Desarrollo de un prototipo de una herramienta para identificar vulnerabilidades de código en programas desarrollados con TypeScript

Leonel Casallas Rodríguez y Jorge Iván Guzmán Godoy

Facultad de Ingeniería

Universidad Distrital Francisco José de Caldas

Especialización en Ingeniería de Software

Director

MSc. Jhon Freddy Parra Peña

Revisor

PhD. Beatriz Elisa Jaramillo Moreno

Octubre 22, 2024

Tabla de contenido

Introducción 7

Parte I. Contextualización de la investigación 9

Capítulo 1. Descripción de la investigación 9

1.1 Planteamiento / Identificación del Problema 9

1.2 Objetivos 9

1.3 Justificación del trabajo / investigación. 10

1.4 Hipótesis 11

1.5 Marco referencial 11

1.6 Metodología de la investigación 15

1.7 Organización del trabajo de grado 18

1.8 Estudio de sistemas previos 19

1.8.1 Estudio #1 19

1.8.2 Estudio #2 20

1.8.3 Estudio #3 22

1.8.4 Estudio #4 23

1.8.5 Estudio #5 24

Parte II. Desarrollo de la investigación 26

Capítulo 2. Recolección y análisis de requisitos 26

2.1 Recolección de información 26

2.1.1 Vulnerabilidades de código....26
2.2 Análisis de requisitos....32
2.3 TypeScript....35
Capítulo 3. Diseño rápido del prototipo....37
3.1 Especificación de requisitos....38
3.1.1 Requisitos funcionales....38
3.1.2 Requisitos no funcionales....39
3.2 Inteligencia artificial....40
3.3 Extensiones en Visual Studio Code....42
Capítulo 4. Desarrollo del prototipo de extensión....43
4.1 Definición de la estructura básica de la extensión....43
4.1.1 Extensión Manager....44
4.1.2 Analyzer....45
4.1.3 Log....45
4.1.4 HoverProviderManager....46
4.2 Creación del proyecto....47
4.3 Incorporación y acoplamiento del modelo de lA en la herramienta....49
Capítulo 5. Pruebas de la extensión....51
Capítulo 6. Refinación del prototipo bajo Grog....53
Parte III. Cierre de la investigación....56

Capítulo 7. Resultados y discusión.....56

7.1 Conclusiones.....56

Bibliografía 57

Anexos.....62

1. Repositorio que contiene el código de la extensión 62

2. Manual de instalación alojado en la descripción del repositorio.....62

3. Resultado de implementación y uso de la herramienta 65

Tabla de ilustraciones

Ilustración 1: Modelo inteligencia artificial VulPecker (Zhen et al., 2016) 16

Ilustración 2: Modelo inteligencia artificial VulDeePecker (Zhen et al., 2018) 17

Ilustración 3: Ejemplo de uso del prototipo para detección de vulnerabilidades usando análisis estático (Moon & Kim, 2021) 18

Ilustración 4: Configuración de búsqueda de vulnerabilidades en National Vulnerability Database 28

Ilustración 5: Búsqueda de vulnerabilidades en Common Vulnerabilities and Exposures 29

Ilustración 6: Búsqueda de vulnerabilidades Open Web Application Security Project (OWASP) 30

Ilustración 7: Búsqueda de vulnerabilidades Common Weakness Enumeration 31

Ilustración 8: Resultado de Búsqueda National Vulnerability DataBase NVD por palabra TypeScript. 33

Ilustración 9: Resultados Common Vulnerabilities and Exposures CVE por filtrado TypeScript 33

Ilustración 10: Resultado de la Base Exploit Data Base 34

Ilustración 11: Diagrama de clases de la extensión de detección de vulnerabilidades 47

Ilustración 12: Código del prototipo en Visual Studio Code 49

Ilustración 13: Repositorio de código de TypeScript de Microsoft 52

Ilustración 14: Configuración de parámetros al modelo Groq con llama 3.1 55

Ilustración 15 Resultado de acceder al repositorio de la extensión. 62

Ilustración 16 Requisitos de instalación previos o entorno de desarrollo 63

Ilustración 17 Configuración de Groq como Api Key mediante cuenta 64

Ilustración 18 Paso a paso de instalación de la extensión en Visual Studio Code.....64

Ilustración 19 Estructura final del proyecto luego de la instalación de instancias.....65

Ilustración 20 Ventana para test del código objetivo.....66

Ilustración 21 Despliegue de Analyze Code al código ts de prueba.....66

Ilustración 22 Inicio de la herramienta Analyze Code en entorno de ejecución.....67

Ilustración 23 Finalización del proceso de análisis del código TS 67

Ilustración 24 Vulnerabilidad 1 de 3 o resultado de Analyze Code.....68

Ilustración 25 Descripción de la vulnerabilidad detectada por Analyze Code 68

Introducción

En el contexto de la seguridad informática, la detección de vulnerabilidades en el código es un área de interés actual (Rebecca L. Russell et al., 2018), lo anterior teniendo en cuenta el creciente número de ataques cibernéticos, analizado en un vecino cercano como lo es (Melissa Valeria Echeverría Joniaux et al., 2020) lo cual permite pensar en la sofisticación de las técnicas empleadas por los atacantes en un contexto más digital.

Es así como hay que buscar asegurar que el software esté libre de vulnerabilidades, hasta el punto de ser esencial para la protección de información y el mantenimiento de la integridad del mismo sistema, En este proyecto nos enfocamos en el desarrollo de un prototipo de herramienta que permita detectar las vulnerabilidades más comunes en códigos de programas desarrollados bajo JavaScript y TypeScript, esto bajo la premisa de la amplia utilización de estos lenguajes en la industria del desarrollo web (Márquez Coca et al., 2023)

Pese a los avances en el despliegue de herramientas y métodos para detectar vulnerabilidades, aún es necesario mejorar la precisión y la efectividad explorando nuevas técnicas, lo que podría beneficiarse de enfoques no convencionales, que permitan respuestas más veloces a los desafíos de la seguridad informática planteados, sin comprometer la integralidad con el flujo de trabajo del desarrollo web, tal como se aplica a la detección de noticias falsas (Escobar Cerón & Quevedo Cortés, 2021)

En este proyecto, proponemos un enfoque que combina la clasificación de vulnerabilidades basadas en su importancia para la seguridad de la aplicación y la aplicación de algoritmos de inteligencia artificial para la detección de dichas vulnerabilidades. Nuestro objetivo es desarrollar un prototipo que no solo identifique las vulnerabilidades, sino que también lo haga de manera eficiente y con un bajo porcentaje de falsos positivos. El prototipo

será implementado como un servicio web, lo que permitirá su fácil integración con gestores de paquetes y/o extensiones, facilitando su uso por parte de los desarrolladores.

En resumen, este trabajo se propone:

- Clasificar las vulnerabilidades de código según su nivel de importancia para la seguridad de la aplicación.
- Aplicar algoritmos de inteligencia artificial para la detección de vulnerabilidades en el código de sistemas desarrollados con TypeScript.
- Implementar el prototipo en un servicio web, permitiendo su integración y uso en programas objetivo.

Estas iniciativas buscan avanzar el estado del arte en la detección de vulnerabilidades, proporcionando una herramienta valiosa para los desarrolladores y contribuyendo a la creación de software más seguro y confiable.Haga clic o pulse aquí para escribir texto.

Parte I. Contextualización de la investigación

Capítulo 1. Descripción de la investigación

1.1 Planteamiento / Identificación del Problema

Actualmente se presenta un incremento constante de ciberataques (Laura Lesmes Diaz 2023 agosto, El Tiempo"Fortinet") estos últimos son realizados a las páginas web y las aplicaciones como Twilio (Valle, 2022). Una de las principales razones por las que se presenta este incremento es debido a las vulnerabilidades que los atacantes encuentran dentro del código fuente del sistema (González, 2023) y que provocan que la integridad de los datos registrados por los usuarios quede expuesta. Que los usuarios (Desarrolladores) pueden llegar a cuestionar la viabilidad del uso de una herramienta para analizar el código de un programa en búsqueda de vulnerabilidades. A partir de esta incógnita, surgen las siguientes preguntas:

- ¿Qué aspectos tienen en cuenta los ciber atacantes antes de vulnerar un sistema informático?
- ¿Cuáles son las principales vulnerabilidades que se pueden encontrar en el código fuente de un sistema informático?
- ¿Qué tan importante es mantener la integridad de la información dentro de un sistema informático?

1.2 Objetivos

El objetivo general de este proyecto es desarrollar un prototipo de una herramienta que permita detectar las vulnerabilidades de código más comunes dentro de un programa desarrollado con TypeScript. Para lograr el objetivo general, se plantean los siguientes objetivos específicos:

- Clasificar las vulnerabilidades de código en base a su nivel de importancia para la seguridad de la aplicación.

- Aplicar algoritmos de inteligencia artificial para la búsqueda de las vulnerabilidades que pueda tener el código de un sistema.
- Implementar el prototipo en un servicio web (gestores de paquetes y/o extensiones) para que pueda ser utilizado por el programa objetivo.

1.3 Justificación del trabajo / investigación.

Hoy en día, existen muchas herramientas que usan la inteligencia artificial para ser utilizadas en un campo en específico para solucionar una necesidad específica como GitHub Copilot, Intellicode, CodeWhisperer, entre otras. Sin embargo, una de las necesidades que ha sido poco explorada hasta la fecha es el análisis del código de una aplicación y/o página web en busca de posibles vulnerabilidades que puedan afectar a un sistema de información, principalmente en el riesgo que pueden correr los datos de los usuarios que se mueven por medio de este sistema, y que pueden generar problemas a nivel legal y económico en las empresas que los implementan y sus desarrolladores.

A nivel metodológico, la elaboración del prototipo requiere de las bases de la inteligencia artificial, por medio de la realización de adaptaciones de la información que actualmente se encuentra disponible con respecto a las vulnerabilidades de código más populares, como bases de datos y/o repositorios que almacenan las principales vulnerabilidades de código que pueden afectar a un sistema y los lenguajes de programación donde es aplicable, generando valor al proponer una solución innovadora con esta herramienta en un campo poco explorado hasta el momento.

Por ende, esta investigación se realiza con el objetivo de proponer una herramienta que permita propiciar la automatización del proceso de verificación de seguridad del código de una aplicación realizada con TypeScript por medio de la inteligencia artificial, incluyendo entre otras cosas las variables y la estructura utilizada en el código del aplicativo analizado.

1.4 Hipótesis

La hipótesis formulada para el desarrollo de este proyecto es de causalidad, y se estipula abajo:

La implementación de la herramienta para el análisis del código desarrollada con este proyecto permitirá identificar las vulnerabilidades dentro del código de los aplicativos desarrollados con TypeScript.

A partir de esta hipótesis se identifican las siguientes variables:

- Variable independiente: Implementar la herramienta para el análisis del código.
- Variable dependiente: Identificar vulnerabilidades dentro del código de un aplicativo desarrollado con TypeScript.

1.5 Marco referencial

En el mundo actual, la informática ha continuado con su evolución en gran medida, especialmente en el área de la ciberseguridad, en donde se puede notar un gran crecimiento tanto en el aprendizaje como en su demanda. Sin embargo, también se puede notar que no posee una estandarización específica, por lo que el estilo de conocimiento empleado es muy variable debido a cómo están compuestos a nivel temático cada uno de los cursos y certificaciones especializadas en la ciberseguridad, y que provocan retrasos o vulnerabilidades al implementarla en un proyecto que dejan la puerta abierta a un posible ciberataque (Elías Carabaguiaz Gonzalez, 2017)

Por este motivo, se requiere que todos los frentes de las aplicaciones estén protegidos ante gran parte de los ciberataques para evitar que la información de los usuarios se vea comprometida. Esto se debe a que el software es moldeable y permite ser modificado con base en las necesidades que debe cumplir por medio de algunas mejoras y debido a esto, llega

al límite de mejora y requiere un cambio total. Este proceso provoca que el software sea cada vez más inseguro cuando se le hagan mejoras y llegará el momento en el que la seguridad se vea tan afectada después de muchos parches que sea mejor pensar en otro software (Wanumen Silva, 2006).

A raíz de esto, han surgido una gran cantidad de herramientas que permiten identificar los posibles riesgos de seguridad que posee una aplicación en cualquier capa del software. Para el análisis de código fiable y seguro, se ha comenzado la implementación de herramientas que utilizan inteligencia artificial (IA), como afirma Ángel Pérez en una entrevista realizada en el ISMS Forum y registrada en la revista Inteligencia Artificial y Ciberseguridad (2023), "la IA ya forma parte de los actuales sistemas de protección en la mayoría de las organizaciones. Sin ir más lejos, un ejemplo actual del uso de inteligencia artificial en la protección de las organizaciones son los sistemas EDR (endpoint detection and response) que contienen algoritmos capaces de detectar comportamientos anómalos en los sistemas; pero también se incorporan en los sistemas de detección antifraude". (p.10)

Algunos conceptos básicos que se relacionan con el propósito de esta investigación son los siguientes:

- Ciberseguridad: Es el de servicios, mecanismos y políticas que aseguran que el modo de operación de un sistema informático sea seguro, es decir: es el conjunto de técnicas y métodos para proteger los activos cibernéticos de ataques. (José Luis Solleiro Rebolledo et al., 2022)
- Vulnerabilidad del código: Es un fallo en un programa o sistema informático. Pero no cualquiera, sino un fallo de seguridad. Es necesaria esta distinción puesto que no todos los errores de programación derivan en fallos de seguridad. Un error en un programa puede llevar a que no funcione correctamente o que su comportamiento no sea el esperado,

pero no todos estos tipos de problemas pueden considerarse fallos de seguridad. Según la capacidad de aprovecharse de este defecto, la vulnerabilidad será más o menos grave (Instituto Nacional de Tecnologías de la Comunicación, 2011).

- Seguridad de la información de la aplicación: Consiste en la preservación de la confidencialidad, integridad y disponibilidad, así como de los sistemas implicados en el tratamiento de la información dentro de una organización (Lyda Janeth Rodríguez Torres, 2017).

- Código seguro: Es el conjunto de buenas prácticas que se usan para determinar si un fragmento de código es seguro o no. Esto permite analizar su estructura de manera sencilla por medio de un conjunto de pasos definidos.

- Inteligencia artificial para ciberseguridad: La ciberseguridad se enfrenta a múltiples retos, como son la detección de intrusiones, la protección de la privacidad, la defensa proactiva, la identificación de comportamientos anómalos o la detección de amenazas sofisticadas, pero, sobre todo, a las cambiantes amenazas que aparecen continuamente. Debido a ello, se están explorando métodos basados en la IA que faciliten el análisis y la toma de decisiones en tiempo real para una rápida detección y reacción ante ciberataques. También se está empleando la IA para desarrollar sistemas auto adaptables y que permitan automatizar las respuestas ante ciber amenazas (Ana Ayerbe, 2020) .

- Prototipo servicio web: Se puede considerar como una pequeña implementación de un sistema de software, que ayuda al diseño y que puede ser usada como una técnica de determinación y validación de requerimientos (MORENO MARÍN JOHN EDISON, 2019) .

- Node.js: Ideado como un entorno de ejecución de JavaScript orientado a eventos asíncronos, Node.js está diseñado para crear aplicaciones network escalables por

medio de un modelo de eventos que se ejecuta de manera directa (OpenJS Foundation, 2023)
.

- React: Es una librería de JavaScript flexible, eficiente y declarativa para construir interfaces de usuario, permitiendo al desarrollador crear interfaces de usuario complejas usando trozos de código pequeños e independientes llamados componentes (Meta Platforms, 2023).
- TypeScript: Es un lenguaje de programación fuertemente tipado que está construido sobre JavaScript, ofreciendo una mejor personalización a cualquier escala (Microsoft, 2023) .

A lo largo de los últimos años, muchos tipos de escáner de vulnerabilidades de uso comercial para aplicaciones y páginas web han sido creados e implementados con éxito como, por ejemplo, Wireshark, Hashcat, NMap, Nikto, entre otros. Además de eso, también han surgido diversas normas y leyes para regular la seguridad de los usuarios en linea:

- Decreto 338 del 8 de marzo de 2022: Establece los lineamientos generales para fortalecer la gobernanza de la seguridad digital, la identificación de infraestructuras críticas cibernéticas y servicios esenciales, la gestión de riesgos y la respuesta a incidentes de Seguridad Digital (MINITIC, 2022) .
- Norma ISO/IEC 27001: Promueve la adopción de un enfoque basado en procesos, para establecer, implementar, operar, hacer seguimiento, mantener y mejorar el sistema de gestión de la seguridad de la información de una organización(ICONTEC, 2006) .
- Norma ISO/IEC 27032: Ofrece unas líneas generales de orientación para fortalecer el estado de la Ciberseguridad en una empresa, utilizando los puntos técnicos y estratégicos más importantes para esa actividad y los que están relacionados con la seguridad

en las redes, seguridad en internet, seguridad de la información y seguridad de las aplicaciones(Grupo ACMS Consultores, 2023.)

1.6 Metodología de la investigación

El tipo de estudio a utilizar es el exploratorio, debido a que se realizará una investigación documental sobre las principales vulnerabilidades que afectan al código sin profundizar mucho en los detalles de cada una de ellas, como su origen y registro detallado. El método utilizado para la investigación será cualitativo, ya que la información recolectada no estará basada en datos estadísticos y/o numéricos medibles.

Para obtener la información necesaria para el desarrollo del proyecto, se empleará la investigación documental, la cual incluirá investigaciones, artículos, libros, documentación, blogs, foros y videos.

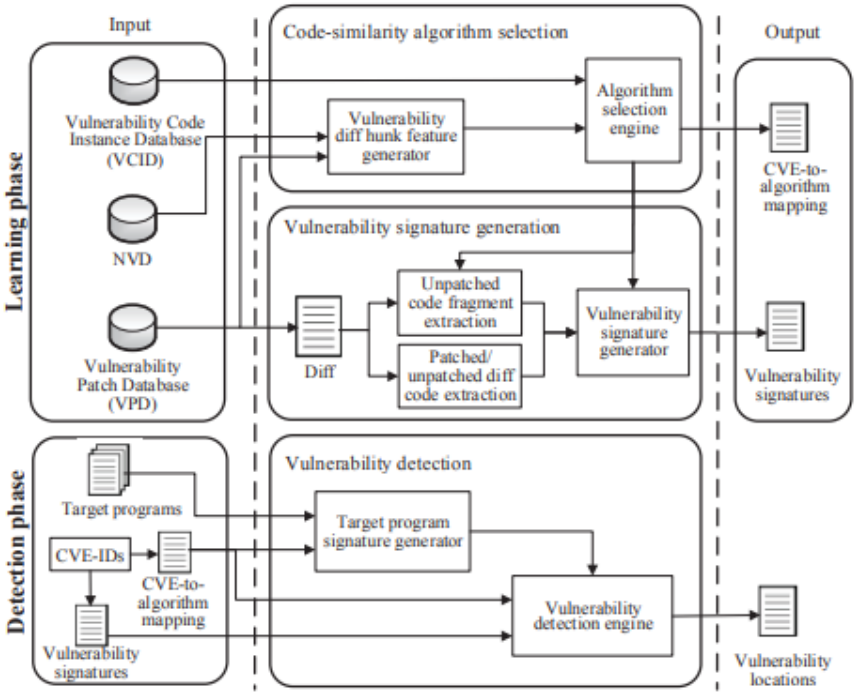
A continuación, se encuentran los resultados de la investigación documental inicial:

Para obtener un mejor contexto de la implementación de herramientas relacionadas con el análisis de vulnerabilidades del código a lo largo del tiempo, se realiza una investigación detallada de artículos y trabajos que generaron una implementación con una funcionalidad similar, incluso si el público objetivo es distinto (por ejemplo, que estén enfocados a un lenguaje de programación diferente).

Una de las implementaciones de este tipo de herramienta más conocida es VulPecker, que funciona como un sistema de detección automática de vulnerabilidades basadas en el análisis de código similar, la cual fue presentada en 2016 y desarrollada para los lenguajes de programación C y C++. Uno de los principales problemas que se encontró durante el desarrollo de la implementación es la falta de información base sobre las vulnerabilidades, el cual fue solucionado con la creación de una base de datos que registre la información de las principales encontró vulnerabilidades y las líneas de código que las generan. Otro problema

de gran importancia que se es el rango de variabilidad que puede tener un código con respecto al ejemplo almacenado en base de datos, por ejemplo, pueden existir diferencias en el nombre de las variables o la estructura del código; el cual fue solucionado por medio del diseño de algoritmos que permiten analizar el código existente y determinar si la vulnerabilidad está presente sin importar la estructura del código.

Ilustración 1: Modelo inteligencia artificial VulPecker (Zhen et al., 2016)

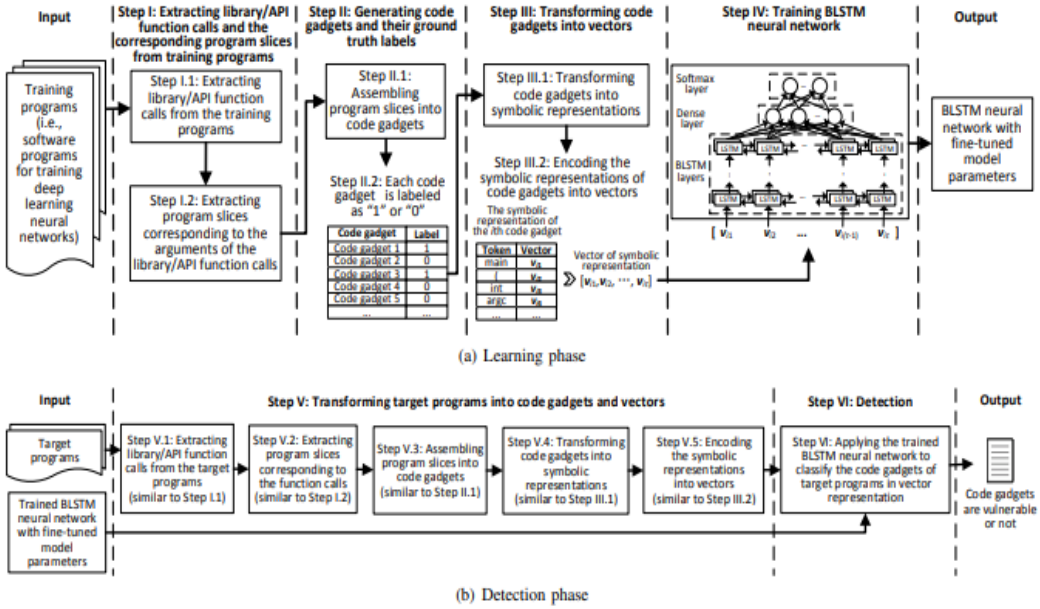


Nota: El anterior grafico permite identificar la arquitectura planteada de un modelo de inteligencia artificial convencional.

En el año 2018, una versión mejorada de esta implementación fue presentada con el nombre de VulDeePecker, la cual profundizó en el uso del Deep Learning para reducir la cantidad de trabajo manual en la detección de las vulnerabilidades, y a su vez, buscando reducir el número de casos detectados como falsos positivos. Para cumplir con este propósito,

se añadieron redes neuronales que permitieron mejorar el análisis de código al añadir algunas capas al proceso de aprendizaje y detección de vulnerabilidades.

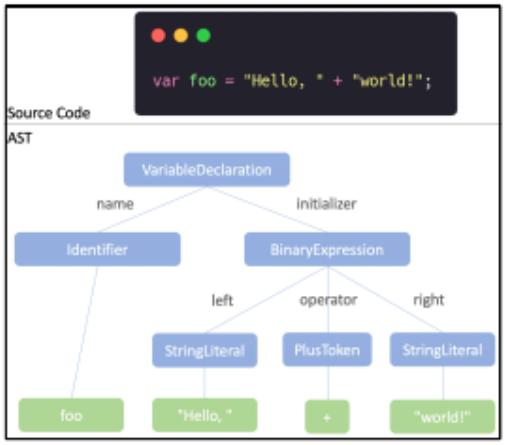
Ilustración 2: Modelo inteligencia artificial VulDeePecker (Zhen et al., 2018)



Nota: El anterior grafico permite identificar los datos de entra y salida del modelo de inteligencia artificial convencional.

En el año 2021, se publicó un artículo que explica la implementación de un prototipo para la detección de vulnerabilidades dentro del código de una aplicación desarrollada con TypeScript por medio del análisis estático de la estructura del código. Esta implementación, a diferencia de las anteriormente ilustradas, puede ser utilizada durante la etapa de desarrollo, lo que evita problemas de seguridad al desplegar la aplicación analizada. Además de esto, también se priorizan vulnerabilidades que puedan amenazar la información almacenada en las bases de datos por medio de inyección SQL.

Ilustración 3: Ejemplo de uso del prototipo para detección de vulnerabilidades usando análisis estático (Moon & Kim, 2021)



Nota: El anterior diagrama permite identificar el uso de análisis estático de código descrito en variable y valor de la variable.

1.7 Organización del trabajo de grado

En la parte 1 de este documento que contiene la contextualización de la investigación, compuesta por la definición del problema que el trabajo de grado busca resolver, así como los objetivos a cumplir relacionados con ese problema. Además, se expone la justificación de la viabilidad del proyecto en el enfoque actual, se define la hipótesis con sus variables y se explica las referencias teóricas consultadas. Finalmente, se ilustra la metodología que se utilizará durante el desarrollo del proyecto y el paso a paso explicado acá.

En la parte 2 se presenta el proceso realizado para llevar a cabo la investigación, dividiéndola por medio de capítulos que recopilan un conjunto de temáticas y eventos similares, comenzando con las fases de recolección y análisis de la información. Después se continúa con el bosquejo inicial del prototipo y su posterior implementación por medio de código utilizando como lenguaje base TypeScript. Se específica a continuación el desarrollo de la herramienta de inteligencia artificial utilizada para detectar los patrones más comunes que surgen a partir de una vulnerabilidad usando el lenguaje Python, y su acoplamiento con la

extensión desarrollada previamente. Finalmente, se somete el desarrollo realizado a un conjunto de pruebas para determinar la viabilidad y efectividad de la extensión, y se realiza el despliegue en el marketplace de extensiones que ofrece Visual Studio Code.

Por último, en la parte 3 se dan a conocer los resultados obtenidos con la investigación, verificando si se logró cumplir con los objetivos propuestos para este proyecto, y explicando la viabilidad a futuro de este proyecto, ya sea en sí mismo o con proyectos derivados, tomando como enfoque primario las limitaciones que puede presentar la extensión por el tiempo y los avances tecnológicos que se encuentran en un punto álgido al momento de presentar este proyecto.

1.8 Estudio de sistemas previos

El siguiente apartado se comprende de una revisión de trabajos de grado relevantes para el desarrollo de la investigación los cuales fueron revisados y posteriormente analizados con el objetivo de realizar una plena identificación de estudios previos que se relacionan con las temáticas de ciberseguridad, Inteligencia artificial y desarrollo de extensiones:

1.8.1 Estudio #1

(Beltrán Ballén Andrés Fernando & Lozano Ruiz Fernando Augusto, 2022). Diseño de un aplicativo para generar documentación de código en el lenguaje de C# bajo el framework de .Net. Investigación de tipo proyectiva. La población es la empresa Hexacta, tomando como muestra algunos de sus empleados. Para obtener la información requerida utilizaron Scopus, Tree of Science, repositorios abiertos, y encuestas.

El objetivo general es diseñar un aplicativo el cual genere la documentación del código con una parametrización previa para el uso de los desarrolladores que trabajan con el lenguaje de programación de C# bajo el Framework de .NET, generada como un aplicativo web con una arquitectura de 3 capas para mantener una solución sostenible y empleando las

buenas prácticas del desarrollo. A raíz de este, Beltrán y Lozano proponen los siguientes objetivos específicos:

- Definir la parametrización pertinente que emplea el aplicativo con los posibles acrónimos o siglas que utilice una organización y teniendo cuenta los estándares del desarrollo para el correcto funcionamiento dentro de la organización.
- Diseñar el prototipo del aplicativo propuesto bajo los parámetros establecidos del lenguaje y el framework usando una arquitectura de 3 capas para la generación de la documentación.
- Validar el funcionamiento del prototipo diseñado realizando pruebas en una organización dedicada al desarrollo de software para evaluar la futura implementación del aplicativo.

Los autores evidenciaron que la funcionalidad que se implementó en el aplicativo, correspondiente a la lógica de negocio que se implemente en el código que se está desarrollando, la documentación que genera es más precisa, puesto que brinda información adicional a la interpretación generada por el engine del API de OpenAI.

El modelo presentado en esta investigación se tomó como referencia para el trabajo actual sobre el mejoramiento de las buenas prácticas de código mediante una herramienta que utiliza inteligencia artificial, siendo la mayor diferencia de ese proyecto enfocado en los comentarios y el desarrollado en este documento se enfoca en las vulnerabilidades de código.

1.8.2 Estudio #2

(Escobar Cerón & Quevedo Cortés, 2021). Modelo prototipo de inteligencia artificial basado en procesamiento de lenguaje natural y redes de neuronas artificiales para la detección de noticias falsas en español. Investigación de tipo proyectiva. La población de este proyecto son las noticias disponibles en la red en español, siendo la muestra el conjunto de noticias

usadas para el análisis del proyecto. La técnica utilizada para obtener información fue la investigación documental y la creación de un dataset para el almacenamiento de las noticias.

El objetivo general de ese proyecto es implementar un modelo prototipo de procesamiento de lenguaje natural y redes de neuronas artificiales, para detectar si una noticia en español es falsa o no. A raíz de este, Escobar y Quevedo proponen los siguientes objetivos específicos:

- Construir un conjunto de datos que contenga un listado de noticias falsas y verdaderas en español, que permita extraer las características más importantes de cada una, con el fin de usarlo como el conjunto fuente de entrenamiento, validación y finalmente de pruebas.
- Diseñar un modelo de Machine Learning haciendo uso de redes neuronales y procesamiento de lenguaje natural, que sea capaz de detectar si una noticia en español es falsa o verdadera, en una fase de entrenamiento y validación.
- Evaluar el modelo implementado a través de pruebas independientes al conjunto de datos de validación y entrenamiento, con el fin de medir el nivel de precisión del modelo.

Los autores lograron obtener una fiabilidad del 98% al analizar el conjunto de noticias por medio del uso del algoritmo clasificador Naive Bayes Multinomial y un conjunto de redes neuronales que analiza el contenido y la estructura de cada noticia para determinar si es verdadera o no.

El modelo presentado en esta investigación se tomó como referencia para el trabajo actual debido al uso que se le da a la inteligencia artificial como herramienta detectora de patrones específicos usando lenguaje natural y redes neuronales, siendo la mayor diferencia el público objetivo al que apunta este proyecto al abarcar las noticias en español, mientras

que este proyecto se enfocará en el código fuente de una aplicación desarrollada en TypeScript.

1.8.3 Estudio #3

(MORENO MARÍN JOHN EDISON, 2019). Modelo base de conocimiento para auditorías de seguridad en Servicios Web con SQL Injection. Investigación de tipo proyectiva. La población de este proyecto son las auditorías realizadas a los desarrollos, siendo la muestra una auditoría llevada a cabo por Moreno, con unos parámetros definidos. La técnica utilizada para obtener información fue la investigación documental y la búsqueda de ontologías relacionadas con este campo.

El objetivo general de ese proyecto es modelar una base de conocimiento para la auditoría de vulnerabilidad a vectores de ataque de inyección SQL en Servicios Web de una entidad, basados en tendencias, metodologías y buenas prácticas de seguridad de la información, gestionando los riesgos o incidentes de seguridad en las organizaciones. Tomando este como base, Moreno plantea los siguientes objetivos específicos:

- Establecer una metodología propia para el modelo, con metodologías definidas, métodos y buenas prácticas en materia de modelos de base de conocimiento integrados con el área de auditorías de seguridad de la información para los Servicios Web de una entidad o un prototipo organizacional.
- Analizar y desarrollar guías, técnicas y herramientas de software de Inyección SQL, sobre uno o varios servicios web, en una entidad o en un prototipo organizacional, para la generación de datos de seguridad de la información.
- Operacionalizar un software de base de conocimiento con datos resultantes de las pruebas de Inyección SQL para tener una base de reporte histórico de auditoría de

seguridad de información y así poder tener un control ante posibles ataques o riesgos en la información.

Al finalizar el proyecto, el autor llegó a la conclusión de que se logró crear un modelo de base de conocimiento implementado sobre una herramienta libre, ejecutando una auditoría en seguridad de servicios web con SQL Injection sobre un prototipo organizacional, usando el software OpenKM como herramienta que sirve como base al modelo propuesto.

Esta investigación se especializa en implementar y usar la inteligencia artificial, en especial de las redes neuronales, para conseguir un aprendizaje bastante preciso, aspecto fundamental en el desarrollo en este proyecto para detectar vulnerabilidades con un mínimo margen de error.

1.8.4 Estudio #4

(Luque Tovar & Bravo Builes, 2021). Diseño e implementación de un sistema de protección contra el ataque de inyección SQL, en un servidor vulnerable utilizando herramientas Open Access. Investigación de tipo cuantitativa, los autores plantean el desarrollo de pruebas experimentales sobre equipos seleccionados, en los cuales primero se recolectó datos y luego se realizó análisis cuantitativo propuesto por el Dr. Roberto Sampieri y colegas en estudio del 2014.

El objetivo general de ese proyecto es la implementación de un sistema de prevención contra ataques de inyección SQL en un servidor web vulnerable utilizando herramientas Open Access. Tomando este como base, Moreno plantea los siguientes objetivos específicos:

- Implementar el ataque de inyección de SQL en un servidor web vulnerable
- Detectar el ataque de inyección en un servidor web vulnerable utilizando Kali Linux.

- Implementar un firewall de aplicaciones Open Access para prevenir esta vulnerabilidad e implementar una prueba de vulnerabilidades para aplicar medidas de corrección.

Después de obtener los resultados de los WAF se tomó una muestra de 2100 ataques desde Python para cada WAF, donde se evaluaron los Tiempos de respuesta y capacidad de los firewalls identificando 11 escenarios de ataque, se obtuvo que los WAF octopus y Modsecurity detectaron 100% de las sentencias en comparación con WAF Raptor que solo detectó el 54.54%

Al final se realzó una comparación del tiempo de respuesta promedio que tarda cada WAF en detección y bloqueo de una sentencia, el resultado evidencia que el WAF Octopus es el que presenta un mayor tiempo de respuesta en bloquear la sentencia, seguido por Modsecurity el WAF que presenta un tiempo menor de respuesta es Raptor, en el eje vertical se observa el tiempo en segundos de cada firewall, sin embargo cuando se comparó tiempo de detección vs WAF utilizado se determina que Octopus WAF corresponde al de mayor respuesta.

Luego de la fase de pruebas y de analizar se concluye que es importante tener en cuenta el eslabón más débil en una aplicación o página web, en cuanto a los ataques de inyección, es no limpiar las variables de entrada cuando son ingresadas en el campo de texto por un agente externo, de esta forma se eliminaran las vulnerabilidades, dicho esto lo esencial es que los desarrolladores estén continuamente actualizados sobre los ataques a la seguridad web y debidamente capacitados de cómo desarrollar de una forma segura, es de subrayar que se debe asegurar los sitios web, lo más recomendable es el uso de los WAF acompañado de un desarrollo seguro.

1.8.5 Estudio #5

(Bedoya De la Cruz, 2023). Analizador de código como extensión de Visual Studio Code**.** Investigación proyectiva la investigación plantea la necesidad de una herramienta de análisis de código en el ámbito académico y desarrollo colaborativo el cual permita la implementación de un plugin Code Analyzer el cual se puede resumir como una extensión de Visual Studio Code, que genera métricas de seguimiento a bugs, duplicación de código, seguridad que incluye vulnerabilidades y complejidad, tanto de estructura como cognitiva, la herramienta fue desplegada en la tienda de Visual Studio Code y su infraestructura desplegada en la nube de la universidad, brindando acceso a estudiantes y docentes.

El objetivo general de la investigación es realizar mejoras sobre un plugin Code Analyzer que permitan generar un producto más versátil y robusto que soporte repositorios

Implementar un firewall de aplicaciones Open Access para prevenir esta vulnerabilidad e implementar una prueba de vulnerabilidades para aplicar medidas de corrección.

Parte II. Desarrollo de la investigación

Para la presente investigación se definió la metodología de Desarrollo Iterativo, ajustándose con un cronograma de actividades desarrollado por capítulos a continuación:

Capítulo 2. Recolección y análisis de requisitos

Se define mediante dos pasos los cuales buscan realizar preguntas entorno a el problema de investigación planteado y que permiten establecer que se busca por parte del usuario o público objetivo.

2.1 Recolección de información

Como se mencionó en el planteamiento del problema, se busca como primer paso realizar un acercamiento a la pregunta ¿Qué aspectos tienen en cuenta los ciber atacantes antes de vulnerar un sistema informático?, para lo anterior se realizó una investigación de sistemas previos la cual permitió identificar conceptos y fuentes de información para final mente definir los siguientes conceptos clave:

2.1.1 Vulnerabilidades de código

En líneas generales se describe las vulnerabilidades de software como debilidades o errores que pueden ser explotados por actores maliciosos para comprometer sistemas informáticos, robar datos o causar daños.

Dentro de las investigaciones encontradas en el repositorio de la Universidad Distrital se identifica la herramienta Kali Linux como un método individual para realizar verificación de vulnerabilidades lo que permite adentrarnos en la segunda pregunta: ¿Cuáles son las principales vulnerabilidades que se pueden encontrar en el código fuente de un sistema informático?

Para responder a esta pregunta hay que establecer bases de datos de vulnerabilidades online, recursos esenciales para la seguridad informática, ya que proporcionan información

detallada sobre las vulnerabilidades conocidas, incluyendo su descripción, impacto potencial y soluciones disponibles, es decir, organizaciones entorno a identificar y clasificar vulnerabilidades.

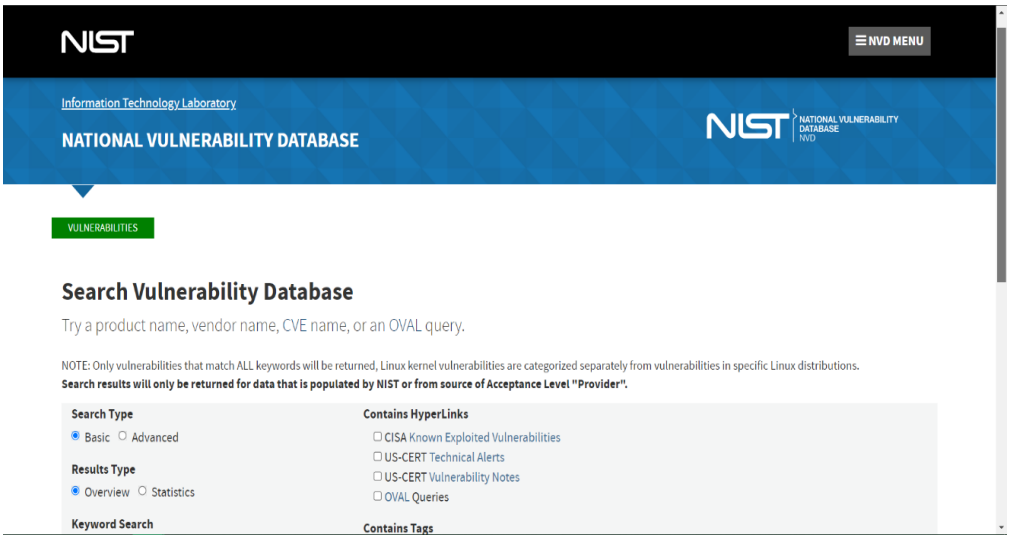
En este orden de ideas existen diferentes tipos de bases de datos de vulnerabilidades, cada una con sus propias características y objetivos:

- **Bases de datos públicas**: Estas bases de datos son de libre acceso y están disponibles para cualquier persona. Algunos ejemplos populares incluyen la National Vulnerability Database (NVD) del NIST, la Common Vulnerabilities and Exposures (CVE) List y la Vulnerabilities Exploit Index (VEX).
- **Bases de datos comerciales:** Estas bases de datos ofrecen información más completa y actualizada, así como herramientas de análisis y gestión de vulnerabilidades. Algunos ejemplos incluyen BugTraq, CoreSec y Tenable.
- **Bases de datos internas:** Algunas organizaciones mantienen sus propias bases de datos de vulnerabilidades internas, que pueden incluir información específica sobre sus propios sistemas y aplicaciones.

Luego de explorar e indagar acerca de las bases de datos de vulnerabilidades más fiables y utilizadas que estén disponibles de manera gratuita actualmente, las más destacadas son:

- **National Vulnerability Database (NVD):** La NVD es una base de datos pública del Instituto Nacional de Estándares y Tecnología (NIST) de los Estados Unidos. Contiene información sobre vulnerabilidades en software y sistemas de información. A su vez, incluye evaluaciones de severidad, descripciones técnicas, soluciones y referencias a parches, funcionando como una fuente de referencia primaria para organizaciones gubernamentales y del sector privado.

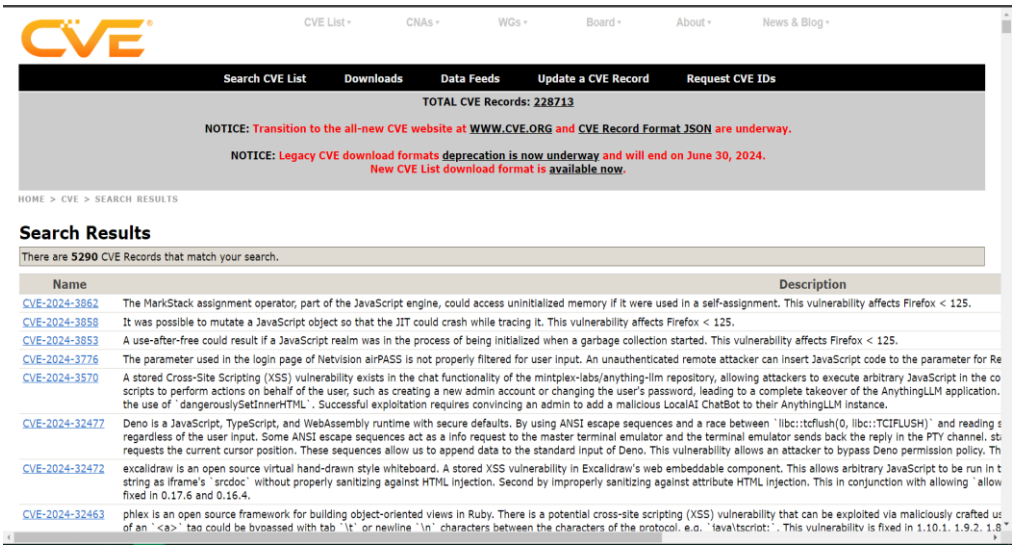
Ilustración 4: Configuración de búsqueda de vulnerabilidades en National Vulnerability Database



Nota: la anterior imagen permite una vista de la página de búsqueda de vulnerabilidades https://nvd.nist.gov/

- **Common Vulnerabilities and Exposures (CVE):** La CVE es un identificador único asignado a cada vulnerabilidad. Es un estándar ampliamente utilizado para la clasificación y comunicación de vulnerabilidades, funcionando como base para muchas otras bases de datos de vulnerabilidades.

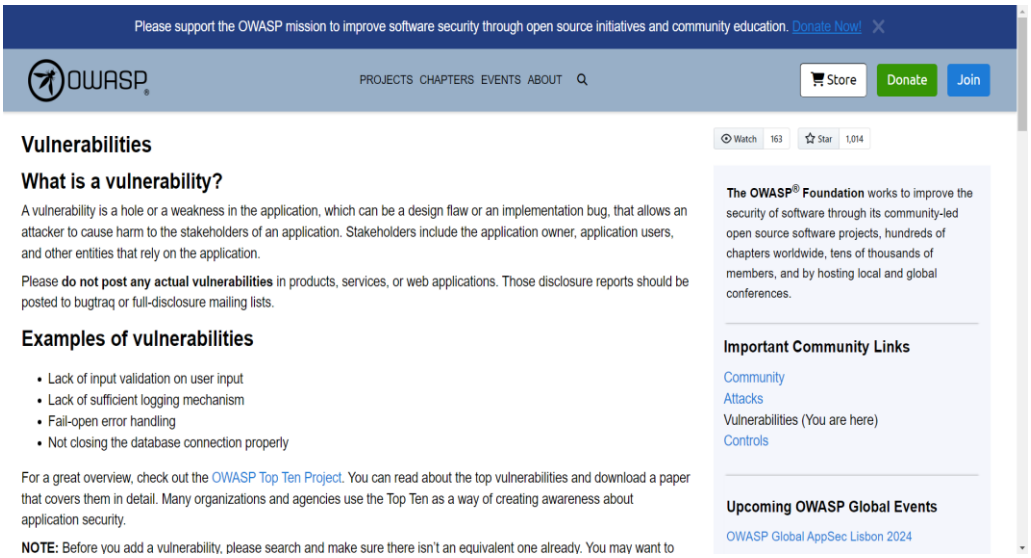
Ilustración 5: Búsqueda de vulnerabilidades en Common Vulnerabilities and Exposures



Nota: la anterior imagen permite una vista de la página de búsqueda de vulnerabilidades https://cve.mitre.org/ con su descripción

- **Open Web Application Security Project (OWASP)**: Ofrece una base de datos de vulnerabilidades web, conocida como OWASP Top 10. Esta lista se actualiza anualmente y destaca las diez vulnerabilidades web más críticas. Brinda recursos educativos, guías de seguridad y herramientas para prevenir y mitigar estas vulnerabilidades.

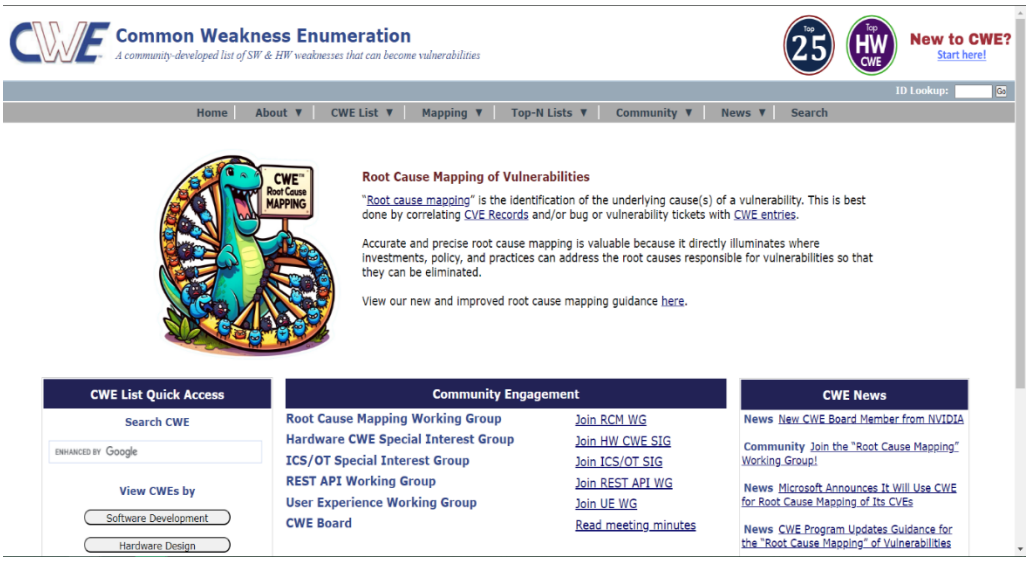
Ilustración 6: Búsqueda de vulnerabilidades Open Web Application Security Project (OWASP)



Nota: la anterior imagen permite una vista de la página de búsqueda de vulnerabilidades https://owasp.org/www-community/vulnerabilities/

- **CWE (Common Weakness Enumeration):** Página web desarrollada por MITRE Corporation. Cataloga debilidades y errores comunes en el software, incluyendo vulnerabilidades de código. Proporciona una clasificación estructurada de estas debilidades, facilitando su identificación y análisis. Es una herramienta valiosa para desarrolladores y profesionales de la seguridad que buscan mejorar la seguridad del código fuente.

Ilustración 7: Búsqueda de vulnerabilidades Common Weakness Enumeration



Nota: la anterior imagen muestra la página de búsqueda https://cwe.mitre.org/

En resumen, se puede afirmar que las bases de datos de vulnerabilidades ofrecen una serie de beneficios para la seguridad informática:

- Conciencia de las vulnerabilidades: Permiten a los administradores de sistemas y profesionales de la seguridad estar al tanto de las últimas vulnerabilidades que afectan a su software y sistemas.
- Priorización de la remediación: Ayudan a priorizar la remediación de vulnerabilidades en función de su gravedad y potencial impacto.
- Búsqueda de soluciones: Proporcionan información sobre parches, soluciones alternativas y mitigaciones disponibles para las vulnerabilidades.
- Análisis de riesgos: Facilitan el análisis de riesgos de seguridad y la toma de decisiones informadas para proteger los sistemas informáticos.

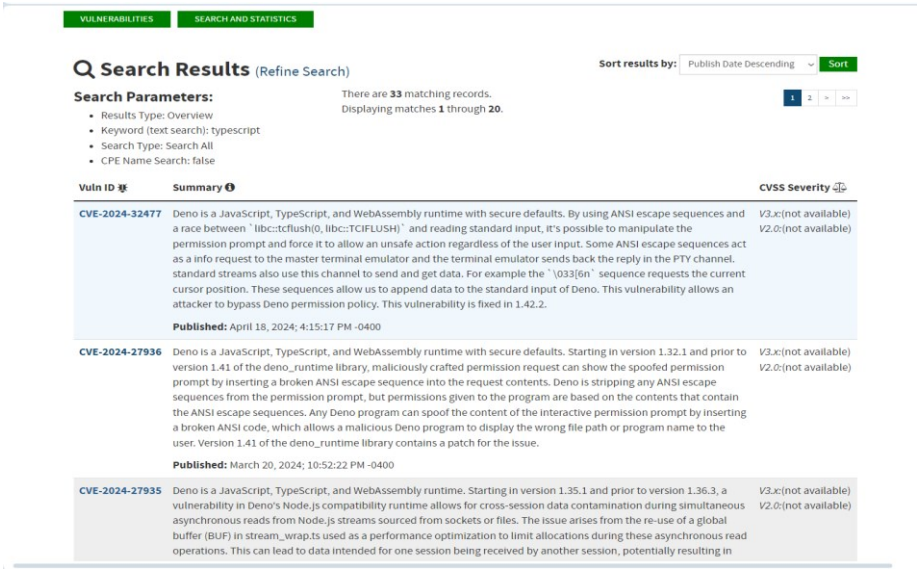
Por último, las bases de datos de vulnerabilidades se pueden utilizar de diversas maneras para mejorar la seguridad informática:

- Escaneo de vulnerabilidades: Se pueden utilizar herramientas de escaneo para identificar vulnerabilidades en los sistemas informáticos, comparando sus configuraciones con las entradas de las bases de datos de vulnerabilidades.
- Gestión de vulnerabilidades: Se pueden utilizar sistemas de gestión de vulnerabilidades (VMS) para centralizar la información sobre las vulnerabilidades, priorizar su remediación y realizar un seguimiento del progreso.
- Inteligencia de amenazas: Se pueden utilizar las bases de datos de vulnerabilidades para identificar las vulnerabilidades que están siendo explotadas activamente por los actores maliciosos.
- Investigación de seguridad: Se pueden utilizar las bases de datos de vulnerabilidades para investigar incidentes de seguridad y comprender el modus operandi de los atacantes.

2.2 Análisis de requisitos

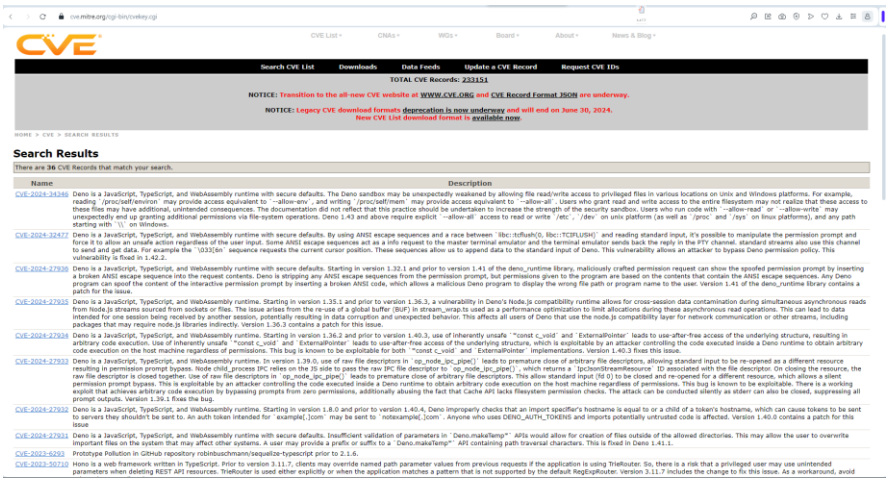
En el presente apartado se realizó una consulta a las bases de datos de vulnerabilidades previamente identificadas en el apartado de recolección de información con el objetivo de definir los requisitos o necesidades del usuario, delimitando el estudio a JavaScript y TypeScript, a continuación, el resultado de la exploración o consulta de las bases de datos.

Ilustración 8: Resultado de Búsqueda National Vulnerability DataBase NVD por palabra TypeScript.



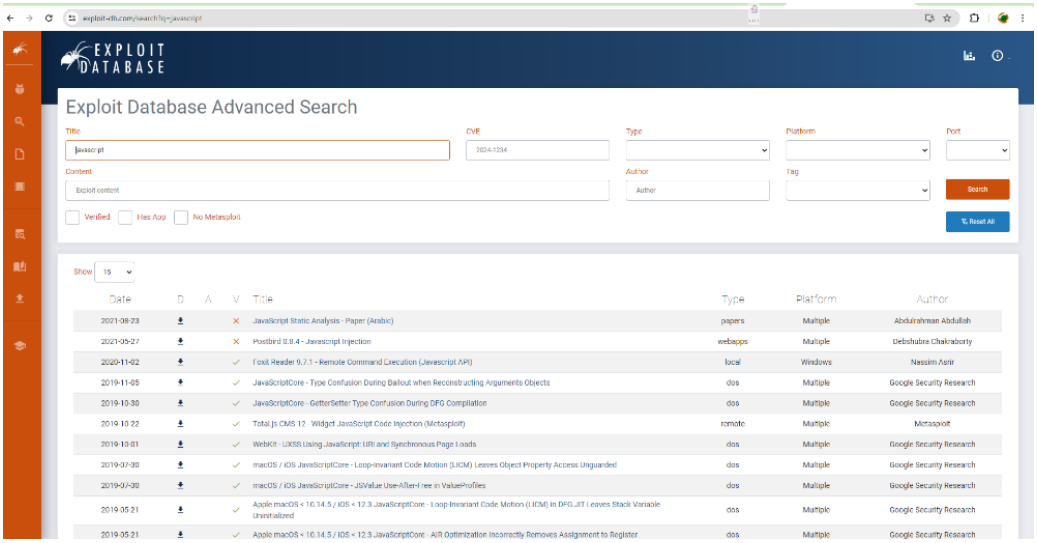
Nota: la anterior imagen muestra la vulnerabilidad con su descripción y una codificación

Ilustración 9: Resultados Common Vulnerabilities and Exposures CVE por filtrado TypeScript



Nota: la anterior imagen muestra la vulnerabilidad con su descripción y una codificación

Ilustración 10: Resultado de la Base Exploit Data Base



Nota: la anterior imagen muestra la vulnerabilidad con su descripción y una codificación junto con el autor.

Una vez consultadas las bases de datos antes mencionadas se identificaron las estructuras definidas por cada página respecto a vulnerabilidades de código TypeScript las cuales fueron analizadas, en este mismo sentido se tomó en cuenta el último reporte de OWASP Top 10 Web Application Security Risks del cual podemos destacar el siguiente listado de vulnerabilidades a estudiar:

- **SQL Injection:** inyección de SQL es un tipo de ciberataque encubierto en el cual un hacker inserta código propio en un sitio web con el fin de quebrantar las medidas de seguridad y acceder a datos protegidos. Una vez dentro, puede controlar la base de datos del sitio web y secuestrar la información de los usuarios. Le explicamos cómo funcionan los ataques de inyección de SQL, cómo combatirlos y cómo una herramienta antivirus potente lo puede proteger contra las consecuencias (Ivan Belcic, 2020).

- **Cross Site Scripting (XSS):** Los ataques de scripting entre sitios (a veces denominados XSS) consisten en inyectar códigos maliciosos en sitios web que suelen ser

fiables. Un ataque de scripting entre sitios se produce cuando los ciberdelincuentes inyectan scripts maliciosos en el contenido del sitio web atacado, que luego se agrega al contenido dinámico que se envía al navegador de la víctima. El navegador de la víctima desconoce que los scripts maliciosos no son fiables y, por lo tanto, los ejecuta (Kaspersky Lab, 2023).

- **Cross Site Request Forgery:** La falsificación de solicitudes entre sitios (CSRF) es un ataque que obliga a un usuario final a ejecutar acciones no deseadas en una aplicación web en la que está actualmente autenticado. Con un poco de ayuda de ingeniería social (como enviar un enlace por correo electrónico o chat), un atacante puede engañar a los usuarios de una aplicación web para que ejecuten acciones de su elección. (OWASP, 2024):
- **Deserialization of untrusted data:** Ocurre cuando admite la lectura de archivos u otros elementos ajenos al código sin verificar si la información de los archivos es la solicitada. Si no lo es, puede haber dejado un hueco por donde el atacante puede alterar la estructura del código con la información de un archivo falso, de manera similar a la inyección SQL.

Según estas definiciones, se puede considerar que las bases de datos analizadas no solo permiten la documentación de entidades públicas o privadas, sino que pueden ser una fuente importante en las estrategias de disminución de vulnerabilidades en el contexto del desarrollo de aplicaciones web.

Teniendo en cuenta lo anterior es pertinente definir que es TypeScript antes de continuar con la investigación de vulnerabilidades.

2.3 TypeScript

TypeScript es un lenguaje desarrollado como un superconjunto de JavaScript que añade características de tipado estático y otras mejoras al lenguaje. Microsoft lo desarrolló y lo lanzó por primera vez en 2012. Desde entonces, ha ganado una gran popularidad entre los

desarrolladores de JavaScript, especialmente en proyectos de gran escala y aplicaciones empresariales.

Las características principales de TypeScript son:

- **Tipado estático:** TypeScript permite a los desarrolladores anotar sus variables, funciones y parámetros con tipos específicos, como number, string o boolean. Esto ayuda a detectar errores de tipo en tiempo de compilación, antes de que el código se ejecute en el navegador o en un entorno Node.js.
- **Interfaces:** TypeScript proporciona soporte para interfaces, que definen la estructura de objetos y sus métodos. Las interfaces permiten a los desarrolladores crear contratos entre diferentes partes del código, mejorando la legibilidad y la mantenibilidad de este.
- **Clases:** TypeScript ofrece soporte para clases, que permiten crear objetos orientados a objetos con propiedades, métodos y herencia. Las clases facilitan la organización del código y la reutilización de componentes.
- **Módulos:** TypeScript utiliza módulos para organizar el código en unidades independientes y reutilizables. Los módulos permiten importar y exportar código entre diferentes archivos, mejorando el uso de los módulos y la organización de proyectos grandes.

Por esto, TypeScript se ha convertido en un lenguaje popular para el desarrollo web moderno, especialmente en proyectos de gran escala y aplicaciones empresariales. Es utilizado por empresas como Google, Microsoft, Amazon, Facebook y Twitter. Algunos ejemplos de frameworks populares que utilizan TypeScript son Angular, React y Vue.js. Si bien JavaScript sigue siendo el lenguaje principal para el desarrollo web front-end, TypeScript se ha convertido en una opción preferida para proyectos de gran escala y

aplicaciones empresariales que buscan una mayor calidad de código, mantenibilidad y escalabilidad.

TypeScript ofrece características que pueden ayudar a prevenir este tipo de vulnerabilidades, sin embargo, es crucial tomar medidas adicionales para garantizar la seguridad de las aplicaciones desarrolladas con este lenguaje para proteger datos confidenciales, mantener la integridad del sistema, reducir costos, mejorar la imagen de la marca y cumplir con las regulaciones. Al aprovechar las características de TypeScript y seguir las prácticas recomendadas, los desarrolladores pueden crear aplicaciones más seguras y confiables.

Por último, una vez identificadas y clasificadas las vulnerabilidades, se procederá con el diseño del modelo que guiará el desarrollo, utilizando el enfoque de prototipado evolutivo. Este modelo establecerá la base para la primera iteración del prototipo inicial, que será ajustado y mejorado en cada ciclo.

Capítulo 3. Diseño rápido del prototipo

El desarrollo de una extensión que permita identificar vulnerabilidades de código TypeScript supone poner en marcha los conocimientos adquiridos a lo largo de los cursos académicos, es así como a continuación se realizara una descripción de las fases de diseño y cómo estás se relacionan con la metodología de desarrollo de proyectos implementada a lo largo del proyecto.

Inicialmente se identificó como metodología del proyecto a Scrum, esta estructura de organización permite a partir de sprint y back log identificar los requerimientos del cliente, para el caso, se realizó un total de 3 sprint los cuales permitieron desarrollar el aparte de recolección y análisis de información, no obstante se logró identificar que el tamaño del equipo supone un reto al momento de su implementación y que la cantidad de encuentros

puede llegar a resultar poco beneficiosa debido a que los requerimientos y requisitos cambian, por lo que se identificó la necesidad de cambiar a la metodología a el método de prototipado teniendo en cuanta como antecedente(Quijano Gonzalez & Rodriguez Espinosa, 2016), en un contexto diferente al de ciberseguridad, no obstante se puede afirmar que un plugin y una extensión son soluciones comparables en un entorno de software.

Ahora bien, resultado del análisis de antecedentes, sistemas previos y demás información relevante permite identificar a Visual Studio Code como uno de los editores de código preferidos por los desarrolladores block (J. Pomeyrol, 2019), teniendo en cuenta este dato se selecciona a Visual Studio Code como Editor objeto del presente estudio y se decide a explorar su entorno de desarrollo.

3.1 Especificación de requisitos

En este apartado de diseño de la Extensión (Prototipo rápido) se identificaron los requisitos del producto a desarrollar definiéndose principalmente dos tipos de requisitos enunciados a continuación:

3.1.1 Requisitos funcionales

- La extensión debe poder detectar vulnerabilidades de código utilizando técnicas de inteligencia artificial enfocadas a la detección de patrones similares.
- La extensión debe integrarse con el entorno de desarrollo de Visual Studio Code, permitiendo su uso únicamente en los archivos con extensión .ts o .tsx, que usan TypeScript como lenguaje base.
- Las funcionalidades de la extensión deben poder ser invocadas a través de la interfaz de Visual Studio Code y/o el terminal de comandos.
- La extensión debe ser robusta y eficiente, minimizando el impacto en el rendimiento de Visual Studio Code.

- La extensión debe generar resultados precisos y confiables.
- La extensión debe tener una conexión directa con un API para la ejecución de la funcionalidad de inteligencia artificial.

3.1.2 Requisitos no funcionales

- La extensión debe ser compatible con las últimas versiones de Visual Studio Code.
- La extensión debe ser compatible con diferentes sistemas operativos (Windows, macOS, Linux).
- La extensión debe cumplir con los estándares de seguridad y privacidad de datos.
- La extensión debe estar bien documentada, incluyendo guías de usuario y referencias de la API.
- La extensión debe ser extensible, permitiendo la integración con otros plugin y herramientas.
- La extensión debe ser traducible a diferentes idiomas.
- La extensión debe ser compatible para publicarse en el Marketplace de Visual Studio Code.

En el entorno antes mencionado se identifica que la base o repositorio a crear se encuentra construido entorno a un identificador, resumen y Common Vulnerability Scoring System CVSS o versión lo que permite tener una estructura inicial acerca de cómo se documenta la vulnerabilidad relacionada a nuestro entorno objetivo TypeScript o en su defecto JavaScript.

Dentro de la etapa de identificación de requisitos se debe hacer una delimitación teórica de algunos conceptos referentes al tipo de tecnología que diferenciara de otras soluciones identificadas las definiciones se precisan a continuación:

3.2 Inteligencia artificial

La inteligencia artificial (IA) está revolucionando diversos campos, incluyendo la seguridad del software. Su capacidad para procesar grandes cantidades de datos, identificar patrones y aprender de experiencias anteriores la convierte en una herramienta poderosa para la prevención de vulnerabilidades de código. Si bien existen algunos desafíos que deben abordarse, la IA se está convirtiendo en una herramienta cada vez más importante para los desarrolladores y equipos de seguridad que buscan crear software más seguro.

Algunos de los casos en los que se ha implementado la IA para detectar vulnerabilidades de código son los siguientes:

- **Análisis de código estático**: Las herramientas de análisis de código estático tradicionales pueden identificar patrones de código potencialmente vulnerables, pero a menudo generan muchos falsos positivos y requieren revisión manual. La IA puede mejorar la precisión de estas herramientas al identificar patrones más sutiles y complejos, y al reducir la cantidad de falsos positivos.
- **Detección de anomalías:** La IA puede analizar el comportamiento del código para detectar anomalías que podrían indicar la presencia de vulnerabilidades. Por ejemplo, puede identificar patrones de acceso a memoria inusuales o llamadas a funciones inesperadas.
- **Predicción de vulnerabilidades:** Los modelos de aprendizaje automático pueden entrenarse con conjuntos de datos de vulnerabilidades conocidas para aprender a identificar código que es probable que contenga vulnerabilidades. Esto puede ayudar a los desarrolladores a enfocar sus esfuerzos de prueba y revisión en las áreas de mayor riesgo.

- **Reparación automática de vulnerabilidades:** La IA puede utilizarse para generar automáticamente parches para vulnerabilidades conocidas. Si bien esta tecnología aún se encuentra en sus primeras etapas de desarrollo, tiene el potencial de revolucionar la forma en que se corrigen las vulnerabilidades.

A pesar de esto, hay algunos obstáculos que la IA debe afrontar para ser eficaz en la detección de vulnerabilidades de código, que incluyen:

- **Disponibilidad de datos:** Los modelos de IA necesitan grandes cantidades de datos de entrenamiento para ser efectivos. Es importante contar con conjuntos de datos de alta calidad que incluyan ejemplos de vulnerabilidades conocidas y código seguro.
- **Facilidad de explicación:** Los modelos de IA a menudo son "cajas negras", lo que dificulta comprender cómo llegaron a una determinada decisión. Esto puede dificultar la confianza en los resultados de estos modelos.
- **Sesgo:** Los modelos de IA pueden estar sesgados si se entrenan con datos sesgados. Es importante tener en cuenta este riesgo y tomar medidas para mitigarlo.

En el panorama actual de la seguridad informática, la detección de vulnerabilidades de código se ha convertido en una tarea crucial para proteger sistemas, datos y la integridad de las aplicaciones. Si bien los métodos tradicionales como las revisiones manuales y las pruebas de penetración han sido útiles, la complejidad y el volumen cada vez mayores del software hacen que estos enfoques sean cada vez más desafiantes e ineficientes. Es aquí donde la inteligencia artificial (IA) emerge como una herramienta poderosa para superar estas limitaciones y revolucionar la detección de vulnerabilidades de código.

Es importante destacar que la IA no reemplaza por completo a los métodos tradicionales de detección de vulnerabilidades, sino que los complementa y potencia. La

combinación de la experiencia humana con las capacidades analíticas de la IA permite obtener mejores resultados y crear un enfoque más integral para la seguridad del software.

3.3 Extensiones en Visual Studio Code

Visual Studio Code (VSCode) se ha convertido en uno de los editores de código más populares entre los desarrolladores de software gracias a su extensibilidad, personalización y amplio conjunto de funcionalidades. Las extensiones de VSCode juegan un papel fundamental en su éxito, ya que permiten ampliar las capacidades del editor y adaptarlo a las necesidades específicas de cada desarrollador o proyecto.

Las extensiones de VSCode son pequeños programas que añaden nuevas funcionalidades al editor. Estas extensiones pueden abarcar una amplia gama de tareas, desde la sintaxis y el resaltado de código hasta la integración con herramientas de control de versiones, depuración y publicación de código.

VSCode cuenta con un Marketplace oficial donde se pueden encontrar miles de extensiones gratuitas y de pago. Es importante buscar extensiones que provengan de desarrolladores confiables y que tengan una buena reputación en la comunidad. Leer las reseñas y valoraciones de otros usuarios también puede ser útil para tomar una decisión informada. El ecosistema de extensiones de Visual Studio Code se encuentra en constante evolución, con nuevas extensiones que se publican regularmente y actualizaciones que mejoran las funcionalidades existentes. Es recomendable mantenerse actualizado sobre las últimas extensiones disponibles para aprovechar al máximo las posibilidades que ofrece VSCode.

La creación de una extensión para Visual Studio Code que detecte vulnerabilidades de código en archivos TypeScript es una iniciativa viable con un gran potencial para impactar positivamente en la seguridad del desarrollo web y aplicaciones móviles. Los argumentos a

favor superan los desafíos, y la existencia de recursos y herramientas disponibles facilita su realización. Sin embargo, es importante considerar la complejidad del análisis de código, la necesidad de minimizar falsos positivos y el compromiso a largo plazo con el mantenimiento y las actualizaciones. Con un enfoque adecuado y una ejecución bien planificada, esta extensión podría convertirse en una herramienta invaluable para la comunidad de desarrolladores de TypeScript.

Capítulo 4. Desarrollo del prototipo de extensión

4.1 Definición de la estructura básica de la extensión

La estructura de una extensión en Visual Studio Code sigue un patrón general, aunque puede variar ligeramente según las necesidades específicas de la extensión. Los componentes que se han definido para integrar la estructura de la extensión son los siguientes:

- **Archivo package.json:** Este es el archivo de manifiesto principal de la extensión que contiene metadatos importantes como el nombre de la extensión, la versión, la descripción, las dependencias, las activaciones y otros detalles más.
- **Carpeta src (código fuente):** Este directorio contendrá el código fuente de tu extensión, organizado en subdirectorios según la funcionalidad. Por ejemplo, puedes tener subdirectorios para los comandos, los proveedores de lenguaje, las vistas, etc.
- **Iconos e imágenes:** Este directorio puede contener los iconos e imágenes utilizados por la extensión, como los iconos de la barra lateral, las imágenes de vista previa, etc.
- **Archivos de configuración:** Estos archivos pueden incluir configuraciones específicas de la extensión, como archivos de configuración JSON para definir opciones personalizadas, archivos de configuración de tareas para definir tareas personalizadas relacionadas con la extensión, etc.

- **Carpeta test (pruebas):** Aquí es donde puedes colocar pruebas unitarias y de integración para tu extensión. Es importante incluir pruebas para garantizar que la extensión funcione como se espera y para evitar problemas de regresión.
- **Carpeta out (compilación):** Este directorio suele contener los archivos compilados de tu extensión. Pueden ser archivos JavaScript o archivos empaquetados en formato .vsix listos para ser distribuidos. Otros archivos y directorios: Dependiendo de las necesidades de tu extensión, es posible que tengas otros archivos y directorios, como archivos de configuración específicos para diferentes entornos, archivos de datos de ejemplo, scripts de automatización, etc.

Una vez definido la estructura del proyecto se hace necesario delimitar de forma técnica la estructura de la extensión mediante el uso de un diagrama de clases el cual permite mostrar al usuario el funcionamiento interno futuro de la extensión, así las cosas, se definen las siguientes clases:

4.1.1 Extensión Manager

Corresponde al conjunto de herramientas que permite instalar, gestionar y actualizar extensiones en el editor de código, para la herramienta en desarrollo se plantea la implementación de las funciones:

- **HoverProviders:** Visual Studio Code permiten a las extensiones mostrar información contextual cuando el usuario pasa el cursor sobre un símbolo en el código. Proporcionan detalles como documentación o tipos de datos, mejorando la experiencia de desarrollo. Son útiles para mostrar firmas de funciones o explicaciones rápidas.
- **Active:** En Visual Studio Code inicia las funcionalidades de una extensión cuando se abre el editor o se activa un evento específico. Es el punto de entrada para registrar

comandos, proveedores y otros servicios. Todo el código de la extensión se ejecuta después de la activación.

- **OnDidChangeActiveTextEditor:** En Visual Studio Code se activa cuando el usuario cambia de un editor activo a otro. Permite a las extensiones reaccionar ante este cambio, como actualizar vistas, realizar análisis de código o modificar comportamientos según el nuevo archivo que se haya enfocado en el editor.

4.1.2 Analyzer

Es una herramienta o extensión que inspecciona el código en tiempo real, detectando errores, advertencias, y sugerencias de mejora. Proporciona análisis estático para asegurar la calidad del código, ofreciendo recomendaciones de optimización, detección de bugs y mejores prácticas para diversos lenguajes.

- **DataList:** Se refiere a una estructura que muestra elementos o sugerencias, como en el autocompletado. Proporciona opciones basadas en contexto mientras el usuario escribe.
- **PerformAnalysis:** El método se utiliza en extensiones para analizar el código fuente, identificando errores, advertencias o sugerencias de mejora, ayudando a optimizar la calidad del código.

Este componente se divide a su vez en otros dos componentes que permiten el manejo de la información identificada por parte de la herramienta respecto a las vulnerabilidades detectadas.

4.1.3 Log

Son registros de actividades o eventos generados por el editor o extensiones. Facilitan la depuración, rastreo de errores y el monitoreo del comportamiento del sistema.

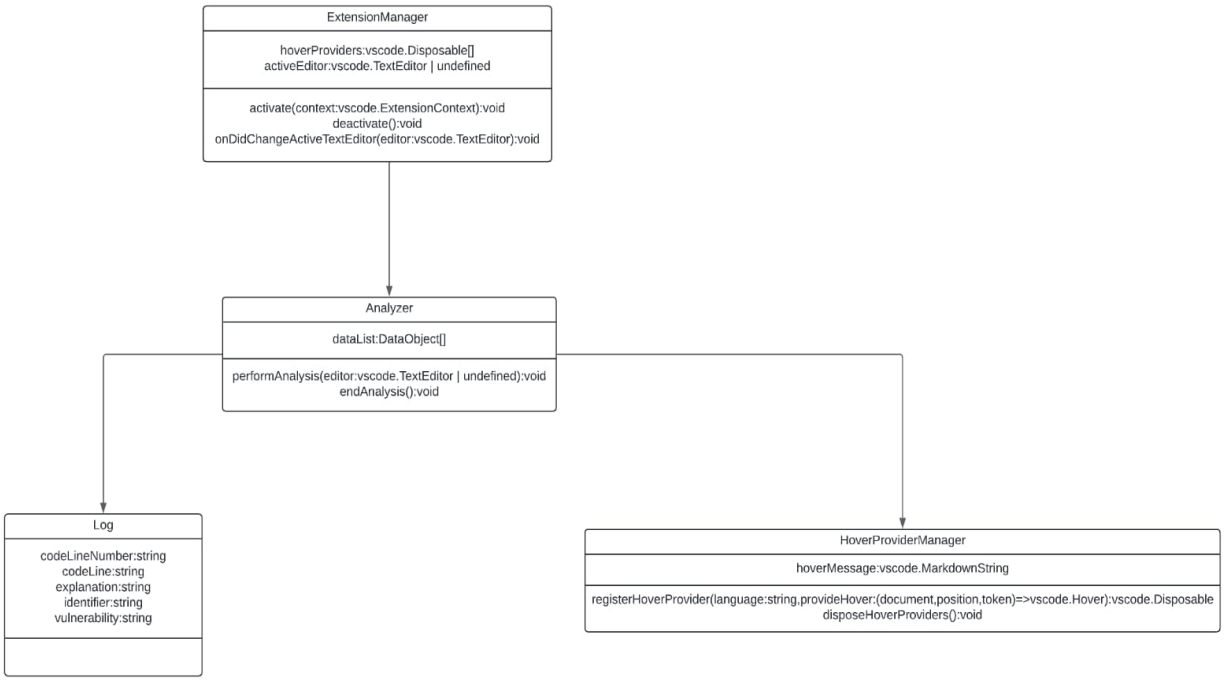
- **CodeLine:** Se refiere a una única línea dentro de un archivo de código fuente el editor muestra y numera las líneas, facilitando la navegación, edición y depuración del código.
- **Explantation:** Se refiere a una breve descripción del código detecta
- **Identifier:** Es un nombre que se usa para referirse a variables, funciones, clases, o cualquier otro elemento en el código fuente. Los identificadores ayudan a distinguir y acceder a diferentes partes del código.
- **Vulnerability:** Es un nombre que se usa para referirse a variables, funciones, clases, o cualquier otro elemento en el código fuente detectado como una vulnerabilidad determinada basado en las bases de datos de Exploit.

4.1.4 HoverProviderManager

Se encarga de gestionar los **HoverProviders**, que son responsables de proporcionar información contextual cuando el usuario pasa el cursor sobre un símbolo en el código. Esta gestión incluye registrar, administrar y manejar los HoverProviders en el editor.

- **HoverProviderManager:** se refiere al contenido o información que se muestra en una ventana emergente (hover) cuando el usuario pasa el cursor sobre un símbolo o elemento del código.
- **RegisterHoverProvider:** se utiliza para registrar un **HoverProvider** que proporciona información adicional cuando el usuario pasa el cursor sobre un símbolo en el código.

Ilustración 11: Diagrama de clases de la extensión de detección de vulnerabilidades



Nota: La anterior imagen corresponde a un diagrama de clases que permite abstraer las clases pertenecientes al prototipo.

4.2 Creación del proyecto

Para el desarrollo de la extensión, se tenía pensado en un principio que esta, al igual que el modelo de inteligencia artificial que se tiene propuesto utilizara el lenguaje de programación Python como su base. Sin embargo, después de investigar acerca del proceso de desarrollo de extensiones en Visual Studio Code a profundidad, se descubrió que debido a que el editor de código está desarrollado utilizando Electron, el cual es un framework que implementa Node.js y Chromium en una aplicación de escritorio, la implementación con

Python, a pesar de ser posible, requiere de la necesidad de convertir el código desarrollado al lenguaje de programación JavaScript o TypeScript, si se decide que los tipos van a ser explícitos. Por lo tanto, se decidió que el desarrollo de la herramienta se realizará con TypeScript.

Con esto en mente, y después de leer y analizar la información de la guía que Visual Studio Code del sitio web oficial, se comenzó con el desarrollo de la extensión.

Como requisito inicial antes de comenzar a desarrollar, se debe instalar NodeJS que es un entorno en tiempo para ejecución de código multiplataforma, usando el enlace oficial de su sitio web.

Una vez instalado, y usando el instalador de paquetes npm, se instalará a continuación Yeoman, el cual permite la instalación del generador de extensiones para Visual Studio Code, ejecutando el siguiente comando: npm install -g yo generator-code.

Después de que finalice la instalación, se ejecuta el comando yo code para comenzar con la creación de un proyecto, tras lo cual se debe agregar la información relacionada con los detalles para la creación del proyecto, como el nombre del autor, la descripción de la extensión, la ruta del punto de entrada y la compatibilidad con diferentes versiones de Visual Studio Code.

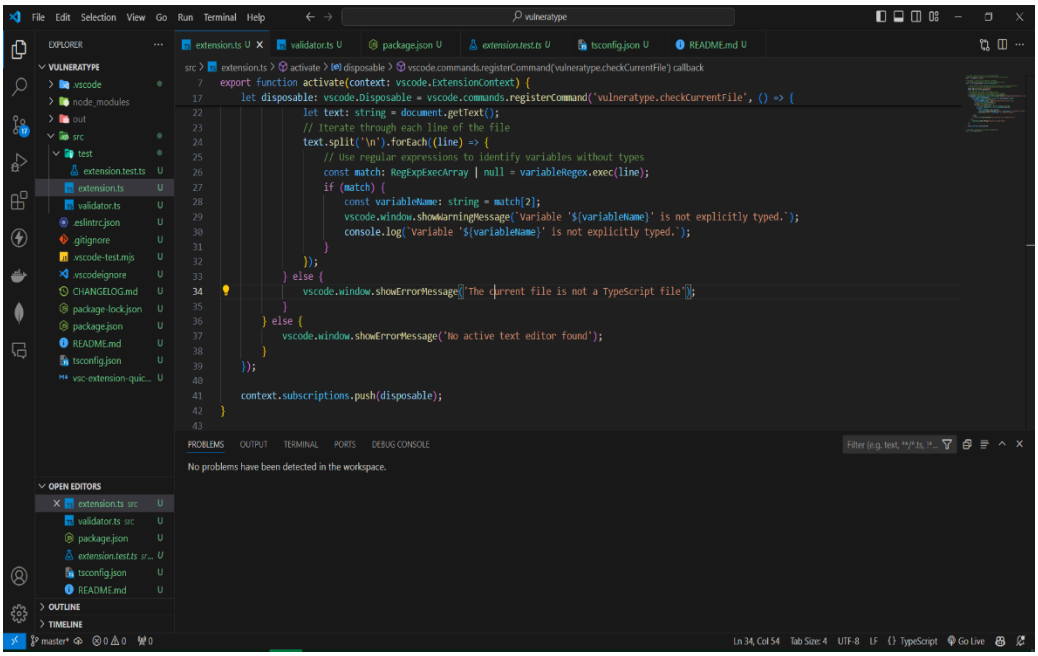
Una vez que Yeoman haya generado la estructura del proyecto, se localiza la carpeta del proyecto y se abre en Visual Studio Code. Después de que abra, se ubica el archivo package.json y se editan los detalles de la extensión que no fueron solicitados previamente, como la versión y las palabras clave.

El archivo extension.ts dentro de la carpeta src contiene la lógica principal de la extensión, incluyendo la definición de comandos, la creación de interfaces de usuario y la

interacción con el API de Visual Studio Code, así que ese es el primer archivo que se modifica para la implementación de la función base que será llamada desde la extensión.

A medida que se desarrolló la extensión, se fue probando con el comando F5 para entrar en modo debug e ir verificando si funcionaba correctamente. Al final del desarrollo base de la extensión, se obtuvo la siguiente estructura:

Ilustración 12: Código del prototipo en Visual Studio Code



Nota: la anterior imagen muestra el código generado para el prototipo en la clase extensión la cual contiene la validación del código.

4.3 Incorporación y acoplamiento del modelo de lA en la herramienta

Lo primero que se debe realizar es la definición de una serie de conceptos para establecer de forma teórica las diversas partes de la implementación planteada comenzando por las siguientes temáticas:

- OpenAI: OpenAI es una organización de investigación que se centra en el desarrollo de inteligencia artificial avanzada y ética, con el objetivo de garantizar que los beneficios de la IA sean compartidos equitativamente con la humanidad. Fundada en 2015, OpenAI se ha convertido en líder en el desarrollo de modelos de lenguaje, como GPT (Generative Pre-trained Transformer), que destacan por su capacidad para entender y generar texto en lenguaje natural. Los avances de OpenAI en el procesamiento del lenguaje han permitido mejoras significativas en áreas como la traducción automática, la asistencia virtual, y la generación de contenido, lo que ha sido crucial para la innovación en inteligencia artificial conversacional y otras aplicaciones relacionadas (Tom B. Brow et al., 2020).

Una de las principales ventajas de los modelos de OpenAI, como GPT-4, es su capacidad para generar respuestas coherentes y contextualmente relevantes a partir de grandes volúmenes de datos. Esto es posible gracias a un proceso de entrenamiento que utiliza millones de parámetros, lo que permite a los modelos aprender patrones complejos en los textos. Además, OpenAI ha implementado esfuerzos importantes para mitigar riesgos asociados con el uso indebido de la IA, adoptando una postura proactiva en torno a la ética y la seguridad de la inteligencia artificial, con especial énfasis en la gobernanza de la IA avanzada para minimizar los posibles impactos negativos sobre la sociedad (OpenAI, 2024).

- Groq: Groq es un lenguaje de consulta específico diseñado para bases de datos orientadas a documentos como Sanity. A diferencia de otros lenguajes de consulta tradicionales, Groq se destaca por su expresividad y flexibilidad, lo que permite a los desarrolladores extraer datos de manera eficiente y precisa. Este lenguaje facilita la creación de consultas más dinámicas y específicas sin necesidad de estructuras complejas como SQL. Además, es especialmente útil para trabajar con datos en formato JSON, permitiendo operaciones avanzadas como filtrado, proyección y desestructuración de datos. Entre sus

principales ventajas, destaca la simplicidad de su sintaxis, lo que reduce la curva de aprendizaje para los desarrolladores, y su capacidad para manejar datos no estructurados o semiestructurados de forma fluida (Sanity, 2023).

El uso de Groq también permite un alto grado de flexibilidad, ya que las consultas se adaptan al formato del documento sin requerir modificaciones en el esquema de la base de datos. Esto es particularmente ventajoso en sistemas donde los datos cambian frecuentemente o no siguen un esquema rígido, brindando así una solución ágil y escalable para la gestión de datos en proyectos con grandes volúmenes de información (Jung A., 2022). En resumen, Groq combina simplicidad, flexibilidad y optimización para manejar datos estructurados en formato JSON, lo que lo convierte en una opción poderosa para desarrolladores que requieren precisión y rendimiento en sus consultas. Además, su principal ventaja con respecto al producto desarrollado por OpenAI es el hecho de no tener costo alguno, mientras proporciona resultados con una confiabilidad alta.

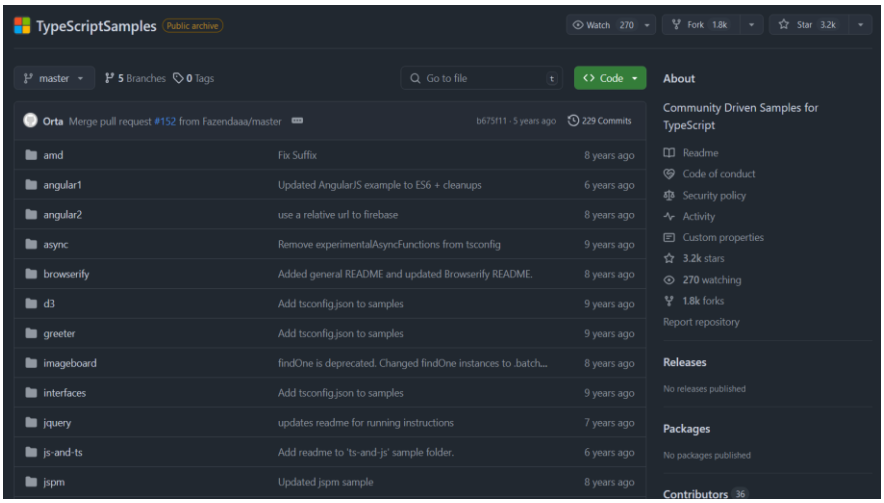
Con las definiciones establecidas, se decidió optar por la implementación del modelo de Groq para generar un prompt que permita analizar el código extraído del editor del usuario tomando en cuenta las vulnerabilidades encontradas en el top 25 generado por la OWASP. Para este fin, se estableció un método encargado de recibir el prompt y ordenar las vulnerabilidades encontradas para que sean más fáciles de ubicar y resaltar en el editor de Visual Studio Code.

Capítulo 5. Pruebas de la extensión

En esta fase del proceso de desarrollo, se evalúa el comportamiento de la extensión en diversos escenarios para verificar que cumple con los requisitos especificados. El objetivo de las pruebas es identificar y corregir posibles errores, además de optimizar su rendimiento. Para este fin, se optó por usar la extensión en proyectos de código abierto disponibles en

línea, por lo que después de realizar la búsqueda del proyecto que cumpla dichas condiciones, se encontró un repositorio público en GitHub creado por Microsoft con ejemplos de proyectos de TypeScript, el cual se encuentra disponible en el siguiente enlace: https://github.com/microsoft/TypeScriptSamples

Ilustración 13: Repositorio de código de TypeScript de Microsoft



Nota: la anterior imagen muestra el conjunto de código disponible en un repositorio abierto de Microsoft.

Con la realización de estas pruebas se encontraron algunos detalles dentro del código que debían ser corregidos, los cuales son:

- Cuando el servidor de Groq no retornaba una respuesta, un error se mostraba al usuario que finaliza la extensión de manera precipitada, por lo que se agregó un ciclo para repetir la petición un número limitado de veces, y si no hay respuesta, devolver un mensaje de error al usuario sin afectar el estado de la extensión.
- En algunos casos, cuando la coincidencia del atributo codeLine abarcaba dos o más líneas de código, la funcionalidad de resaltado y el mensaje de hover no se activaban correctamente. Para esto, se ajustó la lógica de la extensión para manejar correctamente

coincidencias de varias líneas, asegurándose de que el resaltado y el mensaje de hover se activaran adecuadamente incluso en bloques de código más extensos.

- En ocasiones, aunque el resaltado y la información de hover funcionaban correctamente, el enlace "Ver más" que debía abrir un popup con información adicional no permitía clic de parte del usuario. Después de analizar el código, se identificó que el problema estaba relacionado con la falta de registro adecuado del comando en el archivo package.json. Se modificó el código para registrar dinámicamente los comandos asociados a cada elemento de la lista, permitiendo que el botón pudiera ser seleccionado sin importar la cantidad de elementos presentes.
- La función endAnalysis eliminaba el resaltado del código, pero no eliminaba correctamente el elemento visual que aparecía al hacer hover. Debido a esto, se revisó la implementación de la función para asegurarse de que no solo se eliminara el resaltado, sino también cualquier tooltip o información flotante que pudiera quedar activa tras el análisis. Esto mejoró la limpieza de la interfaz.
- Cuando se cambiaba a otro archivo dentro del editor, el análisis anterior no se reiniciaba adecuadamente, lo que podía llevar a errores en el comportamiento de la extensión. Para corregir esto, se implementó un mecanismo para realizar análisis independientes en cada archivo al detectar un cambio, asegurando que los resaltados y mensajes de hover estuvieran sincronizados correctamente con el archivo activo.
- Una vez fueron realizados los cambios y se continuó con el desarrollo de las pruebas, se encontró que la fiabilidad de la herramienta era superior al 90%, solo teniendo algunas dificultades cuando el código era demasiado extenso o la respuesta del prompt no era lo suficientemente precisa para detallar la vulnerabilidad.

Capítulo 6. Refinación del prototipo bajo Grog

Después de finalizar las pruebas, se decidió optimizar el código para que el porcentaje aumente a un mínimo del 95%. Para esto, se realizan iteraciones sobre el prototipo para resolver problemas identificados, optimizar su funcionamiento y garantizar que cumpla con los objetivos del proyecto, y a su vez, se realizaron diversos cambios al código para que el prompt generado sea mucho más preciso.

Lo primero que se realizó fue probar los distintos modelos disponibles en Groq, para determinar cuál proporcionaba las respuestas más precisas. Otro cambio realizado fue la reducción del valor de temperatura configurado para la generación del prompt, lo cual reduce el nivel de aleatoriedad que utiliza Groq para sus respuestas, evitando de esta manera que se reportaran falsas alarmas de vulnerabilidades.

Además de ello, se implementó un ciclo para filtrar los resultados en base a un número determinado de prompts realizados con el fin de contrastar las respuestas y determinar cuáles eran las vulnerabilidades detectadas. Sin embargo, después de realizar las pruebas correspondientes, se determinó que aumentaba en gran medida el tiempo de procesamiento de la extensión sin mostrar un cambio significativo en las vulnerabilidades encontradas, por lo que al final esta implementación fue descartada y solo se optimizó la temperatura del prompt.

Ilustración 14: Configuración de parámetros al modelo Groq con llama 3.1



Nota: la anterior imagen muestra los parámetros definidos respecto al modelo de groq y su implementación en TypeScript

Parte III. Cierre de la investigación

Capítulo 7. Resultados y discusión

7.1 Conclusiones

Las conclusiones de este proyecto de grado destacan la eficacia del prototipo desarrollado para la detección de vulnerabilidades en código TypeScript mediante inteligencia artificial. Se observó que el prototipo logró identificar una amplia gama de vulnerabilidades con precisión, superando en muchos casos a los enfoques tradicionales de análisis estático. Sin embargo, se identificaron desafíos en la disponibilidad de datos de entrenamiento y la interpretación de los resultados, lo que sugiere la necesidad de investigaciones adicionales para abordar estas limitaciones. A pesar de esto, se subraya la relevancia de la integración de inteligencia artificial en la detección de vulnerabilidades y se ofrecen recomendaciones para futuras mejoras, lo que podría tener un impacto significativo en la seguridad informática y la ingeniería de software.

El impacto de este proyecto en trabajos futuros es considerable. La integración exitosa de técnicas de inteligencia artificial en la detección de vulnerabilidades en código TypeScript sienta las bases para investigaciones adicionales en el campo de la seguridad informática. Los resultados y las lecciones aprendidas de este proyecto pueden inspirar y guiar a futuros investigadores y profesionales en la mejora de herramientas de detección de vulnerabilidades, la expansión de la gama de vulnerabilidades detectadas y la optimización del rendimiento de dichas herramientas. Además, este trabajo podría motivar a la industria a adoptar enfoques innovadores basados en inteligencia artificial para fortalecer la seguridad de sus sistemas y aplicaciones, lo que tendría un impacto positivo en la protección de datos y la confianza del usuario.

Bibliografía

Ana Ayerbe. (2020). *La ciberseguridad y su relación con la inteligencia artificial*. Real Instituto Elcano. https://www.realinstitutoelcano.org/analisis/la-ciberseguridad-y-su-relacion-con-la-inteligencia-artificial/

Bedoya De la Cruz, F. (2023). *Analizador de código como extensión de Visual Studio Code* [UNIVERSIDADDELOSANDES]. https://repositorio.uniandes.edu.co/entities/publication/a3acba1c-3bc3-4d4e-96cb-0f0c886e7384

Beltrán Ballén Andrés Fernando, & Lozano Ruiz Fernando Augusto. (2022). *DISEÑO DE UN APLICATIVO PARA GENERAR DOCUMENTACIÓN DE CÓDIGO EN EL LENGUAJE DE C# BAJO EL FRAMEWORK DE .NET* [UDFJC]. https://repository.udistrital.edu.co/server/api/core/bitstreams/495e3caa-2234-4edf-8fb1-fc311db97767/content

Elias Carabaguiaz Gonzalez. (2017). Importancia del Aprendizaje de Ciberseguridad ante los Riesgos de las Tecnologías de Información. *Revista de la Facultad de Tecnología de Información y Comunicación.*, *1*, 3–13.

Escobar Cerón, D. F., & Quevedo Cortés, A. D. (2021). *Modelo prototipo de inteligencia artificial basado en procesamiento de lenguaje natural y redes de neuronas artificiales para la detección de noticias falsas en español*. UDFJC.

González, E. (2023, February 24). *El 84% de las bases de código tiene al menos una vulnerabilidad de código abierto*. Bitlifemedia. https://bitlifemedia.com/2023/02/bases-de-codigo-vulnerabilidad/

Grupo ACMS Consultores. (n.d.). *ISO 27032 Gestión de la CiberSeguridad*. 2023. Retrieved September 23, 2024, from https://www.grupoacms.com/iso-27032-gestion-ciberseguridad

Hai, R., Koutras, C., Quix, C., & Jarke, M. (n.d.). *Data Lakes: A Survey of Functions and Systems*. https://doi.org/10.1109/TKDE.2023.3270101

ICONTEC. (2006). *NORMA TÉCNICA NTC-ISO/IEC COLOMBIANA 27001*. https://img1.wsimg.com/blobby/go/b653c9ee-535c-4528-a9c5-bb00166ad0dc/downloads/1cd65ml0r_919353.pdf

Instituto Nacional de Tecnologías de la Comunicación. (2011). ¿Qué son las vulnerabilidades del software? *Cuaderno de Notas Del OBSERVATORIO*, 1–10. https://www.jesusamieiro.com/wp-content/uploads/2011/08/Que_son_las_vulnerabilidades_del_-software.pdf

Ivan Belcic. (2020, July 22). *¿Qué es la inyección de SQL y cómo funciona?* Avast. https://www.avast.com/es-es/c-sql-injection#:~:text=La%20inyecci%C3%B3n%20de%20SQL%20es,la%20informaci%C3%B3n%20de%20los%20usuarios.

José Luis Solleiro Rebolledo, Rosario Castañón Ibarra, Ángel David Guillén Valencia, Tania Yadira Hernández Molina, & Norma Solís Mérida. (2022). Vigilancia tecnológica en CIBERSEGURIDAD. *Tendencias Tecnológicas*, 2–44. https://www.redinnovagro.in/pdfs/cyber.pdf

J. Pomeyrol. (2019, September 6). *Visual Studio Code, el editor de código preferido por los desarrolladores*. MUY LINUX. https://www.muylinux.com/2019/09/06/visual-studio-code-editor-preferido-desarrolladores/

Jung A. (2022). *Data Query Optimization with Groq*. Sanity. https://www.sanity.io/answers/discussion-of-groq-query-optimization-and-potential-for-a-referencedby-function

Kaspersky Lab. (2023, December 30). *¿Qué es un ataque de scripting entre sitios? Definición y explicación*. Kaspersky. https://www.kaspersky.es/resource-center/definitions/what-is-a-cross-site-scripting-attack

Luque Tovar, A. L., & Bravo Builes, C. B. (2021). *Diseño e implementación de un sistema de protección contra el ataque de inyección SQL, en un servidor vulnerable utilizando herramientas Open Access* [UDFJC]. http://hdl.handle.net/11349/29053

Lyda Janeth Rodríguez Torres. (2017). *CONTROLES DE SEGURIDAD EN APLICACIONES ENTIDAD PROMOTORA DE SALUD*. https://alejandria.poligran.edu.co/bitstream/handle/10823/1042/Controles%20de%20Seguridad%20en%20Aplicaciones.pdf?sequence=1&isAllowed=y

Márquez Coca, W. A., Valenzuela Chicaiza, C. V., Acosta Jaramillo, C. A., Gaón Rojas, N. M., & Chimarro Amaguaña, J. D. (2023). *Desarrollo de aplicaciones web con tecnologías modernas*. Religación Press. https://doi.org/10.46652/ReligacionPress.108

Melissa Valeria Echeverría Joniaux, Melissa Andrea Garaycoa Walker, & Aleksandar Tusev. (2020). ¿ESTÁN PREPARADOS LOS MILLENNIALS ECUATORIANOS CONTRA UN ATAQUE INFORMÁTICO? *CHAKIÑAN, REVISTA DE CIENCIAS SOCIALES Y HUMANIDADES*, *10*, 73–86. https://doi.org/10.37135/chk.002.10.05

Meta Platforms. (2023). *Tutorial: Intro to React*. https://legacy.reactjs.org/tutorial/tutorial.html#what-is-react

Microsoft. (2023). *TypeScript: JavaScript With Syntax For Types*.

MINITIC. (2022). *ABC SEGURIDAD DIGITAL Decreto de Decreto 338 de 2022*. https://gobiernodigital.mintic.gov.co/692/articles-238198_recurso_1.pdf

MORENO MARÍN JOHN EDISON. (2019). *Modelo base de conocimiento para auditorías de seguridad en Servicios Web con SQL Injection* [UDFJC]. https://repository.udistrital.edu.co/server/api/core/bitstreams/da8dc23a-ceaa-4d59-95b2-7fd013084609/content

OpenAI. (2024, September 17). *Introducing OpenAI o1-preview*. https://openai.com/index/introducing-openai-o1-preview/

OpenJS Foundation. (2023). *Acerca de Node.js*. https://nodejs.org/en/about

OWASP. (2024). *Cross Site Request Forgery (CSRF)*. OWASP Foundation. https://owasp.org/www-community/attacks/csrf#

Quijano Gonzalez, S. D., & Rodriguez Espinosa, L. M. (2016). *AirQud: Plugin para Qgis que Permite Realizar un Modelo Básico de la Dispersión de Contaminantes por Fuentes Fijas*. UDFJC.

Rebecca L. Russell, Louis Kim, Lei H. Hamilton, Tomo Lazovich, Jacob A. Harer, Onur Ozdemir, Paul M. Ellingwood, & Marc W. McConley. (2018). *Automated vulnerability detection in source code using deep representation learning*. 757–762.

Sanity. (2023). *Query Language (GROQ)*. Sanity. https://www.sanity.io/docs/groq

Tom B. Brow, Benjamin Mann, & Melanie Subbiah. (2020). Language Models are Few-Shot Learners. *ArXiv*. https://doi.org/10.48550/arXiv.2005.14165

Valle, M. (2022, August 17). *Un ciberataque a Twilio expone datos de 1.900 usuarios de Signal*. Bitlifemedia. https://bitlifemedia.com/2022/08/un-ciberataque-a-twilio-expone-datos-de-1-900-usuarios-de-signal/

Wanumen Silva, L. F. (2006). COMO CREAR HERRAMIENTAS SEGURAS DE DESARROLLO USANDO CODIGO SEGURO. *Revista Vínculos*, *3*, 92–106.

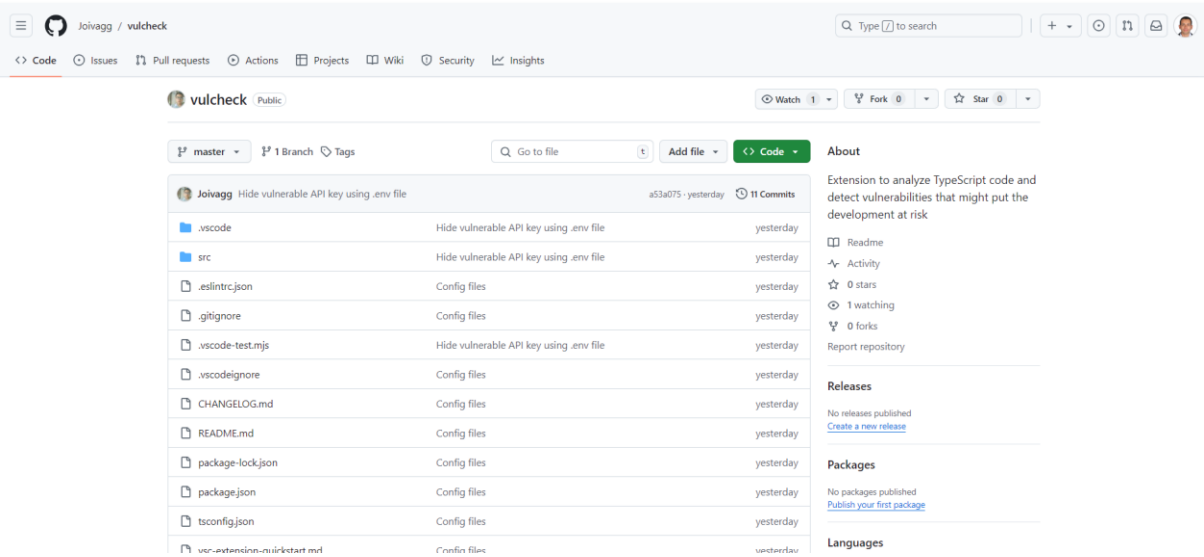
Anexos

El presente Anexo se compone de lo siguiente:

1. Repositorio que contiene el código de la extensión

https://github.com/Joivagg/vulcheck.git

Ilustración 15 Resultado de acceder al repositorio de la extensión.

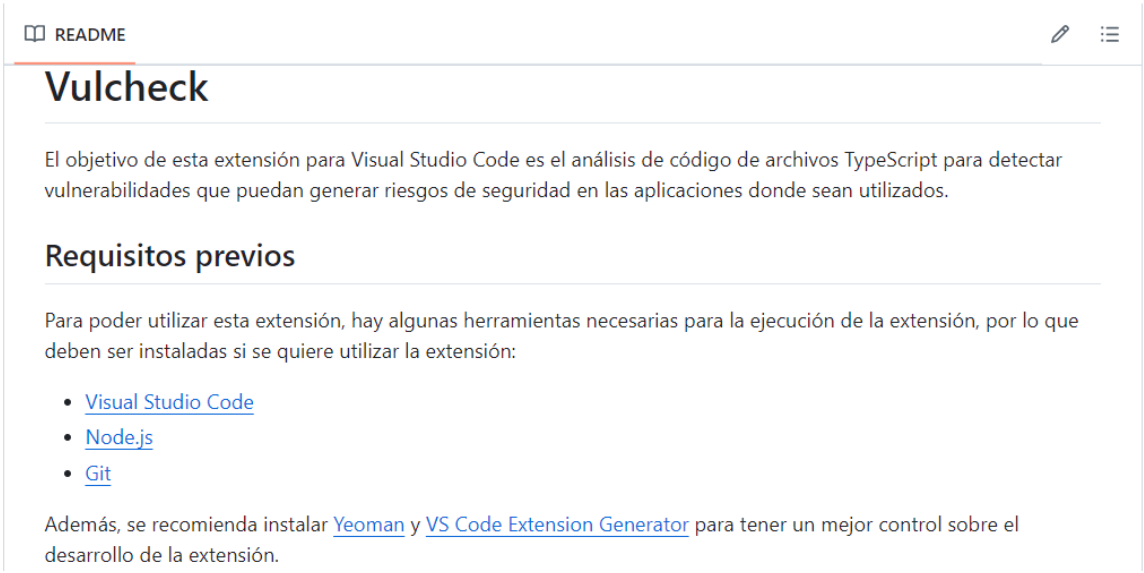


Nota: la anterior imagen muestra los archivos correspondientes a una aplicación Node JS, el cual contiene el código de la extensión a cargar en Visual Studio Code.

2. Manual de instalación alojado en la descripción del repositorio

A continuación, se presenta el paso a paso de instalación para ejecutar el código del prototipo de la extensión en el programa Visual Studio Code:

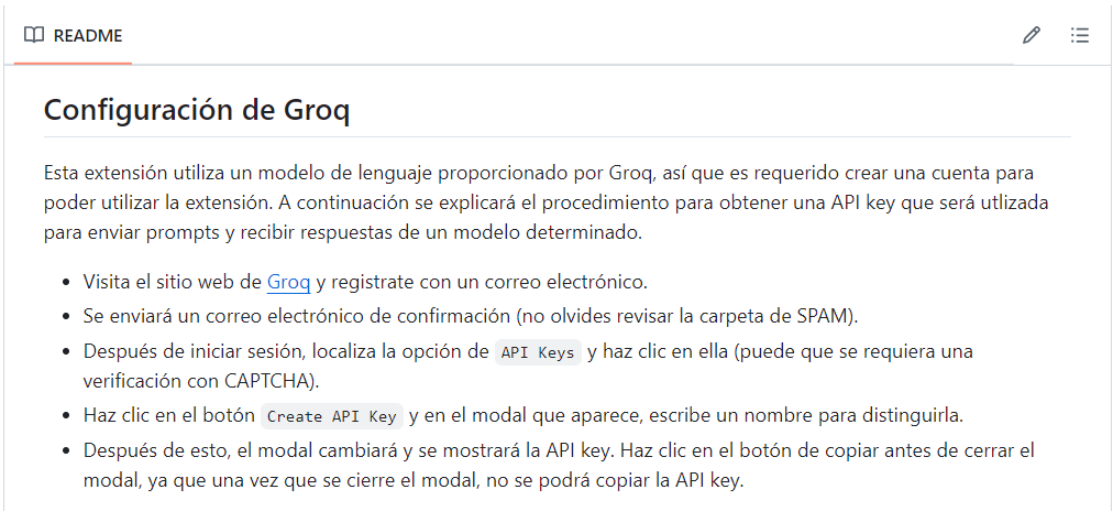
Ilustración 16 Requisitos de instalación previos o entorno de desarrollo



Nota: la anterior imagen muestra el conjunto de librerías y programas necesarios para la implementación de la extensión.

El desarrollo de la extensión en primera instancia condujo a la determinación de requisitos funcionales y no funcionales los cuales permitieron identificar el entorno de desarrollo junto con los programas necesarios para la creación del prototipo de extensión, es así como ahora bien se presenta una de las etapas más relevantes la cual nos permitirá integrar inteligencia artificial a nuestra extensión.

Ilustración 17 Configuración de Groq como Api Key mediante cuenta



README

Configuración de Groq

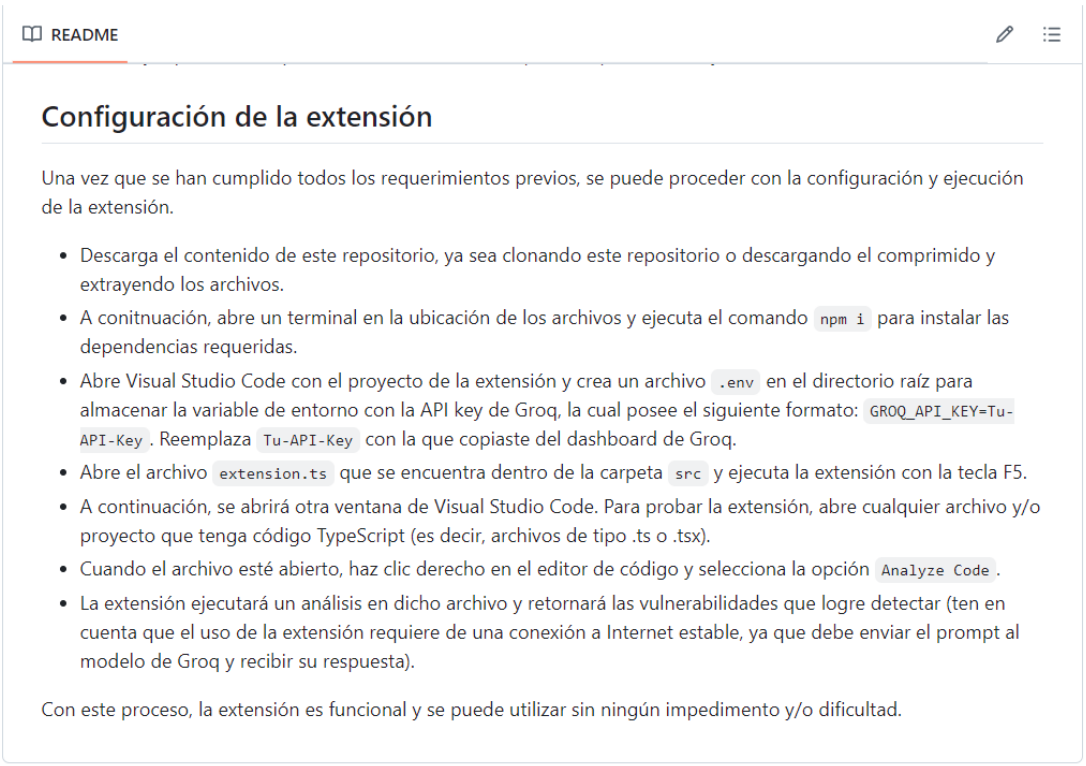
Esta extensión utiliza un modelo de lenguaje proporcionado por Groq, así que es requerido crear una cuenta para poder utilizar la extensión. A continuación se explicará el procedimiento para obtener una API key que será utlizada para enviar prompts y recibir respuestas de un modelo determinado.

- Visita el sitio web de Groq y registrate con un correo electrónico.
- Se enviará un correo electrónico de confirmación (no olvides revisar la carpeta de SPAM).
- Después de iniciar sesión, localiza la opción de `API Keys` y haz clic en ella (puede que se requiera una verificación con CAPTCHA).
- Haz clic en el botón `Create API Key` y en el modal que aparece, escribe un nombre para distinguirla.
- Después de esto, el modal cambiará y se mostrará la API key. Haz clic en el botón de copiar antes de cerrar el modal, ya que una vez que se cierre el modal, no se podrá copiar la API key.

Nota: la anterior imagen muestra el conjunto de instrucciones necesarias para la configuración de Api Key de Groq.

Ahora bien, se realizó un paso a paso para el despliegue de la extensión

Ilustración 18 Paso a paso de instalación de la extensión en Visual Studio Code



README

Configuración de la extensión

Una vez que se han cumplido todos los requerimientos previos, se puede proceder con la configuración y ejecución de la extensión.

- Descarga el contenido de este repositorio, ya sea clonando este repositorio o descargando el comprimido y extrayendo los archivos.
- A conitnuación, abre un terminal en la ubicación de los archivos y ejecuta el comando `npm i` para instalar las dependencias requeridas.
- Abre Visual Studio Code con el proyecto de la extensión y crea un archivo `.env` en el directorio raíz para almacenar la variable de entorno con la API key de Groq, la cual posee el siguiente formato: `GROQ_API_KEY=Tu-API-Key`. Reemplaza `Tu-API-Key` con la que copiaste del dashboard de Groq.
- Abre el archivo `extension.ts` que se encuentra dentro de la carpeta `src` y ejecuta la extensión con la tecla F5.
- A continuación, se abrirá otra ventana de Visual Studio Code. Para probar la extensión, abre cualquier archivo y/o proyecto que tenga código TypeScript (es decir, archivos de tipo .ts o .tsx).
- Cuando el archivo esté abierto, haz clic derecho en el editor de código y selecciona la opción `Analyze Code`.
- La extensión ejecutará un análisis en dicho archivo y retornará las vulnerabilidades que logre detectar (ten en cuenta que el uso de la extensión requiere de una conexión a Internet estable, ya que debe enviar el prompt al modelo de Groq y recibir su respuesta).

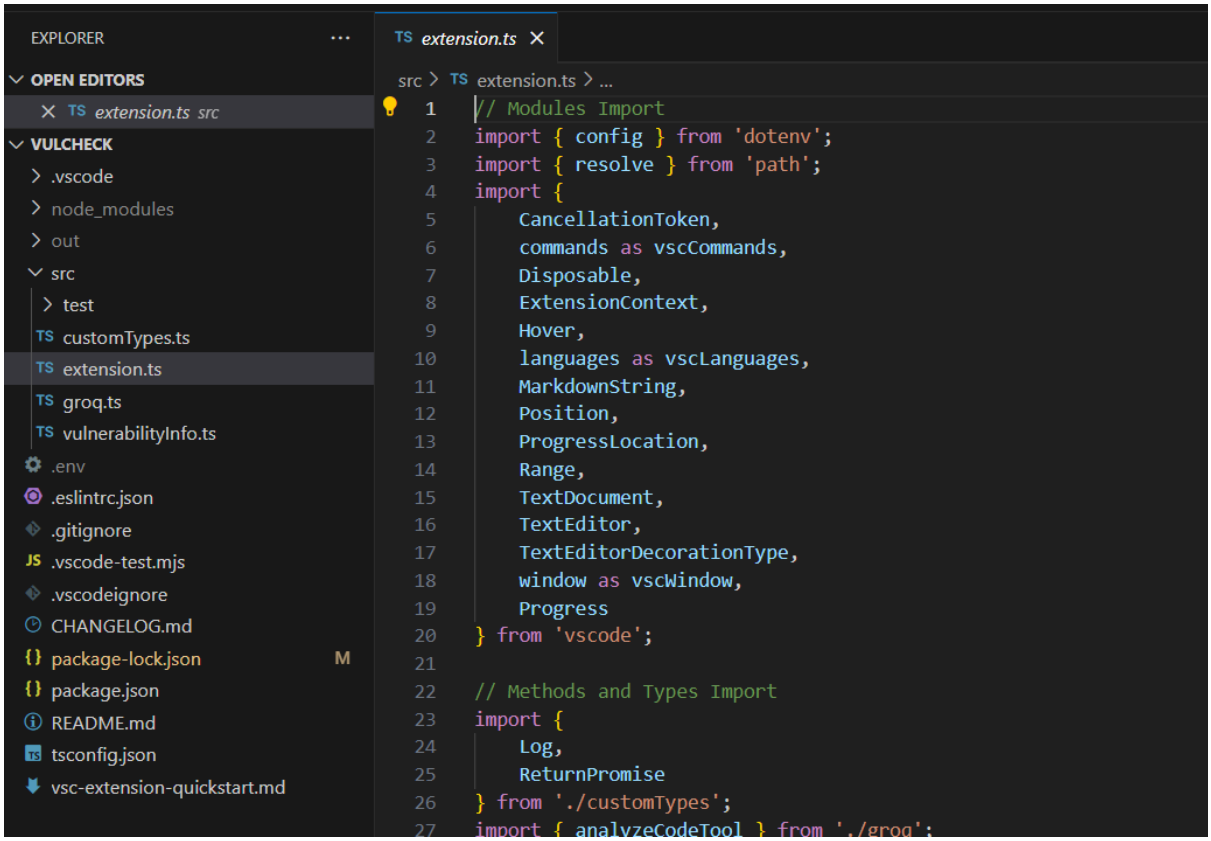
Con este proceso, la extensión es funcional y se puede utilizar sin ningún impedimento y/o dificultad.

Nota: la anterior imagen muestra el conjunto de instrucciones para configurar la extensión dentro de Visual Studio Code.

3. Resultado de implementación y uso de la herramienta

Una vez realizada la correcta instalación de la extensión dentro del entorno Visual Studio Code es momento de llevar a cabo una pequeña guía que describa el funcionamiento de la extensión en un caso real.

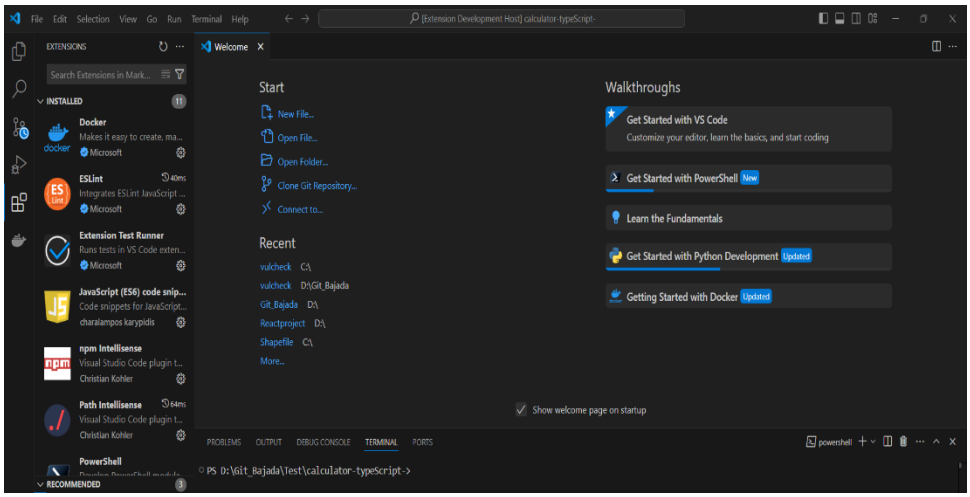
Ilustración 19 Estructura final del proyecto luego de la instalación de instancias



Nota: la anterior imagen muestra el conjunto archivos importados del repositorio junto a los archivos creados durante el proceso de configuración de variables.

El resultado de ejecutar el código anterior es el siguiente:

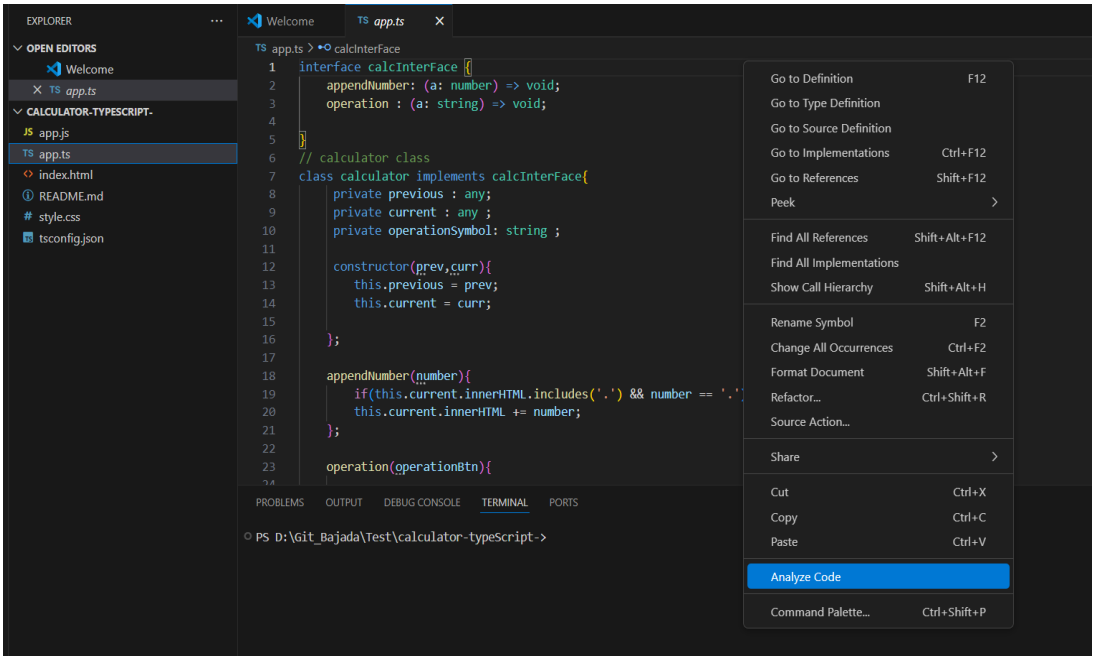
Ilustración 20 Ventana para test del código objetivo



Nota: la anterior imagen muestra el despliegue de la extensión o interfaz para introducir el archivo ts para su análisis.

Para el proyecto se realizó pruebas con diferentes códigos, pero para este anexo se presenta el siguiente Git Hub publico https://github.com/samaghapour/calculator-typeScript- el cual contiene un ejemplo de calculadora desarrollado por el usuario samaghapour.

Ilustración 21 Despliegue de Analyze Code al código ts de prueba



Nota: Despliegue del comando analyze Code dentro del menú de lectura de archivo ts

Una vez se hace clic derecho en el código ts y se selecciona la herramienta Analyze Code este muestra el siguiente estado

Ilustración 22 Inicio de la herramienta Analyze Code en entorno de ejecución



Nota: La imagen muestra la ventana de estado correspondiente al proceso de despliegue la extensión.

El programa tardara unos segundos a minutos dependiendo del tamaño del código analizar, en este caso no existen más variables a configurar por lo que se ha refinado previamente como se mencionó en el capítulo de refinamiento.

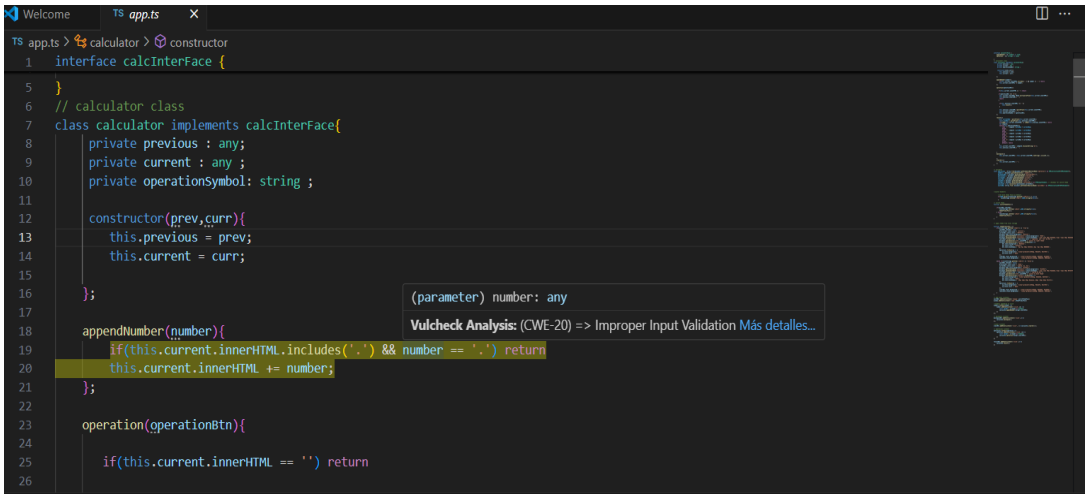
Ilustración 23 Finalización del proceso de análisis del código TS



Nota: La imagen muestra la ventana de estado correspondiente al proceso de final una vez este finaliza e indica el número de vulnerabilidades encontradas.

Analizaremos únicamente un resultado teniendo en cuenta que el resultado son 3 vulnerabilidades.

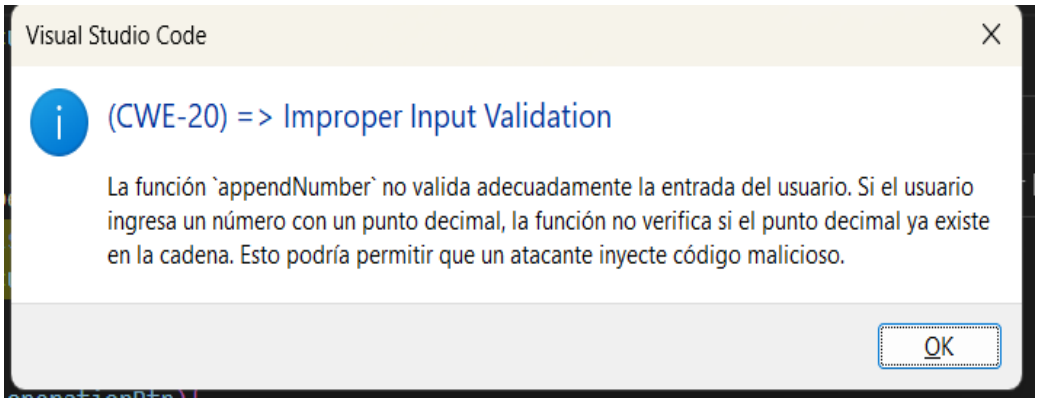
Ilustración 24 Vulnerabilidad 1 de 3 o resultado de Analyze Code.



Nota: La imagen muestra en color amarillo a la vulnerabilidad encontrada, y el cursor permite desplegar información de la vulnerabilidad encontrada.

Una vez se presiona clic sobre más detalles se observa la información más específica de la vulnerabilidad.

Ilustración 25 Descripción de la vulnerabilidad detectada por Analyze Code



Nota: La imagen muestra la ventana emergente que permite entender la vulnerabilidad de código TypeScript encontrada.

Para el caso anterior se encuentra dentro de **CWE (Common Weakness Enumeration):** el cual corresponde a un ataque de inyección SQL, la herramienta despliega en español la información más detallada.