

**DETECCIÓN DE VULNERABILIDADES DE LA CONFIGURACIÓN DE
SEGURIDAD EN APLICACIONES FLASK MEDIANTE EL USO DE
PRUEBAS DE MUTACIÓN**

HAROLD ANDRÉS MONTANO HURTADO

Código: 1968067

harold.montano@correounivalle.edu.co

JUAN STEBAN VELOZA GUEVARA

Código: 1968025

juan.veloza@correounivalle.edu.co

**FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA DE SISTEMAS Y COMPUTACIÓN
PROGRAMA ACADÉMICO DE INGENIERÍA DE SISTEMAS
TULUÁ – VALLE DEL CAUCA**

2024

**DETECCIÓN DE VULNERABILIDADES DE LA CONFIGURACIÓN DE
SEGURIDAD EN APLICACIONES FLASK MEDIANTE EL USO DE
PRUEBAS DE MUTACIÓN**

HAROLD ANDRÉS MONTANO HURTADO

Código: 1968067

harold.montano@correounivalle.edu.co

JUAN STEBAN VELOZA GUEVARA

Código: 1968025

juan.veloza@correounivalle.edu.co

Director

Carlos Andrés Delgado Saavedra, Ing.

carlos.andres.delgado@correounivalle.edu.co

**FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA DE SISTEMAS Y COMPUTACIÓN
PROGRAMA ACADÉMICO DE INGENIERÍA DE SISTEMAS
TULUÁ – VALLE DEL CAUCA**

2024

Agradecimientos

El paso por la carrera universitaria y la ejecución de este trabajo de grado, no habrían sido posibles sin el apoyo de muchas personas que nos han brindado su ayuda, conocimiento y motivación:

Principalmente a Dios y al universo por darnos la salud, la voluntad y oportunidad de estudiar esta carrera.

A nuestras familias por la confianza depositada en nosotros y por su esfuerzo, dedicación y apoyo a lo largo de la vida.

Al profesor Carlos Andrés Delgado Saavedra, nuestro Director, por su profesionalismo, paciencia, apoyo y colaboración en todo momento a lo largo de la carrera y especialmente en este proyecto.

A todo el gremio profesoral que hizo parte del aprendizaje y motivación, para irnos inculcando la pasión por la investigación y la pasión por querer resolver problemas visualizados con la ayuda de los temarios obtenidos a lo largo de la carrera.

A la Universidad del Valle sede Tuluá y a todos nuestros profesores, por permitirnos adquirir conocimientos, ayudarnos a crecer profesional y personalmente.

Tabla de Contenido

Resumen	VI
Introducción	VII
1. Contexto y Objetivos	1
1.1. Problema de Investigación	1
1.1.1. Descripción del problema	1
1.1.2. Formulación del problema	2
1.2. Objetivos	3
1.2.1. Objetivo general	3
1.2.2. Objetivos Específicos y Resultados Esperados	3
1.3. Alcance	3
1.4. Estructura del documento	4
2. Marco Referencial	5
2.1. Marco Teórico	5
2.1.1. Definición de Sistema	5
2.1.1.1. Sistema de Software	6
2.1.1.2. Seguridad en Sistemas de Software	6
2.1.2. Vulnerabilidades en los Sistemas de Software	6
2.1.3. Pruebas de Mutación	7
2.1.3.1. Tipos de Pruebas de Mutación	8
2.2. Glosario	9
2.3. Antecedentes	10
3. Seguridad en Flask	13
3.1. Arquitectura de Flask	13
3.2. Configuraciones Presentes en Flask	14
4. Diseño y Desarrollo del Proyecto	20
4.1. Aplicación de la Metodología XP	20
4.1.1. Exploración	21

4.1.1.1.	Historias de Usuario	22
4.1.1.2.	Definición de Tecnologías	24
4.1.1.3.	Librería AST	25
4.1.1.4.	Arquitectura del Sistema	28
4.1.2.	Iteraciones	30
4.1.2.1.	Descripción de la Herramienta Desarrollada	32
4.1.2.2.	Documentación del Prototipo Aplicación de la Detección de Vulnerabilidades [1]	33
4.1.2.3.	Documentación de Caso de Estudio Desarrollado en Flask [1]	34
4.1.2.4.	Configuraciones de Seguridad a Tener en Cuenta en CORS	36
5.	Operadores de Mutación	38
6.	Validación de Pruebas	42
7.	Conclusiones y Trabajos Futuros	48
7.1.	Conclusiones	48
7.2.	Trabajos futuros	50
	Bibliografía	51

Lista de Figuras

3.1. Arquitectura MVC	14
4.1. Extreme Programming XP	21
4.2. Arquitectura del Sistema	29
4.3. Arquitectura Descrita de AST en el Proyecto	30
4.4. Codigos Presentes en Peticiones	36
5.1. Pruebas de Mutación	39
5.2. App Run	41
5.3. Inyección de Mutantes	41

Lista de Tablas

1.1. Objetivos Específicos	3
2.1. Comparativa Antecedentes	11
3.1. Configuraciones de Seguridad Comunes en Flask	16
3.2. Vulnerabilidades de las Configuraciones de Seguridad Comunes en Flask . . .	18
4.1. Historia de Usuario N°1	22
4.2. Historia de Usuario N°2	23
4.3. Historia de Usuario N°3	23
4.4. Historia de Usuario N°4	24
4.5. Tabla Comparativa de Librerías de Prueba	27
4.6. Tarea N°1 - Iteración 1	31
4.7. Tarea N°2 - Iteración 1	31
4.8. Tarea N°3 - Iteración 2	32
4.9. Tarea N°4 - Iteración 2	32
5.1. Operadores de Mutación de las Configuraciones	40
6.1. Prueba 1	42
6.2. Prueba 2	43
6.3. Prueba 3	43
6.4. Prueba 4	44
6.5. Prueba Unitaria Origin	44
6.6. Prueba Unitaria Methods	45
6.7. Prueba Unitaria Allow Headers	45
6.8. Prueba Unitaria Supports Credentials	45
6.9. Prueba Unitaria Max Age	46
6.10. Prueba Unitaria Send Wildcard	46
6.11. Prueba Unitaria Automatic Options	46
6.12. Prueba Unitaria Vary Headers	47

Resumen

Este proyecto se enfocó en evaluar la seguridad de las aplicaciones Web desarrolladas en el marco de Flask. Implementando pruebas de mutación, en las cuales se buscó identificar vulnerabilidades y malas prácticas comunes en la configuración incorrecta de seguridad. El proceso implica el desarrollo de operadores de mutación, la categorización de casos de configuración incorrecta, y la creación de casos de prueba específicos. La integración de las pruebas de mutación durante el desarrollo permitirá la identificación temprana de posibles errores, evaluando su impacto en las etapas iniciales. Además, estos operadores serán integrados en una aplicación que permita detectar las vulnerabilidades en las configuraciones de seguridad. La validación de la efectividad de las pruebas se llevará a cabo mediante métricas asociadas a las pruebas de mutación, tales como el análisis de cobertura de vulnerabilidades.

Este proyecto representa una contribución a las prácticas de seguridad en el desarrollo de aplicaciones Web, destacando la utilidad y beneficios de las pruebas de mutación en este contexto. Para el desarrollo del proyecto se empleó la metodología XP teniendo en cuenta la facilidad del trabajo en parejas, que proporciona esta metodología.

Introducción

Con el avance tecnológico presente en la actualidad, se puede evidenciar como las aplicaciones Web desempeñan un papel fundamental en la vida cotidiana de muchas empresas, llevando a cabo desde transacciones financieras hasta comunicaciones. Por ende, surge la necesidad apremiante de abordar los desafíos asociados con garantizar la seguridad de los datos que se manejan, considerando el constante aumento de amenazas cibernéticas y la sofisticación de los ataques.

En la actualidad, la seguridad de estas se puede ver amenazada por vulnerabilidades que pueden surgir debido a configuraciones inadecuadas. De acuerdo con esto, las pruebas de software en seguridad son una estrategia para evaluar la robustez de las aplicaciones Web al realizar una evaluación de las prácticas de software que se han implementado.

Se requiere brindar beneficios significativos, mejorando la seguridad general de las aplicaciones Web y contribuyendo al conocimiento en el campo de las pruebas de mutación. Con este fin, se desarrollarán un conjunto de operadores de mutación enfocados en la detección de malas prácticas de seguridad en el código de diseño de las aplicaciones basadas en el Framework Flask de Python, el cual es uno de los más utilizados para su construcción. La estructura de este trabajo de grado tomará en cuenta secciones claves, como la revisión de literatura, descripción del problema, la metodología que se usó, los resultados esperados y las conclusiones.

Capítulo 1

Contexto y Objetivos

1.1. Problema de Investigación

1.1.1. Descripción del problema

Actualmente las aplicaciones web se encuentran dentro del desarrollo de software, pues según Amazon [2] proporciona beneficios que incrementan hasta en un 50 % el rendimiento y alcance de estas, comparadas con aplicaciones de escritorio; estas aplicaciones administran datos sensibles que pueden ser accedidos sin autorización por terceros sino se siguen las prácticas de seguridad necesarias para controlar el acceso y flujo de datos.

Tomando en cuenta que uno de los Framework más populares para la generación de aplicaciones Web es Flask de Python [3], resulta de gran importancia realizar el análisis de seguridad que se puede reflejar en este tipo de aplicaciones web. Se puede visualizar que durante el proceso del desarrollo de las aplicaciones, pueden volverse cada vez más robustas y complejas, es por esto que pueden aparecer brechas, por lo que con el pasar del tiempo, avanzando en el desarrollo van apareciendo vulnerabilidades por diversos factores que no se tienen en cuenta al momento de la implementación a código de la idea del sitio web; como comentan El Mahjoubi Omar & Duch Jordi en [4] este tipo de debilidades pueden ser generadas por diferentes tipos de razones, como lo pueden ser el diseño de interfaz, errores al momento de la programación e incluso una indebida planeación al momento de estructurar el desarrollo del proyecto. Es por esto, que la detección temprana de este tipo de vulnerabilidades es fundamental para evitar posibles fallos en versiones posteriores del desarrollo de la aplicación, fallos de los cuales se pueden aprovechar los atacantes para extraer información o hacer daños que tengan como fin dejar inutilizable la aplicación.

Con el fin de proteger los datos que intercambian las aplicaciones se realizan diversos tipos de pruebas [5] las cuales proporcionan un buen nivel de seguridad, ya que nunca se ha visto asociado con niveles de vulnerabilidad, sin embargo manejan contactos directos con el servicio de soporte por si se encuentra alguna falencia de este tipo; entre las más usadas se encuentran

pruebas unitarias (que analizan de forma individual minuciosamente líneas de código) y las pruebas de seguridad, las cuales son normalmente apoyo fundamental para proporcionar la integridad de las aplicaciones Web; sin embargo, éste tipo de pruebas pueden no resultar totalmente confiables, esto debido a que el nivel de cobertura no es lo suficientemente amplio como para evaluar todas las posibilidades que hay a la hora de medir el nivel de seguridad presente en una aplicación. Actualmente las herramientas que permiten desarrollar aplicaciones web son los framework, que ayudan al ahorro de tiempo automatizando muchas tareas. Las actualizaciones que se le realizan a estas herramientas son el inconveniente más notorio porque deben hacerse gradualmente y no se realizan de una forma automática.

Uno de los problemas más conocidos en seguridad está relacionado con la configuración, la cual ayuda a regular el acceso a los datos, el nivel de acceso que tienen los usuarios, la autenticación que se requiere y las credenciales que tienen los usuarios. Esta configuración puede variar dependiendo del entorno que se maneje para el diseño de la aplicación, pues para que la aplicación se inicie Flask debe tener activa la configuración de seguridad desde un principio [6].

Las pruebas de mutación son una estrategia para evaluar la calidad de las diferentes pruebas que se realizan en un aplicativo, al realizar cambios dentro de las aplicaciones para evaluar si las pruebas de software pueden detectarlas [7]. Este tipo de pruebas son pertinentes para la evaluación de la seguridad en las aplicaciones Web, a través de comprobaciones efectuadas en el código fuente que permiten encontrar fallas que arrojen las pruebas unitarias y dejen al descubierto las vulnerabilidades.

1.1.2. Formulación del problema

Con las consideraciones anteriores sobre el problema a tratar, surge el interrogante para describir el problema y buscar una idea más clara para una posible solución:

¿De qué manera las pruebas de mutación pueden evaluar problemas de configuración de la seguridad de una aplicación Web tipo Flask?

Esta pregunta proviene de haber mencionado que por más pruebas de seguridad que se le apliquen al software, puede que no exista una total cobertura, por eso se desea buscar una solución con algún tipo de prueba a desarrollar para reducir la preocupación por la posibilidad de que algunos errores pasen desapercibidos y puedan llevar a tener vulnerabilidades no detectadas. Se busca esta detección temprana de errores puesto que en cualquier fase del desarrollo pueden aparecer algún tipo de código de programación que pueda incidir en generar vulnerabilidades que pongan en riesgo los datos de los usuarios que usen la aplicación.

1.2. Objetivos

1.2.1. Objetivo general

Implementar pruebas de mutación para evaluar la calidad y seguridad de un software web diseñado con Flask.

1.2.2. Objetivos Específicos y Resultados Esperados

Objetivo Específico	Sección del Documento	Página Objetivo
Seleccionar un conjunto de malas prácticas comunes de configuración incorrecta de seguridad en Aplicaciones Flask.	Capítulo 3	Página 13
Describir un conjunto de operadores de mutación basados en código para la evaluación de configuraciones incorrectas de seguridad seleccionadas.	Capítulo 4	Página 20
Desarrollar los operadores de mutación.	Capítulo 5	Página 37
Validar la efectividad de las pruebas a través de la efectividad mediante un caso de estudio.	Capítulo 6	Página 41

Tabla 1.1: Objetivos Específicos

1.3. Alcance

Mediante este trabajo de grado, se implementaron y evaluaron las pruebas de mutación en aplicaciones web desarrolladas en Python con el framework Flask. El alcance abarca desde la creación de un caso de estudio desarrollado con Flask, desarrollando a su vez un entorno de pruebas de mutación cuyos operadores están basados en la modificación del código fuente (backend), permitiendo detectar vulnerabilidades de las configuraciones de seguridad (CORS) en la aplicación web, teniendo como guía la aplicación original versus la aplicación mutada (aplicación que se le hicieron cambios para buscar errores).

A través de estas mutaciones desarrolladas se aplicaron técnicas de pruebas unitarias propias de Python las cuales respaldan al software, enfatizando en identificar posibles errores en la lógica, la seguridad y la calidad usada para la aplicación evaluada en el código. El proyecto tiene el enfoque de mostrar herramientas de pruebas ya existentes, así como crear casos de pruebas específicos en un prototipo de software que permita detectar vulnerabilidades en las configuraciones de seguridad para la aplicación con dicho framework. Cabe aclarar que, para evaluar las pruebas de mutación, es necesario llevar a cabo un conjunto de pruebas ya diseñadas, buscando también evaluar las ventajas y asimismo sus desventajas en el contexto de aplicarlas en la web.

1.4. Estructura del documento

El presente documento se encuentra dividido en siete capítulos: En el primero se muestra la definición del problema y los objetivos del proyecto. En el segundo, se encuentra el marco teórico y los antecedentes. En el tercero, se muestra el proceso de análisis de las pruebas de mutación. En el cuarto, se presenta el diseño y el desarrollo del proyecto. En el quinto, se analiza la documentación y el proceso usado para implementar el prototipo de aplicación. En el sexto se encuentran documentadas las pruebas realizadas para mostrar la eficacia del aplicativo que permita detectar vulnerabilidades. En el séptimo se encuentran las conclusiones y trabajos futuros.

Capítulo 2

Marco Referencial

2.1. Marco Teórico

Las aplicaciones web y la detección de vulnerabilidades son un problema por atenderse pues es la seguridad de los usuarios finales la que está en juego, con esto hay que desarrollar formas para lograr obtener un mejor producto de software como lo puede ser una aplicación web, un camino a tomar para detectar dichas vulnerabilidades en su seguridad es a través de las pruebas de mutación. El diseño de pruebas de mutación es necesario para lograr robustecer la seguridad de la aplicación web, con el fin de aumentar la dificultad de ser vulnerado y expuesto ante cualquier ciberataque o intento de este. Para tener en claro la seguridad en la web, es necesario integrar prácticas de seguridad desde etapas iniciales del desarrollo del proyecto; al hacerlo, se busca incrementar la dificultad para ser vulnerado y se reduce la exposición ante ciberataques, mejorando la confianza del usuario final en el producto.

La seguridad web además de ser un componente crítico en el desarrollo de software, también tiene cierta susceptibilidad debido a su exposición en línea y la interacción constante con usuarios externos, lo cual, las convierte en un objetivo frecuente para atacantes que buscan explotar puntos débiles. Sin embargo, para detectar vulnerabilidades algunas técnicas de uso de pruebas manuales o herramientas de análisis estático y dinámico han quedado obsoletas, siendo insuficientes para abordar la evolución de ciberataques modernos. Buscando nuevas herramientas, el uso de técnicas avanzadas ha sido una solución, como lo son la instauración de las pruebas de mutación, las cuales han ganado relevancia por la capacidad de simular posibles incursiones de ataques, evaluando el sistema ante su robustez del código fuente frente a posibles amenazas.

2.1.1. Definición de Sistema

Entendiendo que la complejidad del funcionamiento e interacciones que poseen estos productos de software se le toma como sistemas, tal y como Johansen habla en [8], muestra cómo

los sistemas tienen subsistemas, que a su vez son sistemas con más subsistemas, los procesos de retroalimentación que poseen y la sinergia que debe existir dentro de un sistema para que todo funcione correctamente; cabe mencionar que en teoría cada uno de los sistemas tienden a degenerarse o a volverse obsoleto.

2.1.1.1. Sistema de Software

Como se mencionó, los sistemas tienden a desgastarse, y es ello lo que se trata de mantener bajo control con los productos de software. El desarrollo del ciclo de vida del software ayudó a tener un mejor control durante el desarrollo de software, siendo Royce quien propuso el primer modelo, el modelo cascada [9], que consistía en un desarrollo de forma secuencial y rígido, que pasaba por la fase de análisis, diseño, implementación, pruebas, despliegue y mantenimiento, sin posibilidad de regresar, es un modelo simple de seguir pero presenta grandes dificultades para adaptarse a los cambios; por otra parte se tiene el modelo en espiral de Boehm encontrado en [10], el cual es un proceso iterativo que continuamente se evalúa y mejora. Este modelo incluye las fases del modelo de cascada bajo un enfoque basado en un análisis y gestión, de riesgos, la cual forma una espiral que se repite hasta alcanzar el producto deseado. El modelo espiral es mucho más flexible que el modelo cascada permitiendo incorporar cambios, adaptarse a variaciones y expectativas de los clientes, aunque requiere de más tiempo y recursos.

2.1.1.2. Seguridad en Sistemas de Software

En el tema de la seguridad y la ciberseguridad, estas están relacionadas, más no necesariamente son lo mismo. La seguridad está encargada de proteger información en general, sin importar el formato o el medio en que se encuentre. La ciberseguridad se encarga de proteger información en un ámbito digital, es decir, en las redes, los dispositivos y los sistemas informáticos.

2.1.2. Vulnerabilidades en los Sistemas de Software

Una forma desarrollada para detectar vulnerabilidades son las pruebas de mutación, llamadas así por la forma en como al cambiar el código fuente se obtiene una variación en algunas funciones, procesos o aspectos de un sistema como se describe en [11]; estas pruebas pueden ser positivas y negativas, así mismo al desarrollar mutaciones en el código de una aplicación web o casi cualquier sistema informático, se verá reflejado en el comportamiento del sistema. Para ello se debe seguir una secuencia establecida, como la que encontrada en [12]: donde se introduce una prueba en el programa P, (como primer paso) se elige el conjunto de operadores que se van a mutar, (segundo paso) se crea las formas en las que mutará o analizará las posibles mutaciones del código, (tercer paso) se debe filtrar toda mutación equivalente o redundante, (cuarto paso) se diseñan pruebas con la capacidad de matar las mutaciones, (quinto paso) se ejecutan las pruebas con todas las mutaciones, (sexto paso) se califica el funcionamiento de las mutaciones, (séptimo paso) se reduce o prioriza las mutaciones con menos o mayor eficiencia,

respectivamente, en el siguiente paso (octavo paso) desde el cuarto hasta el séptimo se repiten hasta que el proceso dé como resultado una puntuación de mutación aceptable, (noveno paso) finalizando el proceso, se le pide al usuario que evalúe si los resultados de las ejecuciones de las pruebas fueron los esperados, y por ultimo este se refiere al llamado problema de creación del oráculo de prueba que involucra al evaluador para afirmar el comportamiento de la ejecución de la prueba. En caso de que se encuentren fallas, los desarrolladores deben solucionar los problemas y reiniciar el proceso hasta que se alcance un nivel de puntuación aceptable y no se pueda encontrar ninguna falla.

Tomando en cuenta lo dicho en [13] las aplicaciones pueden ser consideradas vulnerables si existen herramientas con las que se crearon partes del código fuente desactualizadas o se usan framework, los cuales deben estar en constante actualización de forma manual; otra razón para catalogar a una aplicación como vulnerable es cuando se instalan funciones innecesarias, donde pueden haber puertos expuestos, servicios o páginas en segundo plano que pueden ser fácilmente vulneradas y la forma que puede distinguirse como sencilla de vulnerar es cuando los datos puestos por los usuarios en la página para posteriormente ser introducidos en la base de datos, no están siendo encriptados, ya que de este modo los atacantes cibernéticos pueden tener un acceso a la base de datos y ver sin ningún problema los datos almacenados allí. Para prevenir alguna de estas características es necesario tener actualizadas las herramientas que se usan a la hora de desarrollar, revisando los permisos de almacenamiento en la nube que se le conceden en el código fuente; tener configurada la automatización que verifique lo efectivo de herramientas en cada entorno de trabajo dentro de la aplicación.

Considerando que la seguridad informática tiene tan alta capacidad de temario, como el descrito en [14], siendo unas de ellas: Las secuencias de comandos en sitios cruzados (a cerca de las etiquetas HTML arbitrarias) o la falsificación de solicitudes en sitios cruzados (sobre la delicada información almacenada en las cookies). El manejo de las configuraciones que se mencionan en [14] es una forma directa de tratar las consideraciones de seguridad mencionadas; también se toma en cuenta que lo que se permite hacer con Flask respecto a la seguridad, encontrado en [14], entre sus posibilidades existen: autenticación basada en sesión, gestión de roles y permisos, pasajero de contraseña, autenticación HTTP básica, autenticación basada en tokens, entre muchas más.

2.1.3. Pruebas de Mutación

Son una técnica de pruebas de software que tienen como objetivo introducir modificaciones controladas en el código fuente o principal de un software para evaluar que tan efectivas son las funciones que se tienen programadas en los casos de prueba. Mediante este método se simulan errores potenciales en el código y verifica si los casos de prueba son lo suficientemente robustos como para detectar estos errores [15].

Este proceso de pruebas de mutación inicia con la creación de ambientes o variantes del código original, llamadas mutantes, en las que se introduce una única modificación. Estas modificaciones suelen ser cambios en el código, como reemplazar operadores, variar constan-

tes, o alternar condiciones lógicas. Cada mutante se prueba con la suite de casos de prueba original para observar si el cambio es detectado. Si el ambiente de pruebas identifica un fallo en el mutante, se considera asesinado; de lo contrario, el mutante sobrevive, lo cual indica posibles falencias en la cobertura de los casos de prueba. Se permite medir la efectividad de los casos de prueba al evaluar si estos pueden detectar errores introducidos en el código. La medida principal en las pruebas de mutación es el índice de mutación, el cual calcula el porcentaje de mutantes asesinados por la suite o ambiente de pruebas respecto al total de mutantes generados. Este índice permite identificar áreas donde la cobertura de pruebas es insuficiente y posibles puntos vulnerables en el código que podrían resultar en errores en un entorno de producción [16].

Para el desarrollo e implementación de las pruebas de mutación, se requiere de desarrollar o hacer uso de un motor de pruebas, el cual se encarga de introducir de forma automática las mutaciones en el código fuente, permitiendo ejecutar los casos de prueba para permitir evaluar los resultados. Los motores de prueba utilizan operadores de mutación, los cuales funcionan como un conjunto de reglas específicas que permiten introducir mutaciones al código.

2.1.3.1. Tipos de Pruebas de Mutación

Existen diversos tipos de pruebas de mutación las cuales se clasifican en varios tipos dependiendo de su nivel y el enfoque al cual se esté apuntando.

- **Pruebas de Mutación Enfocadas en Código Fuente:** Mediante la implementación de este tipo de pruebas, se busca realizar cambios controlados dentro del código fuente. Se modifican fragmentos específicos relacionados con diversas variables, como lo pueden ser condicionales, booleanos, configuraciones, entre otras variables que se deseen cambiar.

En este proyecto, se hizo uso de este tipo de pruebas de mutación, en el cual se hacen cambios directos simulando posibles errores en la configuración. Este tipo de pruebas permite detectar errores en configuraciones especialmente de seguridad como se mencionó anteriormente. Las pruebas de mutación de código fuente realizadas fueron propuestas en este proyecto para evaluar configuraciones inseguras comunes en aplicaciones Flask, presentando una herramienta que permite a los desarrolladores identificar y corregir configuraciones de seguridad inadecuadas. Contribuyendo a la calidad y seguridad del software.

- **Pruebas de Mutación en Esquemas de Datos:** Consiste en la modificación de los esquemas de bases de datos y estructuras de datos presentes en el código de la aplicación, buscando identificar problemas de consistencia en la manipulación de los datos.
- **Pruebas de Mutación a Nivel de Interfaz:** Están centradas en modificar elementos presentes en la interfaz gráfica del usuario, como lo son formularios o puntos de entrada de datos, buscando identificar problemas de validación y autenticación a nivel

de interfaz. Dicho enfoque es común en aplicaciones donde la seguridad del usuario final es primordial, como lo son el comercio electrónico y sectores bancarizados.

- **Pruebas de Mutación en Sistemas Distribuidos:** Se enfatizan en sistemas que afectan la comunicación entre microservicios, pues en estos casos son diversos servidores trabajando en conjunto para crear un mismo sistema. Este enfoque permite evaluar la robustez de una aplicación, dicho enfoque de pruebas no es propuesto para aplicaciones con arquitectura monolíticas.

2.2. Glosario

- **Aplicación Web** [17]: Es un software que se ejecuta en un servidor web y se puede acceder por medio de un navegador de internet. Diferenciadas de aplicaciones de escritorio, no requieren de instalación en el dispositivo del usuario. Las aplicaciones web varían desde simples sitios web informativos hasta complejas plataformas interactivas incluyendo funciones de comercio, gestión de bases de datos, etc.
- **World Wide Web (WWW)** [18]: también conocido como web, es un sistema de documentos de hipertexto interconectados a los que se accede a través de Internet.
- **Sitio Web** [19]: Conjunto de documentos electrónicos de hipertexto que hacen referencia a un tema determinado. Generalmente están definidos por una página de bienvenida (home page), la cual se muestra inicialmente y desde la cual se puede acceder a las demás paginas por medio de enlaces (links). Un Sitio Web poseen un nombre de dominio, conocido popularmente como dirección de internet.
- **Framework Flask** [20]: Es un framework que está diseñado para el desarrollo de aplicaciones web en Python. El cuál es ligero y flexible, permitiendo a los desarrolladores utilizar módulos y bibliotecas según sus necesidades.
- **Seguridad En Aplicaciones Web** [21]: Hace referencia a las medidas y prácticas implementadas para proteger aplicaciones web de ciberataques y accesos no autorizados.
- **Pruebas de Mutación** [22]: Son un tipo de técnica para probar el software, que es utilizada para evaluar la calidad de los casos de prueba existentes. Busca involucrar la introducción de pequeños cambios en el código fuente del programa para crear versiones modificadas llamadas "mutantes".
- **Operadores de Mutación** [23]: Son los mecanismos utilizados para realizar cambios en el código fuente durante la ejecución de las pruebas de mutación. Estos cambios incluyen la modificación de operadores aritméticos, la eliminación de líneas de código, la alteración de condiciones lógicas, entre otros.
- **Metodología XP (Extreme Programming)** [24]: Es una metodología de desarrollo de software que busca promover una alta calidad del software y la capacidad de respuesta a los cambios de requisitos del cliente.

- **Análisis de Vulnerabilidades** [25]: Es el proceso de identificar, cuantificar y priorizar las vulnerabilidades o riesgos presentes en un sistema informático.
- **Árbol de Sintaxis Abstrácta (AST)** [26]: Es la forma de interpretación que tiene un software sobre todas las posibilidades que tiene para abordar operaciones, tiene información concreta y condensada. A través de esta estructura de datos se pueden abordar problemas complejos, teniendo en cuenta su interpretación con la información que se obtiene de este.

2.3. Antecedentes

Por el problema expuesto, se busca acceder a proyectos surgidos para resolver problemas similares, para revisar las finalidades que se usaron para resolverlos:

- **Gestión de un proyecto de control de pruebas de un aplicativo** [27]: Por medio de este proyecto, se busca plasmar como debe ser la gestión de pruebas de software que se debe realizar dentro de un aplicativo, para que éste tenga un estándar de calidad adecuado y exista una planeación en el ámbito de testing para que el proyecto no se quede sólo en ideas y pueda surgir como un proyecto funcional, que logre resolver un problema para algún sector de la población. En este caso, se planea realizar pruebas de software a una aplicación de tipo web .NET, donde se evidencie la necesidad de llevar el paso a paso del desarrollo del proyecto con las pruebas integradas.

El proyecto inicia haciendo énfasis en que se encuentre presente en el desarrollo del desarrollo de un producto de software, las pruebas, describiendo con detalle las necesidades que han llevado a los desarrolladores, principalmente en España, a implementar el recurso óptimo de las pruebas; esto debido a que no se culminaran muchos proyectos de software, se analiza principalmente muchas aplicaciones .NET donde se veían brechas que no permitían que el proyecto tuviera una culminación exitosa, sino que al momento de la entrega de la aplicación se empezaban a manifestar múltiples errores que no cumplían con los requisitos que pidió el cliente. Muestra la necesidad de incorporar a expertos en pruebas dentro del equipo de desarrollo, para que haya un correcto levantamiento de los casos de prueba que se piensan efectuar. Cabe aclarar que en este proyecto se evidencia un estudio muy conceptual respecto a cómo incluir los casos de prueba en un desarrollo de calidad, además describe los diferentes tipos de pruebas que existen y en que ámbitos usarlos, referenciando importantes artículos de prestigiosos sitios como lo son Amazon, IBM, Azure y demás. Enfatiza las pruebas en cómo afectan en el ciclo de vida del software y como alterarlo de una forma positiva y que permita un correcto desarrollo del producto final, el autor permite dividir en las siguientes fases en las cuales, incluye las pruebas de software: planificación, preparación, especificación, ejecución. Finalización y una última fase denominada como control.

Como conclusión se muestran los principios básicos y fundamentos, así como las metodologías orientadas a negocio que se utilizan en proyectos reales, como ha afectado el

desarrollo de las pruebas en proyectos, las cuales han beneficiado de forma positiva a la culminación exitosa de los productos de software.

- **Pruebas de mutación para aplicaciones web y móviles** [28]: En este proyecto se busca encontrar soluciones al problema de incluir las pruebas en las aplicaciones que funcionan tanto para dispositivos móviles como las enfocadas para web, en este caso se busca más el enfoque hacia los dispositivos móviles donde se incluye el tipo de pruebas que ofrece Android: Firebase Test Lab y Android UI Monkey Exerciser, gracias a estas dos herramientas el autor busca introducir en el desarrollo de aplicaciones móviles un conjunto de pruebas que cumplan con la calidad del software; se busca un enfoque hacia HITPA, la cual es una herramienta integradora de pruebas para conseguir abarcar el sistema operativo Android.

Como objetivos, busca integrar mediante la herramienta mencionada anteriormente las pruebas de mutación para aplicaciones de dispositivos móviles, identificar como las pruebas de unidad buscan beneficiar el desarrollo de estas pruebas de mutación y tener un informe relevante con respecto a las comparativas hechas por la aplicación mutada versus la original. Con estos informes se busca demostrar que el desarrollo de aplicaciones móviles se beneficia de las ventajas de las pruebas de mutación y así conseguir un desarrollo del software previniendo futuros errores.

Como resultados obtenidos a este proyecto según Acevedo Correa [28], “La implementación de la herramienta y su alcance cubrieron de manera satisfactoria los objetivos propuestos desde un principio” demostrando que poder integrar las herramientas propuestas por Android con HITPA, tuvo elementos innovadores y alentadores que logran probar aplicaciones móviles, algo que no es muy común realizar, ya que se busca programar una aplicación web, volverla responsive y por último llevarla a ser una aplicación móvil tipo apk.

Por medio de la siguiente tabla, se busca mostrar las características más relevantes respecto a estos proyectos que se tienen como referencias:

Proyectos	Características
Gestión de un proyecto de control de pruebas de un aplicativo. [27]	Enfatizar en la importancia de las pruebas de software, planificar adecuadamente para que exista una ejecución óptima, incorporar en el ciclo de desarrollo de software a las pruebas.
Pruebas de mutación para aplicaciones web y móviles. [28]	Enfocar las pruebas de mutación en aplicaciones móviles, integrar las herramientas de prueba de Android con HITPA, innovar en la incorporación de nuevas herramientas.

Tabla 2.1: Comparativa Antecedentes

Considerando la tabla anterior y en los antecedentes mencionados se evidencia la falta de un enfoque de pruebas de software que busque evaluar la correctitud en las aplicaciones web, ya que algunos proyectos lo muestran conceptualmente y otros se salen hacia aplicaciones móviles, siendo ideal el plano propuesto para que no exista redundancia clara al visualizar el proyecto.

Capítulo 3

Seguridad en Flask

Dando contexto específico a cerca del framework Flask que se trabajó en este proyecto, Flask es un framework back-end que se usa para el desarrollo de aplicaciones web en Python, está diseñado buscando ofrecer facilidad de implementación y flexibilidad sin requerir una estructura compleja. Conociendo a cerca de su historia, fue creado por Armin Ronacher, es muy valorado en la comunidad de desarrollo debido a su diseño minimalista y modular, esto facilita el desarrollo de aplicaciones ligeras y rápidas para entornos de producción y pruebas [29]. Su diseño e implementación ligera permite a los desarrolladores el poder elegir e integrar únicamente los componentes necesarios sin tener que incrementar los recursos, como módulos para autenticación, bases de datos y validación de formularios, según las necesidades de la aplicación [30].

Para conocer a cerca de la seguridad y analizar el interior de Flask es necesario conocer cómo está diseñado y bajo que arquitectura se encuentra estructurado.

3.1. Arquitectura de Flask

Teniendo en cuenta y analizando Flask, es importante conocer su arquitectura, pues esta arquitectura está basa en el patrón de diseño MVC (modelo, vista, controlador), que facilita la comunicación entre el servidor y la aplicación web. Dicho patrón de diseño permite a los desarrolladores manejar cada solicitud HTTP y sus respuestas de forma sencilla, proporcionando el debido control sobre cómo se procesan las peticiones y cómo se manejan en el código [29]. Además, Flask incorpora Werkzeug, una biblioteca o plugin que se encarga de gestionar la ejecución de aplicaciones, y Jinja2, un motor de plantillas que facilita la creación de todo el contenido HTML dinámico. Estos componentes se combinan e integran para crear un entorno robusto y eficiente, lo que suele ser ideal para desarrollar aplicaciones monolíticas, aplicaciones RESTful y prototipos rápidos [31].

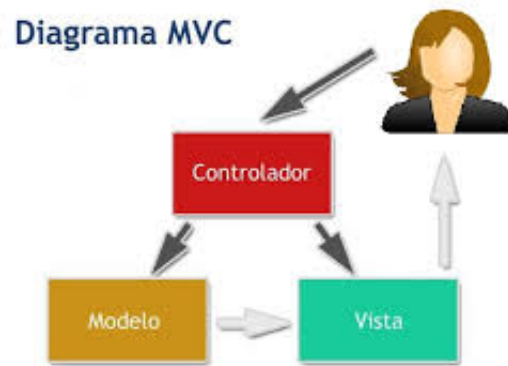


Figura 3.1: Arquitectura MVC

Fuente: [32]

3.2. Configuraciones Presentes en Flask

Flask cuenta con diversas configuraciones predeterminadas diseñadas para simplificar el desarrollo inicial de aplicaciones. Entre estas están, el `DEBUG`, normalmente en su estado `False` por defecto, el `TESTING`, también predeterminado a `False`.

Dichas configuraciones son personalizables fácilmente mediante el tratamiento de un archivo de configuración, variables de entorno o archivos específicos en formatos como JSON, proporcionando flexibilidad y fácil entendimiento para adaptarse a distintos entornos de despliegue.

En este caso, se abordó el análisis de vulnerabilidades de seguridad, por consiguiente, se muestra a continuación las configuraciones y librerías necesarias para ajustar estos parametros en Flask.

- **Protección contra Inyecciones SQL:** Siendo Flask un framework ligero, soporta ORM (Object Relational Mapping) como SQLAlchemy, esto evita las inyecciones SQL. Al hacer uso de ORM, las consultas SQL son generadas por el framework de manera automática, evitando la manipulación directa de las consultas en texto plano, lo cual es una práctica recomendada para prevenir ataques de inyección SQL [31].
- **Cross-Site Scripting (XSS):** Flask protege contra XSS mediante el uso de su sistema de plantillas Jinja2, que permite el escape automáticamente de las variables inyectadas en las plantillas HTML, reduciendo así el riesgo de inyecciones maliciosas en el contenido que se muestra a los usuarios [33].
- **Cross-Site Request Forgery (CSRF):** Buscando prevenir ataques CSRF, Flask ofrece la extensión Flask-WTF (WTForms para Flask), que integra tokens CSRF en los formularios de la aplicación. Los tokens se generan para cada usuario y son validados con cada solicitud de formulario, ayudando a mitigar ataques de falsificación de solicitudes [6].

- **Cabeceras de Seguridad:** Flask permite configurar las cabeceras HTTP de forma segura, como Content Security Policy (CSP), Strict-Transport-Security (HSTS), y X-Content-Type-Options, estas protegen contra ataques de manipulación de contenido y ataques de tipo clickjacking. Las cabeceras se pueden implementar con middleware de seguridad en Flask [29][6].
- **Validación de Entradas del Usuario:** La extensión Flask-WTF proporciona métodos para validar las entradas de usuario, esto es ideal para proteger la aplicación contra entradas maliciosas que puedan comprometer la seguridad del sistema [6].

La unión de estas configuraciones de seguridad y bibliotecas propias del framework permite a los desarrolladores de Flask implementar y tener buenas prácticas en sus aplicaciones buscando garantizar un entorno seguro. Sin embargo, se recomienda mantener los paquetes de Flask actualizados.

El problema a la hora de abordar las configuraciones de seguridad es bastante común en los proyectos que son realizados día a día, pues los desarrolladores se ocupan por construir la plataforma web sin tener cuidado con la parte backend son las configuraciones que proporcionan la seguridad de la aplicación son uno de los problemas más peligrosos en el desarrollo de aplicaciones, incluyendo aquellas desarrolladas haciendo uso del framework Flask. Flask, como microframework para Python, es popular por su flexibilidad, pero también por la responsabilidad que pone en los desarrolladores de configurar adecuadamente la seguridad.

Una configuración incorrecta de seguridad puede exponer a una aplicación a diversos tipos de ataques, como inyecciones de SQL, Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), y otros. Este capítulo explora las configuraciones incorrectas de seguridad más comunes en aplicaciones Flask y sus implicaciones, junto con recomendaciones para mitigarlas. Tomando en cuenta que existen varios tipos de bibliotecas que muestran configuraciones de Flask, en este caso se muestran principalmente las que están presentes en CORS, una biblioteca que permite evidenciar la parte de la seguridad.

Configuración	Descripción	Variables
Origins	Especifica que orígenes son permitidos para solicitudes CORS; define los dominios que pueden acceder a los recursos de la aplicación.	Lista de URLs que son permitidas.
Methods	Define los métodos HTTP que son permitidos; controla los métodos que pueden ser usados en solicitudes.	Metodos de peticiones como lo son get, post, put, delete, entre otros.
Allow Headers	Especifica los encabezados HTTP permitidos; define qué encabezados puede incluir el cliente en las solicitudes.	Lista de encabezados permitidos.
Support Credentials	Habilita o deshabilita el envío de cookies y credenciales en solicitudes CORS. Permite que el navegador incluya cookies, autenticación HTTP y otros tokens en solicitudes CORS.	True o false.
Max Age	Controla la duración de la caché para solicitudes. Permite determinar cuántos segundos se cachean las respuestas antes de ser reenviadas.	Número entero en segundos.
Automatic Options	Indica si Flask debe gestionar automáticamente las solicitudes de options. Permite a Flask procesar solicitudes sin lógica personalizada.	True o false.
Send Wildcard	Habilita o deshabilita el uso de comodines en las respuestas CORS. Teniendo en cuenta que si es True, el servidor responde con *, permitiendo acceso desde cualquier origen.	True o false.
Very Header	Determina si la respuesta incluye el encabezado vary. Ayuda a los sistemas de caché a distinguir respuestas basadas en encabezados específicos de las solicitudes.	True o false.

Tabla 3.1: Configuraciones de Seguridad Comunes en Flask

Fuente: Elaboración propia

A continuación, se muestran otro tipo de configuraciones, en las cuales por defecto, Flask viene configurado en un modo que es seguro para el desarrollo, pero no para producción. Sin una configuración adecuada, estas configuraciones pueden llevar a una exposición innecesaria de la aplicación a ataques potenciales.

Se debe tener en consideración que, cuando se encuentra un valor en estado true, significa que esta configuración está activa, de lo contrario, si se encuentra en false, esta configuración

está desactivada.

- **Origins:** Define qué orígenes están permitidos para solicitudes CORS y los valores predeterminados que tiene Flask es que permite a todos los orígenes permitir el acceso a la aplicación evaluada.
- **Methods** [34]: Por defecto trae la presencia de los métodos http los cuales son: Get, head, post, options, put, patch y delete.
- **Allow headers:** Permite mostrar encabezados en cada solicitud de Cors, su valor por defecto es el Content Type.
- **Support credentials** [35]: Indica si las credenciales (cookies, autenticación HTTP) se permiten en solicitudes CORS. Por defecto viene con false en su configuración asignada.
- **Max age:** Tiempo de caché en segundos para solicitudes pre-flight (solicitudes que están en el espacio). Usa por defecto un tiempo de 21600 segundos, lo cual equivale a 6 horas.
- **Send wildcard:** Se muestra para permitir usar comodines o "adornos" dentro de los encabezados. Por defecto trae el false predeterminado.
- **Automatic options:** Permite hacer uso de activar el options como solicitud http. Viene con el true por defecto.
- **Vary header:** Muestra las solicitudes proporcionadas por CORS. Está en estado de true, como su estado predefinido.

Para tener en cuenta, que estas configuraciones las traiga el framework por defecto, no significa que sean las más apropiadas para tener una seguridad adecuada, garantizando que no hay brechas abiertas; pues el framework propone estas configuraciones buscando ahorrar recursos dentro del servidor del aplicativo en uso.

Teniendo esto claro, a continuación se muestran las configuraciones de la forma que son seguras para el desarrollo de la aplicación y también se muestran las configuraciones que hacen que la aplicación sea vulnerable.

Configuración	Segura	Vulnerable
Origins	Tener las URL de los sitios u orígenes de servidores de donde se puede acceder a la aplicación.	Tener permitido a cualquier origen el acceso a la aplicación, esto significa no tener URLs dentro de Origins.
Methods	Tener especificados los metodos de tipo http que son necesariamente primordiales para su uso.	Permitir todos los métodos HTTP sin restricciones.
Allow Headers	Especificar los encabezados HTTP permitidos	Permitir todos los encabezados.
Support Credentials	Permacer deshabilitada el envío de cookies y credenciales en solicitudes CORS.	Estar soportando credenciales (estado de la configuración en true), lo que puede exponer cookies de sesión.
Max Age	Matener controlada la latencia del tiempo en servidor, tener 3600 segundos es más que suficiente para que existan operaciones de carga de caché.	Incrementar el tiempo de caché de las opciones pre-flight para mantener configuraciones inseguras.
Automatic Options	Debe mantenerse desactivada la automatización de opciones (estado de configuración en false) y que el desarrollador pueda hacerlo de manera manual.	Permitir que Flask maneje las solicitudes OPTIONS automáticamente (estado de configuraciones en true).
Send Wildcard	Mantener deshabilitado el uso de comodines en las respuestas CORS.	Enviar comodines en las respuestas CORS (estado de configuraciones en true).
Very Header	Se debe ayudar a los sistemas de caché a distinguir respuestas basadas en encabezados específicos de las solicitudes (estado de la configuración en true).	No variar la respuesta dependiendo de los encabezados, haciendo el caché más predecible (estado de la configuración en false).

Tabla 3.2: Vulnerabilidades de las Configuraciones de Seguridad Comunes en Flask

Fuente: Elaboración propia

La configuración incorrecta de seguridad en aplicaciones Flask puede tener severas consecuencias, exponiendo las aplicaciones y a las personas que acceden a ellas, a una variedad de ataques y comprometiendo la seguridad de los datos.

Es fundamental que los desarrolladores comprendan las configuraciones por defecto del framework y apliquen las mejores prácticas de seguridad desde el principio del ciclo de desarrollo,

no esperando a tener las configuraciones que proporciona el framework por defecto, pues a veces este tipo de configuraciones no tienen una cobertura completa. Las extensiones de seguridad, el uso de buenas prácticas de codificación y la validación y sanitización adecuadas de las entradas del usuario son elementos cruciales para proteger las aplicaciones web.

Capítulo 4

Diseño y Desarrollo del Proyecto

4.1. Aplicación de la Metodología XP

Esta metodología se basó en prácticas clave el cómo desarrollo guiado por pruebas (TDD), lo cual benefició a este proyecto donde el desarrollo de la herramienta busca que basándose en pruebas unitarias permitan hacer que las pruebas de mutación funcionen para detectar las vulnerabilidades, por otro lado, la programación en parejas, la integración continua, y retroalimentación constante, jugaron un papel fundamental para el aporte, adelantamiento y los logros pertinentes para que el proyecto sea satisfactorio.

- **Creación de Historias de Usuario:** Identificar y documentar los requerimientos funcionales desde la perspectiva del usuario final. Teniendo en cuenta que las historias deben ser claras, concisas y basado en que historia que se haga, historia que deberá ser probada.
- **Desarrollo guiado por pruebas (TDD):** Las pruebas unitarias y de mutación permitieron la implementación del código.
- **Programación en parejas:** Los operadores de mutación, las pruebas de seguridad y los test unitarios se desarrollaron en equipos, asegurando calidad y revisión constante.
- **Pequeñas entregas:** Se realizaron entregas frecuentes buscando la efectividad del aplicativo para detectar vulnerabilidades mediante pruebas de mutación.
- **Estimación:** Asignar puntos de esfuerzo a cada historia basándose en la complejidad que se requiera en cada una.
- **Documentación Básica:** Suministrar documentación útil sobre la implementación e instalación del proyecto entregable.
- **Soporte Post-Entrega:** Resolver errores encontrados después de la demostración del prototipo funcional.



Figura 4.1: Extreme Programming XP

Fuente: [36]

A la hora de implementar la fase de pruebas usando la metodología XP (Extreme Programming) para este proyecto, se reflejan las pruebas unitarias y pruebas de mutación, pues para este proyecto es fundamental evidenciar que se pasan los procesos de los operadores creados y esto se puede verificar teniendo un índice de pruebas adecuado y verificado para mostrar que se han cumplido los objetivos.

4.1.1. Exploración

Antes de especificar la planeación usada para abordar el desarrollo del proyecto, fue necesario especificar los roles de cada uno de los participantes del proyecto:

- **Desarrollador:** Harold y Juan (los autores).
- **Cliente:** Desarrolladores.
- **Tracker:** Carlos Andrés Delgado Saavedra (director de trabajo de grado).
- **Manager:** Carlos Andrés Delgado Saavedra (director de trabajo de grado).

4.1.1.1. Historias de Usuario

Teniendo en cuenta que lo que se busca realizar es una herramienta que permita detectar las vulnerabilidades que están presentes en las configuraciones de un proyecto realizado con Flask, se han previsto tener en cuenta la realización de las historias de usuario, tomando en cuenta que se desea para el desarrollo.

En total, se obtuvieron 4 Historias de Usuarios, las cuales representaban los requisitos tenidos en cuenta (evidenciados en los anexos mostrados a continuación) [37] que debía tener la herramienta a desarrollar.

Así pues, una historia de usuario asignada con estimación de 2, requerirá el doble de esfuerzo que una estimada con 1 y una que esté estimada con 3, requerirá el triple de esfuerzo que una estimada en 1 y así sucesivamente.

A continuación, se muestran historias de usuario:

Historia de Usuario N°1		
Nombre Historia: Creación de un Operador de Mutación para Pruebas de Seguridad		Responsable: Harold Montano
Prioridad: Alta	Puntos Estimados: 3	Iteración: 3
Descripción: Como desarrollador, deseo crear un operador de mutación buscando detectar configuraciones incorrectas de seguridad en Flask para identificar vulnerabilidades.		
Criterios de Aceptación: ■ El operador debe poder detectar configuraciones incorrectas como desactivación de CORS sin restricciones. ■ Las pruebas deben pasar con un código bien configurado y fallar cuando haya configuraciones vulnerables.		

Tabla 4.1: Historia de Usuario N°1

Fuente: Elaboración propia

Historia de Usuario N°2		
Nombre Historia: Generación de Informes de Cobertura de Vulnerabilidades		Responsable: Juan Veloza
Prioridad: Media	Puntos Estimados: 2	Iteración: 1
Descripción: Como desarrollador, quiero generar un informe que muestre la cobertura de vulnerabilidades detectadas por las pruebas de mutación para evaluar la efectividad de las pruebas realizadas.		
Criterios de Aceptación: ■ El informe debe detallar las vulnerabilidades detectadas. ■ El informe debe incluir mediciones sobre la cobertura de los casos de prueba.		

Tabla 4.2: Historia de Usuario N°2

Fuente: Elaboración propia

Historia de Usuario N°3		
Nombre Historia: Validación de la Eficacia del Aplicativo		Responsable: Harold Montano
Prioridad: Media	Puntos Estimados: 2	Iteración: 1
Descripción: Como desarrollador, quiero validar la eficacia del aplicativo que permita detectar vulnerabilidades mediante el uso de pruebas para asegurar que las pruebas detectan errores en las configuraciones de Flask.		
Criterios de Aceptación: ■ Se deben documentar los cambios en la configuración de seguridad.		

Tabla 4.3: Historia de Usuario N°3

Fuente: Elaboración propia

Historia de Usuario N°4		
Nombre Historia: Documentación de la Configuración de Seguridad en Flask		Responsable: Juan Veloza
Prioridad: Media	Puntos Estimados: 2	Iteración: 1
Descripción: Como desarrollador, quiero crear una guía de buenas prácticas de configuración de seguridad en Flask evitando vulnerabilidades comunes.		
Criterios de Aceptación: ■ Debe estar integrada dentro de las pruebas como referencia para usuarios.		

Tabla 4.4: Historia de Usuario N°4

Fuente: Elaboración propia

4.1.1.2. Definición de Tecnologías

Para este proyecto se usaron tecnologías apropiadas para los objetivos conocidos previamente, las cuales serán descritas a continuación junto con las funciones que cumplieron:

- **Python:** Es el lenguaje base de este proyecto el cual está siendo enfocado en la versión 3.10.11 y se utiliza para desarrollar tanto el motor de pruebas, los casos de prueba y análisis de seguridad. Python destaca principalmente por su simplicidad, versatilidad y amplia gama de bibliotecas, lo cual facilita la manipulación de árboles de sintaxis abstracta (AST) y la integración con frameworks como Flask. Se cuenta con gran capacidad de librerías que permiten hacer un amplio análisis, cuenta con un gran soporte para el análisis de operaciones de herramientas diseñadas basadas en Python, es un lenguaje interpretado lo cual facilita la realización de pruebas y la implementación de prototipos rápidos y eficientes.

El uso de este lenguaje es tenido en cuenta para el desarrollo de la herramienta en general, tanto para pruebas como para la interpretación de árbol de sintaxis abstracta, siendo este un elemento a analizar para mostrar los análisis de las configuraciones de seguridad.

- **Flask:** Es un framework para Python que permite la creación de aplicaciones web para la parte del back-end. Se utilizó como ejemplo de tecnología objetivo en las pruebas de seguridad, pues este permite gestionar rutas, recibir parámetros y manipular datos provenientes del usuario, lo que lo convierte en un punto de interés para la detección de vulnerabilidades. Cuenta con beneficios como tener una curva de aprendizaje baja lo que permite un entendimiento fácil, flexible con aplicaciones robustas y popular en la comunidad de Python gracias a su abundante documentación y recursos.

Es el framework en el cual se evaluaron sus configuraciones de seguridad, con la versión 3.0.2.

El caso de estudio para verificar las configuraciones de Flask dentro de este proyecto se encuentran dentro del siguiente repositorio [1].

- **MySQL:** Es la base de datos que se utilizó para el almacenamiento de la información presente en el caso de estudio que se evaluó.

4.1.1.3. Librería AST

Permite trabajar con el Árbol de Sintaxis Abstracta (AST) de un programa hecho en Python. Por medio de este árbol se permite tener una representación intermedia del código fuente que conserva la estructura lógica del código sin necesidad de ejecutarlo [5]. Gracias a este módulo se permite desarrollar, programas con múltiples posibilidades, dentro de las funciones de esta librería se encuentra:

El poder analizar al convertir el código fuente en un árbol sintáctico usando funciones especializadas propias de la librería que permitan convertir el código proporcionado hacia un árbol sintáctico. Recorrer y poder examinar cada nodo del árbol sintáctico usando clases que permitan descubrir las múltiples posibilidades que puede tomar el programa y analizar la más adecuada para cada caso. Transformar y poder modificar el árbol y generar un nuevo código. Tener la posibilidad de convertir un árbol AST en un objeto ejecutable y compilable.

Dentro del funcionamiento de AST, que fue descrito anteriormente, se incluyen los siguientes componentes:

- **Nodos:** Cada nodo dentro del árbol representa una construcción del lenguaje Python, como declaraciones condicionales o bucles, expresiones o funciones y estructuras o clases.
- **Recorrido:** Por medio de una clase especializada en explorar y visitar todos los nodos o posibilidades de las acciones del programa se recorre cada nodo del árbol, permitiendo así extender y tener la mejor elección para alguna situación o acción requerida.
- **Transformación:** Esta clase permite modificar y regenerar cada nodo del árbol dentro del cual se trabaja.

Teniendo en cuenta las funcionalidades con las que cuenta la librería, es preciso mencionar sus ventajas a la hora de usarla y trabajar con esta:

- **Análisis de Código:** Permite analizar código fuente sin necesidad de ejecutarlo, ideal para herramientas de análisis de seguridad, que buscan detectar problemas como inyecciones de código o vulnerabilidades.
- **Refactorizaciones Automáticas:** Permite hacer transformaciones automáticamente al código, como puede ser: Renombrar variables o funciones; actualizar la sintaxis a nuevas versiones; insertar anotaciones de manera automática.
- **Generación Automatizada de Código:** La librería AST permite generar código

Python de forma programada, lo que resulta útil para, generación de pruebas automáticas e inserción de lógica personalizada en tiempo de ejecución.

- **Seguridad:** Resulta útil para analizar código dinámico (código en ejecución) y detectar posibles vulnerabilidades antes de ejecutarlo.
- **Facilidad al Depurar:** Al tener el código en una estructura comprensible y entendible, AST ayuda a comprender cómo Python interpreta las posibles construcciones del lenguaje, esto resulta útil para depurar o hacer debug (recorrer el código paso a paso).

Teniendo claras la ventajas, se muestran las operaciones que se efectúan a la hora de establecer esta librería dentro de este proyecto, haciendola funcional.

1. **Reporte en la Cobertura de las Operaciones de Mutación:** Permite mostrar una puntuación global acerca de mutación, evidenciando qué porcentaje de mutaciones fueron detectadas por las pruebas.
2. **Mutaciones Asesinadas y Supervivientes:** Muestra cada mutación que se generó (este tipo de mutaciones se pueden verificar a través de pruebas unitarias), mostrando cada línea que fue afectada y si hubo detección en las pruebas.
3. **Clasificación de Tipos de Mutaciones:** Los reportes muestran a cada mutación según su tipo, como lo son el reemplazo de operadores o inversión de los condicionales.

A la hora de analizar el proyecto, surge la duda ¿Por qué se usó la librería AST y no otra librería enfocada más hacia probar el software? A continuación, por medio de una tabla comparativa se busca mostrar que otras alternativas se tuvieron en cuenta y cómo se evaluó para escoger una:

Herramienta	Complejidad	Tipos de Mutación	Generación de Reportes	Justificación
MutPy	Media	Amplio	Detallado	Cobertura alta de mutaciones y con reportes específicos para Python.
Cosmic Ray	Media	Moderado	Básico	Baja variedad en las mutaciones y reportes poco detallados.
Hypothesis	Alta	Limitado	Básico	Hypothesis es un motor de pruebas basado en generación de datos. Genera datos aleatoriamente para las pruebas y puede manejar gran variedad en los tipos de datos.
AST	Alta	Depende de lo programado	Depende de lo programado	Esta es una librería que permite efectuar el crecimiento de un programa y establecer parametros específicos para cumplir un fin deseado por el desarrollador.

Tabla 4.5: Tabla Comparativa de Librerías de Prueba

Fuente: Elaboración propia

Los parámetros con los que se evaluaron cada librería de prueba, fueron tenidas en cuenta a partir de tener la información de cada herramienta descrita a continuación:

- **Mutpy:** Se ha detallado teniendo en cuenta que es una herramienta diseñada propia y específicamente para Python, esta es centrada en evaluar los test unitarios diseñados para garantizar la efectividad de mutaciones introducidas al código. Mutpy permite al usuario explorar y controlar los cambios que se le aplican al código, proporcionando información para saber que tan robustas son frente a posibles fallos que existan o a modificaciones no controladas. El problema que se presentó para su elección es la incompatibilidad que se tiene con muchas dependencias de Python y Flask que son el pilar de este proyecto.
- **Cosmic Ray:** Herramienta para realizar mutaciones en Python, esta está diseñada en un enfoque modular y flexible. Su diseño es adaptativo buscando personalizar el

funcionamiento, siendo muy útil al requerir usar las pruebas de mutación dentro de proyectos específicos con configuraciones prediseñadas. El motivo de desventaja se refleja en lo complejo de su instalación y el problema con compatibilidades con sistemas operativos como Windows, pues esta herramienta está diseñada para servidores con sistemas operativos basados en Linux.

- **Hypothesis:** Se usa como una herramienta de pruebas generales, es decir, se enfoca en el proceso de las pruebas de software en Python de una forma innovadora, complementando las comunes, abordando la parte de pruebas de mutación, centrándose en la automatización generativa de datos en diversos escenarios.
- **AST:** Es una librería descrita para Python, la cual, se encarga de realizar el seguimiento sintáctico de un programa, estableciendo de forma autónoma sus funcionalidades y el alcance que puede tener cada una. Se eligió como fuente de los mutantes, pues proporciona facilidad operativa al ser una librería propia de Python y con la operatividad a la hora de relacionarse con otras dependencias.

De forma general, cada una de estas herramientas es óptima para usos especializados de cada una, en el caso de este proyecto, se trabajó con AST pues el enfoque que se buscó es uno más adaptativo buscando realizar cambios con diversos datos y variables, en este caso son las configuraciones de seguridad.

4.1.1.4. Arquitectura del Sistema

Al realizar el análisis de la detección de vulnerabilidades utilizando pruebas de mutación con la librería AST de Python, podemos definir un proceso detallado para implementar operadores de mutación personalizados, ejecutar pruebas unitarias, se muestra una arquitectura orientada a funciones personalizadas, caracterizada por el desarrollo interno y específico de cada funcionalidad. Pues al generar mutantes o cambios en líneas de código, evaluándolos y ejecutando un conjunto de pruebas unitarias existentes, cerciorándose si las pruebas pueden detectar algún tipo cambio. Esto requiere ciertos modulos o extensiones los cuales están organizados en componentes independientes, cada uno con responsabilidades bien definidas. En términos de arquitectura de software [38], la arquitectura modular puede entenderse como una forma de arquitectura basada en componentes. Este tipo de arquitectura permite que cada módulo o componente tenga una funcionalidad específica, y los componentes se integran para formar el sistema completo.

La arquitectura tenida en cuenta por el desarrollo de este proyecto, busca ser lo suficientemente independiente, para que cada elemento presente dentro del sistema no genere conflicto con otras dependencias de otro modulo y así contribuyendo a ejercer una singularidad total dentro del aplicativo.

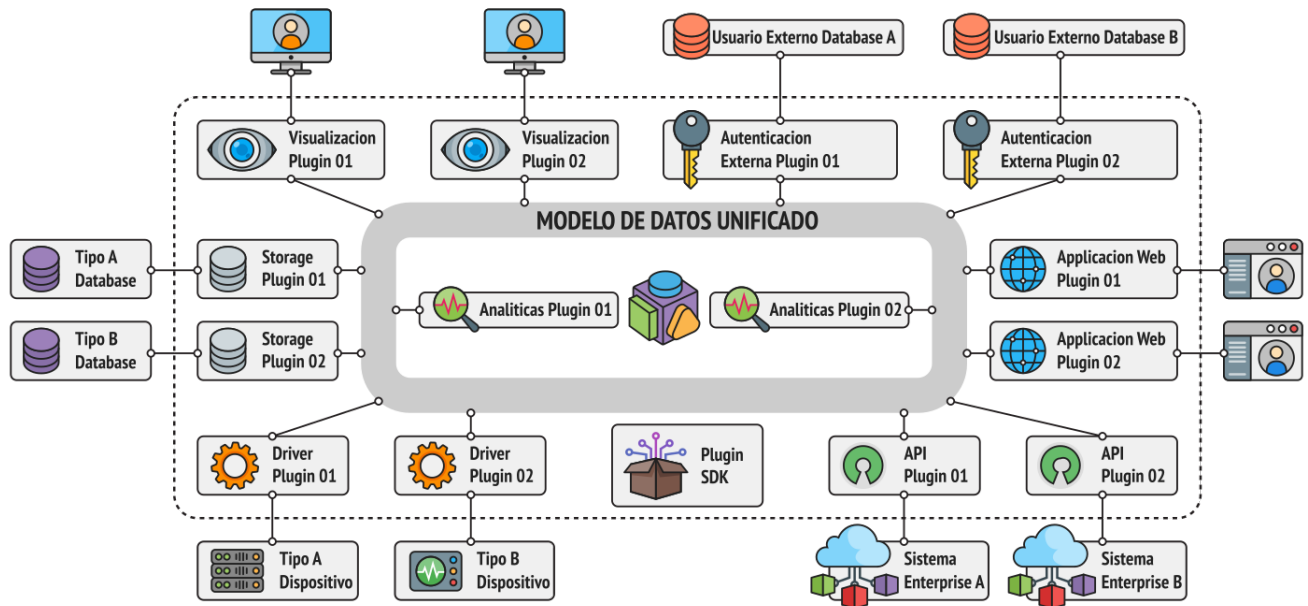


Figura 4.2: Arquitectura del Sistema

Fuente: [39]

En el caso de la generación de operadores de mutación y la evaluación de pruebas de mutación en las configuraciones de seguridad, los módulos serían:

- El parser, el cuál es el encargado de analizar y manipular el árbol de sintaxis abstracta del código fuentes.
- Generador de mutantes, un conjunto de cambios que se le aplican mínimas mutaciones controlando al código fuente, generando un código alterado, denominado código mutante. Este hace uso de una estructura de datos AST, el cual identifica los puntos a mutar dentro del código fuente.
- Ejecutor de mutantes, ejecuta cada mutante en el contexto del conjunto de pruebas existente y vigila los resultados buscando determinar si las pruebas detectan los errores introducidos.
- Ejecutor de pruebas, encargado de ejecutar y gestionar las pruebas unitarias sobre los mutantes generados y verificar si fallan o no.
- Gestión de concurrencia, otro módulo que coordina la ejecución por separado de pruebas, aprovechando múltiples procesos.
- Integrador de scripts, permite de forma programada la carga de código hasta la generación de informes.

Las razones para tener en cuenta a esta arquitectura, es la facilidad de implementar módulos

que se comuniquen pero que a su vez puedan tener cierta independencia para que no generen conflictos internos, además de poder tener la posibilidad de evolucionar a una arquitectura como microservicios.

La forma gráfica de ver la arquitectura reflejada a partir de la construcción del AST sería de la siguiente forma:

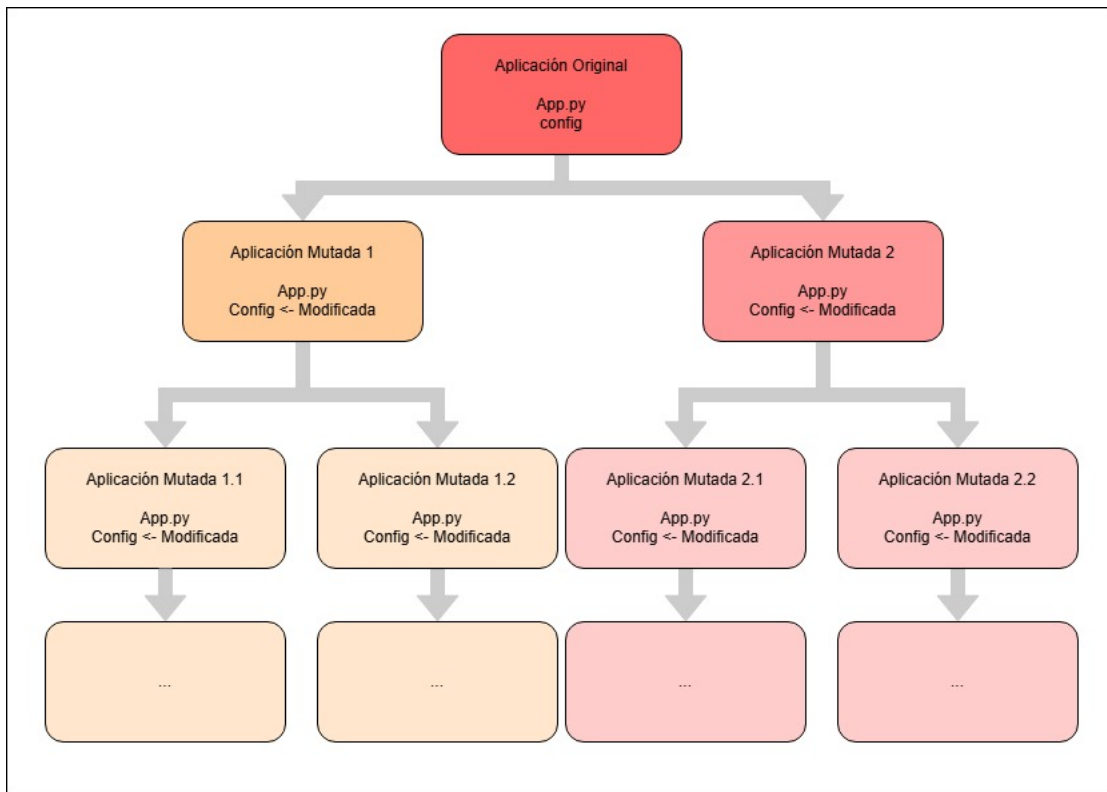


Figura 4.3: Arquitectura Descrita de AST en el Proyecto

Fuente: Elaboración propia

4.1.2. Iteraciones

El proyecto se dividió en 2 iteraciones, cada una con una duración entre una y 5 semanas para su realización y, con un número determinado de historias de usuario, teniendo en cuenta que en este proyecto lo principal fue llevar a cabo pruebas, cada iteración siendo realizada debía ser revisada, probada y verificada que se estén cumpliendo con lo requerido.

Para el desarrollo de cada iteración, se utilizó a GitHub (sistema de control de versiones) para facilitar la integración del código de los dos desarrolladores y controlar el versionamiento del aplicativo que se realizó.

Cabe por resaltar que, al concluir cada iteración, los desarrolladores se reunían con el Manager del proyecto para hacer la respectiva entrega y revisar avances del proyecto. Es importante

resaltar que las entregas fueron planeadas para su realización e incluso en otros casos se había avanzado más de lo acordado.

A continuación, se mencionan las iteraciones principales, junto con sus respectivas historias de usuario y con las diversas pruebas de aceptación que se obtuvo.

Iteración 1:

La iteración 1, tiene como objetivo cumplir las siguientes historias de usuario:

- Historia de usuario N° 1: Creación de un operador de mutación para pruebas de seguridad.
- Historia de usuario N° 3: Validación de la eficacia del aplicativo que detecta vulnerabilidades mediante el uso de pruebas.

Las tareas realizadas en esta iteración fueron:

Tarea	
Número de Tarea: N°1	Historia de Usuario (N° y Nombre): 1 – Creación de un operador de mutación para pruebas de seguridad.
Nombre Tarea: Desarrollo del módulo de mutaciones	
Tipo de Tarea: Desarrollo	Puntos Estimados: 3
Fecha de Inicio: 01/06/2024	Fecha de culminación: 15/07/2024
Programador Responsable: Harold Montano	
Descripción: Desarrollar el módulo de las mutaciones para poder hacer los cambios de código dentro del código fuente de la aplicación a evaluar.	

Tabla 4.6: Tarea N°1 - Iteración 1

Fuente: Elaboración propia

Tarea	
Número de Tarea: N°2	Historia de Usuario (N° y Nombre): 3 – Validación de la eficacia del aplicativo.
Nombre Tarea: Pruebas de la aplicación.	
Tipo de Tarea: Prueba	Puntos Estimados: 3
Fecha de Inicio: 15/07/2024	Fecha de culminación: 01/10/2024
Programador Responsable: Juan Veloza	
Descripción: Probar continuamente las operaciones que se hagan en el aplicativo para que exista un correcto funcionamiento y haga lo deseado.	

Tabla 4.7: Tarea N°2 - Iteración 1

Fuente: Elaboración propia

Iteración 2:

La iteración 2, tiene como objetivo cumplir las siguientes historias de usuario:

- Historia de usuario N° 2: Generación de Informes de Cobertura de Vulnerabilidades.
- Historia de usuario N° 4: Documentación de la Configuración de Seguridad en Flask.

Las tareas realizadas en esta iteración fueron:

Tarea	
Número de Tarea: N°3	Historia de Usuario (N° y Nombre): 2 – Generación de Informes de Cobertura de Vulnerabilidades.
Nombre Tarea: Informes de las pruebas realizadas.	
Tipo de Tarea: Prueba	Puntos Estimados: 2
Fecha de Inicio: 01/10/2024	Fecha de culminación: 01/11/2024
Programador Responsable: Harold Montano	
Descripción: Mostrar en forma ordenada la cantidad de pruebas realizadas y cuántos de los mutantes fueron satisfactorios y cuántos fueron dados de baja.	

Tabla 4.8: Tarea N°3 - Iteración 2

Fuente: Elaboración propia

Tarea	
Número de Tarea: N°4	Historia de Usuario (N° y Nombre): 4 – Documentación de la Configuración de Seguridad en Flask.
Nombre Tarea: Documentación del Proyecto.	
Tipo de Tarea: Documentación.	Puntos Estimados: 1
Fecha de Inicio: 01/01/2024	Fecha de culminación: 21/11/2024
Programador Responsable: Juan Veloza	
Descripción: Evidenciar los procesos realizados e incluir procesos de instalación de la aplicación.	

Tabla 4.9: Tarea N°4 - Iteración 2

Fuente: Elaboración propia

4.1.2.1. Descripción de la Herramienta Desarrollada

Se tiene en cuenta que se desarrolló un caso de estudio donde se busque efectuar las mutaciones producidas por el aplicativo creado por parte de la librería AST, es por eso que a continuación se documenta la creación y el funcionamiento de ambos aplicativos:

4.1.2.2. Documentación del Prototipo Aplicación de la Detección de Vulnerabilidades [1]

A través de diseñar la aplicación que permite implementar el código para la generación de cada uno de los mutantes, se permitió obtener a los operadores que tienen como objetivo ser reglas predefinidas que indican cómo realizar las mutaciones. Como ejemplos de los operadores se tienen la sustitución de constantes o valores booleanos (True por False), la eliminación de bloques de código condicional. En este proyecto, los operadores se aplican al AST (Árbol de Sintaxis Abstracta) del código fuente, permitiendo una manipulación precisa y controlada del comportamiento del programa. Cada mutante es evaluado mediante pruebas unitarias diseñadas para verificar la funcionalidad del sistema. Donde los resultados posibles van desde: Mutantes muertos, si una prueba detecta el cambio realizado, el mutante se considera eliminado (muerto); mutante sobreviviente, si el conjunto de pruebas no detecta el cambio, el mutante sobrevive, indicando una debilidad o vulnerabilidad en las pruebas. Cada análisis de supervivencia de los mutantes y las métricas calculadas permiten ajustar la configuración del sistema de la siguiente manera: Identificando los operadores de mutación más relevantes, detectando pruebas ineficaces y mejorando la cobertura de pruebas mediante la inclusión de casos adicionales para detectar mutantes sobrevivientes.

El proceso respecto a los pasos que tendría la aplicación sería:

1. Como primer entrada se tiene al código fuente de la aplicación a probar.
2. Se aplican los operadores o conjunto de mutantes para tener las reglas predefinidas para evaluar cada configuración.
3. Se prueban los mutantes respecto a la configuración tenida.
4. Por último se analizan las métricas calculadas y ajustes realizados en la configuración.

Para calcular la medida de la tasa de mutantes, se tuvo en cuenta la siguiente fórmula:

$$\text{Tasa de Mutación} = \frac{\text{Mutantes muertos}}{\text{Total de mutantes}}$$

Las métricas se comparan respecto a tener las pruebas unitarias, tomando en cuenta a las configuraciones seguras para compararlo con cada mutante procesado, buscando así si existe alguna vulnerabilidad. Para tener una tasa de mutación apropiada y confiable es necesario tener un buen número de mutantes por evaluar, es por esto que se diseñó una función que permite combinar y replicar las configuraciones línea a línea según sea requerido para evaluar posibles vulnerabilidades, teniendo así una tasa de mutación con un nivel de porcentaje confiable. La capacidad de generar y evaluar mutantes de manera controlada asegura la extensibilidad del sistema, promoviendo su adaptabilidad a diversas necesidades de desarrollo y fortaleciendo su utilidad.

El repositorio donde se encuentra alojado el código fuente de la aplicación desarrollada se encuentra presente en la fuente [1].

4.1.2.3. Documentación de Caso de Estudio Desarrollado en Flask [1]

Antes de conocer a cerca de la manipulación que se le dio a Flask, es importante conocer el proceso de instalación que se debió llevar a cabo para la creación y posterior implementación de las configuraciones dentro de la API creada en este framework:

1. Se debe clonar el Repositorio presente en el siguiente enlace de github: [?].
2. Para facilidad con la interacción de paquetes existentes dentro del proyecto, se debe crear un ambiente virtual donde se procederá a instalar las dependencias o bibliotecas necesarias: **python -m venv venv**. Para proceder a su activación: **.\venv\Scripts\activate** (en Windows) y: **source venv/bin/activate** (en Linux). Se dispone a la creación de ambiente virtual para que los paquetes de instalación no generen conflictos con otros archivos dentro de su computador.
3. Para tener los requerimientos instalados dentro del ambiente virtual se dispone de ejecutar: **pip install -r requirements.txt**

Entrando en contexto se muestra el porqué es necesario usar Flask el cual es un framework de desarrollo web ligero y flexible para Python. Es utilizado en la comunidad de desarrollo debido a su simplicidad, facilidad de uso y extensibilidad. A continuación, se muestran las principales razones para hacer uso de Flask mostrando que es una opción recomendada para el desarrollo de aplicaciones web. Como ventajas se tiene la simplicidad, flexibilidad, documentación estable, adaptabilidad y velocidad de comunicación con los servidores.

En este caso Flask incluye configuraciones específicas para CORS y una clave API, con el objetivo de mostrar como organizar un proyecto Flask con validaciones estrictas de configuración. La estructura está diseñada para ser modular y clara, separando la lógica de la aplicación principal y las configuraciones en archivos distintos.

Este caso de estudio se sigue una organización sencilla y funcional donde existen:

- **Archivo principal (app.py):** Contiene la lógica de la aplicación y el punto de entrada para ejecutar a Flask.
- **Archivo de configuración (config.py):** Define los parámetros configurables clave para el funcionamiento de la aplicación, como las configuraciones de CORS y la API Key.
- **Carpeta de pruebas (tests/):** Contiene pruebas unitarias para verificar que las configuraciones sean correctas y que la aplicación funcione según lo esperado.

Haciendo énfasis en la arquitectura modular, se cuenta con la separación de responsabilidades, donde, la lógica de la aplicación y las configuraciones están en archivos separados, mejorando la organización y facilitando su modificación. Por otro lado, enfatizando más en la parte de la seguridad, la aplicación utiliza una extensión para configurar el control de acceso entre dominios, lo cuál resulta ser útil para proteger y definir qué clientes pueden interactuar con la API. En otro apartado de las configuraciones se hace uso de una clase que busca poner

una seguridad en el uso de una clave API, la cual agrega una capa adicional de seguridad, asegurando que solo usuarios o sistemas autorizados puedan acceder a los recursos protegidos.

El paso a paso para el uso del caso de estudio que permite mostrar los resultados de este proyecto serían:

1. Instalar las dependencias desde el archivo de requisitos.
2. Ejecutar el archivo principal utilizando Python.
3. La aplicación estará disponible en un puerto local predeterminado y preparada para responder a las solicitudes.

El proyecto incluye la verificación de la integridad de las configuraciones antes de iniciar. Buscando garantizar:

- Que la aplicación no funcione si las configuraciones originales (óptimas) se modifican, protegiendo contra errores accidentales o cambios no autorizados.
- Si el sistema detecta cambios en las configuraciones, la aplicación detendrá su ejecución y mostrará un mensaje indicando que las configuraciones no son válidas.

Flask está diseñado para ser un ejemplo sólido de cómo estructurar las aplicaciones web modernas. Con un enfoque en la modularidad y la validación de configuraciones, siendo seguro y escalable. Ideal para proyectos que requieran configuraciones estrictas y soporte para control de acceso entre dominios.

Códigos de Respuesta Presentes:

- **200 OK:** La solicitud se completó correctamente.
- **400 Bad Request:** La solicitud fue incorrecta o no se pudo procesar.
- **404 Not Found:** El recurso solicitado no fue encontrado.
- **500 Internal Server Error:** Error interno del servidor.



Figura 4.4: Codigos Presentes en Peticiones

Fuente: [40]

Para correr la aplicación y tener las pruebas unitarias presentes de las configuraciones permitidas y debidas, se debe tener en cuenta lo siguiente:

- Para tener la aplicación corriendo desde el servidor local: **python App.py**
- Para correr las pruebas unitarias, se debe tener en la terminal de comandos el siguiente:
`python -m unittest -v <ruta>`

Para conocer el funcionamiento de este proyecto basta con tener levantados los test unitarios pues es en estos los que se enfoca el aplicativo para validar las configuraciones correctas.

4.1.2.4. Configuraciones de Seguridad a Tener en Cuenta en CORS

Para tener en cuenta, CORS se usó como una biblioteca presente en Python para poder acceder a las configuraciones que se buscan cambiar.

A continuación, se describen algunas configuraciones presentes para evaluar su eficacia dentro del caso de prueba:

- **HTTPS** es encargado de cifrar la comunicación entre el navegador del usuario y el servidor web, garantizando que los datos transmitidos estén protegidos contra la interceptación y manipulación. Esto es especialmente importante al enviar información confidencial, como contraseñas en este caso. Es un protocolo de cifrado de datos, como tal no espera una cadena específica.
- **CORS** (Cross-Origin Resource Sharing) es un mecanismo de seguridad que permite o restringe las solicitudes de recursos (como archivos JavaScript y CSS) entre diferentes

orígenes de dominio. Ayuda a prevenir ataques de scripts entre sitios al restringir qué recursos pueden acceder desde qué sitios. La configuración de CORS generalmente se realiza utilizando una extensión como Flask-CORS, que permite especificar configuraciones CORS a nivel de aplicación. La extensión Flask-CORS ofrece una variedad de opciones para configurar las políticas de CORS.

- **Supports Credentials**, un valor booleano que indica si el servidor debe permitir el intercambio de cookies y credenciales de autenticación en solicitudes CORS. Por defecto, esto está desactivado (False).
- **Origins**, es una lista de orígenes permitidos que pueden acceder a los recursos del servidor. Puede ser un solo origen o una lista de varios orígenes (direcciones o url permitidos).
- **Methods**, se presenta como una lista de métodos HTTP permitidos para las solicitudes CORS. Por defecto, Flask tiene que los métodos GET, HEAD y POST están permitidos. Puede incluir otros métodos como OPTIONS, PUT, DELETE, etc.
- **Allow Headers**, es una lista de encabezados HTTP permitidos en las solicitudes CORS. Esto permite a los clientes enviar encabezados personalizados en las solicitudes

Capítulo 5

Operadores de Mutación

El diseño del proyecto para evaluar la calidad de las configuraciones en sitios web Flask mediante pruebas de mutación utilizando la librería AST de Python implica descomponer el código base en su representación de árbol de sintaxis abstracta (AST), una estructura jerárquica que permite identificar funciones, variables y bloques de código como nodos susceptibles de mutaciones controladas. Este proceso comienza con la generación de mutantes, que son copias modificadas del código fuente donde se introducen cambios específicos en operadores, valores o estructuras lógicas. Dichos cambios se enfocan en configuraciones críticas de seguridad, como autenticación, manejo de credenciales, validaciones de entrada y mecanismos de autorización. Estas mutaciones permiten simular fallos o vulnerabilidades intencionales que se evalúan mediante pruebas unitarias previamente diseñadas para medir su capacidad de detección. El flujo de trabajo asegura que las mutaciones se apliquen sobre copias temporales del código, preservando la integridad del original, utilizando la librería copy de Python como medio para generar versiones independientes del código en cada iteración. Las mutaciones se combinan estratégicamente, evaluando el impacto acumulativo y el nivel de detección que logran las pruebas unitarias, permitiendo así calcular métricas como la tasa de mutación, que mide el porcentaje de mutantes eliminados (pruebas exitosas) frente a los sobrevivientes (fallas en las pruebas). Este enfoque no solo destaca las debilidades en la configuración actual, sino que también identifica áreas críticas para su mejora.

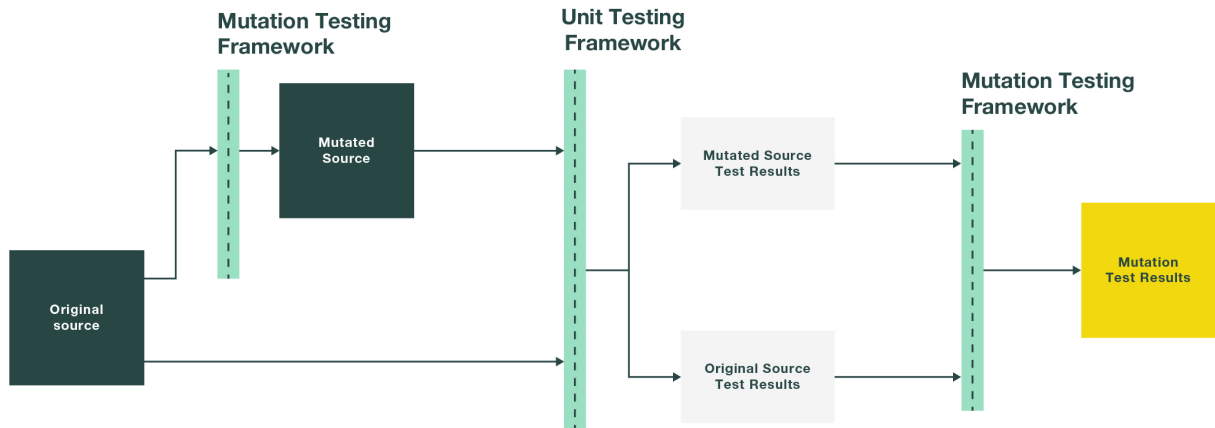


Figura 5.1: Pruebas de Mutación

Fuente: [41]

Adicionalmente, el proyecto adopta una arquitectura modular que divide el sistema en componentes funcionales, incluyendo un módulo de análisis de AST para identificar patrones clave, un módulo de generación de mutantes para realizar cambios en el código, y un módulo de evaluación para ejecutar pruebas unitarias y analizar los resultados. Este diseño asegura la extensibilidad del sistema y facilita la integración de nuevos operadores de mutación, adaptados a las necesidades específicas del proyecto, como mutaciones en configuraciones de CORS y restricciones de origen en aplicaciones Flask. Como propósito del sistema es proporcionar un análisis exhaustivo de seguridad, basado en datos concretos obtenidos durante las pruebas. La documentación resultante no solo detalla los cambios realizados y los resultados obtenidos, sino que también proporciona recomendaciones claras para optimizar la configuración y prevenir vulnerabilidades. Este enfoque asegura que las pruebas de mutación sirvan como una herramienta robusta para garantizar la calidad y seguridad del sistema en contextos reales, permitiendo una validación continua y adaptable a las necesidades específicas del entorno de desarrollo.

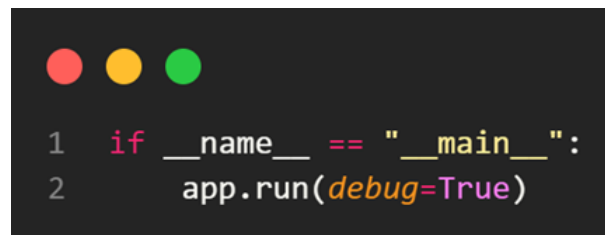
Operador de Mutación	Proceso	Descripción
Change Origins	Modificar las configuraciones en CORS añadiendo orígenes inválidos.	Permite verificar si las solicitudes desde dominios no autorizados son correctamente bloqueadas para asegurar la integridad de la aplicación.
Change Methods	Alterar los métodos HTTP permitidos como GET, POST, DELETE, etc.	Prueba si la configuración del sistema restringe métodos HTTP no autorizados que podrían comprometer recursos sensibles.
Alterar Allow Headers	Modificar las listas de encabezados permitidos en las solicitudes.	Garantiza que solo los encabezados seguros sean aceptados por la aplicación, reforzando la seguridad contra encabezados maliciosos.
Vary Supports Credentials	Alternar el valor de SUPPORTS CREDENTIALS entre True y False.	Examina cómo se gestionan las credenciales en diferentes configuraciones, asegurando que no se exponga información sensible de manera innecesaria.
Change Max Age	Modificar el tiempo máximo de almacenamiento en caché para pre-flight requests.	Este operador evalúa si el sistema gestiona adecuadamente las configuraciones de caché para evitar riesgos de desactualización en las políticas CORS.
Automatic Options	Desactivar la respuesta automática a solicitudes OPTIONS.	Comprueba si la aplicación responde correctamente a pre-flight requests, una parte esencial en la configuración de CORS.
Send Wildcard	Usar comodines (*) en configuraciones de CORS.	Prueba la robustez del sistema al manejar configuraciones abiertas que podrían ser peligrosas en entornos productivos.
Vary Header	Agregar o modificar el encabezado Vary en las respuestas.	Evalúa si el encabezado Vary está configurado adecuadamente para evitar problemas de caché que expongan datos a solicitudes no autorizadas.

Tabla 5.1: Operadores de Mutación de las Configuraciones

La tabla anterior contiene los operadores de mutación presentes en este proyecto, cabe aclarar

que, estos son configuraciones de seguridad propias de Flask en una librería llamada CORS. Los operadores de mutación típicos están diseñados para modificar variables y expresiones mediante operadores de reemplazo, inserción o eliminación. Además de modificar el código fuente del programa, los operadores de mutación también se pueden definir como reglas que buscan modificar la gramática utilizada para capturar la sintaxis de un artefacto de software. Al iniciar el análisis de mutaciones, este conjunto de pruebas debe ejecutarse con éxito contra el programa original para verificar su correctitud para el caso de prueba.

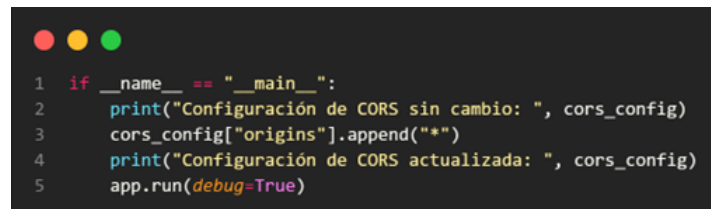
La forma de inyectar estos operadores de mutación en el código es encontrar el método principal de la clase, es decir, el que permite ejecutar la aplicación:

A screenshot of a code editor with a dark background. At the top left, there are three colored circles: red, yellow, and green. Below them, the following Python code is displayed:

```
1 if __name__ == "__main__":  
2     app.run(debug=True)
```

Figura 5.2: App Run
Fuente: Propia

Teniendo presente este método, se procede a inyectar los operadores de mutación antes de la ejecución de la aplicación. Para tener las múltiples variaciones de los mutantes y compararlos con la aplicación principal, se procede a mutar de todas las formas posibles, dependiendo de las variaciones que pueda tener cada configuración, obteniendo así múltiples archivos mutados para después compararlos con la aplicación principal.

A screenshot of a code editor with a dark background. At the top left, there are three colored circles: red, yellow, and green. Below them, the following Python code is displayed:

```
1 if __name__ == "__main__":  
2     print("Configuración de CORS sin cambio: ", cors_config)  
3     cors_config["origins"].append("")  
4     print("Configuración de CORS actualizada: ", cors_config)  
5     app.run(debug=True)
```

Figura 5.3: Inyección de Mutantes
Fuente: Propia

El enfoque tenido en cuenta para la implementación de operadores de mutación para configuraciones de seguridad, permite tanto identificar posibles vulnerabilidades, como poder optimizar y validar configuraciones deficientes.

Capítulo 6

Validación de Pruebas

Para la fase de tener cada prueba documentada, se buscó tener descrita cada historia de usuario, donde se busca tener cada necesidad del usuario cubierta:

- **Historia de Usuario 1 Creación de un Operador de Mutación para Pruebas de Seguridad**

Descripción	Criterios de Aceptación	Resultados Esperados	Resultados Obtenidos	Estado
Validar acceso autorizado con origen permitido	Detectar configuraciones incorrectas y permitir acceso solo con un origen autorizado.	Respuesta con datos con código 200.	Respuesta correcta con datos y código 200.	Aprobado
Validar acceso no autorizado con origen no permitido	Bloquear acceso desde orígenes no autorizados.	Mensaje "Unauthorized".	Mensaje "Unauthorized".	Aprobado
Validar métodos permitidos	Métodos no permitidos deben devolver un error adecuado (405), y los permitidos deben devolver datos.	Código 200 para métodos permitidos, 405 para métodos no permitidos.	Resultados coinciden con las expectativas.	Aprobado

Tabla 6.1: Prueba 1

Fuente: Elaboración propia

- **Historia de Usuario 2 Generación de Informes de Cobertura de Vulnerabilidades**

Descripción	Criterios de Aceptación	Resultados Esperados	Resultados Obtenidos	Estado
Generar informe de pruebas aprobadas.	Informe debe detallar configuraciones incorrectas detectadas por las pruebas.	Informe incluye descripciones de vulnerabilidades relacionadas en las configuraciones.	Informe generado con detalles esperados.	Aprobado
Validar cobertura de casos de prueba	Informe que incluye métricas de cobertura, mostrando casos exitosos y fallidos.	Métricas de cobertura incluidas con claridad.	Métricas generadas correctamente.	Aprobado

Tabla 6.2: Prueba 2

Fuente: Elaboración propia

■ Historia de Usuario 3 Validación de la Eficacia del Aplicativo

Descripción	Criterios de Aceptación	Resultados Esperados	Resultados Obtenidos	Estado
Validar detección de errores en configuraciones	La aplicación debe detectar errores en configuraciones de seguridad.	Configuración vulnerable detectada correctamente.	Detecta correctamente configuraciones inseguras.	Aprobado
Documentar cambios en configuraciones	Cambios realizados en configuraciones deben estar reflejados en los reportes de pruebas.	Documentación de cambios clara y precisa.	Documentación generada con los detalles requeridos.	Aprobado

Tabla 6.3: Prueba 3

Fuente: Elaboración propia

■ Historia de Usuario 4 Documentación de la Configuración de Seguridad en Flask

Descripción	Criterios de Aceptación	Resultados Esperados	Resultados Obtenidos	Estado
Crear guía de buenas prácticas	Guía debe estar integrada en las pruebas para referencia de usuarios.	Documento que incluye ejemplos y explicaciones claras.	Documento generado correctamente.	Aprobado
Verificar integración con código existente	Configuraciones de seguridad deben alinearse con el marco de trabajo utilizado (Flask).	Configuraciones y código deben ser consistentes.	Configuraciones alineadas y funcionales.	Aprobado

Tabla 6.4: Prueba 4

Fuente: Elaboración propia

Teniendo en cuenta que para este proyecto es necesario las pruebas unitarias para la mantenibilidad de las mutaciones efectuadas, es necesario mostrar los resultados que se esperan de estas mismas, cabe aclarar que las configuraciones que se muestran para aprobar cada prueba, son las que hacen a la aplicación segura, recomendado por la documentación de Flask [6]. A continuación se muestran:

- **Prueba Unitaria del Test Origin:** Consiste en dar orígenes de los cuales es permitido acceder a la aplicación, tales como URLs o IPs de alguna máquina.

Descripción	Criterios de Aceptación	Resultados Esperados	Resultados Obtenidos	Estado
Configuración utilizada para permitir orígenes en CORS.	Los aplicación debe poder ser accedida desde los orígenes permitidos.	Orígenes propuestos accedidos.	Acceso concedido.	Aprobado

Tabla 6.5: Prueba Unitaria Origin

Fuente: Elaboración propia

- **Prueba Unitaria del Test Methods:** Por medio de esta configuración se puede permitir que metodos de consultas se pretenden usar en la aplicación.

Descripción	Criterios de Aceptación	Resultados Esperados	Resultados Obtenidos	Estado
Permite verificar los métodos que permite la aplicación en CORS.	Los métodos permitidos deben coincidir con los permitidos en la aplicación.	Métodos propuestos permitidos.	Métodos autorizados.	Aprobado

Tabla 6.6: Prueba Unitaria Methods

Fuente: Elaboración propia

- **Prueba Unitaria Allow Headers:** Permite dar acceso dependiendo de la autorización disponible en el header.

Descripción	Criterios de Aceptación	Resultados Esperados	Resultados Obtenidos	Estado
Verifica que encabezados permite la aplicación.	Los encabezados deben ser los mismos configurados y los permitidos.	Acceso concedido.	Acceso concedido.	Aprobado

Tabla 6.7: Prueba Unitaria Allow Headers

Fuente: Elaboración propia

- **Prueba Unitaria Support Credentials:** Da la posibilidad de soportar o no credenciales.

Descripción	Criterios de Aceptación	Resultados Esperados	Resultados Obtenidos	Estado
Verifica si la aplicación permite o concede credenciales.	Para evitar vulnerabilidades se deben bloquear las credenciales.	False.	False.	Aprobado

Tabla 6.8: Prueba Unitaria Supports Credentials

Fuente: Elaboración propia

- **Prueba Unitaria Max Age:** Se verifica el valor de segundos que tiene la aplicación para acceder a cookies.

Descripción	Criterios de Aceptación	Resultados Esperados	Resultados Obtenidos	Estado
Tiempo de las cookies permitidos en la aplicación.	El tiempo configurado debe ser de 3600 segundos suficiente para intercambio de datos.	3600	3600	Aprobado

Tabla 6.9: Prueba Unitaria Max Age

Fuente: Elaboración propia

- **Prueba Unitaria Send Wildcard:** Se verifica si se permite el uso de comodines dentro de la aplicación.

Descripción	Criterios de Aceptación	Resultados Esperados	Resultados Obtenidos	Estado
Verifica si se puede acceder usando comodines.	La aplicación no debe permitir los comodines.	False.	False.	Aprobado

Tabla 6.10: Prueba Unitaria Send Wildcard

Fuente: Elaboración propia

- **Prueba Unitaria Automatic Options:** Verifica si están activas las opciones y configuraciones automáticas.

Descripción	Criterios de Aceptación	Resultados Esperados	Resultados Obtenidos	Estado
Habilita las configuraciones automáticas.	La configuración y opciones automáticas deben estar desactivadas y configuradas manualmente por el desarrollador.	False.	False.	Aprobado

Tabla 6.11: Prueba Unitaria Automatic Options

Fuente: Elaboración propia

- **Prueba Unitaria Vary Header:** Verifica la variedad de encabezados dentro de la aplicación.

Descripción	Criterios de Aceptación	Resultados Esperados	Resultados Obtenidos	Estado
Habilita los encabezados vary.	Deben estar habilitados los encabezados vary.	True.	True.	Aprobado

Tabla 6.12: Prueba Unitaria Vary Headers

Fuente: Elaboración propia

Capítulo 7

Conclusiones y Trabajos Futuros

7.1. Conclusiones

El presente trabajo de grado ha tenido como objetivo principal la implementación de pruebas de mutación para evaluar la calidad y seguridad de un software web diseñado con Flask. En el desarrollo del trabajo de grado se ha alcanzado los objetivos que han sido planteados inicialmente, y por consiguiente ha dejado como aprendizaje:

- La identificación de malas prácticas en la seguridad, específicamente en las configuraciones de un sitio web permite entender los puntos críticos donde las aplicaciones, especialmente las desarrolladas con Flask suelen ser más vulnerables, ofreciendo una base para desarrollar medidas correctivas. Un análisis previo permite tener un mejor diseño en los operadores de mutación específicos, buscando cubrir un amplio espectro de vulnerabilidades.
- Una inadecuada selección de malas prácticas contribuyó a definir los criterios de evaluación de seguridad a abordar, para el proyecto, brindando un enfoque que asegure que las pruebas de mutación vayan acorde con los riesgos de seguridad más relevantes para aplicaciones web desarrolladas en Flask.
- La descripción de los operadores de mutación permitió establecer una metodología que simula los probables errores de configuración en el código de seguridad de Flask que pueden ocurrir en cualquier desarrollo de un proyecto, lo que facilita la evaluación de la robustez de las pruebas de seguridad. Esto asegura que el proyecto abarque un espectro amplio de vulnerabilidades comunes.
- Definir los operadores de mutación basados en código permite brindar un marco de desarrollo de pruebas dirigidas a vulnerabilidades específicas, permitiendo que el sistema de mutación sea flexible y adaptable a distintas configuraciones de seguridad.
- La implementación de los operadores de mutación en una aplicación que detecte las

malas prácticas de seguridad en la configuración busca asegurar que la herramienta de evaluación de seguridad cumpla con su propósito, proporcionando una plataforma para realizar pruebas automatizadas en múltiples escenarios y niveles de complejidad.

- El desarrollo de un software que detecte vulnerabilidades facilita la replicación de pruebas de seguridad, permitiendo que otros desarrolladores hagan uso de la herramienta como un soporte continuo en la evaluación de aplicaciones Flask, contribuyendo así a la mejora de prácticas de seguridad en el desarrollo web.
- La validación mediante un caso de estudio demostró que los operadores de mutación implementados son eficaces para identificar errores en las configuraciones de seguridad, que respalda la funcionalidad y precisión de la herramienta desarrollada.
- El sistema de mutación como método de evaluación de seguridad se comprobó mediante métricas de cobertura y detección de vulnerabilidades, lo cual evidencia que la herramienta es adecuada para detectar configuraciones inseguras en aplicaciones Flask.

Se tuvo en cuenta que, al tener experiencia en el manejo de Python, hubo una facilidad al manejar el framework, sin embargo, hubo que realizar estudio de la implementación de API's, pues a la hora de desarrollar aplicaciones de tipo web se requiere una lógica que adapte la creatividad con las bases obtenidas.

7.2. Trabajos futuros

Teniendo en cuenta que existen varios ámbitos a considerar dentro de las configuraciones de seguridad:

- Se propone a extender la herramienta para evaluar aplicaciones desarrolladas en otros frameworks de Python, como Django, adaptándola a sus respectivas configuraciones de seguridad y prácticas de desarrollo. Por otro lado, también se busca extender estas prácticas de detectar vulnerabilidades mediante pruebas de mutación para otros lenguajes de programación permitiendo extender la cobertura de la herramienta.
- Crear una interfaz gráfica que permita la facilidad en el uso del motor de pruebas, permitiendo a usuarios no técnicos ejecutar e interactuar con las pruebas de seguridad en sus aplicaciones Flask sin necesidad de interacción directa con la línea de comandos. Incorporar opciones de visualización de resultados y generación de reportes detallados sobre las vulnerabilidades encontradas, haciendo la herramienta más accesible y útil para gran variedad de usuarios y perfiles dentro de los equipos de desarrollo.
- Permitir el acceso de técnicas de aprendizaje automático que permita establecer un análisis de patrones de seguridad en el código de Flask y sugerir configuraciones seguras o identificar configuraciones incorrectas de forma automática. Estos modelos entrenados con conjuntos de datos de configuraciones seguras e inseguras proporcionarían un soporte automatizado en tiempo real durante el desarrollo.

Bibliografía

- [1] Propia, “Repositorio github,” 2024. <https://github.com/HaroldMontanoHurtado/Prueba2.git>.
- [2] Amazon Web Services, “¿qué es una aplicación web? - explicación de las aplicaciones web,” 2023. <https://aws.amazon.com/es/what-is/web-application/>.
- [3] J. Domingo Muñoz, “Qué es flask y ventajas que ofrece — openwebinars,” 2023. <https://openwebinars.net/blog/que-es-flask/>.
- [4] O. E. Mahjoubi and J. Duch, “Detección de vulnerabilidades y generación de alertas de seguridad para aplicaciones web,” *Universitat Oberta de Catalunya*, 2019.
- [5] MIT, “pytest: helps you write better programs — pytest documentation,” 2023. <https://docs.pytest.org/en/latest/>.
- [6] Flask, “Manejo de la configuración,” 2023. <https://flask.palletsprojects.com/es/latest/config/>.
- [7] I. Ribera Díaz, “Mutation testing: quality is a priority,” 2023. <https://www.mytra.es/blog-post/test-de-mutacion-la-calidad-es-prioritaria>.
- [8] O. J. Bertoglio, *Introducción a la TEORIA GENERAL DE SISTEMAS*. Limusa S.A, 2004.
- [9] W. W. Royce, “Managing the development of large software systems,” 2023. <https://blog.jbrains.ca/assets/articles/royce1970.pdf>.
- [10] B. W. Boehm, *A spiral model of software development and enhancement*. Long Beach Calif, 1998.
- [11] S. Collado González, “Teoría de la evolución - universidad de navarra,” 2023. <https://www.unav.edu/web/ciencia-razon-y-fe/teoria-de-la-evolucion#toc1>.
- [12] M. Papadakis, M. Kintis, J. Zhang, Y. Jia, Y. Le Traon, and M. Harman, “Mutation testing advances: An analysis and survey,” 2018. *Advances in Computers*, vol. 112, pp. 275–378, Jan. 2019, doi: 10.1016/bs.adcom.2018.03.015.
- [13] OWASP, “Security misconfiguration,” 2021. https://owasp.org/Top10/A05_2021-Security_Misconfiguration/.

- [14] Python Software Foundation, “Consideraciones de seguridad — documentación de flask (2.3.x).,” 2023. <https://flask.palletsprojects.com/es/latest/security/>.
- [15] Smart Bear, “Mutation testing: What is it, and how can it help your code?,” 2023. <https://smartbear.com/blog/what-is-mutation-testing-and-how-can-it-help-your-co/>.
- [16] Synopsys, “Mutation testing: A critical part of software testing,” 2023. <https://www.synopsys.com/software-integrity/resources/knowledge-database/mutation-testing.html>.
- [17] MDN Web Docs, “What is a web app?,” 2024. https://developer.mozilla.org/es/docs/Learn/Common_questions/What_is_a_web_app.
- [18] D. Jones, “World wide web,” *Embroidery*, 2012.
- [19] P. Vázquez, *Creacion De Sitios Web*. Colecciones Manuales USERS, 2008.
- [20] Flask, “Flask documentation.,” 2024. <https://flask.palletsprojects.com/en/2.0.x/>.
- [21] OWASP Foundation, “Owasp top ten web application security risk.,” 2024. <https://owasp.org/>.
- [22] R. Gopinath, C. Jensen, and A. Groce, “Code coverage for suite evaluation by developers.,” 2014. <https://ieeexplore.ieee.org/>.
- [23] Y. Jia and M. Harman, “An analysis and survey of the development of mutation testing.,” 2011. <https://link.springer.com/>.
- [24] K. Beck and C. Andres, “Metodología xp (extremme programming).,” 2004. <https://www.agilealliance.org/>.
- [25] NIST, “Guide to vulnerability analysis.,” 2024. <https://csrc.nist.gov/>.
- [26] ULL, “Árboles de análisis abstracto,” 2023. <https://ull-esit-gradoii-pl.github.io/temas/syntax-analysis/ast.html>.
- [27] M. Fernández Ciruela, “Proyecto final de carrera gestiÓN de un proyecto de control de pruebas de un aplicativo.,” 2010.
- [28] J. Santiago Acevedo Correa and A. Mario Linares Vázquez, “Pruebas de mutaciÓN para aplicaciones web y mÓviles hitpa: Herramienta integradora de pruebas para android.,” 2018.
- [29] Real Python, “Introduction to flask,” 2023. <https://realpython.com/tutorials/flask/>.
- [30] Digital Ocean, “How to use flask for web development,” 2023. <https://www.digitalocean.com/community/tutorials/how-to-use-flask-for-web-development>.
- [31] SQLAlchemy, “Orm in python,” 2023. <https://docs.sqlalchemy.org/>.

- [32] DevCamp, “¿qué es flask?,” 2024. Disponible en: <https://bit.ly/3EjrFlask>.
- [33] OWASP, “Cross-site scripting (xss),” 2023. <https://owasp.org/www-community/attacks/xss>.
- [34] K. R. Murali, V. S. Anitha, “Security and vulnerability analysis of flask web framework,” 2020.
- [35] P. B. Prakash, M. N. Jhanjhi, M. K. S. Khan, “A comprehensive study on vulnerabilities of web application frameworks,” 2019.
- [36] Amoeboids, “Extreme programming: What is it exactly?,” 2024. <https://amoeboids.com/blog/extreme-programming/>.
- [37] Harold Montano - Juan Veloza, “Anexos tg,” 2024. <https://drive.google.com/drive/u/1/folders/1vGBevC1t2xzkReGmvVrjcwvwE3DKYQwM>.
- [38] L. Bass, P. Clements, R. Kazman, “Software architecture in practice,” 2012. 3rd ed. Addison-Wesley.
- [39] Tibbo, “Arquitectura modular,” 2024. <https://aggregate.digital/es/technology/architecture/modular-architecture.html>.
- [40] Cássio Scofield, “Códigos http com propósito,” 2021. <https://www.linkedin.com/pulse/c>
- [41] Ignacio Ribera Díaz, “Test de mutación: la calidad es prioritaria,” 2021. <https://www.mytra.es/blog-post/test-de-mutacion-la-calidad-es-prioritaria>.

ANEXOS

Requerimientos Funcionales

Requerimiento Funcional 1	
Nombre Requerimiento	Descripción
Ejecución Pruebas de Mutación.	El sistema debe permitir la ejecución de pruebas de mutación en el código fuente de aplicaciones Flask, generando variantes del código original para evaluar la capacidad de las pruebas en detectar cambios o vulnerabilidades.

Requerimiento Funcional 1	
Nombre Requerimiento	Descripción
Ejecución Pruebas de Mutación.	El sistema debe permitir la ejecución de pruebas de mutación en el código fuente de aplicaciones Flask, generando variantes del código original para evaluar la capacidad de las pruebas en detectar cambios o vulnerabilidades.

Requerimiento Funcional 2	
Nombre Requerimiento	Descripción
Generación y Aplicación de Operadores de Mutación	El sistema debe contar con un conjunto de operadores de mutación que se puedan aplicar al código fuente de aplicaciones Flask. Estos operadores deben estar categorizados y ser configurables para que el usuario pueda seleccionar y ejecutar solo los necesarios para su evaluación de seguridad.

Requerimiento Funcional 3	
Nombre Requerimiento	Descripción
Reporte de Resultados de Pruebas	Al finalizar una prueba de mutación, el sistema debe generar un informe detallado que incluya los resultados de cada caso de prueba, las mutaciones detectadas, el nivel de cobertura alcanzado y recomendaciones para mejorar la seguridad según las vulnerabilidades encontradas.

Requerimiento Funcional 4	
Nombre Requerimiento	Descripción
Evaluación de la Cobertura de Seguridad.	El sistema debe proporcionar métricas para evaluar la cobertura de seguridad de las aplicaciones Flask probadas, determinando la

	cantidad de vulnerabilidades detectadas y su impacto potencial. Estas métricas deben estar accesibles en el informe final para ayudar en la toma de decisiones de mejora de seguridad.
--	--

Tomando en cuenta la detección de vulnerabilidades en aplicaciones basadas en Flask, es fundamental comprender cómo se prueba la seguridad en dicho entorno. Las pruebas de seguridad en aplicaciones Flask involucran técnicas como el análisis estático del código, pruebas de penetración y auditorías regulares (2024, Python software foundation) para identificar y mitigar posibles vulnerabilidades, donde se busca considerar que la aplicación tenga una seguridad que le permita cierto nivel de confiabilidad para el usuario; esto teniendo en cuenta un código limpio, una conexión con la base de datos que tenga un correcto cifrado de datos y permitir una conexión con el servidor fiable para que los puertos por donde se inserte información, no existan vulnerabilidades.

Las pruebas de mutación se aplican introduciendo deliberadamente mutaciones en el código para evaluar la efectividad de las pruebas existentes en detectar posibles ataques (2020, Martin Thoma), especialmente aquellos comunes en aplicaciones web como inyecciones de SQL y XSS.

La aplicación de estas pruebas de mutación pueden automatizarse utilizando herramientas específicas para Python, como mutmut (2020, Martin Thoma), el cual se puede considerar como una herramienta propia del lenguaje de programación que busca la facilidad a la hora de implementar casos de pruebas, especificando las pruebas de mutación contando con funciones que facilitan su implementación.

Por otro lado, surgen interrogantes a la hora de automatización de pruebas de seguridad, cómo mitigar las vulnerabilidades comunes en aplicaciones Flask, y cómo comparar la efectividad de las pruebas de mutación con otras metodologías de seguridad.

Entre estos nuevos interrogantes surgen:

- ¿Cómo se puede automatizar el proceso de pruebas de seguridad en aplicaciones Flask?

El proceso de automatización de pruebas de seguridad en aplicaciones Flask puede lograrse mediante el uso de herramientas especializadas en pruebas de seguridad, como OWASP ZAP (Zed Attack Proxy) o Burp Suite (2024, OWASP), que pueden escanear automáticamente las aplicaciones en busca de vulnerabilidades conocidas, como inyecciones de SQL, XSS (Cross-Site Scripting) y CSRF (Cross-Site Request Forgery). Además, se pueden implementar scripts personalizados o pruebas de penetración automatizadas para simular ataques y evaluar la resistencia de la aplicación a diferentes

escenarios de amenazas. La combinación de estas técnicas ayuda a garantizar que las aplicaciones Flask se mantengan seguras y protegidas contra posibles ataques.

- ¿Cuáles son las vulnerabilidades de seguridad más comunes en aplicaciones web desarrolladas con Flask y cómo se pueden mitigar?

Las vulnerabilidades de seguridad más comunes en aplicaciones web desarrolladas con Flask incluyen inyecciones de SQL, XSS (Cross-Site Scripting), CSRF (Cross-Site Request Forgery) (2024, OWASP), autenticación y autorización inadecuadas, y gestión insegura de sesiones. Para mitigar estas vulnerabilidades, es fundamental implementar buenas prácticas de seguridad, como validar y sanitizar todas las entradas de usuario para prevenir inyecciones de SQL y XSS, utilizar tokens CSRF para protegerse contra ataques de falsificación de solicitudes entre sitios, implementar mecanismos robustos de autenticación y autorización, como hashes seguros y cifrado de contraseñas, y gestionar las sesiones de manera segura, utilizando técnicas como el almacenamiento seguro de cookies y la expiración adecuada de sesiones.

- ¿Cuáles son las mejores prácticas para desarrollar y mantener un conjunto de pruebas de seguridad robusto para aplicaciones web basadas en Flask?

Para garantizar la seguridad en aplicaciones web basadas en Flask, es fundamental implementar un conjunto de pruebas robustas. Implicando identificar y priorizar áreas críticas, utilizar herramientas de prueba automatizadas. Además, se deben aplicar pruebas de mutación para evaluar la capacidad de detección de cambios maliciosos en el código, incluir casos de prueba de seguridad durante el desarrollo, mantener actualizados los casos de prueba y fomentar la colaboración entre equipos de desarrollo, pruebas y seguridad. Estas prácticas ayudarán a proteger la aplicación contra vulnerabilidades comunes y a mantener la seguridad de manera efectiva.