SISTEMAS ÁGILES Y COGNITIVOS PARA MEJORAR LAS BUENAS PRÁCTICAS EN CIBERSEGURIDAD PARA EL FRONT-END

Luisa Fernanda Quintero Fernández

Julián Andrés Rivera Carrillo

Universidad Icesi

Facultad de Ingeniería

Ingeniería de Sistemas

Cali

2022

SISTEMAS ÁGILES Y COGNITIVOS PARA MEJORAR LAS BUENAS PRÁCTICAS EN CIBERSEGURIDAD PARA EL FRONT-END

Luisa Fernanda Quintero Fernández

Julián Andrés Rivera Carrillo

Proyecto de grado

Christian Camilo Urcuqui López, MSc.

Universidad Icesi

Facultad de Ingeniería

Ingeniería de Sistemas

Cali

2022

Tabla de Contenidos

RESUMEN

Con el rápido crecimiento de tecnologías para el Front-End y la poca sanitización del código, han surgido múltiples formas de atacar a través de este, permitiendo a los cibercriminales acceder a información confidencial, suplantar identidades, robar dinero, acceder a las bases de datos, entre otros.

Nuestro proyecto se acerca a las 3 de las 10 vulnerabilidades identificadas por OWASP como las más reconocidas y comunes en las aplicaciones web, para identificarlas y evaluarlas a nivel de código en JavaScript. Para esto se propuso un modelo de machine learning, detectando la frecuencia en que sale una palabra con la vulnerabilidad identificada; dando como resultado un modelo con una exactitud del 89%.

Finalmente, se implementó una extensión en Visual Studio Code para leer en tiempo real el código que la persona va escribiendo para así identificar que vulnerabilidades posee.

ABSTRACT

With the rapid growth of Front-End technologies and the lack of code sanitization, multiple ways of attacking through code have emerged. Allowing cybercriminals to access confidential information, impersonate identities, steal money, access databases, among others.

Our project approaches the 3 of 10 vulnerabilities identified by OWASP as the most recognized and usual in web applications, to identify and evaluate them at the JavaScript code level. For this, a machine learning model was proposed, detecting the frequency of occurrence of a word with the identified vulnerability; resulting in a model with an accuracy of 89%.

Finally, an extension was implemented in Visual Studio Code to read in real time the code that the person is writing in order to identify which vulnerabilities it has.

LISTA DE ACRÓNIMOS

- **URL:** Localizador de Recursos Uniforme (*Uniform Resource Locator*)
- **OWASP:** Proyecto abierto de seguridad de aplicaciones web (*Open Web Application Security Project*)
- **NLP:** Procesamiento de Lenguaje Natural (*Natural Language Process)*
- **HCI:** Interacción Humano – Computador (*Human – Computer Interaction*)
- **HTTP:** Protocolo de transferencia de Hipertexto (*HyperText Transfer Protocol*)
- **XSS:** Secuencia de comandos en sitios cruzados (*Cross-site Scripting*)
- **XML:** Lenguaje de Marcado Extensible (*Extensible Markup Language*)
- **HTML:** Lenguaje de Marcas de Hipertexto (*HyperText Markup Language*)
- **CSS:** Hojas de estilo en cascada (*Cascading Style Sheets*)
- **ML:** Aprendizaje Automático *(Machine Learning)*
- **DeepSQLi:** es una herramienta basada en el procesamiento profundo del lenguaje natural para el testeo SQLi.
- **SQLi:** Ataques de Inyección SQL
- **SQL:** Lenguaje de Consulta Estructurada *(Structured Query Language)*
- **CNN:** Red Neuronal Convolucional *(Convolutional Neural Network)*
- **RF:** Algoritmo de Bosque Aleatorio *(Random Forest)*
- **RNN:** Red Neuronal Recurrente *(Recurrent Neural Network)*
- **IDE**: Entorno de desarrollo integrado (*Integrated Development Environment*)
- **API**: Interfaz de programación de aplicaciones (*Application Programming Interfaces*)

- **REST:** Transferencia de Estado Representacional (*Representational State Transfer*)
- **VSCode**: Visual Studio Code
- **CVE**: Vulnerabilidades y exposiciones comunes (*Common Vulnerabilities and Exposures*)
- **CWE**: Enumeración de debilidades comunes (*Common Weakness Enumeration*)
- **SHA**: Algoritmo de Hash Seguro (*Secure Hash Algorithm*)
- **TF-IDF**: Frecuencia de término – Frecuencia inversa de documento (*Term Frequency-Inverse Document Frequency*)

GLOSARIO DE TÉRMINOS

Ciencia de datos: obtener elementos de valor (patrones) de distintas fuentes de información, a través de métodos estadísticos, minería de datos, *machine learning*, entre otros. [1]

Ciberseguridad: área de la ciencia de la información que se encarga de desarrollar e implementar mecanismos de protección de la información e infraestructura tecnológica. [1]

Machine Learning: parte de la IA, que busca dotar a un sistema con la capacidad de aprender en entornos variados, a través de unos algoritmos matemáticos.

Front-End: área del desarrollo web se enfoca en lo que el usuario visualiza y con lo que interactúa.

Frameworks: estructura real o conceptual que sirve como guía para desarrollar aplicaciones de software de forma más fácil.

Solución cognitiva: es el uso de modelos computarizados para simular el pensamiento y cerebro humano, en cuanto a velocidad y capacidad cognitiva. [2]

Agilismo: postulado de cuatro valores y doce principios fundamentales para tener la habilidad de producir entregas de valor más rápidas, continuas y de calidad.

Procesamiento de lenguaje natural (NLP): habilidad de una inteligencia artificial para tomar la información del mundo real y procesarla; para que el computador pueda entender el lenguaje natural humano, en su forma escrita y hablada.

Human-Computer Interaction (HCI): campo de estudio multidisciplinario que se enfoca en el diseño de tecnología computacional y en especial, la interacción entre humanos y computadores. Por un lado, se centra en el estudio de como las personas interactúan con la

tecnología. Por otro lado, se basa en cómo crear intervenciones con la tecnología que afecten de manera positiva a las personas.

Análisis estático: técnica que evalúa los comportamientos maliciosos en el código fuente, los datos o los archivos binarios sin ejecutar explícitamente la aplicación. [1]

Scraping: Tecnología capaz de extraer datos desde un programa o código base. [26]

Análisis dinámico: proceso de evalución y testeo de un codigo en tiempo de ejecución y/o escritura. [27]

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INDICE DE ILUSTRACIONES

INDICE DE GRAFICAS

MOTIVACIÓN Y ANTECEDENTES

A medida que pasa el tiempo, las tecnologías como: *frameworks*, librerías, paquetes, arquitecturas, entre otros; van aumentando exponencialmente, y a su vez, las vulnerabilidades a las que se exponen estos sistemas también. Dicha situación afecta directamente a las páginas web, puesto que, a los desarrolladores les resulta difícil mantenerse actualizados de cada una de las vulnerabilidades que aparecen en las nuevas tecnologías.

En febrero del 2017, se descubrió un ataque en Steam de *Cross-site scripting* (XSS), en donde, al visitar un perfil de un usuario se podía ejecutar un código malicioso a través de inyectar código JavaScript en cualquier tag de HTML. Esto conllevó a que el cibercriminal pudiera, a través del perfil infectado, redirigir a la persona hacia otra página para suplantar su inicio de sesión. Además, podía utilizar CSS para cambiar su foto de perfil y así engañar a los usuarios; también drenar el dinero que tenga en su billetera de Steam.[3] La vulnerabilidad surgió debido a que los desarrolladores no 'desinfectaron el código', es decir, no tuvieron buenas prácticas para evitar que un usuario no pudiera ingresar texto malicioso (SQL, JavaScript), o hacer solicitudes en el HTML.

Los reportes de Snyk [4] y Varonis [5], hay una gran deficiencia frente a la ciberseguridad, en promedio más del 50% de la población no conoce cuales son las distintas formas para protegerse y/o combatir algún ataque cibernético; sumándole que el tiempo que toma en resolver una sola vulnerabilidad suele ser en promedio 3 meses.

Por otro lado, la creciente demanda de empleo y la baja cantidad de programadores capacitados en la ciberseguridad también influye en la problemática, se pronostica que siga aumentando el desbalance en cuanto a la relación de vacantes-empleados. Actualmente, solo en Estados Unidos, hay más de 700.000 vacantes abiertas para empleos de ciberseguridad sin

ocupar [6], y se estima que para el 2025 dicha cifra sea de 3.5 millones de vacantes a nivel global [7].

El problema de las vulnerabilidades en aplicaciones web se ha presentado en distintas compañías al no estar debidamente protegidas y/o tener malas prácticas dentro del código, lo que genera un mayor incremento en las oportunidades de ataques debido a la facilidad y rapidez en poder realizarlos.

Aunque existen ciertas soluciones y metodologías ágiles que pueden ayudar a contrarrestar este problema, se siguen presentando altos de niveles de vulnerabilidades; debido a que la industria se encuentra cambiando y cada vez demandan tiempos de entrega más cortos para un producto terminado, poniendo a los desarrolladores a entregar un producto funcionalmente terminado, pero poco seguro.

Por todo lo anterior, es necesario seguir trabajando en formas que mejoren las tecnologías digitales; a través de una solución cognitiva se podría facilitar el proceso de programación en el desarrollo del software, y así mismo, el conocimiento de los programadores.

Esto le permite tanto a los desarrolladores de web Front-End como a los expertos en ciberseguridad reducir el tiempo de análisis de vulnerabilidades en el código mientras un proyecto se encuentra en desarrollo, permitiendo minimizar futuras vulnerabilidades cuando la aplicación web se encuentre desplegada como también cumplir los tiempos de entrega puntualmente.

DESCRIPCIÓN DEL PROBLEMA

Los desarrolladores Front-End no tienen suficientes herramientas que abarquen la compleja diversidad de desafíos en ciberseguridad. Parte de esa insuficiencia viene de los

frameworks y las extensiones actuales al utilizar sistemas de análisis estático. A su vez, generan confusión por el lenguaje técnico que manejan dado el desconocimiento general sobre ciberseguridad. Lo anterior, deja como consecuencia que la mayoría de las aplicaciones web sean inseguras, corriendo el riesgo de que un cibercriminal pueda: i) aprovecharse de sus vulnerabilidades y, ii) ejecutar actividades maliciosas sobre ellas, que perjudiquen a usuarios, organizaciones, entre otros.

OBJETIVOS

Objetivo General

Desarrollar una extensión cognitiva y ágil para reducir el número de vulnerabilidades en el Front-End.

Objetivos Específicos

- Evaluar un modelo para detectar las vulnerabilidades en aplicaciones web.
- Implementar la solución cognitiva a través de un entorno de desarrollo integrado.
- Revisar un conjunto de proyectos con vulnerabilidades para futuras validaciones de las predicciones de la extensión.

MARCO TEÓRICO

Inteligencia artificial

Es el área donde se desarrollan computadoras que, además de intentar emular el pensamiento y/o razonamiento humano, estudian al ser humano, su comportamiento y procesos cognitivos; para así, realizar una creación artificial del conocimiento. [1]

Machine Learning

Es una rama perteneciente a la Inteligencia Artificial, en donde se busca dotar a un sistema de información para que pueda aprender en entornos variados. Se basan en la aplicación de algoritmos matemáticos, y conjuntos de registros y variables [8]. El entrenamiento de la máquina puede tener tres enfoques distintos: [9]

Aprendizaje supervisado: aprendizaje basado en un conjunto de datos etiquetados con una clase o valor. A través de las etiquetas el modelo aprende y mejora su precisión a medida que pasa el tiempo.

Aprendizaje no supervisado: aprendizaje que se basa en descubrir factores no observados o implícitos, y agruparlos de acuerdo con un patrón o tendencia encontrada.

Aprendizaje por refuerzo: el algoritmo aprende por medio de prueba/error, y cuando la máquina obtiene un resultado correcto, se le refuerza dicha decisión para que con el tiempo tome mejores decisiones.

Procesamiento natural del lenguaje

Habilidad de un programa de computador para entender el lenguaje natural humano, tanto en su forma escrita como hablada. Este sistema utiliza inteligencia artificial para tomar la información del mundo real, procesarla y generarle un sentido en una forma que el computador pueda entender el lenguaje humano. [9]

Redes neuronales

Redes que emulan el proceso de decisión realizado por las neuronas del sistema nervioso central biológico, a través del análogo "perceptrón", el cual consta de varias entradas que se combinan normalmente con una adición básica, la adición de entradas se modifica por una función de transferencia y el valor de la salida de la función de transferencia pasa a la salida del perceptrón. La salida del perceptrón puede conectarse a la entrada de otro perceptrón, de modo que una red neuronal consiste en conjunto de perceptrones organizados en niveles o capas. [8]

Deep Learning

Es un grupo de algoritmos de *machine learning* que representan el mundo como una jerarquía anidada de conceptos, donde cada uno de estos es definido en relación con conceptos más simples. Su arquitectura se basa principalmente en redes neuronales con capas ocultas. Hay distintas arquitecturas de redes neuronales como: redes neuronales profundas, las redes neuronales convolucionales y las redes neuronales recurrentes. [8]

Solución cognitiva

Uso de modelos computarizados para simular el cerebro humano en cuanto a velocidad y capacidad cognitiva usando NLP [10]. Una solución cognitiva debe ser:

Adaptativo: Habilidad de ser flexible para que el modelo aprenda mientras los datos y la información van cambiando rápidamente.

Interactivo: la interacción humano-computadora (HCI) es una parte importante, debido a que el usuario tendrá que interactuar con estos sistemas cognitivos, y definir sus necesidades continuamente.

Iterativo y con estado: identifica los problemas y los aclara a través de preguntas y extracción de datos adicionales. Además, mantiene información sobre problemas similares que han ocurrido previamente.

Contextual: los sistemas cognitivos deben entender, identificar y extraer datos contextuales. Dichos datos pueden ser tomados de distintas fuentes como datos estructurados y no estructurados: sensoriales, visuales y auditivos. [2]

Software

Conjunto de instrucciones, datos o programas que le dicen al computador que tarea debe ejecutar.

Desarrollo web

Software que utilizan los navegadores web para realizar actividades vía web. Se pueden utilizar tecnologías como ASP y PHP para almacenar información. Por otro lado, para desplegar esa información al cliente se usa HTML, CSS y JavaScript. [13]

Desarrollo Front-End

Es el desarrollo desde el lado del cliente, utiliza las tecnologías HTML, CSS y JavaScript para realizar una aplicación con la que el usuario va a interactuar. A su vez, transforma el código escrito en el Back-End, en una interfaz gráfica, fácil de leer y entendible en cualquier dispositivo, sistema operativo o navegador. [11]

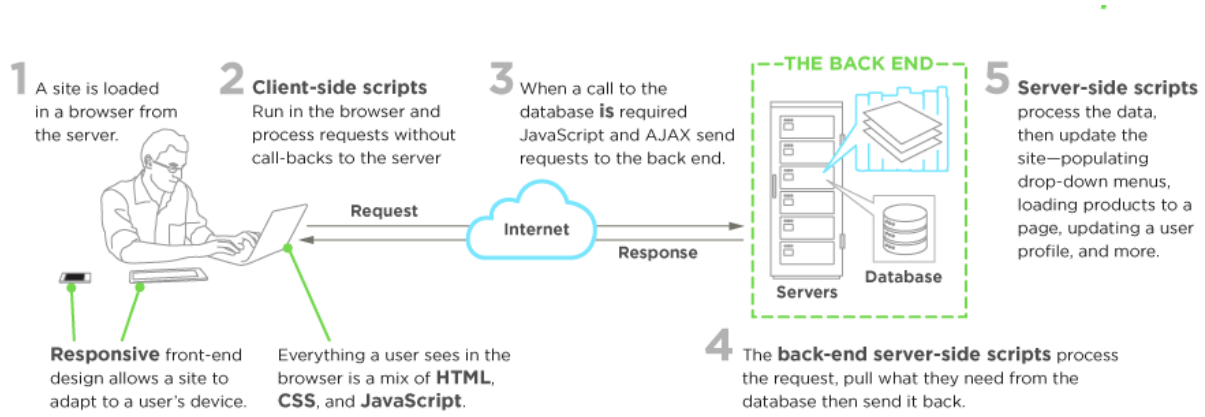


Ilustración 1 *Proceso del desarrollo de aplicaciones web.*
Fuente: https://www.upwork.com/hiring/development/front-end-developer/

DevSecOps

Conjunto de prácticas y herramientas que agrupan al desarrollo del software, la gestión de operaciones TI y la seguridad; para entregar aplicaciones y servicios a gran velocidad de forma segura [12]. Estas prácticas abordan los problemas de vulnerabilidades a medida que van surgiendo, por lo cual, su implementación resulta ser más fácil, rápida y menos costosa.

Ciberseguridad

Área de las ciencias de la computación que se encarga del desarrollo e implementación de mecanismos que protegen la información y la infraestructura tecnológica. Actualmente, es sumamente importante contar con sistemas seguros que garanticen los tres pilares de la ciberseguridad: confidencialidad, integridad y disponibilidad de los datos. Lo anterior, se debe al crecimiento acelerado de la tecnología, la falta de personas especializadas en seguridad digital, las ciberguerras y las vulnerabilidades no detectadas. [1]

Análisis estático

Técnica que evalúa los comportamientos maliciosos en el código fuente, datos o archivos binarios sin ejecutar explícitamente la aplicación [1]. Dicho método ha perdido fuerza debido a que los cibercriminales han adquirido más experiencia con dicho análisis.

Vulnerabilidades

Una vulnerabilidad es un hueco o debilidad en la aplicación, la cual puede constar de una falla de diseño o un bug implementado que permite al atacante (hacker) causar daño a los *stakeholders* de una aplicación.[15]

OWASP

Es una fundación sin ánimo de lucro que se dedica a desarrollar, mantener y comprar aplicaciones que sean de fiar [16]. OWASP publica periódicamente un top 10 con los riesgos informáticos más críticos, lo que estos representan, sus posibles repercusiones y finalmente una serie de recomendaciones para minimizar los riesgos.

Broken Access Control

Política que prohíbe a los usuarios actuar fuera de sus permisos previstos. Esta falla generalmente conduce a la divulgación, modificación o destrucción no autorizada de todos los datos o la realización de una función comercial fuera de los límites del usuario. [14]

Cryptographic Failures

Divulgación accidental de datos confidenciales. A menudo, se trata de información directamente vinculados a los clientes, por ejemplo: nombres, fechas de nacimiento, información financiera, etc., o incluso datos técnicos como nombres de usuario y contraseñas. Esto se puede deber al cifrado débil en cualquier dato transmitido para obtener acceso a la información interceptada. [14]

Injection

Envío de datos no confiables que hacen parte de un comando o consulta y pueden ejecutar comandos involuntarios o acceder a datos sin la debida autorización.[1]

Insecure design

Representa las diferentes debilidades expuestas en una implementación que no puede ser arreglada ya que nunca tuvo diseños de controles de seguridad para poder manejarlos. [14]

Security Misconfiguration

Vulnerabilidad que abusa de las características de los datos y permite que un atacante interactúe con cualquier Back-End o sistema externo al que pueda acceder la aplicación, donde permite que el atacante lea el archivo en ese sistema. Además, puede provocar un ataque de denegación de servicio o podrían realizar una falsificación de solicitud del lado del servidor que induce a la aplicación web a realizar solicitudes a otras aplicaciones. [14]

Vulnerable and Outdated Components

Software vulnerable, no compatible o desactualizado. Incluye el sistema operativo, el servidor web de aplicaciones, el sistema de administración de bases de datos, las aplicaciones, las API y todos sus componentes, los entornos de tiempo de ejecución y las bibliotecas. [14]

Identification and Authentication Failures

Vulnerabilidad que permite ataques de fuerza bruta o automatizados, tales como el relleno de credenciales, donde el atacante tiene una lista de nombres de usuario y contraseñas válidos. También utiliza recuperación de credenciales débil o ineficaz y procesos de contraseña olvidada como "respuestas basadas en el conocimiento", que no se pueden hacer seguras.[14]

ESTADO DEL ARTE

Para el estado del arte del proyecto se presentan 6 proyectos previos con diferentes formas de abordar el problema y distintas soluciones.

Automated reverse engineering of role-based access control policies of web applications

Ha Thanh Le, Lwin Khin Shar, Domenico Bianculli, Lionel Claude Briand y Cu Duy Nguyen desarrollaron un *framework* que se implementó para automatizar la exploración y detección en el control de accesos, e inferir políticas para el modelo en la implementación de aplicaciones web. Por defecto, tiene unas reglas de etiquetas genéricas para usar inicialmente, además se pueden ingresar etiquetas manualmente para deducir efectivamente más políticas de acceso de control.

Dicho Framework cuenta con una precisión del 97.8% para las políticas deducidas, ayudando así a la detección de vulnerabilidades en los controles de accesos, ya que con la herramienta probada en 4 sitios web, se detectaron 64 problemas de control de accesos que fueron reportados respectivamente. [17]

Este sistema da una solución aproximada para la automatización del proceso de detección, corrección y mejoría de las aplicaciones web, el cual es similar al enfoque del proyecto. Sin embargo, nuestro proyecto consiste en la implementación de una extensión que muestre sugerencias en tiempo real, y de esta manera se mejore la seguridad durante el desarrollo del ciclo de vida del software.

Employing and improving machine learning of detection of Phishing URLs

Yaitskyi, A. desarrolla un método de *machine learning* de automatización para la detección de phishing en aplicaciones web, dando una precisión del 98.4% y mostrando una mejoría frente a las detecciones regulares de phishing URL. Con esta mejoría, se aumenta la eficacia de la precisión y mejora el tiempo que toma en detectarse la vulnerabilidad, lo que, a su vez reduce el tiempo de detección manual que tendría un desarrollador para encontrar el phishing. [18]

Dicho método aporta en la parte de automatización y reducción del tiempo que se toma en la detección de vulnerabilidades con distintos algoritmos de clasificación. También se ve limitado a detección de phishing y por eso, nuestro proyecto no tiene el enfoque de analizar las URL.

DEKANT: A Static Analysis Tool That Learns to Detect Web Application Vulnerabilities

El artículo se enfoca en una herramienta de análisis estático, que inspirada en el NLP, aprende a detectar vulnerabilidades automáticamente con *machine learning*. Se usa un modelo de secuencia (HMM) para aprender a caracterizar vulnerabilidades basado en un conjunto de porciones de código anotadas. Este modelo se implementó en la herramienta y se experimentó con un conjunto de aplicaciones *open source* PHP y *plugins* de WordPress. [19]

Este documento aporta con el modelo de machine learning basado en NLP, esto porque es una aproximación a la solución cognitiva que se busca implementar, pero no es suficiente para el alcance del proyecto, pues está basado en análisis estático.

WAF-A-MoLE: Evading Web Application Firewalls through Adversarial Machine Learning

En este estudio se realizó una herramienta (WAF-A-MoLE) entrenada haciendo uso de un dataset público de consultas SQL. Sin embargo, se demostró que los WAF (Web Application Firewall) basados en ML pueden ser evadidos. Se usó una técnica de enfoque antagónico para crear *payloads* maliciosos que se clasifiquen como benignos. El estudio sugiere que el futuro se debería enfocar en testear WAF basado en otras técnicas, como la híbrida, para mejorar la detención de *payload* maliciosos.[20]

El aporte de este documento es el de poder identificar consultas SQL que puedan ser clasificadas como falsos positivos, aun así, en lo que carece el documento es que el WAF es utilizado una vez una página web es desplegada, por otro lado, en el proyecto se contempla evitar ataques desde el proceso de codificación.

Automated Vulnerability Detection in Source Code Using Deep Representation Learning

Este proyecto desarrolla una herramienta a partir de un conjunto de datos de ejemplos de código que se clasificaron con los hallazgos de tres analizadores estáticos para la creación de una representación sencilla y genérica para el entrenamiento de ML. Dicha herramienta trabaja sobre una red neuronal convolucional y clasificada con un algoritmo de árbol ensamblado mostrando que los modelos CNN funcionan mejor que los RNN. Además, un clasificador RF junto con

estas redes neuronales funciona mejor que las redes independientes. [21]

El anterior proyecto nos aporta un modelo de ML que nos puede ayudar con el análisis estático y ser base para el análisis dinámico de datos clasificados del código fuente.

DeepSQLi: Deep Semantic Learning for Testing SQL Injection

En este proyecto se propone una herramienta llamada DeepSQLi para generar casos de prueba y así detectar vulnerabilidades SQLi, enfocada al procesamiento profundo del lenguaje natural. A través de un modelo de lenguaje neural basado en Deep Learning, esta herramienta tiene la capacidad de aprender la semántica de los ataques SQLi y traducirlo en un nuevo caso de prueba. DeepSQLi muestra tener la capacidad de encontrar un numero significante de vulnerabilidades de manera eficiente, gracias al conocimiento semántico obtenido, además de, detectar vulnerabilidades ocultas y posiblemente desconocidas.[22]

El trabajo anterior permite usar el procesamiento natural de leguajes a través de redes neuronales profundas para la detección de vulnerabilidades de inyección SQL que es una de las vulnerabilidades que se tendrá en cuenta en nuestro proyecto.

Matriz de Estado del Arte

En la tabla 1 se realizó un resumen de lo mencionado en el estado del arte de acuerdo con los siguientes criterios claves:

i. **IA:** hace referencia al uso de la inteligencia artificial o *Machine Learning* de cualquier categoría en el funcionamiento o desarrollo del proyecto.

ii. **Front:** hace referencia al lado Front-End en un desarrollo de una aplicación, que es la vista en la que el usuario o cliente final interactúa con el sistema.

iii. **DevSecOps:** hace referencia al uso de metodologías ágiles implementadas en conjunto con Ciberseguridad mientras se realiza el ciclo de vida del proyecto.

iv. **Detects Vulnerabilities:** hace referencia a que el proyecto tiene relación o su alcance es detectar algún tipo de vulnerabilidad informática que haga parte de las 10 vulnerabilidades OWASP.

***Tabla* 1** *Matriz del Estado del Arte*

	IA	Front	DevSecOps	Detects Vulnerabilities

Emploging and improving machine learning of detection of Phishing URLs	X			X
Automated reverse engineering of role-based access control policies of web applications	X	X		X
DEKANT: A Static Analysis Tool That Learns to Detect Web Application Vulnerabilities	X			X
WAF-A-MoLE: Evading Web Application Firewalls through adversarial Machine Learning	X			X
Automated Vulnerability Detection in Source Code Using Deep Representation Learning				X
DeepSQLi: Deep Semantic Learning for Testing SQL Injection	X	X		X
Sistemas Ágiles y cognitivos para mejorar las buenas prácticas en Ciberseguridad para Front-End.	X	X	X	X

METODOLOGÍA

Para la metodología del proyecto se decidió utilizar CRISP-DM y el Ciclo de vida Incremental.

Ciclo de vida incremental

En nuestro proyecto utilizamos un modelo de ciclo de vida incremental que nos permitió tener un control total sobre el flujo de trabajo. También se encaja con el cronograma propuesto, permitiendo tener un desarrollo progresivo de la solución, especialmente en el entorno de la parte del software.

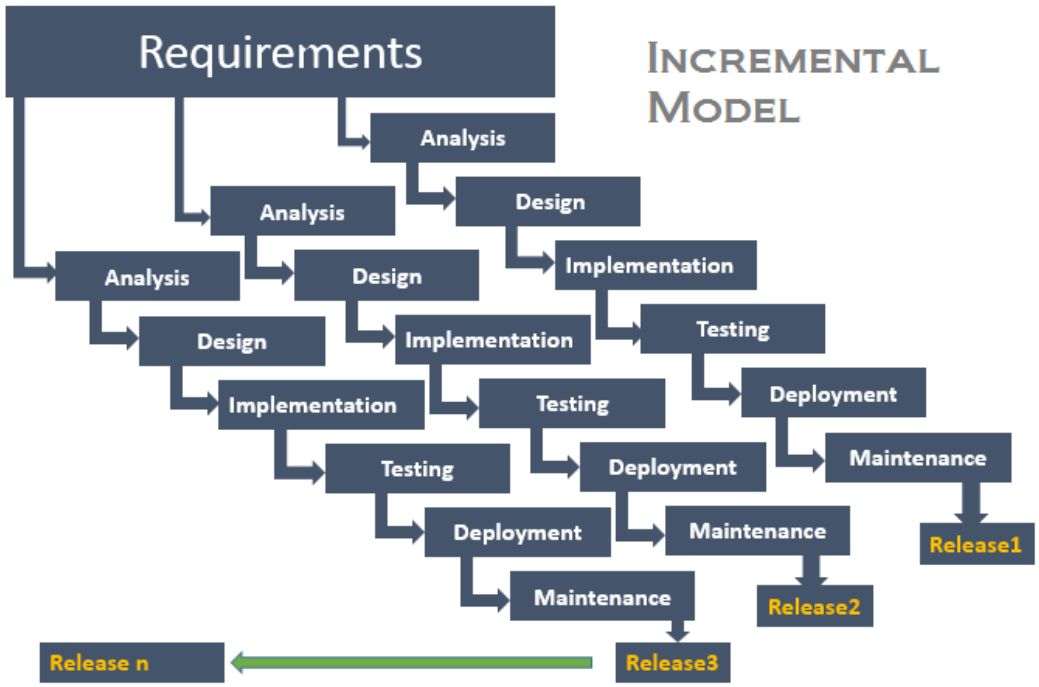


Ilustración 2 *Estructura del modelo incremental*

Fuente: https://testingfreak.com/incremental-model-software-testing-advantages-disadvantages-incremental-model/

Fase 1

Para la primera fase se llevó a cabo el análisis y elicitación de los requerimientos funcionales y no funcionales del proyecto. Adicionalmente, se hizo el particionamiento de los requerimientos a través del método Dorfman para poderlos clasificar y separar correctamente.

Fase 1.1 Requerimientos funcionales

- El sistema debe tener la capacidad de hacer lectura del código a medida que un desarrollador lo escriba.

- El sistema debe tener la capacidad de desplegar una lista de opciones de autocompletado para que un desarrollador pueda escoger la que más se ajuste a su necesidad mientras escribe.

- El sistema debe tener la capacidad de detectar vulnerabilidades sobre el código que un desarrollador se encuentre escribiendo.

- El sistema debe tener la capacidad de generar o señalizar alertas que le indiquen al desarrollador posibles errores en su código que se puedan traducir en vulnerabilidades.

- El sistema debe tener la capacidad de generar recomendaciones sobre errores y posibles vulnerabilidades en el código para el desarrollador.

- El sistema debe tener la capacidad de realizar correcciones sobre el código con respecto a los errores que encuentre.

Fase 1.2 Requerimientos no funcionales

1. El modelo es desarrollado en Python.

1. La extensión debe ser un plugin montado en un entorno de desarrollo integrado. (IDE)
1. El modelo utiliza análisis dinámico.

Fase 1.3 Análisis de Dorfman

En el último ítem de la fase 1 se realizó el Dorfman para identificar los sistemas y subsistemas pertenecientes en el proyecto.

Dicho análisis puede verse dentro de la carpeta: https://github.com/i2tResearch/Ciberseguridad_web/blob/master/Cognitive%20Solution%20for%20Vulnerability%20Detection%20on%20Front-End/Documentos/Dorfman.docx

Fase 2

Después, se realizó el primer acercamiento al diseño de nuestra solución. Se desarrolló el diagrama de *deployment* representado en la *figura 2* teniendo en cuenta los componentes, estructura y lógica base entre los diferentes módulos del software y como se conectan entre sí.

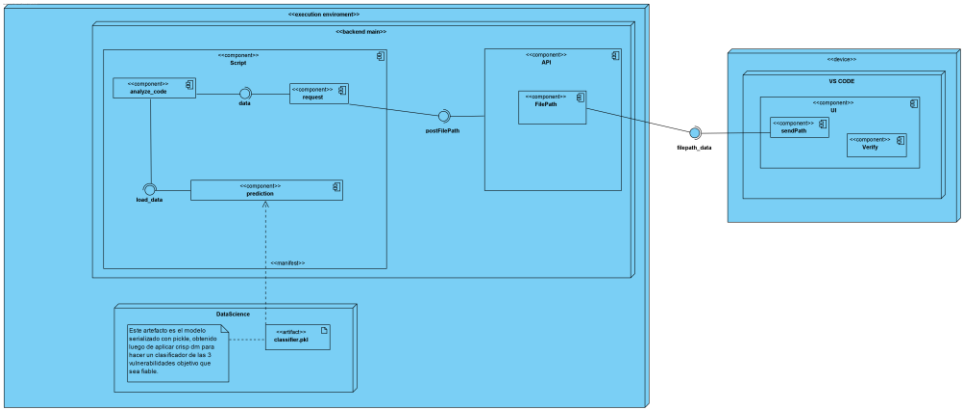


Figura 1 *Diagrama de Deployment*

Ver en detalle en:
https://github.com/i2tResearch/Ciberseguridad_web/blob/master/Cognitive%20Solution%20for%20Vulnerability%20Detection%20on%20Front-End/Documentos/deployment.vpp

Además, se realizó los diagramas de casos de uso (*figura 3*) para visualizar el flujo de cómo se espera que un usuario interactúe con la extensión y a su vez, esta interactúe con el modelo. Para mayor claridad se puede encontrar el documento en la carpeta.

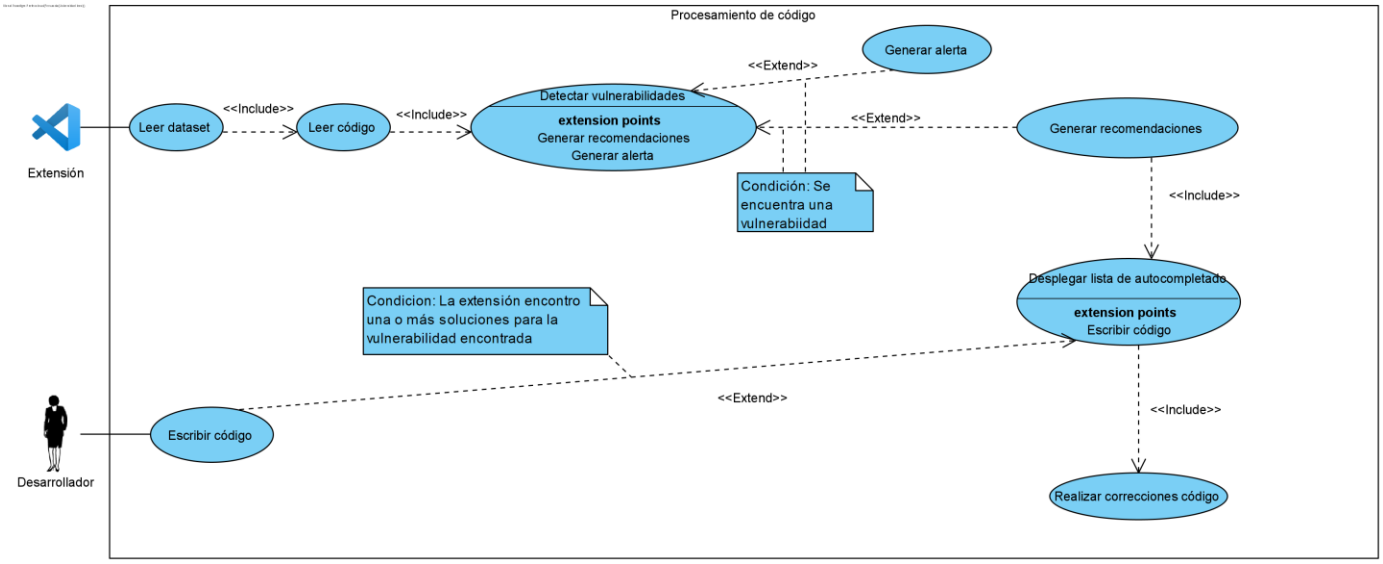


Figura 2*. Diagrama de casos de uso*

Ver en detalle en:

https://github.com/i2tResearch/Ciberseguridad_web/blob/master/Cognitive%20Solution%20for%20Vulnerability%20Detection%20on%20Front-End/Documentos/casos-uso.vpp

Fase 3

Para la tercera fase se llevó a cabo el proceso de búsqueda y recolección de los datos necesarios que nos permitieran desarrollar y probar la solución planteada. Para lograrlo, se realizó un tratamiento al conjunto de datos que pudieran generar problemas. A continuación, comenzó la construcción de un modelo propio

Co	cve_id	diff	cvss3_integrity_impact	cvss3_confidentiality_impact	cvss3_availability_impact	filename
0	CVE-2016-7103	@@ -206,7 +206,7 @@ test("closeOnEscape", function() { }); test("	LOW	LOW	NONE	options.js
1	CVE-2016-7103	@@ -426,7 +426,7 @@ $.widget("ui.dialog", { // dialog in IE (#9312)	LOW	LOW	NONE	dialog.js
2	CVE-2021-41184	@@ -113,7 +113,9 @@ QUnit.test("positions", function(assert) { });	LOW	LOW	NONE	core.js
3	CVE-2021-41184	@@ -148,7 +148,12 @@ $.fn.position = function(options) { options =	LOW	LOW	NONE	position.js
4	CVE-2022-31160	@@ -131,4 +131,41 @@ QUnit.test("Calling checkboxradio on an inp	LOW	LOW	NONE	core.js
5	CVE-2022-31160	@@ -96,4 +96,42 @@ QUnit.test("Input wrapped in a label preserve	LOW	LOW	NONE	methods.js
6	CVE-2022-31160	@@ -50,8 +50,7 @@ $.widget("ui.checkboxradio", [$.ui.formReset	LOW	LOW	NONE	checkboxradio.js

complexity	severity	code_before	code_after	cwe_id
41.0	MEDIUM	define(["jquery","./helper","ui/widgets/dialog","ui/effects/effect-blind","ui/efi	define(["jquery","./helper","ui/widgets/dialog","ui/effects/effect-blind","ui/effects,	CWE-79
159.0	MEDIUM	/*! * jQuery UI Dialog @VERSION * http://jqueryui.com * * Copyright jQuery Four	/*! * jQuery UI Dialog @VERSION * http://jqueryui.com * * Copyright jQuery Foundati	CWE-79
43.0	MEDIUM	define(["qunit","jquery","lib/common","lib/helper","ui/position"], function(QU	define(["qunit","jquery","lib/common","lib/helper","ui/position"], function(QUnit, $	CWE-79
102.0	MEDIUM	/*! * jQuery UI Position @VERSION * http://jqueryui.com * * Copyright jQuery Fo	/*! * jQuery UI Position @VERSION * http://jqueryui.com * * Copyright jQuery Founda	CWE-79
19.0	nan	define(["qunit","jquery","lib/helper","ui/widgets/checkboxradio"], function(QU	define(["qunit","jquery","lib/helper","ui/widgets/checkboxradio"], function(QUnit,	CWE-79
15.0	nan	define(["qunit","jquery","lib/helper","ui/widgets/checkboxradio"], function(QU	define(["qunit","jquery","lib/helper","ui/widgets/checkboxradio"], function(QUnit,	CWE-79
54.0	nan	/*! * jQuery UI Checkboxradio @VERSION * http://jqueryui.com * * Copyright jQu	/*! * jQuery UI Checkboxradio @VERSION * http://jqueryui.com * * Copyright jQuery I	CWE-79
126.0	MEDIUM	/*! * jQuery UI Dialog @VERSION * http://jqueryui.com * * Copyright 2012 jQuery	/*! * jQuery UI Dialog @VERSION * http://jqueryui.com * * Copyright 2012 jQuery Foui	CWE-79
22.0	MEDIUM	(function($) {module("tooltip: options");test("disabled: true", function() {expe	(function($) {module("tooltip: options");test("disabled: true", function() {expect(1	CWE-79
79.0	MEDIUM	/*! * jQuery UI Tooltip @VERSION * http://jqueryui.com * * Copyright 2012 jQuery	/*! * jQuery UI Tooltip @VERSION * http://jqueryui.com * * Copyright 2012 jQuery Fou	CWE-79

Ilustración 3 *CSV generado con el dataset CVEFixes.*

Fase 4

En esta fase se desarrolló el modelo de Machine Learning para usarse en la solución cognitiva. Para ello, se desarrolló diferentes modelos que pasaron por una evaluación que nos permitió determinar de manera objetiva que modelo es el más eficiente y eficaz. Dicho proceso se evidencia en el desarrollo a explicar en CRISP-DM

Fase 5

Por último, se realizó la implementación del modelo y la conexión de este con la herramienta. Para lograrlo, primero se programó la extensión para un entorno de desarrollo integrado (IDE), en este caso se escogió Visual Studio Code con su API para extensiones, usando específicamente el *CodeAction* y sus funciones, el cual permite realizar cambios en el código en tiempo real, ya sea para arreglar una línea de código o refactorizarlo. Además de este, también se utilizó el *Diagnostic* el cual sirve para crear nuevos diagnósticos con su severidad, la cual puede ser de error o advertencia; su nombre y descripción a mostrar en tiempo real.[23]

Después de tenerla implementada, se desarrolló un API para lograr tener una conexión de esta con el modelo. Finalmente, se realizó las pruebas de la integración con la extensión en el IDE escogido.

CRISP-DM

CRISP-DM (Cross-Industry Estándar Process for Data Mining), es la guía de referencia más ampliamente utilizada en el desarrollo de proyectos de *Data Mining*.

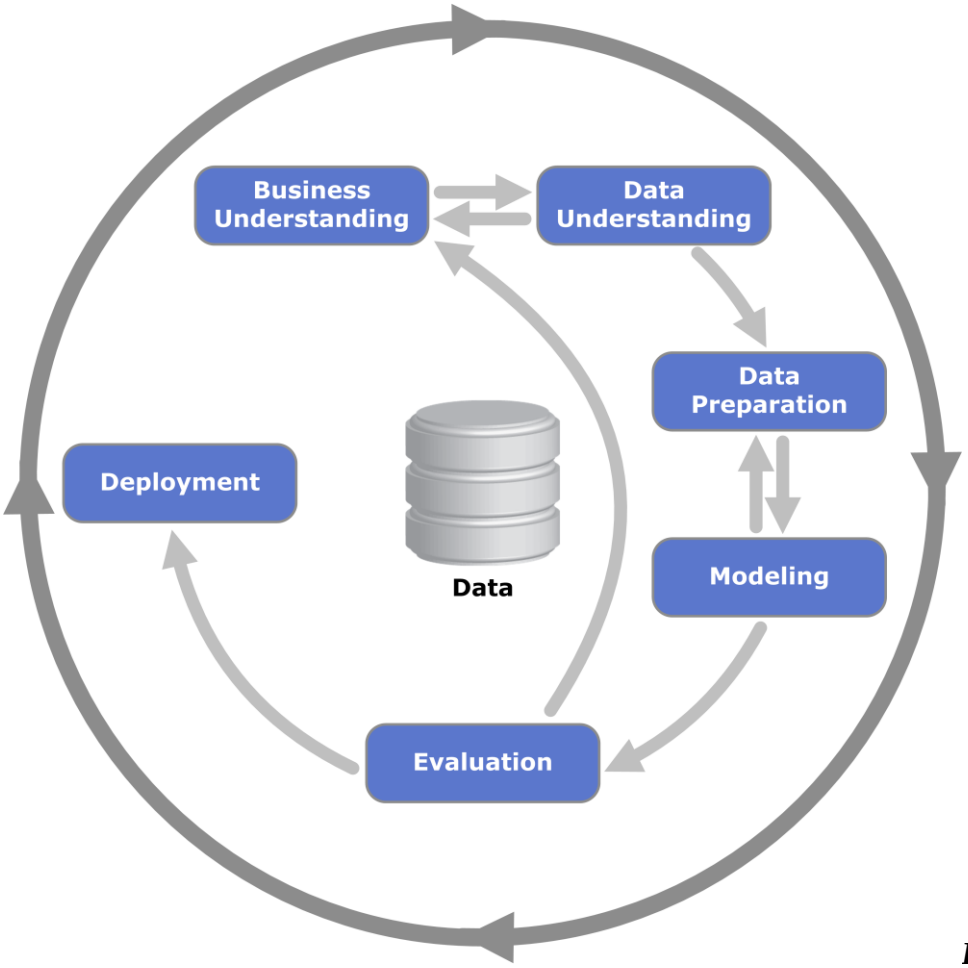


Ilustración 4

Diagrama del proceso de CRISP-DM

Fuente: https://www.datascience-pm.com/crisp-dm-2/

Entendimiento del Negocio

En esta fase, se realizó un estudio previo basado en las investigaciones presentadas en el estado del arte, el cual contiene 6 artículos semejantes o relacionados a nuestro tema de investigación y cuyo objetivo es mostrar el valor presente del proyecto. Adicionalmente, un

marco teórico basado en tres pilares importantes para nuestro proyecto: ciberseguridad, inteligencia artificial y desarrollo de software.

Entendimiento de los Datos

En esta sección se realizó la recolección de los datos necesarios para lograr en su totalidad los objetivos de nuestro proyecto, y así continuar con la identificación de problemas que pudiera tener y hacer el tratamiento respectivo para ser usados de forma eficiente por la solución propuesta.

A partir de eso, se indagó sobre cómo se realizaban cada uno de los ataques de las 10 vulnerabilidades en OWASP para poder comprender como se comportaba cada vulnerabilidad y así mismo, entender que significaban los datos que se iban a buscar y como clasificarlos.

Para ello, se recurrió a varias búsquedas de conjuntos de datos disponibles, entre las cuales está el primer dataset encontrado que fue desarrollado en la Universidad de Lisbon, Portugal. Ellos realizaron un *scrapping* de repositorios en GitHub open-source y almacenaron todos los códigos, con o sin vulnerabilidades; junto con su id, código inseguro, código corregido, fecha, lenguaje, puntaje y nivel de gravedad. [24]

Debido a que las columnas del código están encriptadas con SHA, para nuestro proyecto se tendrían que desencriptar, y la única forma seria a través del método de fuerza bruta, lo que podría resultar en añadir más complejidad a nuestra solución. Además de esto, no posee muchos

datos de archivos en JavaScript, que es actualmente el lenguaje destacado para el Front-End.

El siguiente dataset encontrado fue desarrollado por *Simula Research Laboratory* en Noruega, además, es financiado por el consejo de investigación de Noruega. Extrae los repositorios open-source de Github y utiliza la base de datos *U.S. National Vulnerability Database* (NVD) para recolectar los CVE. Se terminó escogiendo este dataset debido a su compatibilidad con la predicción a través de sus métricas y la completitud de dichas para poder entrenar el modelo con *machine learning*. [25]

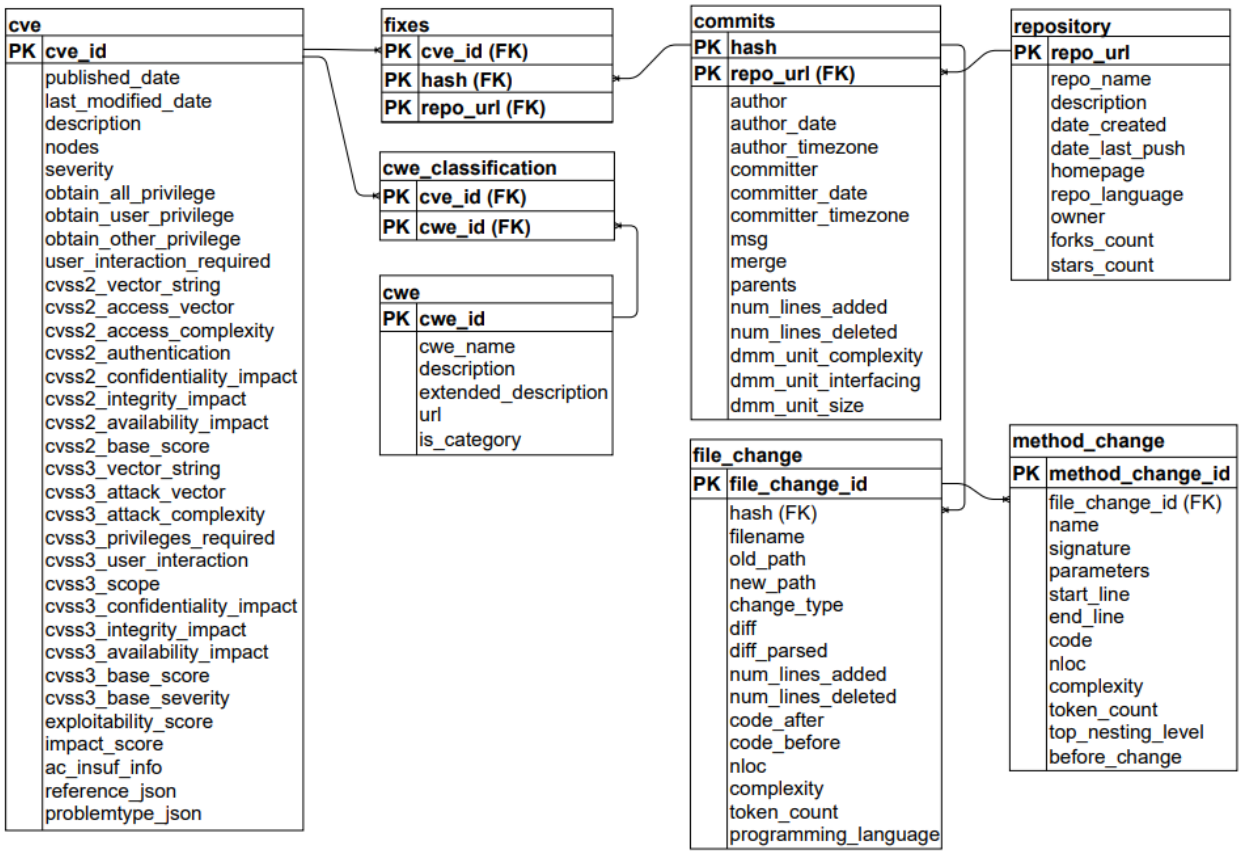


Ilustración 5 *Diagrama de entidad-relación del dataset (pág.5) [25]*

Finalmente, con el conjunto de datos ya escogido, se procedió a la comprensión de los datos de este, en el cual se observó que habían aproximadamente ≈ 10.000 datos extraídos de JavaScript, también se seleccionó que atributos se utilizarían finalmente en el entrenamiento. Las variables presentes en el dataset son:

1. **Cwe_id:** identificador para cada CWE.
2. **Cwe_name:** nombre de cada vulnerabilidad encontrada.
3. **Code_before:** código antes de ser corregido, hace referencia al código aún vulnerable.
4. **Code_after:** código después del commit corrigiéndolo, ya se añadiendo o eliminando.
5. **Filename:** nombre del archivo.
6. **Complexity:** complejidad de la vulnerabilidad dada en números.
7. **Severity:** gravedad de la vulnerabilidad, puede ser alta/media/baja.
8. **Cve_id:** identificador para cada CVE.
9. **Diff:** diferencia entre los dos commits, antes y después, en formato Git.
10. **Diff_parsed:** diccionario de líneas añadidas, se separa en líneas añadidas y líneas eliminadas.
11. **Cvss3_confidentiality_impact:** impacto en la confidencialidad en una vulnerabilidad, puede ser alta/baja/ninguna.
12. **Cvss3_integrity_impact:** impacto en la integridad en una vulnerabilidad, puede ser alta/baja/ninguna.
13. **Cvss3_availability_impact:** impacto en la disponibilidad en una vulnerabilidad, puede ser alta/baja/ninguna.

Preparación para los Datos

En esta fase, se seleccionó los datos de proyectos relacionados con la detección de vulnerabilidades. Se realizó una limpieza y exploración de estos para:

- Eliminar datos NaN y None.
- Cambiar el tipo de variable de algunas columnas (Ej: Object, Int64).
- Eliminar datos atípicos.
- Seleccionar y determinar los atributos más característicos dentro de los datos.
- Realizar una limpieza de caracteres especiales, espacios o comentarios de código con la expresión regular (regEx).
- Vectorizar los datos de texto en números.

Modelado

En esta fase de la metodología, se realizó un conjunto de modelos construido a partir de los siguientes algoritmos clasificatorios de *Machine Learning*:

- Random Forest Classifier.
- Support Vector Machine.
- Gaussian Naive Bayes.
- Stochastic Gradient Descent Classifier.
- Deep Learning.

Las variables más representativas escogidas fueron **code_before** y **cwe_name**, a esta última se le realizó la conversión con un *label encoder* para tomar los valores existentes en dicha columna, y reemplazarlos por números.

En el atributo code_before se utilizó la técnica TF-IDF, la cual consiste en colocar un valor numérico que expresa cuán relevante es una palabra dentro del documento. Dicho número aumenta de acuerdo con la cantidad de veces que una palabra aparece en el texto.

$$TF(i,j) = \frac{\text{Term i frequency in document j}}{\text{Total words in document j}}$$

$$IDF(i) = \log_2 \left(\frac{\text{Total documents}}{\text{documents with term i}} \right)$$

Ilustración 6 *Fórmulas de TF-IDF*

Fuente: https://mungingdata.files.wordpress.com/2017/11/equation.png?w=430&h=336

A su vez, se realizó un *Hold-out* de los datos para tener un 70% de ellos para entrenar, y un 30% restante para probar el modelo.

Evaluación

En este punto de la metodología, se realizó la evaluación del modelo para determinar que cumpla con los objetivos que tiene la solución que se propuso.

Se tuvieron en cuenta las siguientes métricas para evaluarlo:

- Matriz de confusión
- Precisión
- Recall
- F1-Score
- Accuracy
- Support
- Kappa

Despliegue

En esta última fase de la metodología, se realizó el despliegue del modelo para la solución que se propuso. Adicionalmente, se hizo la inclusión de dicho modelo, en un script que recibe los datos del código y da respuesta con el archivo actual abierto en Visual Studio Code.

ANÁLISIS DE RIESGOS Y LIMITACIONES

Se han identificado los siguientes riesgos que pueden afectar el proyecto y su ejecución.

Tabla 2 *Tabla de riesgos, limitaciones y contingencias identificadas para el proyecto*

Riesgos	Limitaciones	Contingencias
El dataset quede desactualizado	Retraso en el cronograma.	Utilizar los datos lo más recientes posibles a la fecha actual, y que contengan las vulnerabilidades actuales.
El dataset mismo tenga vulnerabilidades en sus datos	Retraso en el cronograma al tener que volver a iniciar la búsqueda de datos y su preparación.	Revisar con cuidado que dataset se está encontrando, donde y/o de quien. Además, hacer una preparación detallada de los datos para identificar alguna anomalía.
No se encuentren dataset	Retraso al comienzo de la búsqueda y análisis de datasets disponibles	Ir buscando previamente datos o formas de extraer dichos datos desde páginas o estudios previos de otras personas.
Los recursos computacionales no den abasto	Complejidad en probar la extensión propia y verificar las funcionalidades de dicha.	Tratar de revisar la complejidad que tenga el código y como ir disminuyéndolo.
El diseño de las pruebas no esté bien estructurado	Retraso el cronograma al momento de probar la	Generar pruebas de valor y con casos críticos, puntuales y significativos en como

	aplicación y realizar el despliegue final.	reaccionaria la aplicación a diferentes casos.
No se seleccionó bien el modelo al momento de hacer la evaluación de estos	Retrasa el cronograma en el análisis del dataset.	Elegir varios modelos que hayan tenido un buen resultado en la evaluación y limpieza, o modelos que ya hayan sido previamente usados en otros proyectos de detección.
Falta de conocimientos para desplegar y conectar la extensión con el modelo	Retraso en el cronograma en la construcción y conexión para el despliegue.	Ir adelantando continuamente la estructura básica de la extensión e investigar como conectar ambos lenguajes.

EXPERIMENTOS Y RESULTADOS

Aportes relacionados con el objetivo del proyecto

La idea de realizar este proyecto surge a partir de la necesidad de reducir los tiempos en el desarrollo de aplicaciones web, en donde se invierte una gran cantidad de tiempo a la revisión y corrección de vulnerabilidades en términos de ciberseguridad. Se contribuye con una herramienta hibrida entre análisis estático y dinámico, con un alto enfoque en este último; que permite analizar el código de un proyecto en tiempo real a medida que es implementado, mostrando posibles vulnerabilidades y ofreciendo posibles soluciones.

Aportes relacionados con el desarrollo de capacidades del investigador

Este proyecto nos permitió como investigadores e Ingenieros de Software poner en práctica los conocimientos relacionados con la ciencia de datos a través de la metodología CRISP DM y del proceso de desarrollo de software por medio del ciclo de vida incremental; aprendidos durante nuestra formación profesional.

Tecnologías utilizadas

- Front-End

1. TypeScript

2. VSCode Api

3. JavaScript

- Back-End

  Para el desarrollo del Back de la solución cognitiva, se utilizó Flask, el cual es un *framework* que permite crear aplicaciones web a través de métodos REST y RESTful. Además, se realizó un script en Python para la transformación de las líneas de código en el archivo a revisar y el posterior uso del modelo de IA.

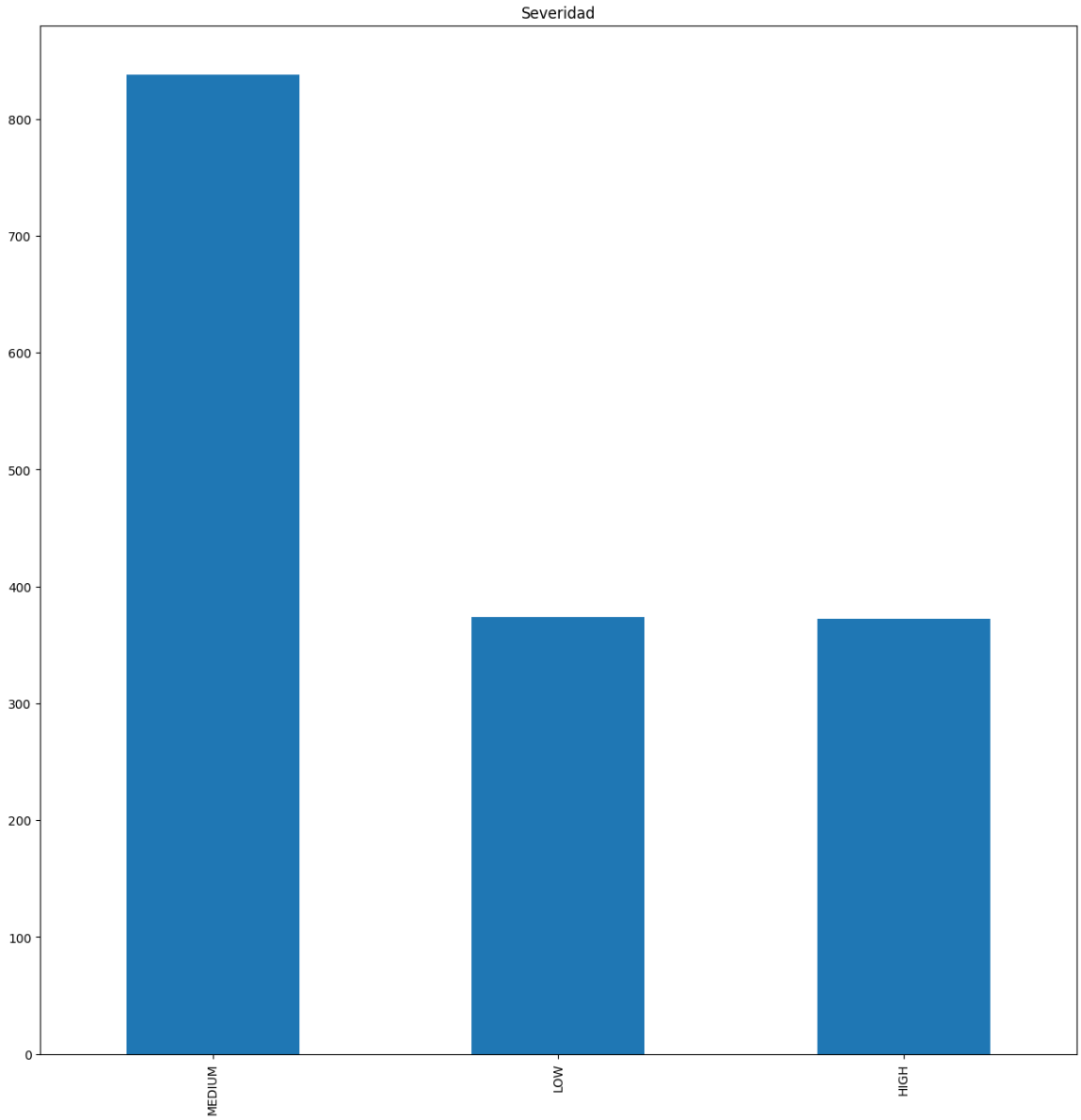
- Analítica de datos

  Para el modelo se usó Python3 junto con la herramienta Jupyter Notebook para tener un seguimiento del proceso llevado a cabo durante todas las fases del proyecto. También se utilizaron librerías como: NumPy, Pandas, scikit-learn, Matplotlib, entre otras.

Experimentos

Experimento 1

Primero se revisaron como estaban los datos frente a distintas variables para explorar como estaban distribuidos los datos:



Grafica 1 *Cantidad de severidades con respecto a cada categoría dentro de esta.*

En este experimento la variable objetivo fue **cwe_name**, en donde se tomaron todos los datos que tuvieran más de 40 apariciones en los **code_before**. Luego, se utilizó la técnica de *Hold-Out* con 70-30 para probarlo con los algoritmos de *machine learning* previamente mencionados.

Los resultados de cada algoritmo fueron los siguientes:

Tabla 3 *Accuracy en cada algoritmo implementado en el experimento 1.*

Modelo	Accuracy
RandomForest	31.1%
SVC	65.8%
Naive-Bayes	55%
Deep Learning	3%
SGD	63.5%

Con este experimento se encontró que el *accuracy* era bastante bajo, por lo que se comenzó a plantear si había algún desbalance entre la cantidad de vulnerabilidades por tipo, donde se denotó que la vulnerabilidad Cross-Site Scripting tiene la mayoría de los datos (573) frente algunas otras vulnerabilidades que únicamente tenían alrededor de 40 o menos, por lo que se decidió recortar más la cantidad de datos para ayudar a mejorar el balance entre estos.

Para revisar que no ocurra dicho desbalanceo se implementó la métrica Cohen Kappa que revisa la fiabilidad y precisión entre dos evaluadores (Jueces-Observadores).

Experimento 2

Teniendo en cuenta el experimento 1, se redujo entonces la cantidad mínima de datos por cada tipo de vulnerabilidad a 60.

Los resultados de cada algoritmo fueron los siguientes:

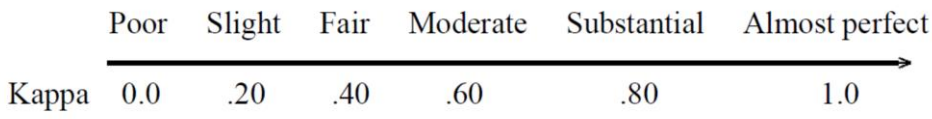
Tabla 4 *Accuracy y Kappa en cada algoritmo implementado en el experimento 2.*

Modelo	Accuracy	Kappa
RandomForest	30.12%	0.3%
SVC	82%	56.25%
Naive-Bayes	72.2%	42.38%
Deep Learning	69%	N/A
SGD	80%	52.74%

Los dos mejores modelos en este experimento fueron SVC y SDG, con un accuracy de 82% y 80% respectivamente. El SVC con unos parámetros de random_state= 22 y shuffle= True y el SGD con parámetros adicionales de loss="hinge" y penalty="l2". Dichas configuraciones son propias de cada algoritmo o hacen parte de los parámetros propios de Sklearn para realizar la clasificación.

El Kappa del SVC fue de 56% y de SGD fue de 52%. La siguiente tabla representa la interpretación de los valores que arroja esta métrica:

Ilustración 7 *Interpretación del puntaje de Cohen Kappa*



Kappa	Agreement
< 0	Less than chance agreement
0.01–0.20	Slight agreement
0.21– 0.40	Fair agreement
0.41–0.60	Moderate agreement
0.61–0.80	Substantial agreement
0.81–0.99	Almost perfect agreement

De acuerdo con dicha ilustración, ambos algoritmos obtuvieron una concordancia moderada, siendo 40% para arriba un buen porcentaje de concordancia. Por lo cual en este experimento hubo una mejoría considerable frente al *accuracy* y kappa.

Con los mejores dos modelos encontrados, se procedió a revisar en detalle sus resultados y sus gráficas.

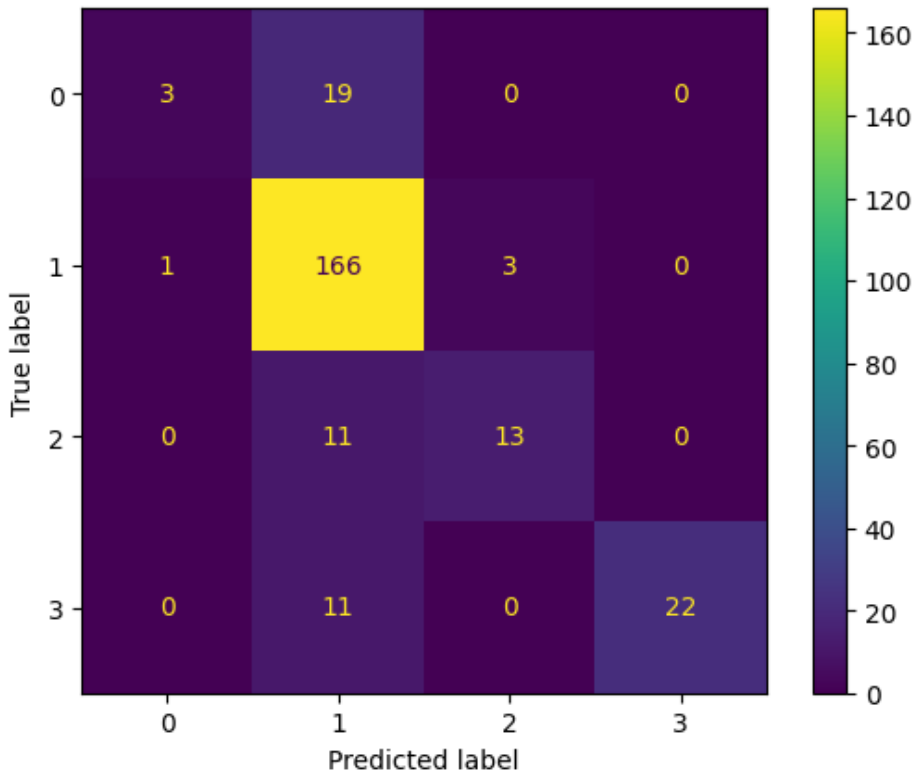
```
[[  3  19   0   0]
 [  1 166   3   0]
 [  0  11  13   0]
 [  0  11   0  22]]
              precision    recall  f1-score   support

           0       0.75      0.14      0.23        22
           1       0.80      0.98      0.88       170
           2       0.81      0.54      0.65        24
           3       1.00      0.67      0.80        33

    accuracy                           0.82       249
   macro avg       0.84      0.58      0.64       249
weighted avg       0.82      0.82      0.79       249

Accuracy: 0.8192771084337349
Kappa:
```

```
0.5625268418381291
```

Grafica 2 *Métricas del Support Vector Machine Classifier en el Experimento 2.*

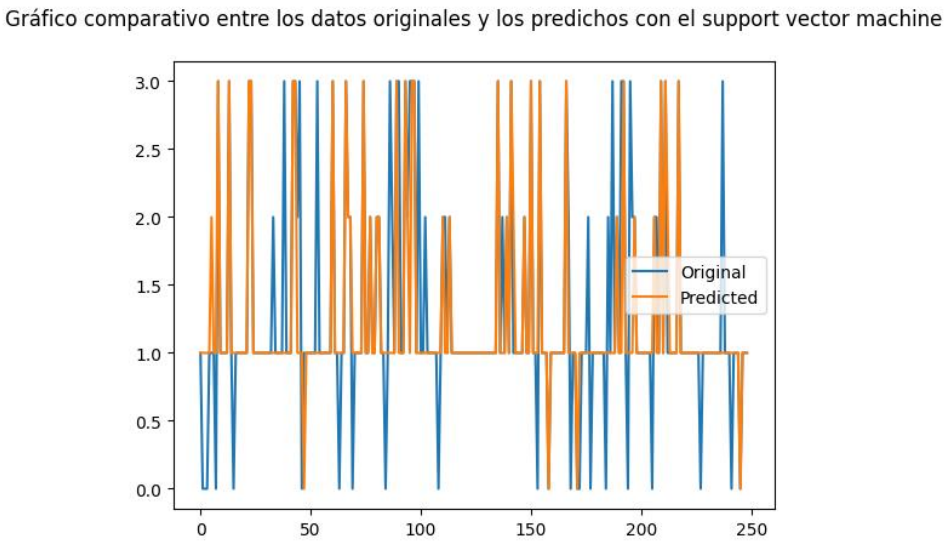
Como se evidencia en la gráfica 6, las otras métricas (precision, recall, f1-score y support) también tuvieron buenos porcentajes.

También se volvió a confirmar que debido a la gran mayoría de datos siendo de Cross-Site y se puede evidenciar a continuación:



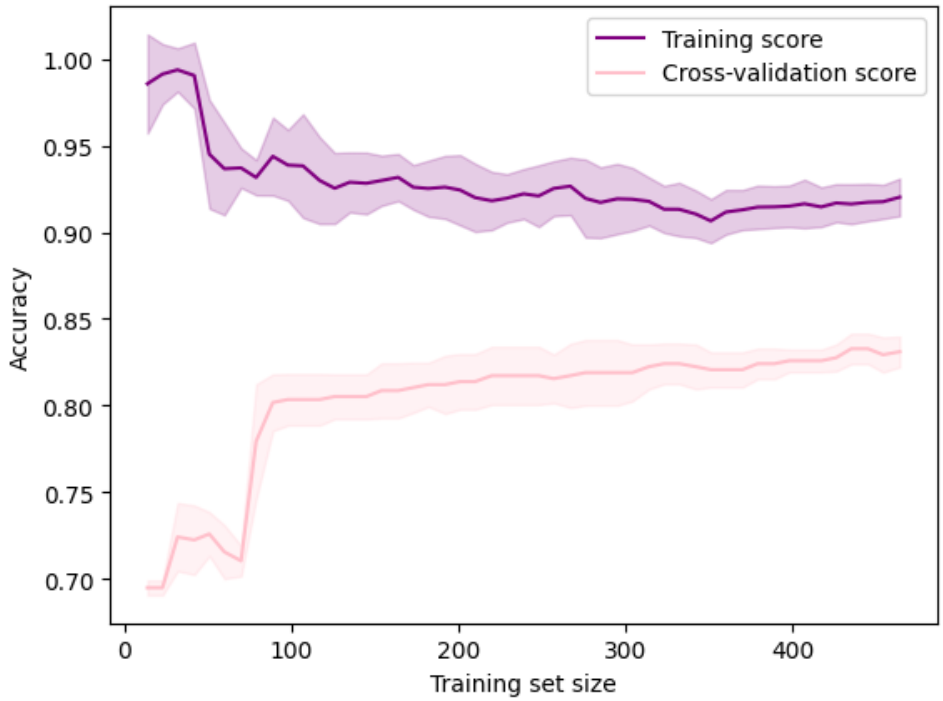
***Grafica** 3 Matriz de confusión Support Vector Machine Classifier en el Experimento 2.*

Se revisó a través de una gráfica que tan acertados eran los resultados con respecto a los datos originales:



***Grafica** 4 Gráfico comparativo de los datos originales con los predichos con Support Vector Machine en el experimento 2.*

Otro acercamiento para revisar el *accuracy* fue con el learning curve comparando igualmente los puntajes del training con los de prueba.



Grafica 5. *Learning curve del experimento 2.*

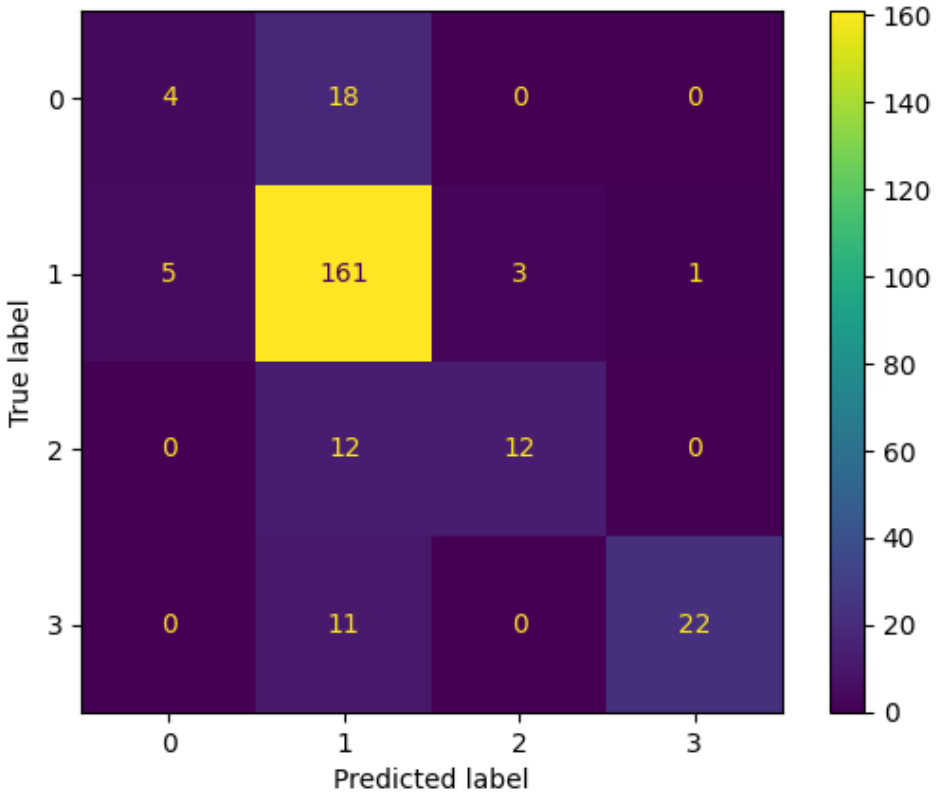
Con respecto al SGD las métricas obtenidas fueron las siguientes:

```
[[  4  18   0   0]
 [  5 161   3   1]
 [  0  12  12   0]
 [  0  11   0  22]]
              precision    recall  f1-score   support

           0       0.44      0.18      0.26        22
           1       0.80      0.95      0.87       170
           2       0.80      0.50      0.62        24
           3       0.96      0.67      0.79        33

    accuracy                           0.80       249
   macro avg       0.75      0.57      0.63       249
weighted avg       0.79      0.80      0.78       249

0.7991967871485943
Kappa:
```

```
0.5274066201032492
```

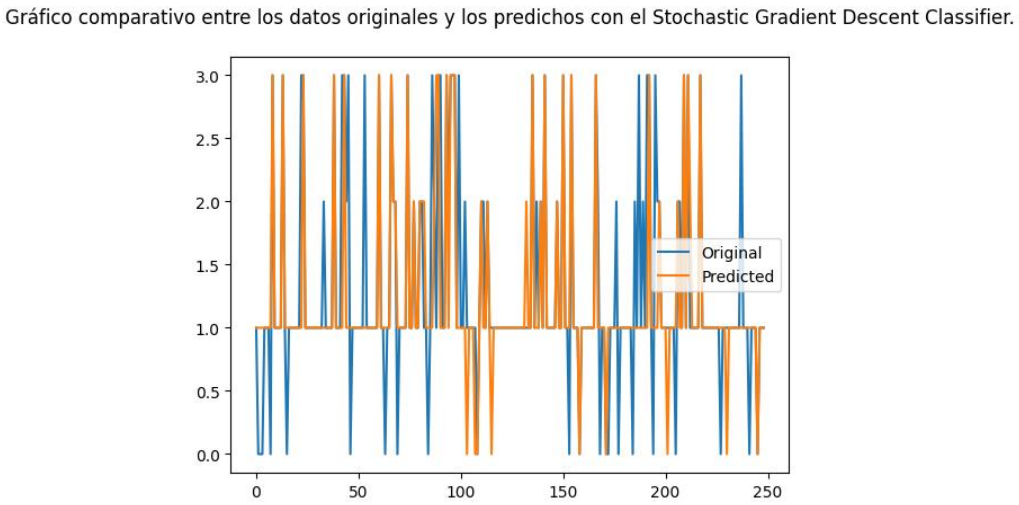
Grafica 6 *Métricas del Stochastic Gradient Descent Classifier en el Experimento 2.*

La matriz de confusión disminuye ligeramente con respecto al otro algoritmo:



Grafica 7 *Matriz de confusión Stochastic Gradient Descent Classifier en el Experimento 2.*

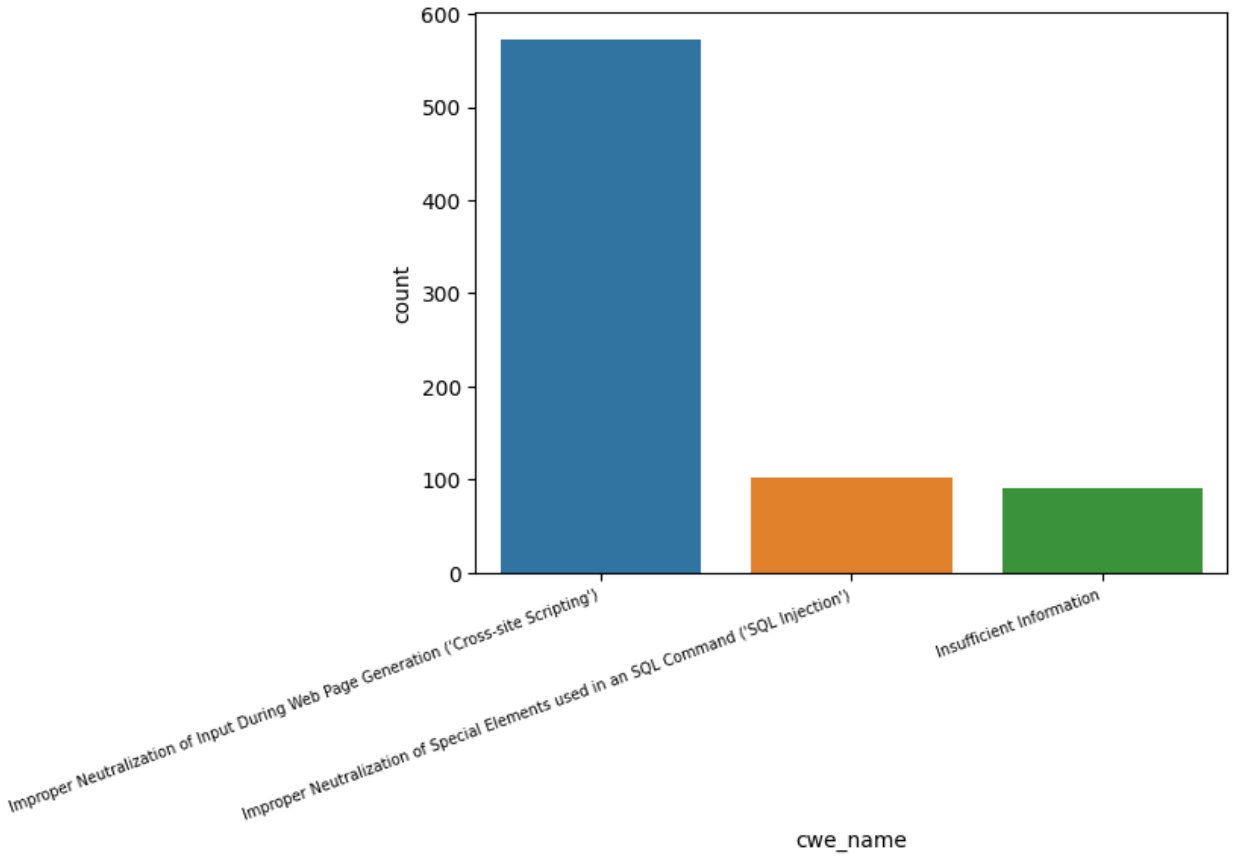
Finalmente, la gráfica de comparación entre los datos originales y los predichos también da muy buenos resultados, similar al SVC, empeorando tan solo levemente.



Grafica 8 *Gráfico comparativo de los datos originales con los predichos con Stochastic Gradient Descent Classifier en el experimento 2.*

Experimento 3

Con los resultados anteriormente evidenciados, se llegó a la conclusión de que, debido a la poca cantidad de datos para los otros tipos de vulnerabilidades, el modelo perdía precisión y *accuracy*. Por lo que en el experimento 3 se redujo a solamente identificar 3 vulnerabilidades. Las vulnerabilidades escogidas fueron:



Grafica 9 *Nombre de las vulnerabilidades escogidas y la cantidad de datos que tiene cada una.*

Si bien Cross-site Scripting sigue siendo la vulnerabilidad con mayor número de datos, SQL Injection e Insufficient Information eran los siguientes con mayor cantidad de datos, aproximadamente 100 por cada uno, por lo cual también fueron tomados en cuenta para realizar el modelo.

Los resultados de cada algoritmo fueron:

Tabla 5 *Accuracy y Kappa en cada algoritmo implementado en el experimento 3.*

Modelo	**Accuracy**	**Kappa**
RandomForest	69%	0.4%

SVC	89.13%	71.55%
Naive-Bayes	85.65%	60.34%
Deep Learning	76.07%	N/A
SGD	86.52%	65.63%

Como se puede evidenciar, solo con 3 tipos de vulnerabilidades, todos los algoritmos mejoraron significativamente. A su vez, SVC y SGD se mantienen como los mejores algoritmos para este modelo con accuracy de 89% y 86% respectivamente.

Los resultados de las métricas del SVC son los siguientes:

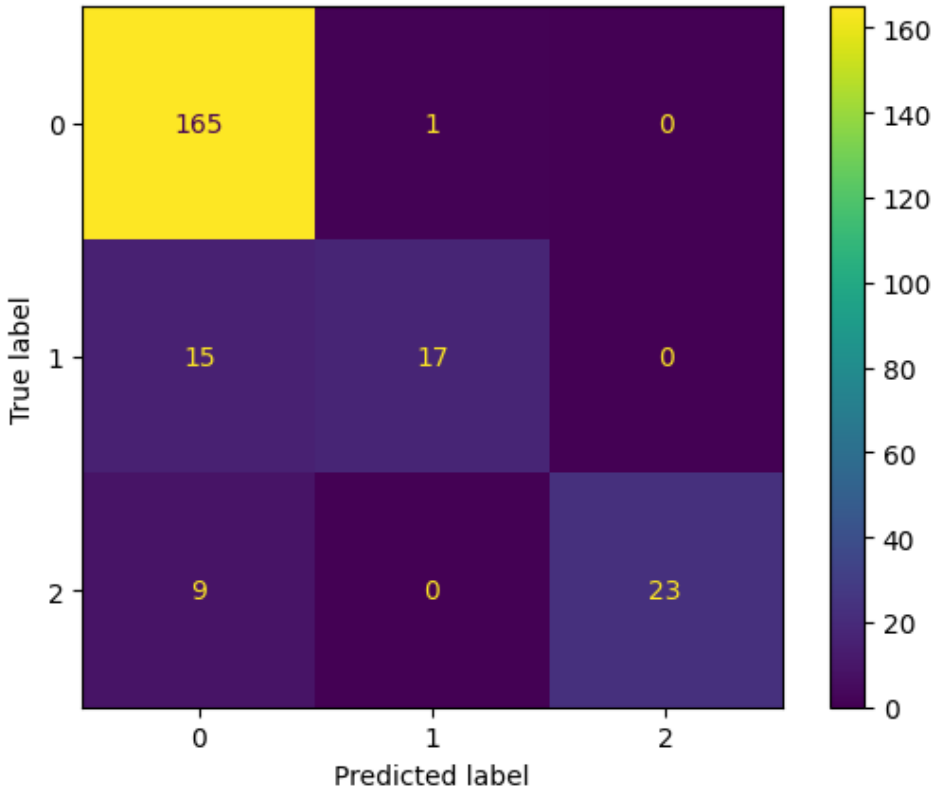
```
[[165   1   0]
 [ 15  17   0]
 [  9   0  23]]
              precision    recall  f1-score   support

           0       0.87      0.99      0.93       166
           1       0.94      0.53      0.68        32
           2       1.00      0.72      0.84        32

    accuracy                           0.89       230
   macro avg       0.94      0.75      0.82       230
weighted avg       0.90      0.89      0.88       230

Accuracy: 0.8913043478260869
Kappa:
```

```
0.7155436825962205
```

Grafica 10 *Métricas del Support Vector Machine Classifier en el Experimento 3.*

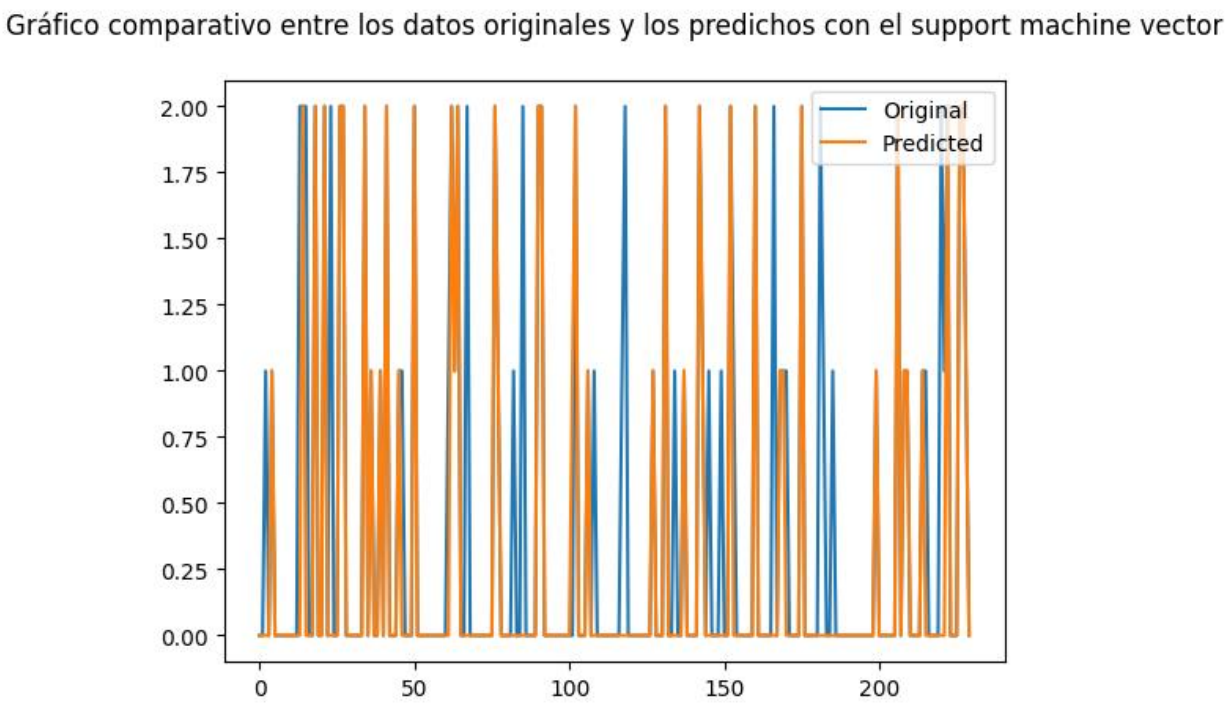
Como se puede observar en la gráfica las demás métricas son muy buenas y los resultados son en su gran mayoría cercanos al 1.0.

También en la matriz de confusión de este experimento se obtuvieron más aciertos y menos clasificaciones erróneas, como se puede evidenciar en el siguiente gráfico:



Grafica 11 *Matriz de confusión Support Vector Machine Classifier en el Experimento 3.*

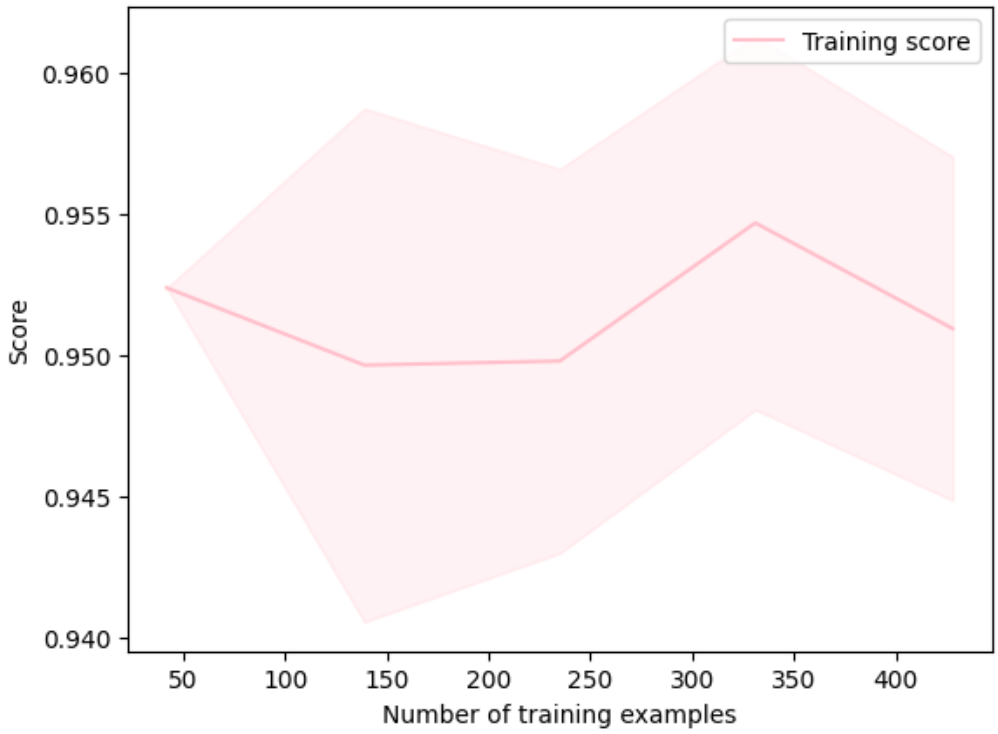
El modelo también mejoró en sus predicciones con respecto a los datos originales.



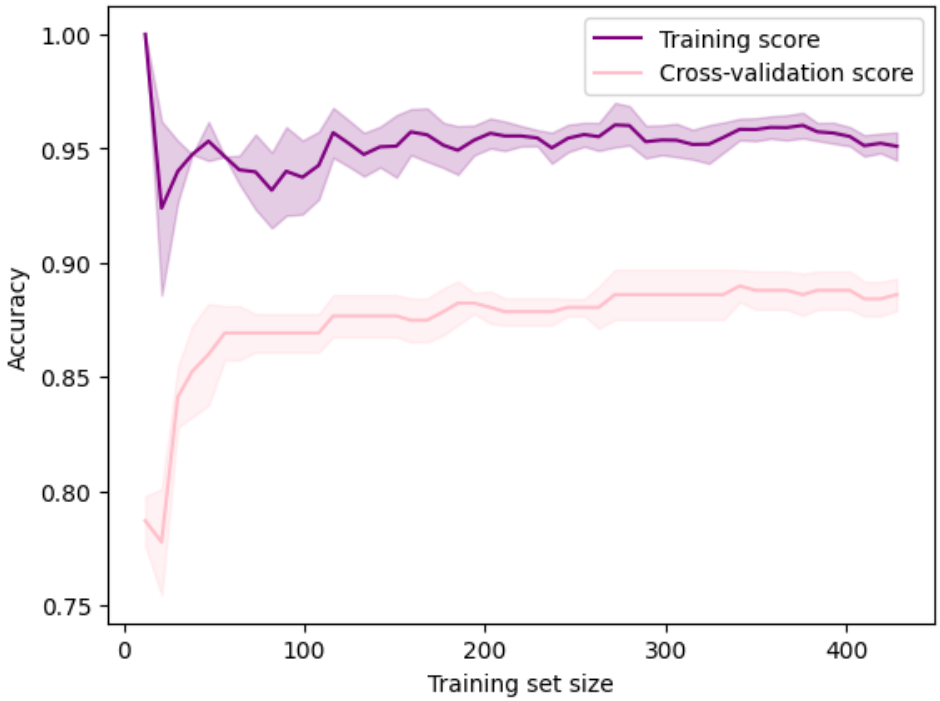
Grafica 12 *Gráfico comparativo de los datos originales con los predichos con Support Vector Machine en el experimento 3.*

Aquí la gran mayoría de los datos arrojaron los mismos resultados que los originales, por lo que con este experimento se obtuvo el mejor modelo para predecir vulnerabilidades.

Adicionalmente se realizó otro learning curve para este experimento, donde se revisó como cambiaba el puntaje de acuerdo con la cantidad de datos para entrenar.



Grafica 13 *Gráfica del learning curve del experimento 3.*



***Grafica** 14 Gráfica del learning curve con el puntaje del training y el puntaje de la prueba del experimento 3.*

Aquí se puede visualizar que el pico del puntaje en el modelo se alcanza con aproximadamente 300-350 datos separados para entrenar.

Resultados

A continuación, se muestra una tabla comparativa entre los 3 experimentos realizados, solamente con los dos mejores algoritmos.

***Tabla 6** Comparación con los dos algoritmos frente a todos los experimentos realizados.*

Número de experimento	SVM	SGD
1	65.8%	63.5%
2	82%	80%
3	89.13%	86.52%

Con base en los resultados mostrados en la tabla, por su alto porcentaje de *accuracy* se escogió utilizar el algoritmo SVM para desplegar el modelo y ser usado en el Script.

La extensión entonces retorna el *path* al API, junto con la información que tiene adentro del archivo actual abierto en Visual Studio Code.

```
Received data: c:\Users\Ferna\Downloads\prueba-pdg\prueba.js
The data: /*jshint node:true*/
'use strict';

var express = require('express');
var app = express();
// app.disable('x-powered-by');
var bodyParser = require('body-parser');
var cookieParser = require('cookie-parser');
var favicon = require('serve-favicon');
var logger = require('morgan');
var port = process.env.PORT || 8001;
var four0four = require('./utils/404')();

var environment = process.env.NODE_ENV;

app.use(favicon(__dirname + '/favicon.ico'));
app.use(bodyParser.urlencoded({extended: true}));
app.use(bodyParser.json());
app.use(cookieParser());
app.use(logger('dev'));

// app.use(function(req, res, next) {
//     res.setHeader('Content-Security-Policy', 'script-src \'self\' ajax.googleapis.com');
//     return next();
// });

app.use('/api', require('./routes'));

console.log('About to crank up node');
console.log('PORT=' + port);
console.log('NODE_ENV=' + environment);

switch (environment){
    case 'build':
        console.log('** BUILD **');
```

Ilustración 8 *Ejemplo de código siendo entregado al API junto con toda su información.*

A partir de esto, se realizó una tabla comparativa de analisis de proyectos con vulnerabilidades que puede ser encontrada en la carpeta:

https://github.com/i2tResearch/Ciberseguridad_web/blob/master/Cognitive%20Solution%20for%20Vulnerability%20Detection%20on%20Front-End/Aplicaciones%20Vulnerables.docx

ENTREGABLES

Tabla 7 *Entregables esperados para entregar cada objetivo específico.*

	Entregables
1. Evaluar un modelo para detectar las vulnerabilidades en aplicaciones web.	• Modelo Seleccionado. • Dataset con los datos seleccionados, limpios, eliminar datos atípicos.
2. Implementar la solución cognitiva a través de un entorno de desarrollo integrado.	• Solución Cognitiva. • Estructura y conexión con la extensión. • Código de preprocesamiento de la información. • Código del entrenamiento de los modelos y evaluación.
3. Revisar un conjunto de proyectos con vulnerabilidades para futuras validaciones de las predicciones de la extensión.	- Proyecto de Grado. - Análisis de proyectos con vulnerabilidades

CONCLUSIONES

- En el experimento 1, la predicción empeora a mayor cantidad de tipos de vulnerabilidades debido a la dispersión y poca frecuencia de algunas de ellas en casos reales. Mientras que el experimento 2 se pudo corregir el desbalance de los datos; validado a partir de la implementación de la métrica Cohen Kappa que permitió revisar la fidelidad y precisión de los mismos.

- En el experimento 3 usando SVM y un conjunto de datos que poseía mayor cantidad de datos en Cross-site, el modelo predijo con una precisión del 89% a todos los tipos de vulnerabilidades propuestos.

- Con un modelo de inteligencia artificial es posible realizar una solución cognitiva que a través de un análisis dinámico identifique que tipos de vulnerabilidades existen en el código.

- El algoritmo de SVM, con un total de 230 datos de pruebas de 3 tipos de vulnerabilidades, obtuvo 205 clasificadas correctamente, donde 165 fueron de Cross-Site, 17 de SQL Injection y 23 de Insufficient Information; mejorando la predicción del modelo enfocándose en estas, siendo Cross-Site la más popular actualmente

- A través de una extensión implementada en Visual Studio Code, es posible mejorar la calidad del software con la metodología DevSecOps al realizar analisis en tiempo real para prevenir problemas de seguridad antes de llegar a producción.

TRABAJO A FUTURO

Como trabajo a futuro se tiene:

- Generalizar la solución para que detecte vulnerabilidades no solo en JavaScript, sino en cualquier otro lenguaje de programación.

- Añadir más vulnerabilidades que hagan enfoque al top 10 de OWASP, y después ampliarlo más allá de solamente OWASP.

- Mejorar la precisión y el kappa de los modelos actuales para que resulte en resultados más precisos y confiables.

- Probar la extensión fuera del ámbito académico y evaluar el recibimiento por parte de los usuarios y el feedback para postular una posible iteración de mejora para la extensión.

- Buscar más datos para las otras vulnerabilidades y construir un modelo que soporte la clasificación de las 10 vulnerabilidades de OWASP, buscando un balanceo equitativo y óptimo entre todos los datos de cada uno.

- Desplegar en servidores más robustos el API y la extensión para que puedan ser accesibles a un mayor número de personas.

ANEXOS

Figura 3 *Análisis de Participación*

Beneficiarios Directos	**Beneficiarios Indirectos**	**Neutrales**	**Oponentes**
Desarrolladores Web Front-End	Personas que navegan a través de las aplicaciones web	Programadores no interesados en la seguridad de su código	Cibercriminales
Programadores de Ciberseguridad	Dueños de empresas que les interesa tener su organización cibersegura		Empresas con desconocimiento sobre la importancia de la inversión en ciberseguridad

Figura 4 *Cronograma de Actividades*

Nombre de tarea	Comienzo real	Fin real	Duración real	Predecesoras
Proyecto de grado	**dom 28/08/22**	**NOD**	**96 días**	
Anteproyecto	**dom 28/08/22**	**NOD**	**66 días**	
Primera entrega	**dom 28/08/22**	**NOD**	**15 días**	
Creacion del arbol del problema	dom 28/08/22	vie 2/09/22	5 días	
Creacion del arbol de objetivos	vie 2/09/22	vie 2/09/22	1 día	3
Analisis de participación	sáb 3/09/22	NOD	0 días	
Escribir motivación y antecedentes	lun 5/09/22	vie 9/09/22	5 días	5
Formulación del problema	sáb 10/09/22	vie 16/09/22	19,75 días	6
Formulación de los objetivos	mié 14/09/22	vie 16/09/22	8,75 días	7
Segunda entrega	**lun 19/09/22**	**vie 28/10/22**	**30 días**	
Estado del arte	lun 19/09/22	vie 14/10/22	20 días	8
Marco teorico	lun 19/09/22	vie 28/10/22	30 días	10CC
Tercera entrega	**lun 31/10/22**	**lun 28/11/22**	**21 días**	
Selección de metodología	lun 31/10/22	lun 28/11/22	21 días	11
Definición de entregables	lun 31/10/22	vie 18/11/22	15 días	13CC
Definición de analisis de riesgos	lun 31/10/22	vie 11/11/22	10 días	13CC

Proyecto de grado II	**vie 14/10/22**	**lun 9/01/23**	**62 días**	
Fase 1	**vie 14/10/22**	**vie 28/10/22**	**11 días**	
Elicitación de requerimientos	vie 14/10/22	sáb 15/10/22	1 día	
Revisión de datasets del estado del arte	mar 18/10/22	vie 28/10/22	9 días	
Dorfman	vie 14/10/22	mar 18/10/22	3 días	18CC
Fase 2	**vie 14/10/22**	**jue 10/11/22**	**20 días**	
Diagrama deployment	vie 14/10/22	mié 26/10/22	9 días	20CC
Casos de uso	jue 27/10/22	mar 1/11/22	4 días	22
Implementación base de la extensión en VSCode	jue 27/10/22	vie 4/11/22	7 días	22
Implementación del API	lun 7/11/22	jue 10/11/22	4 días	24
Fase 3	**vie 11/11/22**	**lun 5/12/22**	**17 días**	
Recopilar datasets	vie 11/11/22	lun 21/11/22	7 días	25
Recopilar modelos	vie 11/11/22	lun 21/11/22	7 días	25
Probar modelos	mar 22/11/22	lun 5/12/22	10 días	28
Fase 4	**mar 6/12/22**	**jue 29/12/22**	**18 días**	
Crear modelo	mar 6/12/22	jue 29/12/22	18 días	27
Entrenar modelo	mar 6/12/22	jue 29/12/22	18 días	31CC
Fase 5	**vie 30/12/22**	**lun 9/01/23**	**7 días**	
Conectar el modelo con la extensión	vie 30/12/22	jue 5/01/23	5 días	32
Probar el modelo	vie 6/01/23	lun 9/01/23	2 días	34

Ver en detalle en:

https://github.com/i2tResearch/Ciberseguridad_web/blob/master/Cognitive%20Solution%20for%20Vulnerability%20Detection%20on%20Front-End/Documentos/cronograma.mpp

REFERENCIAS BIBLIOGRÁFICAS

1. Urcuqui López, C. C., García Peña, M., Osorio Quintero, J. L. y Navarro Cadavid, A. (2018). *Ciberseguridad: un enfoque desde la ciencia de datos.* Cali: Editorial Universidad Icesi. DOI: https://doi.org/10.18046/EUI/ee.4.2018

2. Gillis, A. S. y Botelho, B. (2022). *Cognitive computing*. SearchEnterpriseAI. https://www.techtarget.com/searchenterpriseai/definition/cognitive-computing

3. Abela, R. (2022). *Steam Gaming & Entertainment Platform Vulnerable to Cross-site Scripting Vulnerability*. Invicti. https://www.invicti.com/blog/web-security/steam-entertainment-platform-gaming-vulnerable-xss/

4. Snyk. (2020). *State of Open-Source Security 2022*. https://snyk.io/reports/open-source-security/

5. Sobers, R. (2021). *64% of Americans Don't Know What to Do After a Data Breach—Do You? (Survey).* Varonis. https://www.varonis.com/blog/data-breach-literacy-survey

6. Cyberseek. (s.f). *Cybersecurity Supply And Demand Heat Map.* https://www.cyberseek.org/heatmap.html

7. Morgan, S. (2021). *Cybersecurity Jobs Report: 3.5 Million Openings In 2025*. Cybercrime Magazine. https://cybersecurityventures.com/jobs/

8. Urcuqui López, C.C. y Navarro Cadavid, A. (coords.) (2022). *Ciberseguridad: los datos tienen la respuesta.* Cali: Editorial Universidad Icesi. DOI: https://doi.org/10.18046/EUI/ee.4.2022

9. Brown, S. (2021). *Machine learning, explained.* MIT Sloan. https://mitsloan.mit.edu/ideas-made-to-matter/machine-learning-explained

10. IBM. (s.f) *IBM Cognitive–Get started with cognitive technology*. https://www.ibm.com/watson/advantage-reports/getting-started-cognitive-technology.html

11. Front-End Masters. (2018). *What Is a Front-End Developer?* https://frontendmasters.com/guides/front-end-handbook/2018/what-is-a-FD.html

12. CyberArk Software. (2021). *What is DevOps Security? DevSecOps Definition.* CyberArk. https://www.cyberark.com/what-is/devops-security/

13. Hammoudeh, M. A. A., Alobaid, A., Alwabli, A., & Alabdulmunim, F. (2021). *The Study on Assessment of Security Web Applications.* International Journal of Interactive Mobile Technologies (iJIM), 15(23), pp. 120–135. https://doi.org/10.3991/ijim.v15i23.27357

14. OWASP. (2021). *OWASP Top Ten.* https://owasp.org/www-project-top-ten/

15. OWASP. (2022) *Vulnerabilities.* https://owasp.org/www-community/vulnerabilities/

16. OWASP. (2022) *About the OWASP Foundation*. https://owasp.org/about/

17. Le, H. T., Shar, L. K., Bianculli, D., Briand, L. C., & Nguyen, C. D. (2022). *Automated reverse engineering of role-based access control policies of web applications*. Journal of Systems and Software, 184, 111109. https://doi.org/10.1016/j.jss.2021.111109

18. Yaitskyi, A. (2022). *Emploging and improving machinelearning of detection of Phishing URLs.* DIVA. http://urn.kb.se/resolve?urn=urn:nbn:se:bth-23360

19. Ibéria Medeiros, Nuno Neves, and Miguel Correia. 2016. DEKANT: a static analysis tool that learns to detect web application vulnerabilities. In Proceedings of the 25th International Symposium on Software Testing and Analysis (ISSTA 2016). Association for Computing Machinery, New York, NY, USA, 1–11. https://nebulosa.icesi.edu.co:2144/10.1145/2931037.2931041

20. Luca Demetrio, Andrea Valenza, Gabriele Costa, and Giovanni Lagorio. 2020. WAF-A-MoLE: evading web application firewalls through adversarial machine learning. In Proceedings of the 35th Annual ACM Symposium on Applied Computing (SAC '20). Association for Computing Machinery, New York, NY, USA, 1745–1752. https://nebulosa.icesi.edu.co:2144/10.1145/3341105.3373962

21. Russell, R., Kim, L., Hamilton, L., Lazovich, T., Harer, J., Ozdemir, O., Ellingwood, P., & McConley, M. (2018). Automated vulnerability detection in source code using deep representation learning. *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*. DOI: https://doi.org/10.48550/arXiv.1807.04320

22. Liu, M., Li, K., & Chen, T. (2020). DeepSQLi: deep semantic learning for testing SQL injection. *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*. DOI: https://doi.org/10.48550/arXiv.2005.11728

23. Documentación de Visual Studio Code.

24. Rei, S., & Abreu, R. (2017). *A database of existing vulnerabilities to enable controlled testing studies*. International Journal of Secure Software Engineering (IJSSE), 8(3), 1-23.

25. Bhandari, G., Naseer, A., & Moonen, L. (2021). *CVEfixes: automated collection of vulnerabilities and their fixes from open-source software.* In Proceedings of the 17th International Conference on Predictive Models and Data Analytics in Software Engineering (pp. 30-39).

26. Techopedia. (2018, enero 23). *Data Scraping*. Techopedia.com. https://www.techopedia.com/definition/33132/data-scraping

27. What Is Dynamic Analysis? (2020, July 10). TotalView by Perforce. https://totalview.io/blog/what-dynamic-analysis