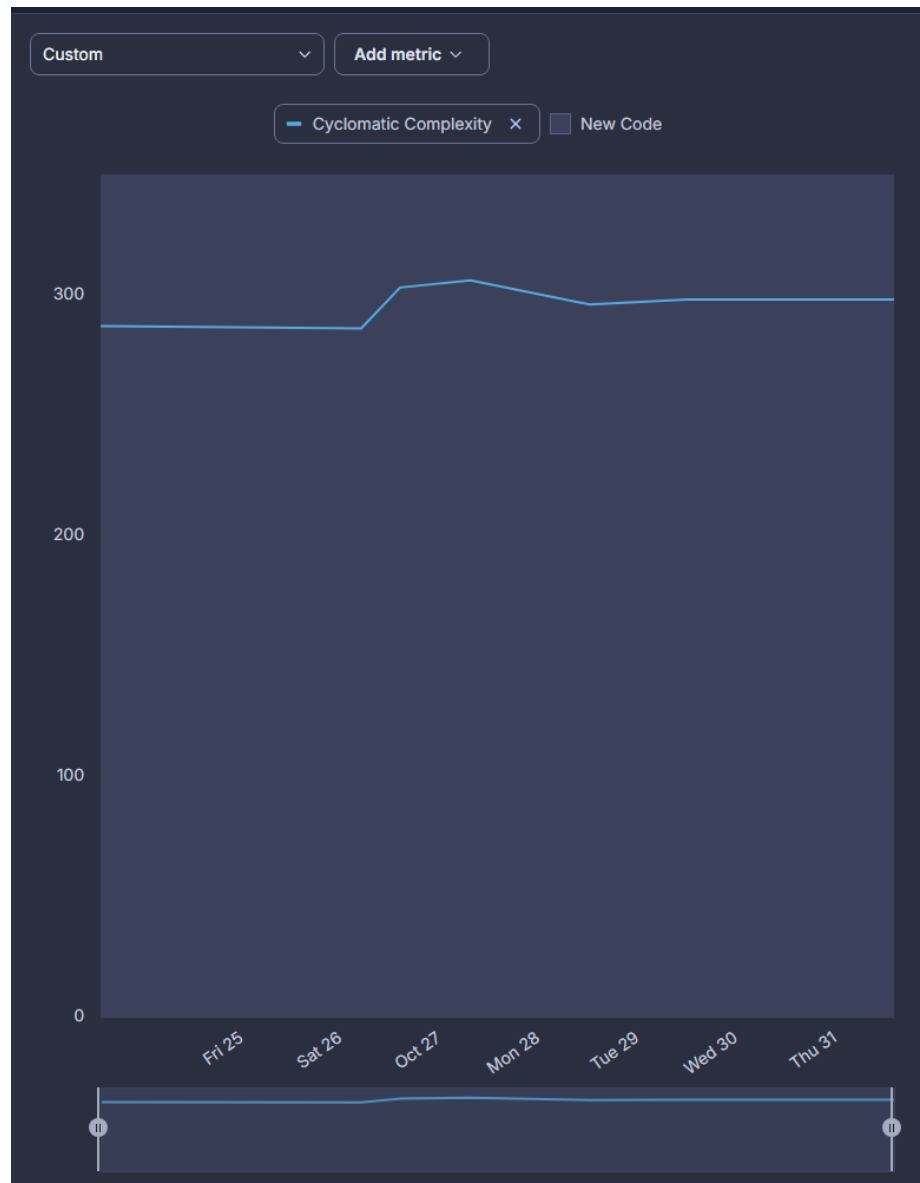


Figura 6.7: Gráfica de complejidad en SonarCloud.

6.2.6. Densidad de Comentarios

La densidad de comentarios mide el porcentaje de comentarios en el código. Tener un equilibrio en esta métrica es importante: demasiados comentarios pueden ser un signo de código complicado, mientras que muy pocos comentarios pueden indicar una falta de documentación. Se trabajó en mantener este equilibrio, asegurando que el código fuera comprensible sin necesidad de comentarios excesivos, pero con la documentación necesaria para la claridad.

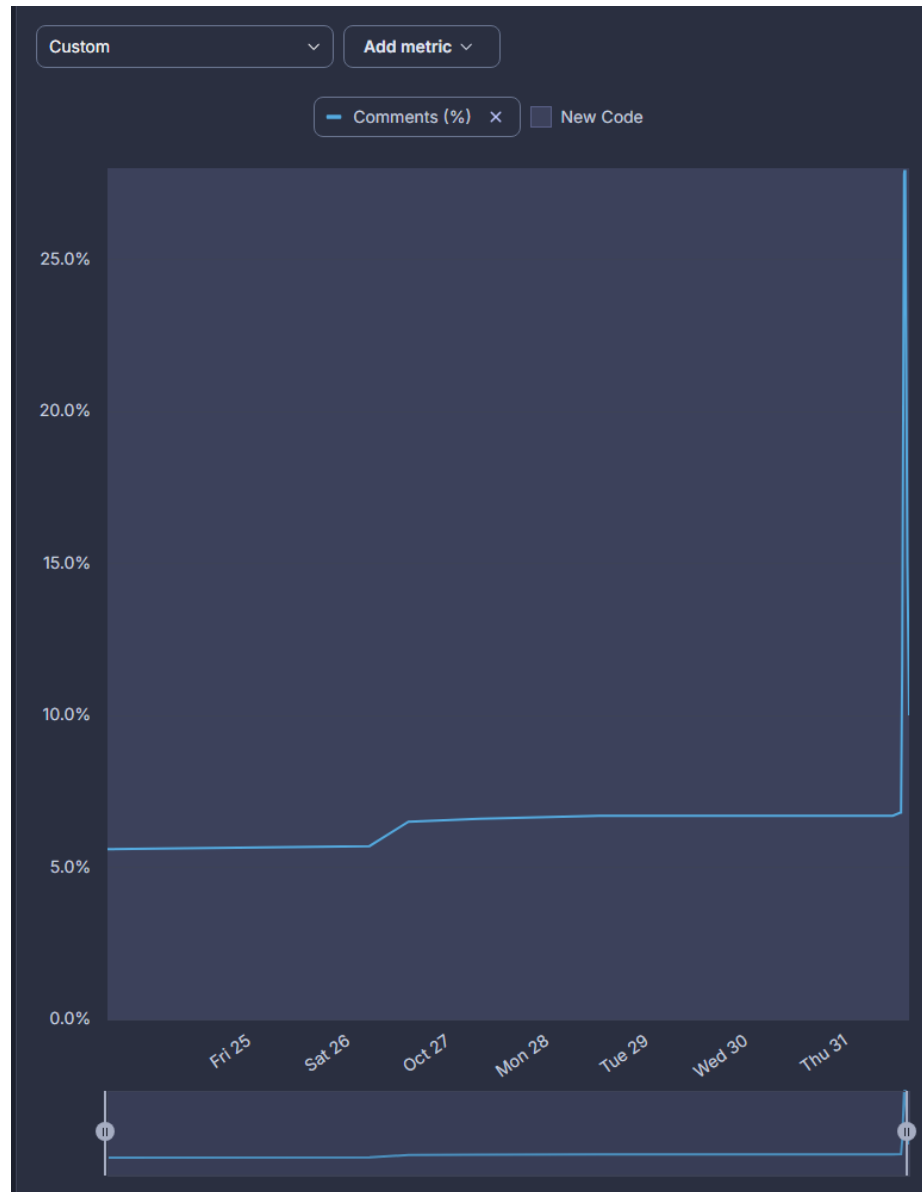


Figura 6.8: Gráfica de densidad de comentarios en SonarCloud.

6.3. Escaneo de Vulnerabilidades en Aplicación Móvil

Se logró realizar el escaneo de la aplicación móvil utilizando las herramientas *MobSF* y *AppSweep*, permitiendo tanto el análisis estático como el dinámico de la aplicación. Este proceso de escaneo reveló vulnerabilidades adicionales en el código móvil que no habían sido identificadas previamente en otras fases del proyecto.

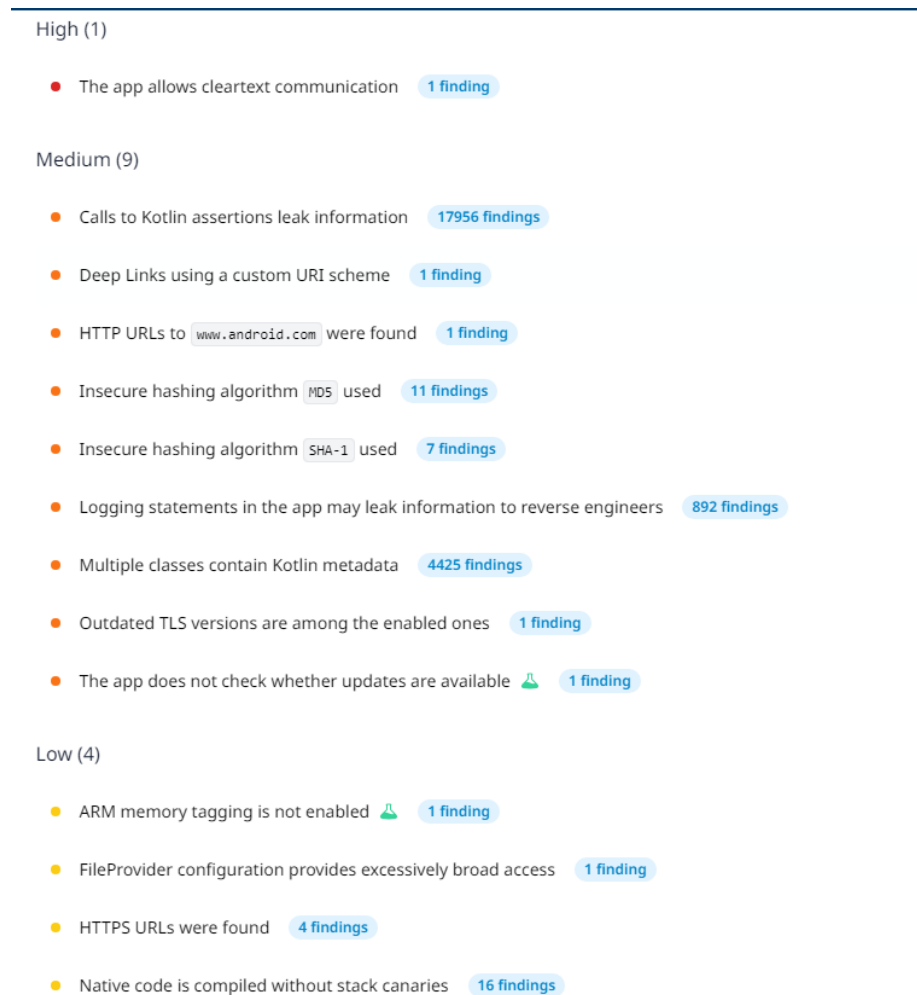


Figura 6.9: Vulnerabilidades detectadas en AppSweep.

Estas nuevas vulnerabilidades detectadas abren el camino para una mejora continua del código y proporcionan una perspectiva valiosa sobre otras áreas de vulnerabilidades potenciales en el desarrollo de aplicaciones móviles. Este análisis contribuye no solo al cumplimiento de estándares de seguridad, sino también a un aprendizaje continuo, apoyando la implementación de buenas prácticas en código y configuraciones futuras que refuercen la protección y fiabilidad de la aplicación móvil.

6.4. Análisis del Código de los Modelos de IA

En el trabajo realizado por el encargado de los modelos de inteligencia artificial, Pedro Pablo, se observó un manejo adecuado y consistente en la implementación del código. A lo largo de todo el proyecto, Pedro Pablo siguió rigurosamente las buenas prácticas de desarrollo seguro, lo que incluyó la modularidad del código, el uso de comentarios claros y la implementación de técnicas de evaluación continua del modelo.

6.4.1. Análisis de Seguridad y Cumplimiento de Estándares

Los análisis de seguridad realizados con *SonarQube*, tanto al inicio como al final del proyecto, no detectaron problemas ni vulnerabilidades en el código. Este resultado refleja un cumplimiento constante de los estándares de calidad y seguridad, atribuible al enfoque proactivo de Pedro Pablo en la implementación de técnicas de monitoreo, tales como gráficos para la precisión y la pérdida del entrenamiento, y la validación del rendimiento mediante una matriz de confusión y un reporte de clasificación.

6.4.2. Prácticas de Seguridad en el Chatbot

Las prácticas seguidas en el código del *chatbot* también evidenciaron un enfoque integral en la seguridad, con una gestión adecuada del registro de *logs* y el manejo seguro de excepciones. Estas medidas contribuyeron a evitar vulnerabilidades comunes, y la política de limitar el historial del chat y el uso seguro de las claves de la *API* reforzaron la protección del sistema, logrando un entorno seguro y confiable.

6.5. Análisis de Permisos en AWS

Durante el desarrollo del proyecto, se realizó un análisis de los permisos otorgados a cada miembro del equipo en *AWS* para garantizar la aplicación del principio de mínimo privilegio. Este análisis se llevó a cabo en conjunto con el encargado de la infraestructura, Gabriel, y se identificaron varios permisos que podrían ser ajustados para mejorar la seguridad general.

6.5.1. Oscar López: Permisos de *DatabaseAdministrator*

Oscar López, encargado del *backend* del proyecto, fue asignado al rol de *DatabaseAdministrator*, el cual contiene permisos específicos para la gestión de bases de datos en *AWS*. Sin embargo, se detectaron algunos permisos que no eran necesarios para sus tareas, como:

- `rds:*` – Permisos relacionados con *Amazon RDS* que podrían ser reducidos para limitar el acceso.
- `s3:CreateBucket` – Permiso innecesario para la creación de *buckets* en *S3*.
- `iam` – Permiso crítico que no debería ser accesible sin restricciones adicionales.

Después de analizar estos permisos, se discutió con Gabriel Vicente, encargado de la infraestructura sobre la necesidad de restringir o eliminar estas capacidades, ajustando el rol de Oscar para cumplir con las mejores prácticas de seguridad.

IAM > Políticas > DatabaseAdministrator

DatabaseAdministrator

Información

Grants full access permissions to AWS services and actions required to set up and configure AWS database services.

Detalles de la política

Tipo

Administrada por AWS: función de trabajo

Hora de creación

November 10, 2016, 11:25 (UTC-06:00)

Permisos

Entidades asociadas

Versiones de la política

Last Accessed

Permisos definidos en esta política

Información

Los permisos definidos en este documento de política especifican qué acciones se permiten o deniegan. Para definir permisos para una identidad de IAM (usuario, grupo de usuarios o rol), asíciela a una política

Buscar

Permitir (13 de 423 servicios)

Servicio	Nivel de acceso	Recurso	Condición de solicitud
CloudWatch	Completo: Enumerar Limitado: Leer, Escribir	Todos los recursos	None
CloudWatch Logs	Limitado: Enumerar, Leer, Escribir	Todos los recursos	None
Data Pipeline	Limitado: Enumerar, Leer, Escribir	Todos los recursos	None
DynamoDB	Acceso completo	Todos los recursos	None
EC2	Limitado: Enumerar	Todos los recursos	None
ElastiCache	Acceso completo	Todos los recursos	None
IAM	Limitado: Enumerar, Leer, Escribir	Multiple	None
KMS	Limitado: Enumerar	Todos los recursos	None
Lambda	Limitado: Enumerar, Leer, Escribir	Todos los recursos	None
RDS	Acceso completo	Todos los recursos	None
Redshift	Acceso completo	Todos los recursos	None
S3	Completo: Enumerar Limitado: Administración de permisos, Leer, Escribir, Etiquetado	Todos los recursos	None
SNS	Completo: Enumerar Limitado: Administración de permisos, Leer, Escribir	Todos los recursos	None

Figura 6.10: Permisos del rol DatabaseAdministrator asignados a Oscar López.

6.5.2. Pedro Pablo: Permisos de SystemAdministrator

Pedro Pablo, responsable de la implementación de los modelos de inteligencia artificial, fue asignado al rol de *SystemAdministrator*. Durante el análisis, se encontraron permisos adicionales que no eran necesarios para sus responsabilidades, como:

- `trustadvisor:*` – Permisos para *AWS Trusted Advisor* que no son relevantes para su función.
- `config:*` – Permisos relacionados con AWS Config que tampoco son necesarios.
- `iam` – Permiso que podría ser removido para fortalecer la seguridad.

Se discutieron estos hallazgos con Gabriel, y se acordó realizar las modificaciones pertinentes para asegurar que Pedro Pablo tuviera únicamente los permisos imprescindibles para su trabajo.

SystemAdministrator

Información

Grants full access permissions necessary for resources required for application and development operations.

Detalles de la política

Tipo

Administrada por AWS: función de trabajo

Hora de creación

November 10, 2016, 11:23 (UTC-06:00)

Permisos

Entidades asociadas

Versiones de la política

(5)

Last Accessed

Permisos definidos en esta política

Información

Los permisos definidos en este documento de política especifican qué acciones se permiten o deniegan. Para definir permisos para una identidad de IAM (usuario, grupo de usuarios o rol), asíciela a una política

Q

Buscar

Permitir (29 de 423 servicios)

Servicio	Nivel de acceso	Recurso	Condición de solicitud
Certificate Manager	Completo: Enumerar Limitado: Leer, Escribir	Todos los recursos	None
CloudTrail	Limitado: Leer, Escribir	Todos los recursos	None
CloudWatch	Acceso completo	Todos los recursos	None
CloudWatch Logs	Acceso completo	Todos los recursos	None
CodeCommit	Completo: Enumerar Limitado: Leer, Escribir	Todos los recursos	None
CodeDeploy	Acceso completo	Todos los recursos	None
CodePipeline	Acceso completo	Todos los recursos	None
Config	Acceso completo	Todos los recursos	None
Directory Service	Acceso completo	Todos los recursos	None
EC2	Completo: Etiquetado Limitado: Enumerar, Administración de permisos, Leer, Escribir	Todos los recursos	None
EC2 Auto Scaling	Acceso completo	Todos los recursos	None
ELB	Acceso completo	Todos los recursos	None
ELB v2	Acceso completo	Todos los recursos	None
EventBridge	Acceso completo	Todos los recursos	None
IAM	Limitado: Enumerar, Leer, Escribir	Multiple	None
Kinesis	Limitado: Enumerar, Escribir	Todos los recursos	None
KMS	Completo: Enumerar, Leer Limitado: Escribir	Todos los recursos	None
Lambda	Completo: Enumerar, Leer Limitado: Escribir	Todos los recursos	None
Pinpoint Email	Acceso completo	Todos los recursos	None
RDS	Completo: Enumerar Limitado: Leer	Todos los recursos	None

Figura 6.11: Permisos del rol SystemAdministrator asignados a Pedro Pablo.

6.5.3. Santiago Taracena y Yongbum Park: Permisos de ReadOnlyAccess

Santiago Taracena y yo (Yongbum Park) fuimos asignados al rol de *ReadOnlyAccess*, el cual es adecuado para nuestras funciones, ya que no requerimos realizar cambios en el sistema. Santiago no necesita subir nada directamente, ya que todo se gestionará a través de *GitHub Actions* para su parte del código, por lo que este rol es suficiente para sus tareas de configuración y observación.

ReadOnlyAccess Información
Provides read-only access to AWS services and resources.

Detalles de la política

Tipo Administrada por AWS: función de trabajo	Hora de creación February 06, 2015, 12:39 (UTC-06:00)
--	--

Permisos | Entidades asociadas | Versiones de la política (12/1) | Last Accessed

Permisos definidos en esta política información
Los permisos definidos en este documento de política especifican qué acciones se permiten o deniegan. Para definir permisos para una identidad de IAM (usuario, grupo de usuarios o rol), asíciela a una política

Servicios no reconocidos

Servicio	Nivel de acceso	Recurso	Condición de solicitud
iotroborunner ⚠	-	-	-

Permitir (297 de 423 servicios)

Servicio	Nivel de acceso	Recurso	Condición de solicitud
Access Analyzer	Completo: Enumerar Limitado: Leer	Todos los recursos	None
Account	Completo: Enumerar, Leer	Todos los recursos	None
Alexa for Business	Completo: Enumerar Limitado: Leer	Todos los recursos	None
AMP	Completo: Enumerar, Leer	Todos los recursos	None
Amplify	Limitado: Enumerar, Leer	Todos los recursos	None
API Gateway	Completo: Leer	Todos los recursos	None
API Gateway V2	Completo: Leer	Todos los recursos	None
App Mesh	Completo: Enumerar Limitado: Leer	Todos los recursos	None
App Runner	Completo: Enumerar Limitado: Leer	Todos los recursos	None
App Studio	Completo: Leer	Todos los recursos	None
AppConfig	Limitado: Enumerar, Leer	Todos los recursos	None
AppFabric	Completo: Enumerar, Leer	Todos los recursos	None
AppFlow	Completo: Enumerar, Leer	Todos los recursos	None
Application Auto Scaling	Completo: Leer	Todos los recursos	None
Application Discovery	Completo: Enumerar, Leer	Todos los recursos	None
Application Signals	Completo: Leer Limitado: Enumerar	Todos los recursos	None
AppStream 2.0	Completo: Enumerar, Leer	Todos los recursos	None
AppSync	Completo: Enumerar Limitado: Leer	Todos los recursos	None
Artifact	Completo: Enumerar Limitado: Leer	Todos los recursos	None
Athena	Completo: Enumerar, Leer	Todos los recursos	None

Figura 6.12: Permisos del rol ReadOnlyAccess asignados a Santiago Taracena y Yongbum Park.

A lo largo del proyecto, los resultados obtenidos evidencian un progreso significativo en la implementación de seguridad, desde la capacitación del equipo hasta la reducción de vulnerabilidades en el código y la cobertura de pruebas alcanzada. Este análisis reflexiona sobre los logros y desafíos del proyecto, abordando cómo cada aspecto se alineó con los objetivos propuestos y cómo los resultados obtenidos fortalecieron la postura de seguridad general de la aplicación.

7.1. Cultura de Seguridad y Capacitación en el Equipo

Uno de los primeros logros clave fue el desarrollo de una cultura de seguridad entre los miembros del equipo, evidenciada en las pruebas de *phishing*. Inicialmente, los resultados mostraron un bajo desempeño, con un promedio de 4.5 respuestas correctas, lo cual indicó una falta de experiencia en la identificación de amenazas comunes, como correos de *phishing*. Tras implementar sesiones de capacitación específicas, el equipo logró una mejora notable, alcanzando un promedio final de 8.25 respuestas correctas en la segunda prueba y 9 en la prueba final de octubre, realizada sin previo aviso para evaluar el aprendizaje sostenido.

Además de las sesiones de *phishing*, se establecieron reuniones regulares para profundizar en temas clave de buenas prácticas y seguridad en el desarrollo, como los principios de *Clean Code* y el *OWASP Top 10*. Estas reuniones proporcionaron una base de conocimientos sobre la importancia de mantener un código claro y seguro, junto con la prevención de las vulnerabilidades más comunes en aplicaciones web. A lo largo de varias sesiones, el equipo fue instruido sobre:

Este enfoque continuo de capacitación no solo mejoró la habilidad del equipo para identificar amenazas, sino que también les brindó herramientas prácticas y conocimientos fundamentales para aplicar directamente en sus códigos, asegurando que las buenas prácticas de seguridad fueran consideradas durante todo el desarrollo. Este avance reflejó un compromiso con la seguridad desde la perspectiva de la gobernanza, alineándose

con el marco *OWASP SAMM*, y contribuyendo a construir una base sólida de conocimientos de ciberseguridad dentro del equipo.

7.2. Reducción de Vulnerabilidades mediante SonarQube y SonarCloud

La implementación de *SonarQube* y *SonarCloud* como herramientas de análisis de código fue determinante en el cumplimiento de los objetivos específicos de reducir vulnerabilidades y mejorar la calidad del código. A lo largo del proyecto, se observó una disminución gradual en el número de vulnerabilidades detectadas, lo cual evidencia que las prácticas de desarrollo seguro fueron integradas con éxito en el ciclo de vida del *software*. Este logro muestra cómo el equipo adoptó una mentalidad de mejora continua, que permitió no solo detectar y corregir errores, sino también prevenir su reaparición en etapas posteriores.

El uso de estas herramientas ha funcionado de manera destacada en la reducción de malas prácticas de codificación que provocan vulnerabilidades y otros problemas de seguridad. *SonarQube* y *SonarCloud* no solo identifican las áreas problemáticas del código, sino que también proporcionan ejemplos detallados que explican por qué una práctica es problemática y cómo puede ser corregida. Además, ofrecen referencias bibliográficas y enlaces a documentación adicional, permitiendo que los desarrolladores profundicen en el conocimiento de las mejores prácticas de seguridad y programación. Esto ha facilitado la capacitación personalizada de los miembros del equipo, ya que cada uno pudo identificar sus puntos débiles y mejorar sus habilidades de codificación.

Al recibir retroalimentación específica y ejemplos de solución, los desarrolladores se acostumbraron gradualmente a las buenas prácticas de codificación. Como resultado, el código se volvió más limpio y seguro, reduciendo significativamente las vulnerabilidades a lo largo del desarrollo del proyecto. Este enfoque de mejora continua ha sido esencial para reforzar una cultura de desarrollo seguro y eficiente.

El cumplimiento del estándar de cobertura de código en *SonarCloud*, alcanzando un 82 %, complementó este logro al asegurar que el código estuviera correctamente probado y validado, incrementando así la confiabilidad de la aplicación. Este aspecto fue fundamental para garantizar un análisis exhaustivo y efectivo del código en consonancia con los principios del *OWASP Top 10*, aportando una base sólida en la seguridad y calidad de la aplicación.

7.3. Exclusiones de Cobertura y Configuración de Calidad de Código

En el contexto del análisis de *SonarCloud*, fue necesario realizar exclusiones específicas en la cobertura de código y en el análisis de algunos archivos. Los archivos excluidos de la cobertura contenían lógica que carece de aplicabilidad directa para pruebas de cobertura, lo que justifica su exclusión. Estos archivos no afectan la funcionalidad o seguridad del sistema, por lo que mantenerlos fuera del análisis de cobertura permitió una evaluación más precisa y relevante del código activo.

Además, los archivos excluidos del análisis corresponden a código de prueba cuyo propósito es exclusivamente de *testing* y, por lo tanto, no requieren de un análisis de seguridad ni de calidad de código, ya que su

relevancia para los objetivos de seguridad es limitada. Estos archivos, al estar diseñados específicamente para probar componentes y funcionalidades del sistema, contienen configuraciones y dependencias que generan resultados que no aplican a la lógica principal del software en un entorno de producción.

Específicamente, el análisis de estos archivos en *SonarCloud* generaba una gran cantidad de vulnerabilidades e incidencias de seguridad catalogadas como *security hotspots*. Sin embargo, estas detecciones eran falsos positivos, ya que el código de *testing* contiene deliberadamente fragmentos y configuraciones que simulan condiciones de prueba, sin afectar la seguridad o funcionalidad del sistema final. Mantener estos archivos en el análisis hubiera inflado artificialmente los reportes de vulnerabilidad y reducido la precisión de los resultados.

Por tanto, excluir estos archivos de *testing* permitió obtener un reporte de calidad y seguridad más preciso y relevante, enfocado exclusivamente en el código que tiene impacto directo en la seguridad y funcionalidad de la aplicación.

Asimismo, la configuración personalizada de calidad de código fue una decisión estratégica clave. Se diseñó un *quality check* manual que incorporó dos requisitos adicionales a los criterios de *SonarCloud*: asegurar que la cobertura del código completo, no solo del nuevo, superara el 80%, y que no existiera ningún *security hotspot* en el código. Estas condiciones adicionales fueron esenciales para reforzar el nivel de seguridad, asegurando que el código cumpliera consistentemente con altos estándares y que cualquier riesgo potencial fuera identificado y mitigado. Las demás condiciones de *SonarCloud* se mantuvieron, ya que cumplían con buenas prácticas de desarrollo y fortalecían el control de calidad del proyecto.

7.4. Code Smells y Resolución

Tal como se observó en la gráfica de los *Code Smells*, hubo un aumento significativo debido a un tipo de comentario repetitivo en varias partes del código. Sin embargo, estos problemas fueron resueltos de inmediato tras un análisis que determinó que dichos comentarios eran innecesarios. El equipo eliminó estos comentarios, logrando una reducción de la cantidad de *Code Smells*, aunque el cambio no se refleje perfectamente en la gráfica, esto demostrando mejoras en el diseño y en la aplicación de principios de *Clean Code*.

7.5. Código Duplicado y Principio DRY

Respecto al código duplicado, *SonarCloud* establece un estándar de calidad de no superar el 3% en código duplicado. Al acercarse a este límite, se comunicó al equipo las áreas problemáticas, mayormente en los archivos de estilos del *frontend*. Se mostró al encargado cómo reestructurar y aplicar correctamente el principio *DRY* (*Don't Repeat Yourself*), reduciendo así la duplicación de código al 1%. Aunque lo ideal es alcanzar un 0%, se planea continuar aplicando estas mejoras conforme avanza el proyecto.

7.6. Complejidad Ciclomática

La complejidad ciclomatica tuvo un breve aumento debido a la adición de nuevas líneas de código. Sin embargo, se logró reducirla, y el cálculo de la complejidad por línea mostró un valor de:

$$\text{Complejidad por Línea} = \frac{\text{Complejidad Total}}{\text{Número de Líneas de Código}} = \frac{298}{4019} = 0.074$$

Un ratio menor a 0.1 es considerado aceptable, indicando que la complejidad está bien distribuida y no representa un gran riesgo en términos de mantenibilidad. Esto demuestra que las prácticas de *Clean Code* se han aplicado adecuadamente, simplificando el código y facilitando su comprensión, prueba y mantenimiento.

7.6.1. Complejidad por Línea de Código (Ratio)

- Menor a 0.1: Se considera un rango aceptable y razonable para la mayoría de los proyectos grandes. Indica que la complejidad está bien distribuida y no representa un gran riesgo en términos de mantenibilidad.
- Entre 0.1 y 0.2: Puede considerarse moderado, y es un signo de que se deben de empezar a revisar áreas específicas del código que podrían simplificarse. No es necesariamente grave, pero vale la pena monitorearlo.
- Mayor a 0.2: Se considera alto y potencialmente problemático. Un ratio por encima de este valor sugiere que el proyecto tiene secciones complejas que pueden ser difíciles de mantener, entender y probar. A este nivel, es probable que se necesite una refactorización significativa para mejorar la calidad.

El valor obtenido de 0.074 en nuestro proyecto cae dentro del rango aceptable, lo que indica que el código está bien estructurado y no presenta riesgos significativos en términos de complejidad. Esto es un reflejo positivo del uso de prácticas de *Clean Code* y otras estrategias de simplificación, que han contribuido a mantener la complejidad del proyecto bajo control y a facilitar su mantenimiento.

7.7. Densidad de Comentarios

La densidad de comentarios se mantuvo en un rango del 10 %, un valor adecuado para asegurar que el código esté bien documentado sin ser excesivamente comentado. Un porcentaje muy bajo puede indicar falta de documentación, mientras que un porcentaje muy alto puede sugerir un código complejo.

El aumento observado en la densidad de comentarios se debió a una mala comunicación de mi parte, haciendo pensar al encargado que era necesario comentar todo el código. Sin embargo, aclaré que los comentarios solo debían agregarse en áreas ambiguas. Tras esta corrección, se logró mantener el porcentaje dentro del rango ideal.

Este ajuste en la densidad de comentarios, combinado con la aplicación de *Clean Code*, aseguró que el código fuera comprensible y fácil de mantener sin necesidad de explicaciones excesivas.

7.8. Integración de Seguridad en Componentes de Inteligencia Artificial y Chatbot

Finalmente, la integración de prácticas de seguridad en los modelos de inteligencia artificial (*IA*) y en el código del *chatbot* refleja el compromiso del encargado con una protección integral de todos los componentes de la aplicación. Este trabajo fue desarrollado por mi compañero Pedro Pablo, quien desde el inicio aplicó rigurosamente las buenas prácticas de codificación. Los resultados de los escaneos de seguridad fueron una clara evidencia de esto: el modelo de *IA* desarrollado no presentó ningún problema de seguridad en los análisis iniciales, lo que demuestra la efectividad de las medidas preventivas que implementó.

En cuanto al escaneo de seguridad del *chatbot*, que hace uso del modelo de *IA* entrenado por Pedro Pablo, solo se detectó un problema menor, el cual fue corregido de inmediato. Este éxito refleja no solo la calidad del trabajo de Pedro Pablo, sino también el impacto positivo de las sesiones de capacitación en seguridad que se llevaron a cabo semanalmente. De mi parte, me aseguré de transmitirle de manera efectiva los conceptos y buenas prácticas discutidos durante las reuniones, lo cual él supo aplicar de manera óptima a lo largo de todo su desarrollo.

Este enfoque colaborativo y de aprendizaje continuo garantizó que los componentes críticos, como el *chatbot* y el modelo de *IA*, estuvieran alineados con los estándares de seguridad del proyecto. La aplicación de estas prácticas desde el principio no solo facilitó la identificación y corrección rápida de problemas, sino que también fortaleció la postura general de seguridad de la aplicación.

7.9. Análisis de Vulnerabilidades en la Aplicación Móvil con AppSweep y MobSF

La incorporación de *AppSweep* y *MobSF* para el análisis de vulnerabilidades en la aplicación móvil fue un paso clave para cumplir con el objetivo de realizar evaluaciones exhaustivas de seguridad. Estas herramientas proporcionaron información crítica sobre posibles debilidades en el código y configuraciones específicas de la aplicación móvil, cubriendo aspectos de seguridad que complementan los análisis realizados en el *backend*.

Sin embargo, aunque se realizó un análisis y se identificaron vulnerabilidades, el proyecto aún no ha logrado aplicar correcciones para eliminar estos problemas de seguridad en la aplicación móvil. Este punto refleja una oportunidad de mejora, pues los hallazgos de *AppSweep* y *MobSF* constituyen un primer paso hacia la mitigación de riesgos, pero se necesita tiempo adicional para implementar las recomendaciones propuestas.

A pesar de no haber concluido con la remediación de todas las vulnerabilidades detectadas, esta etapa es un avance importante y forma parte del ciclo continuo de mejora del proyecto. Dado que este desarrollo

se lleva a cabo en colaboración con la Municipalidad de Guatemala, se cuenta con la oportunidad de aplicar las correcciones necesarias en futuras fases del proyecto. Esto asegurará que la aplicación móvil cumpla con altos estándares de seguridad antes de ser lanzada al público, minimizando así los riesgos de filtración de información y mejorando la protección de los datos de los usuarios.

Esta situación también contribuye al aprendizaje continuo del equipo, ya que al aplicar las correcciones y buenas prácticas sugeridas por *AppSweep* y *MobSF*, será necesario capacitar al equipo en estas nuevas áreas de seguridad. Esto no solo permitirá fortalecer el conocimiento en el manejo de vulnerabilidades móviles, sino también consolidar una cultura de seguridad orientada a la proactividad y a la mejora continua.

7.10. Configuraciones de Permisos en AWS

La gestión de permisos en AWS fue una parte crítica para asegurar que se cumpla con el principio de mínimo privilegio, minimizando así los riesgos de seguridad asociados a una configuración incorrecta o a la asignación excesiva de permisos. A continuación, se detalla el análisis y los cambios realizados en los permisos asignados a los miembros del equipo.

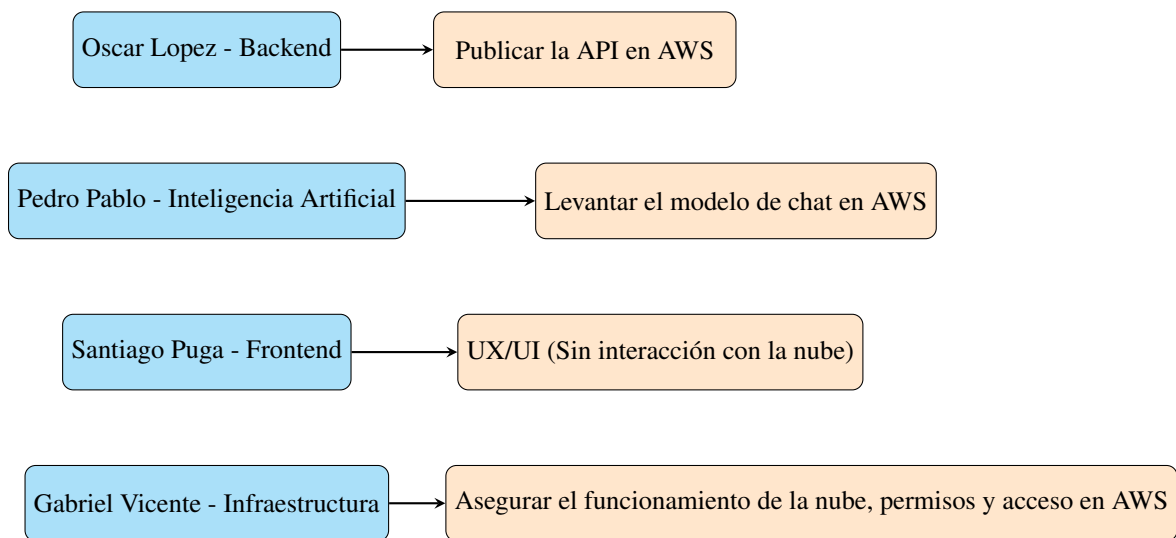


Figura 7.1: Diagrama de Roles y Responsabilidades del Equipo

7.10.1. Oscar López: Revisión de Permisos de DatabaseAdministrator

En el caso de Oscar López, quien tiene el rol de *DatabaseAdministrator*, se identificaron varios permisos innecesarios que se propusieron eliminar para mejorar la seguridad:

- *dynamodb:** – Este permiso permite gestionar *DynamoDB*, un recurso que no se utiliza en este proyecto, por lo que se decidió retirarlo.
- *s3:CreateBucket* – Se propuso eliminar este permiso, ya que Oscar solo debe poder utilizar los

buckets S3 creados por el encargado de la infraestructura, Gabriel, en lugar de tener la capacidad de crear nuevos *buckets*.

- *iam* – Este permiso fue considerado un riesgo, ya que permite a Oscar delegar permisos a funciones (roles). Se decidió quitarlo para evitar la posibilidad de un escalado de privilegios no autorizado. Limitar los roles específicos que Oscar pueda pasar es crucial para prevenir accesos indebidos y garantizar la seguridad del sistema.

Estos cambios aseguran que los permisos de Oscar se mantengan en un nivel mínimo necesario, evitando problemas potenciales derivados de un exceso de privilegios y asegurando un mejor cumplimiento de las prácticas de seguridad.

7.10.2. Pedro Pablo: Revisión de Permisos de SystemAdministrator

Para Pedro Pablo, quien tiene el rol de *SystemAdministrator*, también se identificaron permisos que no eran necesarios:

- *trustadvisor:** – Este permiso permite el acceso a *AWS Trusted Advisor*, que no es necesario para Pedro Pablo, ya que el monitoreo y asesoramiento sobre la configuración y el estado de la cuenta es responsabilidad de Gabriel.
- *config:** – Permisos relacionados con *AWS Config*, que tampoco son relevantes para las tareas de Pedro Pablo.
- *iam* – Al igual que con Oscar, se decidió eliminar este permiso para evitar riesgos de seguridad, ya que permite que Pedro Pablo delegue permisos a otros roles de entidades. Limitar esta capacidad es fundamental para proteger contra posibles escaladas de privilegios.

Estas modificaciones aseguran que Pedro Pablo solo tenga los permisos necesarios para su trabajo, manteniendo una configuración segura y eficiente.

7.10.3. Santiago Taracena y Yongbum Park: Revisión de Permisos de ReadOnlyAccess

Santiago Taracena y yo (Yongbum Park) tenemos asignado el rol de *ReadOnlyAccess*, lo cual es adecuado para nuestras responsabilidades actuales, ya que ninguno de los dos necesita realizar modificaciones en la infraestructura de AWS. Esta configuración permite que Santiago trabaje a través de *GitHub Actions* para subir su parte del código sin intervenir directamente en AWS, y que yo supervise y verifique la infraestructura sin hacer cambios.

Sin embargo, se identificó que en mi caso, Yongbum Park, sería útil tener permisos adicionales para poder observar contraseñas y configuraciones relacionadas. Actualmente, *ReadOnlyAccess* no permite ver detalles como las contraseñas, lo cual es importante para revisar que se sigan las buenas prácticas de seguridad,

como el uso de contraseñas complejas y únicas. Se planea ajustar mi configuración para incluir permisos de observación más completos, sin permitir ninguna modificación.

Estos ajustes en los permisos reflejan nuestro compromiso con una gestión de seguridad efectiva, alineada con el principio de mínimo privilegio, asegurando que cada usuario tenga solo los permisos estrictamente necesarios para desempeñar sus funciones sin comprometer la seguridad del sistema.

Conclusiones

- El desarrollo de este proyecto permitió abordar la problemática de la falta de prácticas de seguridad en el ciclo de vida del *software*, logrando una implementación efectiva de medidas que fortalecen tanto la seguridad como la calidad del código en aplicaciones móviles y *backend*.
- El proyecto se basó en el marco de *OWASP SAMM*, aplicando metodologías de Gobernanza, Diseño, Implementación y Verificación, lo que contribuyó a una metodología de trabajo integral y proactiva para abordar la seguridad desde el inicio de la creación del *software*.
- Se realizó un análisis exhaustivo utilizando *SonarQube* y *SonarCloud* para la detección y mitigación de vulnerabilidades en el análisis estático del código, contribuyendo a una disminución continua de las fallas de seguridad a lo largo del proyecto. Estas herramientas no solo permitieron identificar problemas en el código, sino también aplicar parches cuando fue necesario, elevando el estándar de calidad y seguridad en el desarrollo del *backend*.
- Para el análisis de la aplicación móvil, se emplearon las tecnologías *MobSF* y *AppSweep*, que facilitaron la identificación de vulnerabilidades específicas del entorno móvil. Sin embargo, debido a limitaciones de tiempo, no se lograron implementar las correcciones recomendadas por estas herramientas. Este proyecto no se detendrá en esta etapa, ya que el equipo continuará trabajando en futuras mejoras y aplicará las recomendaciones obtenidas, con el fin de alcanzar los estándares de seguridad que *MobSF* y *AppSweep* requieren para una aplicación móvil robusta y confiable.
- El equipo experimentó un crecimiento significativo en su cultura de seguridad a través de capacitaciones regulares y pruebas de *phishing*. Las evaluaciones realizadas evidenciaron mejoras en la habilidad del equipo para identificar correos maliciosos, lo que redujo el riesgo de vulnerabilidades humanas en la seguridad del sistema.
- Se lograron detectar algunos permisos que no son necesarios para ciertos roles del equipo, lo cual resalta la importancia de revisar los permisos generados por defecto en *AWS*. Esto permitió aplicar

principios de mínimo privilegio, evitando riesgos potenciales por configuraciones de acceso excesivas y mejorando la seguridad general de la infraestructura.

- La cobertura de código alcanzada en *SonarCloud* superó el 80 %, validando la calidad de las pruebas y el compromiso con un desarrollo seguro. La configuración de un *quality check* adicional aseguró que el código cumpliera consistentemente con altos estándares de seguridad y calidad.
- El análisis de seguridad aplicado al código de modelos de inteligencia artificial y *chatbot* desarrollados en el proyecto reflejó una implementación rigurosa de buenas prácticas, asegurando que estos componentes también cumplieran con altos estándares de protección de datos y seguridad operativa.
- Las herramientas y metodologías aplicadas en el proyecto, junto con las prácticas de capacitación continua, no solo contribuyeron a resolver el problema original planteado sino que también establecieron una base sólida para el desarrollo de futuras aplicaciones seguras y confiables.
- Este proyecto no solo facilita la identificación de vulnerabilidades en el código, sino que también apoya el desarrollo de una cultura de seguridad en el equipo y fortalece la postura de seguridad general de la organización, contribuyendo a los estándares de calidad en la industria de desarrollo de *software* seguro.
- El aprendizaje obtenido a nivel académico y profesional resultó invaluable, abarcando desde temas de gestión de accesos y configuraciones de seguridad en infraestructura en la nube hasta el análisis y mitigación de vulnerabilidades en el ciclo de desarrollo de *software*.

Recomendaciones

- Se recomienda establecer un programa continuo de capacitación en ciberseguridad para el equipo, basado en las sesiones de formación que mejoraron la detección de *phishing* y fortalecieron la cultura de seguridad del equipo. Esto permitirá que el personal se mantenga actualizado frente a nuevas amenazas y adopte prácticas seguras de forma constante. Además, sería ideal implementar herramientas específicas para la capacitación, tales como simuladores de ataques de *phishing*, plataformas de evaluación de habilidades y recursos de entrenamiento interactivos, que faciliten la práctica de escenarios de seguridad en un entorno controlado y seguro.
- Para futuras etapas del proyecto, se sugiere investigar y evaluar otras herramientas de análisis de código que puedan complementar las capacidades de *SonarQube* y *SonarCloud*, particularmente en áreas donde se requiera mayor especificidad en la detección de vulnerabilidades o una integración más robusta en entornos cambiantes de desarrollo. Aunque *SonarQube* es una herramienta poderosa y ampliamente utilizada para el análisis de calidad de código y detección de vulnerabilidades, presenta algunas limitaciones. En primer lugar, su cobertura en la detección de vulnerabilidades específicas puede ser limitada, ya que se enfoca principalmente en la calidad del código y errores comunes, pero no siempre detecta vulnerabilidades avanzadas, especialmente aquellas relacionadas con el análisis dinámico o la seguridad de la infraestructura. Además, aunque *SonarQube* permite la integración continua con diversas plataformas, equipos de desarrollo que operan en entornos altamente cambiantes podrían encontrar dificultades al configurar y personalizar la herramienta para su flujo de trabajo o tecnologías emergentes. Por último, *SonarQube*, a pesar de ser compatible con muchos lenguajes, puede no ser tan eficiente al detectar problemas en lenguajes menos comunes o más nuevos, lo que podría requerir el uso de herramientas adicionales especializadas en esas áreas.
- Dado el uso de bibliotecas de terceros en el proyecto, se sugiere integrar herramientas de monitoreo de vulnerabilidades en dependencias externas en los *pipelines* de desarrollo. Estas herramientas permiten detectar vulnerabilidades en tiempo real y aplicar actualizaciones de seguridad de forma proactiva, reduciendo los riesgos asociados con dependencias externas en la aplicación.

- Para mejorar la confiabilidad del código, se sugiere aumentar la cobertura de pruebas en áreas críticas de la aplicación, superando el 82 % alcanzado en este proyecto. Enfocarse en módulos de alto impacto permitirá asegurar una mayor estabilidad y seguridad en componentes que puedan representar riesgos significativos para la aplicación.
- Se sugiere seguir los consejos y recomendaciones proporcionados por las tecnologías de *MobSF* y *AppSweep* en el análisis de aplicaciones móviles, ya que sus reportes detallados ayudan a identificar configuraciones deficientes y áreas de mejora en la seguridad del código. Además, para futuras etapas del proyecto, se recomienda investigar y evaluar herramientas adicionales de análisis de seguridad que profundicen en las evaluaciones de seguridad de aplicaciones móviles, ofreciendo mayor cobertura y especificidad en la detección de vulnerabilidades que sean críticas para este tipo de entornos.
- A fin de optimizar el flujo de trabajo y asegurar escaneos constantes de seguridad, se recomienda implementar un *pipeline* de *CI/CD* para la aplicación móvil. Esto permitirá que herramientas como *MobSF* y *AppSweep* se ejecuten automáticamente en cada despliegue o construcción de la aplicación, eliminando la necesidad de cargas manuales y garantizando que cada actualización pase por un análisis de seguridad inmediato. Esta integración continua contribuirá a mantener un alto estándar de seguridad en cada versión publicada de la aplicación.

Referencias

- (AWS), A. W. S. (2024). *¿qué es el ciclo de vida del desarrollo de software (sdlc)?* Descargado de <https://aws.amazon.com/es/what-is/sdlc/>
- Bernal, C. F. (2024). *Guía de buenas prácticas de seguridad en el desarrollo de software con base en estándares reconocidos en empresas de desarrollo de software.* Descargado de <https://repository.unad.edu.co/jspui/bitstream/10596/20449/1/1088303409.pdf>
- Blog, C. (2024). *What is clean code? a guide to principles and best practices.* Descargado de <https://blog.codacy.com/what-is-clean-code>
- Caro, A. (2023). *La importancia del desarrollo seguro de software en la era digital.* Descargado de <https://web.fdi.ucm.es/posgrado/conferencias/AndresCaroLindo-slides.pdf>
- Chávez, J. J. S. (2024). *4 buenas prácticas para un desarrollo seguro de software: ¿por qué es importante?* Descargado de <https://www.deltaprotect.com/blog/desarrollo-seguro-de-software> (Publicado el 9 de febrero de 2024, actualizado el 25 de abril de 2024. Último acceso: 15 de octubre de 2024)
- Colombia, U. (2024). *¿qué es la ciberseguridad? la importancia de la ciberseguridad.* Descargado de <https://colombia.unir.net/actualidad-unir/que-es-ciberseguridad/#:~:text=La%20importancia%20de%20la%20ciberseguridad&text=La%20ciberseguridad%20no%20solo%20salvaguada,la%20evoluci%C3%B3n%20de%20las%20amenazas>
- Consulting, I. (2022a). *Art. 5 gdpr - principles relating to processing of personal data.* Descargado 2024-09-22, de <https://gdpr-info.eu/art-5-gdpr/>
- Consulting, I. (2022b). *Art. 6 gdpr - lawfulness of processing.* Descargado 2024-09-22, de <https://gdpr-info.eu/art-6-gdpr/>
- de Ambiente y Recursos Naturales (MARN), M. (2024a). *Políticas ambientales y acuerdo gubernativo 189-2017.* Descargado de https://www.marn.gob.gt/wp-admin/admin-ajax.php?juwpfisadmin=false&action=wpfd&task=file.download&wpfd_category_id=96&wpfd_file_id=8524&token=&preview=1
- de Ambiente y Recursos Naturales (MARN), M. (2024b). *Reglamento para la gestión integral de los residuos y desechos sólidos comunes (acuerdo gubernativo 164-2021).* Descargado de <https://www.marn.gob.gt/reglamento-164-2021/>
- de Cataluña, U. (2024). *Seguridad informática: la importancia y lo que debe saber.* Descargado de <https://www.ucatalunya.edu.co/blog/seguridad-informatica-la-importancia-y-lo-que-debe-saber>
- de Derechos Humanos (Corte IDH), C. I. (2024). *Ley de protección y mejoramiento del medio ambiente.* Descargado de <https://www.corteidh.or.cr/tablas/20491a.pdf>
- de Problemas Nacionales de la USAC (IPN), I. (2023). *Análisis de la generación de desechos sólidos en guatemala.* Descargado de <https://ipn.usac.edu.gt/wp-content/uploads/2023/09/Analisis-de-la-generacion-de-desechos-solidos-en-Guatemala.pdf>
- Electro, S. (2011). *Principios de un diseño seguro.* Descargado de <https://serzaelectro.wordpress.com/2011/03/10/principios-de-un-diseno-seguro/>
- Europa, Y. (2024). *¿cómo reducir mi huella de carbono?* Descargado de https://youth.europa.eu/get-involved/sustainable-development/how-reduce-my-carbon-footprint_es
- Formación, I. (2024). *¿qué es sonarqube?* Descargado de <https://imaginaformacion.com/tutoriales/que-es-sonarqube> (Último acceso: 15 de octubre de 2024)

- Foundation, O. (2024a). *Owasp software assurance maturity model (samm)*. Descargado de <https://owasp samm.org/>
- Foundation, O. (2024b). *Owasp top 10 web application security risks*. Descargado de <https://owasp.org/www-project-top-ten/>
- GADISA. (2024). *Gestión de residuos: Todo lo que debes saber*. Descargado de <https://www.gadisa.es/blog/gestion-de-residuos/#:~:text=En%20primer%20lugar%2C%20ayuda%20a,relativos%20con%20los%20residuos%20peligrosos>
- Guatemala, I. S. (2021). *Situación de guatemala en materia de seguridad cibernética*. Descargado de <https://www.isoc.org.gt/wp-content/uploads/2021/11/Situacion-de-Guatemala-en-materia-de-Seguridad-Cibernetica.pdf>
- Gupta, R. (2024). *Sonarqube security features*. Descargado de https://media.licdn.com/dms/document/media/D4D1FAQHlnis2swS0zg/feedshare-document-pdf-analyzed/0/1729849905729?e=1730937600&v=beta&t=BV9P0AtmIfhLi6gMVyamdLwVKoAZSH13Vw6eCH13Z_4
- Hat, R. (2024). *Software development lifecycle security*. Descargado de <https://www.redhat.com/es/topics/security/software-development-lifecycle-security#sdlc-devops-y-el-enfoque-%C3%A1gil>
- Hernández, H. I. (2021). *Qué es el gdpr y cómo afecta a latinoamérica*. Descargado 2024-09-22, de <https://news.registro.gt/2021/09/27/que-es-el-gdpr-y-como-afecta-a-latinoamerica/#:~:text=Guatemala%20carece%20de%20una%20normativa,datos%20personales%20y%20datos%20sensibles>
- Iberdrola. (2024). *¿qué es la huella de carbono y cómo reducirla?* Descargado de <https://www.iberdrola.com/sostenibilidad/huella-de-carbono>
- Laotshi. (2020). *Secure by design: Fundamentos del diseño seguro*. Descargado de <https://laotshi.medium.com/secure-by-design-fundamentos-b4d25ed3a9ac>
- Libre, P. (2024). *Separación de desechos en guatemala: así debe clasificar su basura con los cambios al reglamento y lo que será obligatorio en 2025*. Descargado de <https://www.prensalibre.com/guatemala/comunitario/separacion-de-desechos-en-guatemala-asi-debe-clasificar-su-basura-con-los-cambios-al-reglamento-y-lo-que-sera-obligatorio-en-2025-beaking/>
- López, K. R. (2022). *Ley contra ciberdelincuencia, la normativa que puso en riesgo la libertad de expresión y la crítica ciudadana*. Descargado de <https://www.plazapublica.com.gt/content/ley-contr-a-ciberdelincuencia-la-normativa-que-puso-en-riesgo-la-libertad-de-expresion-y-la>
- Martin, R. C. (2008). *Clean code: A handbook of agile software craftsmanship*. Upper Saddle River, NJ: Prentice Hall. Descargado de <https://github.com/jnguyen095/clean-code/blob/master/Clean.Code.A.Handbook.of.Agile.Software.Craftsmanship.pdf>
- Microsoft. (2021). *Principios de seguridad en el marco microsoft well-architected*. Descargado de <https://learn.microsoft.com/es-es/azure/well-architected/security/principles>
- Micucci, M. (2023). *Metodologías de desarrollo seguro de software e integración de aplicaciones*. Descargado de <https://www.welivesecurity.com/es/recursos-herramientas/metodologias-desarrollo-seguro-software-integracion-aplicaciones/>
- Pineda, G. (2023). *Conflictos en la gestión de los residuos y desechos sólidos: impactos económicos,*

- ambientales y sociales de prácticas e implementación de políticas no consensuadas en guatemala*. Descargado de <https://capitalisthistory.com/2023/08/07/conflictos-en-la-gestion-de-los-residuos-y-desechos-solidos-impactos-economicos-ambientales-y-sociales-de-practicas-e-implementacion-de-politicas-no-consensuadas-en-guatemala/>
- Rodriguez, C. (2024). Tecnologías modernas para el tratamiento de residuos sólidos urbanos. *ResearchGate*. Descargado de https://www.researchgate.net/publication/348280674_Tecnologias_Modernas_para_el_tratamiento_de_Residuos_Solidos_Urbanos
- Romuno, J. (2024). *Smart waste management technologies*. Descargado de <https://www.rts.com/es/blog/smart-waste-management-technologies/> (Publicado el 15 de mayo de 2023. Actualizado el 1 de enero de 2024. Último acceso: 15 de octubre de 2024)
- Security, T. (2024a). *Owasp samm: Cómo mejorar la seguridad del software empresarial*. Descargado de <https://www.tarlogic.com/es/blog/owasp-samm-mejorar-seguridad-software-empresarial/>
- Security, T. (2024b). *Owasp top 10: Riesgos en aplicaciones móviles*. Descargado de <https://www.tarlogic.com/es/blog/owasp-top-10-riesgos-aplicaciones-moviles/>
- Seguras, S. (2022). *¿cuáles son los principales desafíos de ciberseguridad en guatemala en este 2022?* Descargado de <https://www.solucionesseguras.com/noticias/noticias-y-eventos/cuales-son-los-principales-desafios-de-ciberseguridad-en-guatemala-en-este-2022>
- Sonar. (2022). *Sonarqube 10.7 documentation*. Descargado de <https://docs.sonarsource.com/sonarqube/latest/>

Prácticas de Clean Code

El uso de las prácticas de *Clean Code* es esencial para mejorar la legibilidad, mantenibilidad y calidad general del código. A continuación, se detallan algunos de los principios y recomendaciones claves extraídos del libro de Robert C. Martin (*Clean Code: A Handbook of Agile Software Craftsmanship*) (Martin, 2008).

A.1. Uso de Nombres que Revelan la Intención

Es crucial que los nombres de variables, funciones o clases revelen claramente su propósito. Un buen nombre responde a preguntas como por qué existe, qué hace y cómo se usa. Si un nombre necesita un comentario adicional para explicar su intención, entonces no es suficientemente claro.

```
int d; // tiempo transcurrido en días (nombre poco claro)
int elapsedTimeInDays; // nombre descriptivo que revela la intención
int daysSinceCreation;
```

Elegir nombres descriptivos puede simplificar considerablemente la comprensión y modificación del código, reduciendo la necesidad de comentarios adicionales.

A.2. Evitar la Desinformación

Es importante evitar nombres que puedan confundir al lector. Utilizar palabras con significados ambiguos o que se asocian comúnmente con otros conceptos puede inducir a errores. Por ejemplo, evitar nombres de

variables como *hp*, *aix*, o *sco* que pueden confundirse con plataformas *Unix*.

Además, utilizar nombres que difieran ligeramente entre sí, como *XYZControllerForEfficientHandlingOfStrings* y *XYZControllerForEfficientStorageOfStrings*, puede causar confusión. Elige nombres que claramente destaquen las diferencias y que sean fácilmente distinguibles.

A.3. Crear Funciones Pequeñas

Las funciones deben ser lo más pequeñas posible. La práctica de escribir funciones breves facilita su comprensión y reduce los errores. Como regla general, las funciones no deben tener más de 20 líneas. Las funciones más cortas, de dos a cinco líneas, suelen ser más eficaces y manejables.

```
public List<Cell> getFlaggedCells() {
    List<Cell> flaggedCells = new ArrayList<>();
    for (Cell cell : gameBoard) {
        if (cell.isFlagged()) {
            flaggedCells.add(cell);
        }
    }
    return flaggedCells;
}
```

El código anterior se ha simplificado al encapsular la funcionalidad y evitar repeticiones innecesarias.

A.4. Evitar la Duplicación de Código

La duplicación es uno de los principales enemigos de un código limpio. Evitar duplicar fragmentos de código no solo mejora la claridad, sino que también facilita la implementación de cambios futuros y reduce el riesgo de errores.

A.5. Nombres Descriptivos y Consistentes

Los nombres deben ser descriptivos, incluso si esto significa hacerlos más largos. Un nombre bien definido es preferible a un nombre breve y enigmático, ya que mejora la claridad y facilita el mantenimiento del código.

```
int realDaysPerIdealDay = 4;
const int WORK_DAYS_PER_WEEK = 5;
int sum = 0;
for (int j=0; j < NUMBER_OF_TASKS; j++) {
```

```
int realTaskDays = taskEstimate[j] * realDaysPerIdealDay;  
int realTaskWeeks = (realTaskDays / WORK_DAYS_PER_WEEK);  
sum += realTaskWeeks;  
}
```

A.6. Usar Nombres Pronunciables

Los nombres deben ser fácilmente pronunciables para que los programadores puedan comunicarse sobre el código de manera efectiva. Nombres como `genymdhms` pueden confundir a los programadores y dificultar el entendimiento de la lógica.

A.7. Comentarios No Sustituyen el Código Limpio

Es común querer añadir comentarios para explicar un código desorganizado, pero un código claro y expresivo sin comentarios adicionales es preferible. Si el código necesita comentarios extensos, es una señal de que debe ser limpiado y reorganizado.

A.8. Usar Excepciones en Lugar de Códigos de Retorno

Utilizar excepciones para el manejo de errores, en lugar de códigos de retorno, mejora la claridad y separa la lógica de control de errores de la lógica principal. Esto facilita la lectura y mantenimiento del código.

A.9. Mantener Pruebas Unitarias Limpias

Las pruebas unitarias deben tener el mismo nivel de calidad que el código de producción, siendo claras y estructuradas. Esto asegura su mantenibilidad y permite que el código se ajuste a futuros cambios sin aumentar la posibilidad de errores.

A.10. Organización de Clases y Principio de Responsabilidad Única

Las clases deben seguir el Principio de Responsabilidad Única (*SRP*), donde cada clase debe tener una sola responsabilidad o razón para cambiar. Esto organiza el código en clases pequeñas y enfocadas, facilitando el mantenimiento y los cambios futuros.

A.11. Evitar Comentarios Redundantes o Engañosos

Evita los comentarios que solo repiten el código o que no aportan claridad. Los comentarios innecesarios pueden incluso generar confusión, mientras que un código claro comunica de forma efectiva sin necesidad de explicaciones adicionales.

OWASP Top 10

El *OWASP Top 10* es un documento de referencia globalmente reconocido que ayuda a los desarrolladores y equipos de seguridad en aplicaciones web a entender los riesgos de seguridad más críticos. Es un primer paso esencial hacia una codificación más segura y una herramienta fundamental para cambiar la cultura de desarrollo de software dentro de las organizaciones, promoviendo prácticas que producen código más seguro (Foundation, 2024b).

B.1. A01:2021 - Control de Acceso Quebrantado (Broken Access Control)

Descripción: Este riesgo sube de posición debido a la alta prevalencia de problemas de control de acceso. Aproximadamente el 94% de las aplicaciones probadas muestran algún tipo de vulnerabilidad de control de acceso quebrantado, que resulta en la exposición o modificación de información no autorizada. Ejemplos de vulnerabilidades incluyen la violación del principio de privilegio mínimo, manipulación de *URL* o parámetros, y el acceso no autorizado a *API*.

Prevención: Implementar controles de acceso efectivos en el código del lado del servidor y en *APIs*, asegurando que solo los usuarios autorizados puedan acceder a recursos específicos. Limitar el acceso según los roles, y validar los *tokens* y *cookies* de control de acceso. Mantener un registro de intentos fallidos y alertar a los administradores cuando sea necesario.

Escenarios de ataque: Un atacante modifica un parámetro en la *URL* para obtener acceso a la información de una cuenta no autorizada, o navega a una página restringida sin las credenciales adecuadas

(Foundation, 2024b).

B.2. A02:2021 - Fallos Criptográficos (Cryptographic Failures)

Descripción: Anteriormente conocido como **Exposición de Datos Sensibles**, este riesgo ahora se enfoca en fallas criptográficas. Las vulnerabilidades típicas incluyen la falta de cifrado de datos en tránsito, uso de algoritmos débiles o criptografía insuficiente, y manejo inadecuado de claves de cifrado.

Prevención: Asegurarse de que todos los datos sensibles estén cifrados tanto en tránsito como en reposo, y de que se utilicen algoritmos de cifrado fuertes y prácticas de gestión de claves adecuadas. Además, se debe aplicar autenticación y validación de certificados para las conexiones.

Escenarios de ataque: Un atacante intercepta la comunicación en texto claro para robar datos sensibles como contraseñas o números de tarjeta de crédito (Foundation, 2024b).

B.3. A03:2021 - Inyección (Injection)

Descripción: Este riesgo se presenta cuando datos de usuario no validados se insertan en una consulta o comando, exponiendo a las aplicaciones a *SQL Injection*, *NoSQL Injection*, *Cross-site Scripting*, entre otros. La inyección ocurre en el 94 % de las aplicaciones probadas.

Prevención: Usar *APIs* seguras que no dependan de la inyección de datos en consultas o comandos, validando y escapando correctamente los datos de usuario.

Escenarios de ataque: Un atacante inserta código malicioso en una consulta *SQL*, permitiéndole extraer o modificar datos en una base de datos (Foundation, 2024b).

B.4. A04:2021 - Diseño Inseguro (Insecure Design)

Descripción: Un diseño inseguro se refiere a la ausencia de controles de seguridad efectivos en el diseño de la aplicación. No puede ser mitigado con implementaciones seguras ya que los controles necesarios no existen en el diseño inicial.

Prevención: Usar modelos de amenazas y patrones de diseño seguro desde el inicio del desarrollo, integrando controles de seguridad en las historias de usuario y casos de uso.

Escenarios de ataque: Un flujo de recuperación de credenciales inseguro podría permitir el acceso no autorizado a cuentas (Foundation, 2024b).

B.5. A05:2021 - Configuración de Seguridad Incorrecta (Security Misconfiguration)

Descripción: Este riesgo aumenta debido a la prevalencia de configuraciones de seguridad incorrectas en los servidores y componentes. Errores comunes incluyen cuentas y contraseñas por defecto, y aplicaciones de prueba en servidores de producción.

Prevención: Implementar procesos de configuración y endurecimiento de la seguridad, removiendo funcionalidades innecesarias y revisando configuraciones regularmente.

Escenarios de ataque: Un servidor que permite la visualización de directorios permite a un atacante listar archivos y obtener información sensible (Foundation, 2024b).

B.6. A06:2021 - Componentes Vulnerables y Desactualizados (Vulnerable and Outdated Components)

Descripción: La falta de actualización o el uso de componentes vulnerables representan un riesgo considerable en las aplicaciones, ya que pueden contener vulnerabilidades conocidas.

Prevención: Realizar un inventario continuo de los componentes usados en el cliente y en el servidor, sus dependencias y sus versiones. Implementar un proceso de gestión de parches y actualizaciones.

Escenarios de ataque: Un atacante explota una vulnerabilidad en una versión desactualizada de un componente para ejecutar código remoto en el servidor (Foundation, 2024b).

B.7. A07:2021 - Fallos en la Identificación y Autenticación (Identification and Authentication Failures)

Descripción: Incluye problemas como el uso de contraseñas débiles, autenticación de un solo factor y gestión de sesiones inadecuada, lo cual expone a ataques de autenticación como el *credential stuffing*.

Prevención: Implementar autenticación multifactor, no permitir el uso de contraseñas por defecto y limitar los intentos fallidos de inicio de sesión.

Escenarios de ataque: Un atacante utiliza listas de contraseñas conocidas para comprometer cuentas de usuario en una aplicación que carece de protección contra ataques automatizados (Foundation, 2024b).

B.8. A08:2021 - Fallos en la Integridad de Software y Datos (Software and Data Integrity Failures)

Descripción: Este riesgo se enfoca en las fallas relacionadas con la integridad de *software* y datos. Los problemas de integridad permiten la ejecución de código no autorizado, comprometido o modificado por un atacante.

Prevención: Verificar las actualizaciones de *software* mediante firmas digitales y asegurarse de que los repositorios y bibliotecas utilizadas son de confianza.

Escenarios de ataque: Un atacante explota la falta de verificación de firmas en una actualización para introducir una versión maliciosa del *software* (Foundation, 2024b).

B.9. A09:2021 - Fallos en el Registro y Monitoreo de Seguridad (Security Logging and Monitoring Failures)

Descripción: La falta de un monitoreo efectivo y de registros de seguridad permite que las brechas de seguridad pasen desapercibidas, impidiendo una detección temprana y respuesta a incidentes.

Prevención: Asegurarse de que todos los intentos de acceso, fallidos y exitosos, estén registrados, y establecer un sistema de alertas y respuesta ante actividades sospechosas.

Escenarios de ataque: Una organización sufre una brecha de seguridad sin ser consciente de ella debido a la falta de monitoreo de su aplicación (Foundation, 2024b).

B.10. A10:2021 - Falsificación de Solicitudes del Lado del Servidor (Server-Side Request Forgery, SSRF)

Descripción: Las vulnerabilidades *SSRF* permiten a un atacante manipular las solicitudes enviadas por la aplicación a otros servidores sin validación, posibilitando el acceso a recursos no autorizados.

Prevención: Implementar políticas de *firewall* de denegación por defecto, segmentar las funciones de acceso remoto en redes separadas, y validar todas las entradas de *URL* proporcionadas por el cliente.

Escenarios de ataque: Un atacante usa *SSRF* para realizar un escaneo de puertos en la red interna o acceder a datos sensibles expuestos en la misma (Foundation, 2024b).

OWASP Software Assurance Maturity Model (SAMM)

El *OWASP Software Assurance Maturity Model (SAMM)* es un modelo de madurez ampliamente reconocido que ofrece una forma efectiva y medible para que todo tipo de organizaciones analicen y mejoren su postura de seguridad en el *software*. *SAMM* abarca todo el ciclo de vida del *software*, incluyendo el desarrollo y la adquisición, y es agnóstico tanto en tecnología como en procesos. Su diseño es intencionalmente evolutivo y orientado al riesgo, lo que permite una mejora continua (Foundation, 2024a).

Historia: El modelo original (v1.0) fue creado por Pravir Chandra en 2009 y ha demostrado ser un modelo efectivo y ampliamente adoptado para mejorar las prácticas de *software* seguro en organizaciones de diferentes tipos alrededor del mundo. Con la versión 2.0, el modelo se mejoró para abordar algunas de sus limitaciones actuales mediante un enfoque basado en retroalimentación de la comunidad *OWASP* y expertos en cumbres en Europa y EE.UU.

C.1. Prácticas de Seguridad en SAMM

SAMM cubre el ciclo de vida del *software* en múltiples dominios de prácticas de seguridad, estructurados en cuatro funciones principales: *Governance*, *Design*, *Implementation*, *Verification*, y *Operations*. Cada función contiene prácticas de seguridad específicas organizadas en flujos de trabajo, diseñados para orientar a las organizaciones en la integración de seguridad en cada fase del desarrollo de *software*.

C.1.1. Governance

Esta función se enfoca en los procesos y actividades relacionados con cómo una organización gestiona sus actividades generales de desarrollo de *software*. Incluye las siguientes prácticas:

- **Strategy and Metrics:** Construir un plan estratégico de seguridad del *software*, que incluya métricas para evaluar la efectividad del programa.
- **Policy and Compliance:** Alinear los requisitos legales y regulatorios con los estándares de seguridad internos de la organización.
- **Education and Guidance:** Proporcionar capacitación y recursos a los equipos de desarrollo para asegurar que el software se diseñe y desarrolle de manera segura.

Niveles de Madurez en Governance:

- **Nivel 1:** Identificar los objetivos y las métricas iniciales de seguridad y determinar un marco básico de cumplimiento.
- **Nivel 2:** Establecer una hoja de ruta estratégica unificada para la seguridad de aplicaciones, y expandir el cumplimiento a requisitos específicos de aplicaciones.
- **Nivel 3:** Alinear los esfuerzos de seguridad con indicadores organizacionales y valores de activos, y medir el cumplimiento tanto interno como externo.

C.1.2. Design

Esta función aborda cómo la organización define los objetivos de seguridad y establece el diseño de *software* dentro de los proyectos de desarrollo. Las prácticas incluyen:

- **Threat Assessment:** Identificación y evaluación de riesgos de seguridad a nivel de proyecto.
- **Security Requirements:** Definición de requisitos de seguridad para proteger servicios y datos esenciales.
- **Secure Architecture:** Selección y composición de componentes y tecnologías en el diseño arquitectónico de la aplicación.

Niveles de Madurez en Design:

- **Nivel 1:** Identificación de amenazas a nivel general y definición básica de requisitos de seguridad.
- **Nivel 2:** Estandarización y análisis organizacional de amenazas, junto con modelos de amenazas detallados.
- **Nivel 3:** Mejoras proactivas en la cobertura de amenazas, con revisiones periódicas de perfiles de riesgo y arquitecturas seguras.

C.1.3. Implementation

Esta función se centra en los procesos de construcción y despliegue de componentes de *software*. Las prácticas principales incluyen:

- **Secure Build:** Construcción de *software* de manera automatizada y repetible, utilizando componentes seguros.
- **Secure Deployment:** Asegurar que la integridad y la seguridad de las aplicaciones no se vean comprometidas durante el despliegue.
- **Defect Management:** Registro y análisis de defectos de seguridad del *software* para guiar las decisiones estratégicas.

Niveles de Madurez en Implementation:

- **Nivel 1:** Construcción y despliegue de *software* con procesos documentados y gestión de defectos básica.
- **Nivel 2:** Automatización de la construcción y despliegue, con pruebas de seguridad incluidas en cada etapa.
- **Nivel 3:** Integración total de pruebas de seguridad automatizadas y gestión proactiva de defectos en el proceso de desarrollo.

C.1.4. Verification

La función de verificación se centra en cómo una organización comprueba y prueba los artefactos producidos durante el desarrollo de *software*. Las prácticas incluyen:

- **Architecture Assessment:** Validación de la arquitectura de la aplicación para asegurar que cumple con los requisitos de seguridad y amenaza.
- **Requirements-driven Testing:** Pruebas basadas en requisitos de seguridad para asegurar que los controles implementados operan como se espera.
- **Security Testing:** Identificación de debilidades técnicas y de lógica empresarial mediante pruebas de seguridad manuales y automáticas.

Niveles de Madurez en Verification:

- **Nivel 1:** Pruebas de seguridad básicas tanto manuales como automáticas para descubrir defectos.
- **Nivel 2:** Automatización de pruebas de seguridad y pruebas manuales periódicas para componentes de alto riesgo.
- **Nivel 3:** Integración de pruebas de seguridad en el proceso de desarrollo y despliegue, asegurando una evaluación continua de seguridad.

C.1.5. Operations

Esta función abarca las actividades necesarias para mantener la confidencialidad, integridad y disponibilidad durante la vida operativa de una aplicación y sus datos. Las prácticas son:

- **Incident Management:** Gestión de incidentes para reducir el tiempo de detección y mejorar la respuesta ante incidentes de seguridad.
- **Environment Management:** Aseguramiento de la seguridad del entorno operativo mediante la aplicación de configuraciones seguras y parcheo continuo.
- **Operational Management:** Mantenimiento de la seguridad en las funciones de soporte operativo, incluyendo provisión y administración de sistemas.

Niveles de Madurez en Operations:

- **Nivel 1:** Detección y gestión de incidentes basada en mejores esfuerzos y endurecimiento básico del entorno.
- **Nivel 2:** Procesos de gestión de incidentes y configuración seguros, documentados y repetibles.
- **Nivel 3:** Detección y respuesta a incidentes proactiva y mejora continua en la gestión de la seguridad del entorno operativo.

C.2. Evolución y Futuro del OWASP SAMM

El *OWASP SAMM* está diseñado para evolucionar junto con las necesidades de la industria, permitiendo a las organizaciones adaptarse a un panorama de amenazas en constante cambio. Con una metodología basada en riesgos y una implementación flexible, *OWASP SAMM* ofrece una base robusta para construir programas de seguridad de software que sean medibles y efectivos, promoviendo un cambio cultural hacia una mayor seguridad en el desarrollo de software.

Análisis con Mobile Security Framework (MobSF)

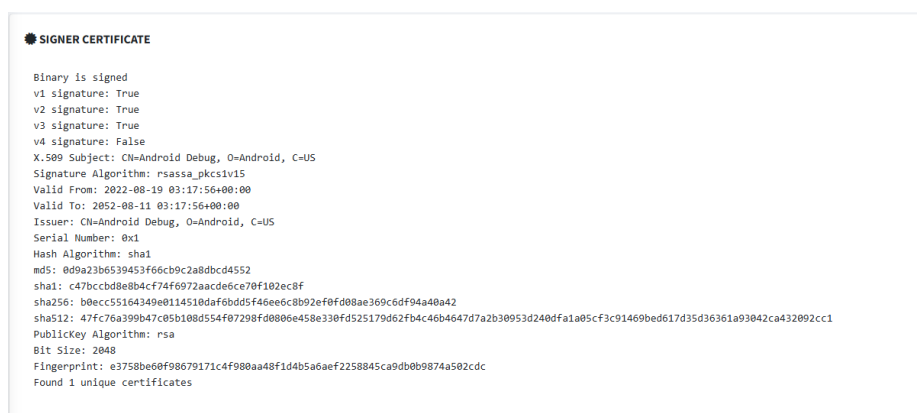


Figura D.1: Certificado de firma de MobSF.

Esta sección proporciona la información básica de los certificados, como las versiones, *hashes*, entre otros. Además, realiza revisiones para identificar configuraciones incorrectas.

APP APPLICATION PERMISSIONS						Search:
PERMISSION	STATUS	INFO	DESCRIPTION	CODE MAPPINGS		
android.permission.INTERNET	allowed	full internet access	Allows an application to create network sockets.	Show Log		
android.permission.READ_EXTERNAL_STORAGE	denied	read external storage contents	Allows an application to read from external storage.	Show Log		
android.permission.SYSTEM_ALERT_WINDOW	denied	display system-level alerts	Allows an application to show system-alert windows. Malicious applications can take over the entire screen of the phone.	Show Log		
android.permission.VIBRATE	allowed	control vibrator	Allows the application to control the vibrator.	Show Log		
android.permission.WRITE_EXTERNAL_STORAGE	denied	read/modify/delete external storage contents	Allows an application to write to external storage.	Show Log		
com.yongbumpark.ecsscanfrontend.DYNAMIC_RECEIVER_NOT_EXPORTED_PERMISSION	denied	Unknown permission	Unknown permission from android reference			

Showing 1 to 6 of 6 entries

Previous 1 Next

Figura D.2: Permisos de aplicaciones de MobSF.

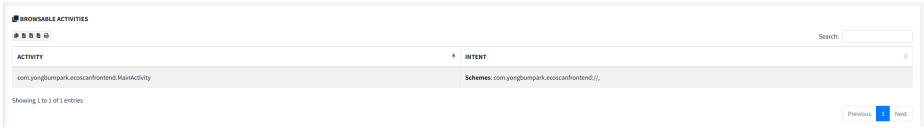
En este caso se muestran los permisos que utiliza la aplicación. Esto es útil para verificar los permisos necesarios en la aplicación, siendo ideal mantener permisos mínimos y solo los necesarios. Dependiendo del propósito de la aplicación, algunos permisos pueden representar riesgos que podrían ser explotados.



API	FILES
Base64 Decode	View Info
Base64 Encode	View Info
Crypto	View Info
Get System Service	View Info
Inter Process Communication	View Info
Java Reflection	View Info
Loading Native Code (Shared Library)	View Info
Local File I/O Operations	View Info
Message Digest	View Info
Starting Activity	View Info

Figura D.3: APIs de Android en MobSF.

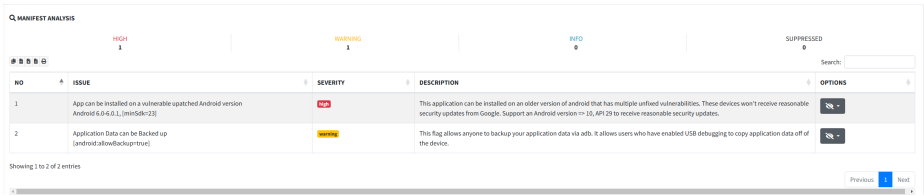
Esta sección muestra las *APIs* que se están utilizando, lo cual es útil al realizar un escaneo para investigar o atacar *APIs* críticas.



ACTIVITY	INTENT
com.yongjupark.ecoscarfrontend.MainActivity	Scheme: com.yongjupark.ecoscarfrontend://

Figura D.4: Actividades navegables en MobSF.

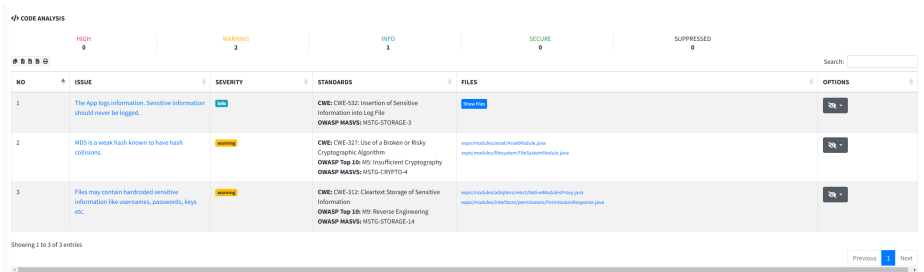
Aquí se muestran todas las actividades navegables, permitiendo observar cómo actúan dichas actividades en la aplicación.



ID	ISSUE	SEVERITY	DESCRIPTION	OPTIONS
1	App can be installed on a vulnerable/patched Android version	HIGH	This application can be installed on an older version of android that has multiple unfixed vulnerabilities. These devices won't receive reasonable security updates from Google. Support an Android version >= 15, API 29 to receive reasonable security updates.	View Info
2	Application Data can be Backed up (android:allowBackup=true)	WARNING	This flag allows anyone to backup your application data via adb. It allows users who have enabled USB debugging to copy application data off of the device.	View Info

Figura D.5: Análisis de manifiesto en MobSF.

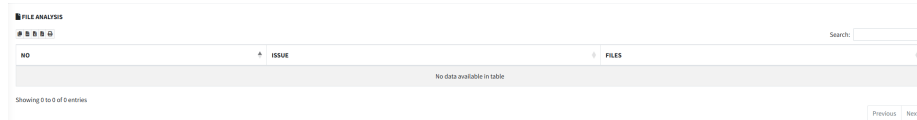
Esta parte muestra las vulnerabilidades detectadas en el análisis estático del manifiesto de la aplicación.



ID	ISSUE	SEVERITY	STANDARDS	FILES	OPTIONS
1	The App logs information. Sensitive information should never be logged.	HIGH	CWE: CWE-532: Invention of Sensitive Information into Log File OWASP MASVS: INSTG-STORAGE-3	View Info	View Info
2	MD5 is a weak hash known to have hash collisions.	WARNING	CWE: CWE-327: Use of a Broken or Risky Cryptographic Algorithm OWASP Top 10: IN: Insufficient Cryptography OWASP MASVS: INSTG-CRYPTO-4	View Info	View Info
3	Files may contain hardcoded sensitive information like usernames, passwords, keys etc.	WARNING	CWE: CWE-312: Hardcoded Storage of Sensitive Information OWASP Top 10: IN: Reverse Engineering OWASP MASVS: INSTG-STORAGE-14	View Info	View Info

Figura D.6: Análisis de código en MobSF.

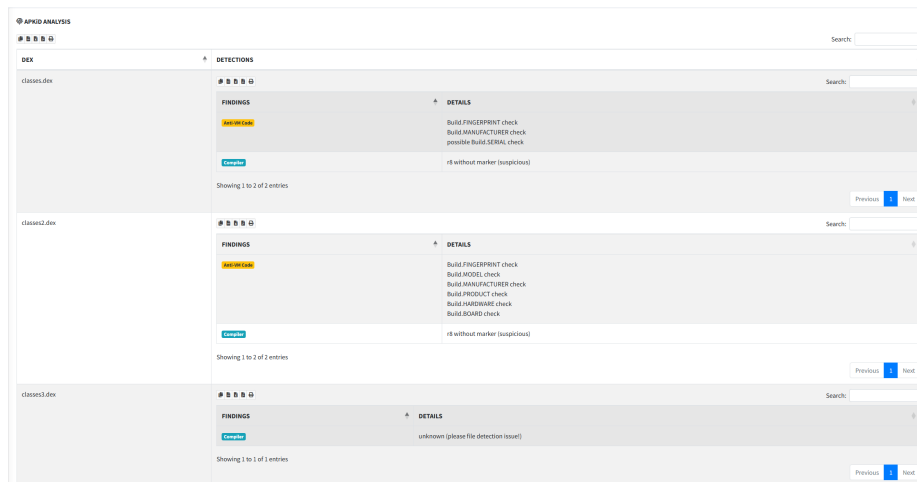
Es la sección donde *MobSF* realiza un análisis estático en todos los archivos *Java* descompuestos, mostrando cualquier vulnerabilidad o información sensible que se encuentre, como *logs* o datos confidenciales.



ID	ISSUE	FILES
No data available in table		

Figura D.7: Análisis de archivos en MobSF.

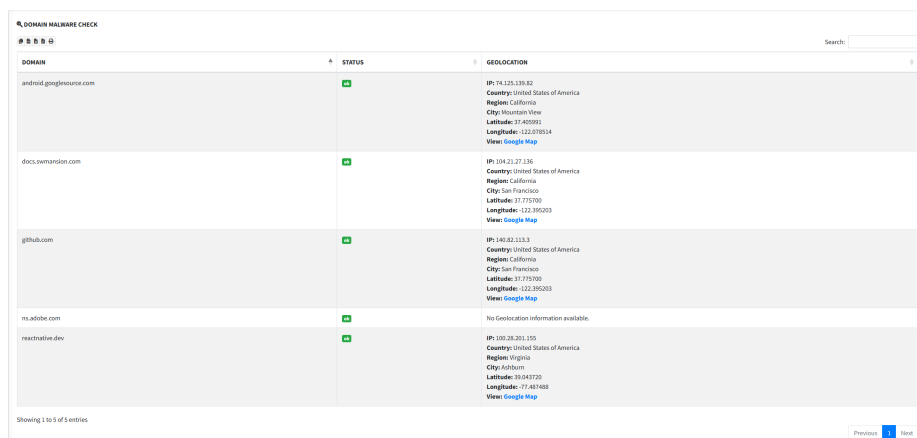
Aquí se listan todos los certificados incrustados en el código, incluyendo claves privadas u otros datos sensibles.



DEX	FINDINGS	DETAILS
classes.dex	Signature	BUILD.FINGERPRINT check BUILD.MANUFACTURER check possible BUILD.SERIAL check Complete r8 without marker (suspicious)
class2.dex	Signature	BUILD.FINGERPRINT check BUILD.MODEL check BUILD.MANUFACTURER check BUILD.PRODUCT check BUILD.HARDWARE check BUILD.BOARD check Complete r8 without marker (suspicious)
class3.dex	Signature	unknown (please file detection issue)

Figura D.8: Análisis de APKid en MobSF.

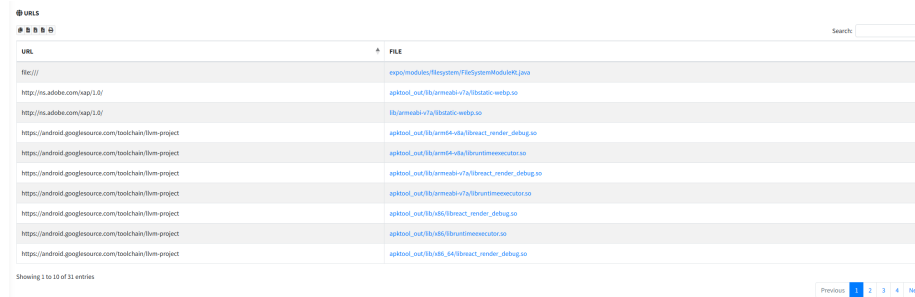
Esta sección muestra los archivos en los entornos de desarrollo y los tipos de compilador u ofuscador utilizados, brindando una buena perspectiva del comportamiento del código de la aplicación.



DOMAIN	STATUS	GEOLOCATION
android.googleusercontent.com	OK	IP: 14.125.138.82 Country: United States of America Region: California City: Mountain View Latitude: 37.405991 Longitude: -122.078554 View Google Map
docs.warshaw.com	OK	IP: 194.21.27.136 Country: United States of America Region: California City: San Francisco Latitude: 37.773930 Longitude: -122.395033 View Google Map
github.com	OK	IP: 140.82.113.3 Country: United States of America Region: California City: San Francisco Latitude: 37.773930 Longitude: -122.395033 View Google Map
mx.xdobe.com	OK	No Geolocation information available.
reactnative.dev	OK	IP: 192.28.201.155 Country: United States of America Region: Virginia City: Ashburn Latitude: 39.047220 Longitude: -77.481688 View Google Map

Figura D.9: Análisis de malware de dominio en MobSF.

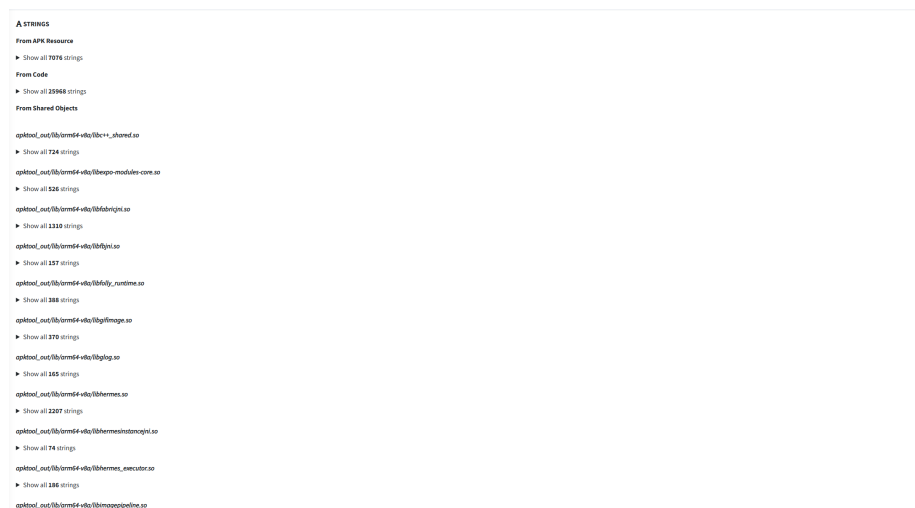
Esta sección extrae todos los dominios del binario y verifica si están en listas negras de dominios o *IPs* sospechosas.



URL	FILE
file://	expo/modules/FileSystem/FileSystemModule.js
http://m.adobe.com/api/1.0/	apktool_out\lib\armeabi-v7a\libstatic-webp.so
http://m.adobe.com/api/1.0/	lib\armeabi-v7a\libstatic-webp.so
https://android.googlesource.com/boochain/Item-project	apktool_out\lib\armeabi-v7a\libreact_render_debug.so
https://android.googlesource.com/boochain/Item-project	apktool_out\lib\armeabi-v7a\libruntimeexecutor.so
https://android.googlesource.com/boochain/Item-project	apktool_out\lib\armeabi-v7a\libreact_render_debug.so
https://android.googlesource.com/boochain/Item-project	apktool_out\lib\armeabi-v7a\libruntimeexecutor.so
https://android.googlesource.com/boochain/Item-project	apktool_out\lib\armeabi-v7a\libreact_render_debug.so
https://android.googlesource.com/boochain/Item-project	apktool_out\lib\armeabi-v7a\libruntimeexecutor.so
https://android.googlesource.com/boochain/Item-project	apktool_out\lib\armeabi-v7a\libreact_render_debug.so

Figura D.10: Listado de URLs en MobSF.

Lista todas las *URLs* detectadas en las distintas fuentes de código, proporcionando una visión de las conexiones externas de la aplicación.



STRINGS
From APK Resource
► Show all 7076 strings
From Code
► Show all 20968 strings
From Shared Objects
apktool_out\lib\armeabi-v7a\libc++_shared.so
► Show all 724 strings
apktool_out\lib\armeabi-v7a\liblog-jni.so
► Show all 526 strings
apktool_out\lib\armeabi-v7a\liblog.so
► Show all 2310 strings
apktool_out\lib\armeabi-v7a\liblog.so
► Show all 187 strings
apktool_out\lib\armeabi-v7a\liblog.so
► Show all 388 strings
apktool_out\lib\armeabi-v7a\liblog.so
► Show all 370 strings
apktool_out\lib\armeabi-v7a\liblog.so
► Show all 185 strings
apktool_out\lib\armeabi-v7a\liblog.so
► Show all 2287 strings
apktool_out\lib\armeabi-v7a\liblog.so
► Show all 74 strings
apktool_out\lib\armeabi-v7a\liblog.so
► Show all 186 strings
apktool_out\lib\armeabi-v7a\liblog.so

Figura D.11: Strings en MobSF.

Lista todas las cadenas de texto incrustadas en el código, mostrando cualquier *API* o secreto sensible que pueda representar un riesgo.

ANEXO E

Google Test de Phishing

Estas son las preguntas de phishing de google que se realizaron al equipo.

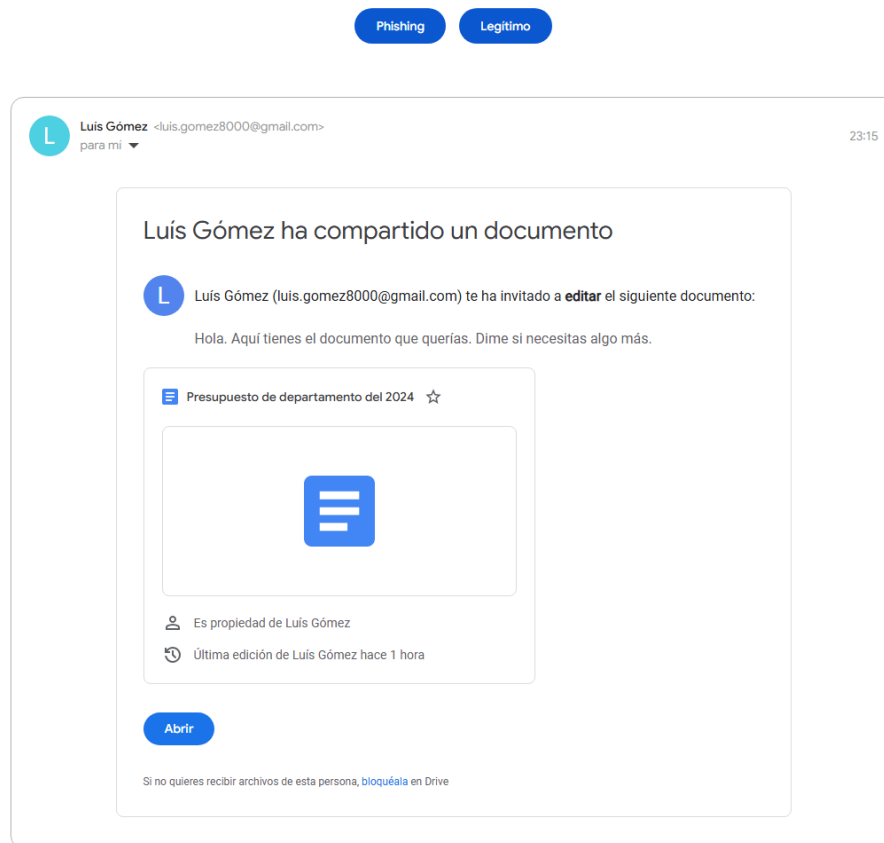


Figura E.1: Pregunta 1 del test de phishing de google.

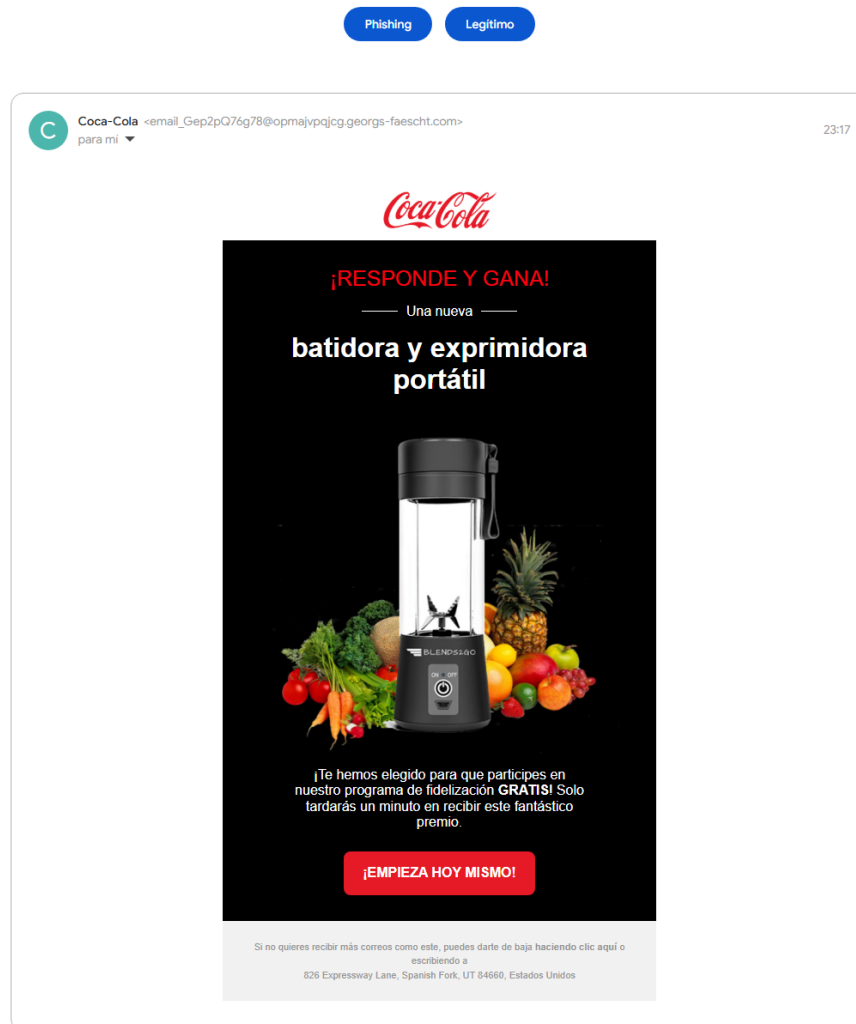


Figura E.2: Pregunta 2 del test de phishing de google.

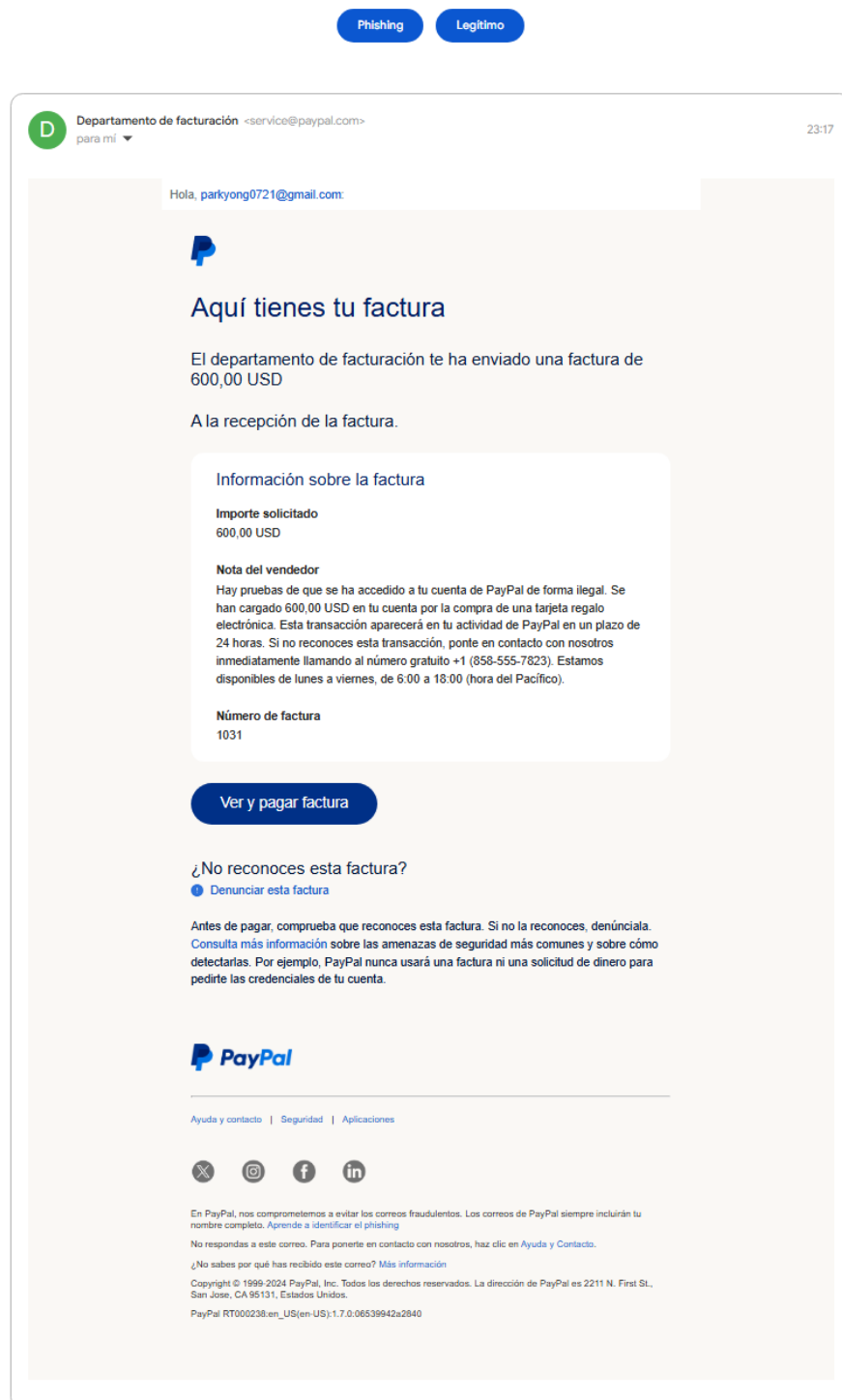


Figura E.3: Pregunta 3 del test de phishing de google.

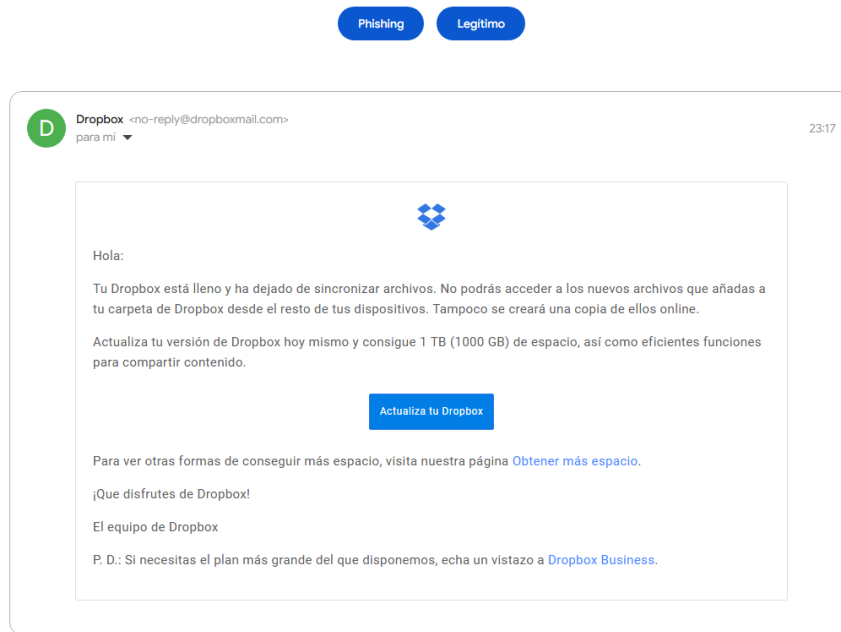


Figura E.4: Pregunta 4 del test de phishing de google.

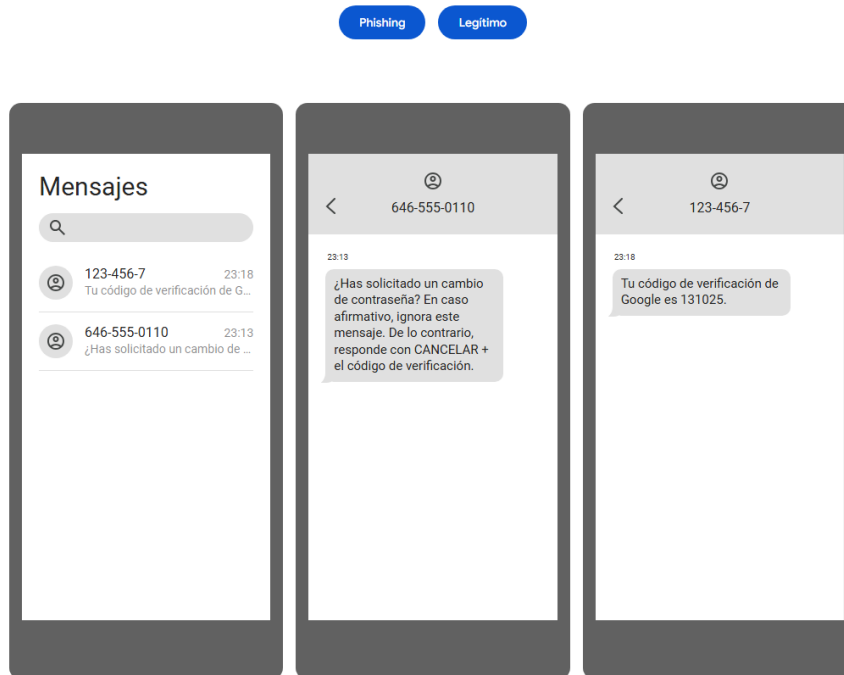


Figura E.5: Pregunta 5 del test de phishing de google.

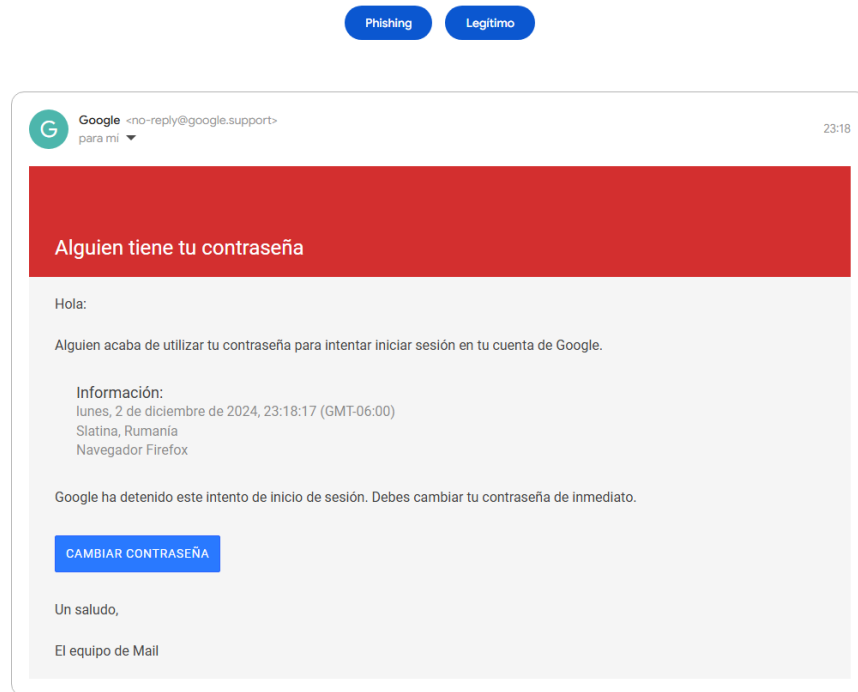


Figura E.6: Pregunta 6 del test de phishing de google.

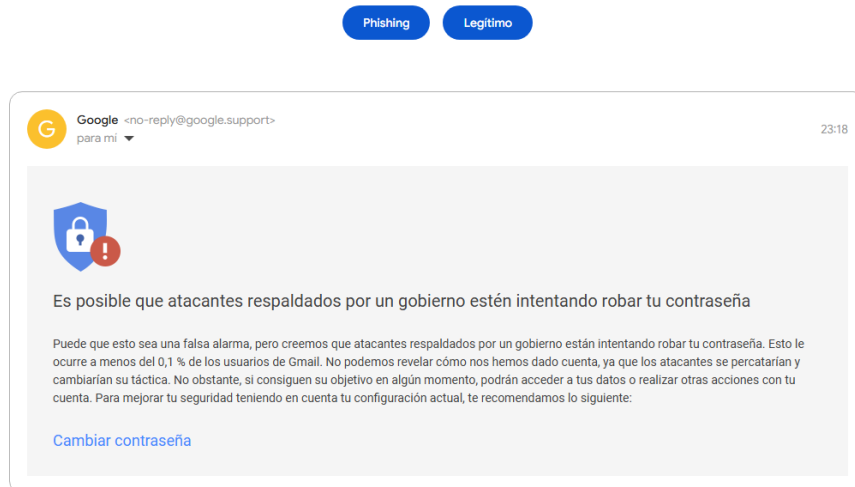


Figura E.7: Pregunta 7 del test de phishing de google.

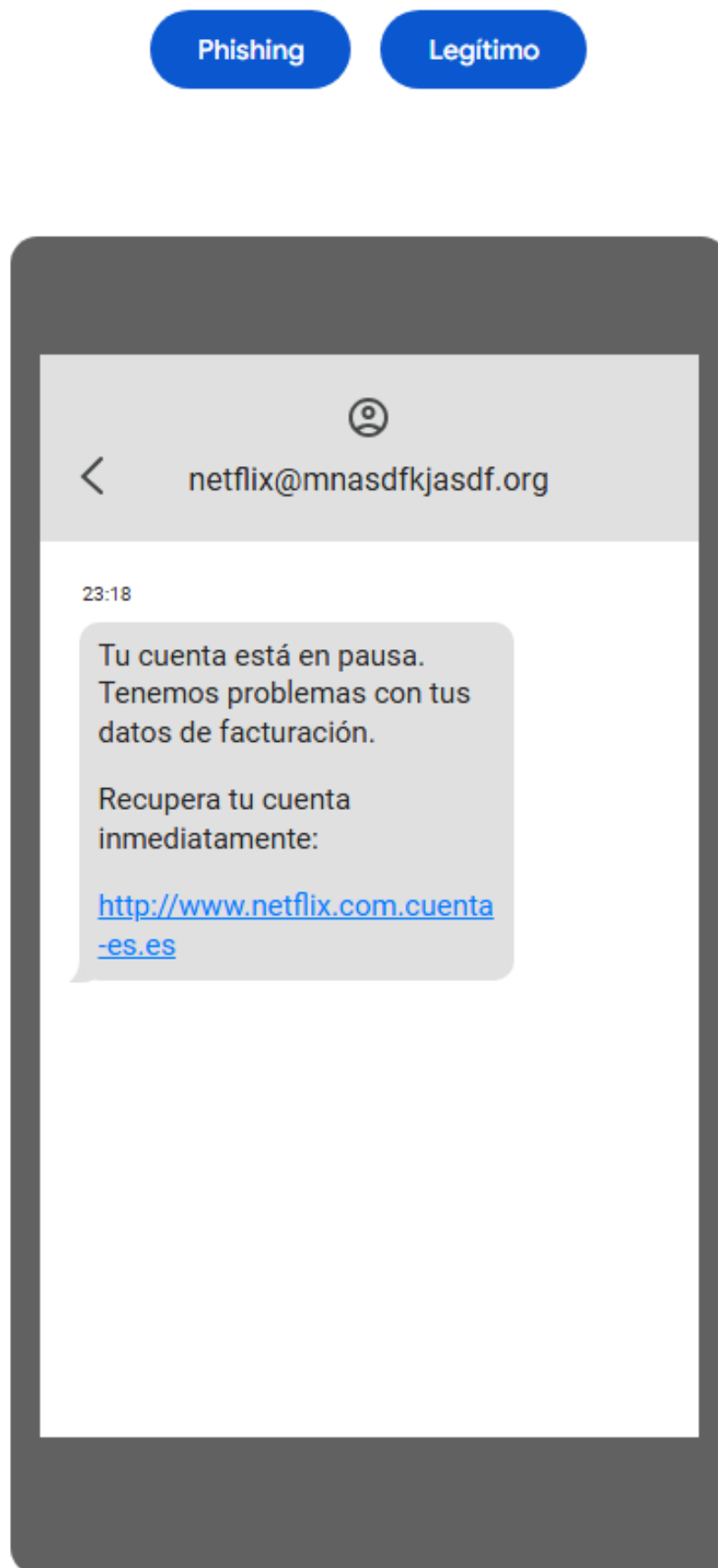


Figura E.8: Pregunta 8 del test de phishing de google.

Phishing

Legítimo

J

Juan Gómez <pixel-giveaway@google.com>
para mí

23:19

dic

3

mar

Pixel 7 Pro gratis - Entrevista de confirmación
[Ver en Google Calendar](#)

Cuándo

mar dic 3, 2024 De 13:00 a 14:00

Quién

Juan Gómez

✚

Si

A lo mejor

No

¡Enhorabuena! Te hemos seleccionado entre los candidatos para ganar un Pixel 7 Pro gratis. Esta invitación es para una entrevista de confirmación. [Haz clic aquí para unirte o gestionar tu cita.](#)

Cuándo
martes dic 3, 2024 · De 13:00 a 14:00

Invitados
Juan Gómez
YongBum
[Ver toda la información de los invitados](#)

Responder a parkyong0721@gmail.com

Si

No

Quizás

Más opciones

Invitación de [Google Calendar](#)

Te hemos enviado este correo porque te has suscrito a las notificaciones del calendario. Si quieres dejar de recibir estos correos, ve a la [configuración de Calendar](#), selecciona este calendario y cambia Otras notificaciones.

Si reenvías esta invitación, los destinatarios podrían enviar una respuesta al organizador para que los añada a la lista de invitados, invitar a otras personas independientemente del estado de su propia invitación, o cambiar tu respuesta de confirmación de asistencia. [Más información](#)

Figura E.9: Pregunta 9 del test de phishing de google.

Phishing

Legítimo

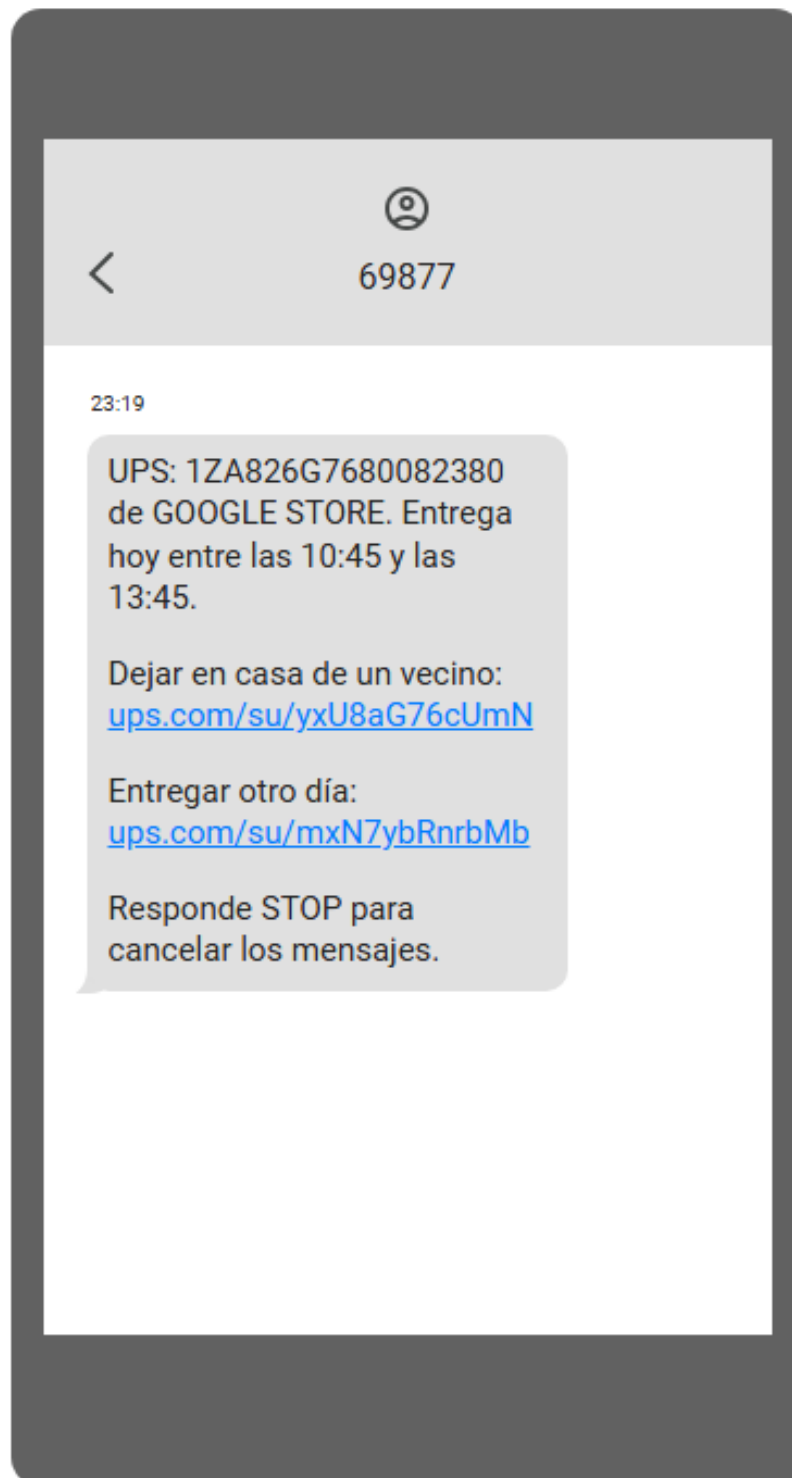


Figura E.10: Pregunta 10 del test de phishing de google.