

UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE INFORMÁTICA
VALPARAÍSO - CHILE



“CVE-HUNTER: SISTEMA DE DETECCIÓN Y ALERTAS TEMPRANAS DE COMMON VULNERABILITIES AND EXPOSURES (CVE), PARA EVITAR EXPLOTACIÓN DE VULNERABILIDADES EN LAS ORGANIZACIONES.”

CAMILO ANDRÉS DUQUE VILLAR

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN INFORMÁTICA

Profesor Guía: Mauricio Figueroa Colarte
Profesor Correferente: Pedro Godoy Barrera

Julio - 2025



CONSTANCIA DE VALIDACIÓN Y CONFIDENCIALIDAD DE MONOGRAFÍA A REPOSITORIO ACADÉMICO

1.- IDENTIFICACIÓN DEL TRABAJO ACADÉMICO

Tipo de monografía (marcar una opción): ☒ Memoria o trabajo de título; ☐ Tesis de Postgrado;

Título del trabajo: "CVE-HUNTER: Sistema de detección y alertas tempranas de Common Vulnerabilities and Exposures (CVE), para evitar la explotación de vulnerabilidades en las organizaciones"

Nombre del candidato(a): Camilo Andrés Duque Villar

Carrera / Grado: Ingeniería Civil Informática

Campus: Casa Central Valparaíso ; **Departamento:** Informática

2.- VALIDACIÓN DEL PROFESOR GUÍA/DIRECTOR DE TESIS

Yo, Mauricio Figueroa Colarte, en mi calidad de profesor(a) guía/director(a) del trabajo académico mencionado anteriormente **DEJO CONSTANCIA** que:

- He revisado esta versión del documento y corresponde a la versión final aprobada del trabajo.
- El trabajo cumple con los requisitos académicos y de formato establecidos por la institución

3.- EVALUACIÓN DE CONFIDENCIALIDAD POR PROPIEDAD INDUSTRIAL

☒ El trabajo **NO contiene información que amerite confidencialidad** y puede ser publicado de inmediato en repositorio con acceso abierto.

☐ El trabajo **CONTIENE** información con potenciales implicancias de propiedad industrial o intelectual y requiere un periodo de confidencialidad (embargo) por:

☐ 6 meses; ☐ 12 meses; ☐ 2 años; ☐ 3 años; ☐ 5 años; ☐ 10 años

Fundamentación de la necesidad de confidencialidad (obligatorio si se solicita embargo):

4.- FIRMAS

Profesor(a) guía o director(a) de memoria o tesis:

Fecha: ; **Firma:** 

Estudiante o Candidato(a):

Fecha: ; **Firma:** 

Este formulario debe ser insertado como página 2 de la memoria o tesis, completado y firmado por estudiante y profesor(a) antes de la entrega en portal PRISMA de Biblioteca USM.

DEDICATORIA

Esta Tesis se la quiero dedicar a mi familia y amigos, que siempre estuvieron acompañandome en este importante proceso.

AGRADECIMIENTOS

En primer lugar, quiero agradecer a mi madre, quien me apoyó desde el inicio de esta travesía universitaria. Siempre estuvo orgullosa de que siguiera adelante, a pesar de las dificultades que se me presentaban en el camino. También agradezco a mi abuela, que siempre me recordaba la importancia de pensar en positivo, de no agachar la cabeza, de mirar hacia adelante y no ponerme el parche antes de la herida. A mi hermano, quien en uno de los momentos más difíciles de mi paso por la universidad, me brindó su apoyo y me motivó a darlo todo en la cancha.

Quiero dar las gracias a mis amigos y a todas las personas que conocí en este proceso: a los *cauros*, con quienes compartimos tantos momentos, como las pichangas, los partidos, los videojuegos y los carretes. Agradezco especialmente a Javier, Marla y Sebastián, quienes fueron mis consejeros, compañeros de entrenamiento y estudio. También a todos quienes me enseñaron cosas nuevas, como mis compañeros del laboratorio.

Agradezco profundamente a *Mataya* y *Vicente*, quienes me acompañaron en el desarrollo del proyecto, apoyándome en la toma de decisiones, definición de reglas de negocio y aspectos técnicos del desarrollo.

Finalmente, quiero agradecer a *María Paz*, quien me acompañó en este último período, dándome ánimo y su constante apoyo, recordándome que sí iba a terminar esta memoria.

RESUMEN

Resumen— Los riesgos crecientes en ciberseguridad exigen soluciones proactivas para proteger los activos digitales. Se propone **CVE-Hunter**, un sistema de monitoreo pasivo de vulnerabilidades diseñado para automatizar la recolección y notificación de *Common Vulnerabilities and Exposures* (CVE). Su objetivo es asistir a equipos de desarrollo en la identificación temprana de amenazas emergentes. Para su validación, **CVE-Hunter** se evaluó rigurosamente con la empresa *Develrox*. La evaluación implicó el análisis de tecnologías específicas mediante una verificación aleatoria estratificada contra la *National Vulnerability Database* (NVD). Los resultados fueron positivos: el sistema detectó **242 vulnerabilidades** asociadas a las tecnologías monitoreadas, de un total de **7503 CVEs** generales. A pesar de limitaciones puntuales, el sistema demostró una alta efectividad. **CVE-Hunter** se posiciona como una primera línea de defensa, capacitando a las organizaciones para tomar decisiones informadas y responder ágilmente, optimizando su postura de ciberseguridad.

Palabras Clave— Vulnerabilidades; Notificación; Ciberseguridad; Desarrollo de software; CVE (Common Vulnerabilities and exposures).

ABSTRACT

Abstract— The escalating risks in cybersecurity demand proactive solutions to safeguard digital assets. This thesis proposes **CVE-Hunter**, a passive vulnerability monitoring system designed to automate the collection and notification of *Common Vulnerabilities and Exposures* (CVEs). Its primary objective is to assist development teams in the early identification of emerging threats. For validation, **CVE-Hunter** was rigorously evaluated in collaboration with *Develrox*, a cybersecurity consulting firm. This evaluation involved the analysis of specific technologies through a stratified random verification against the *National Vulnerability Database* (NVD). The results were significantly positive: the system detected **242 vulnerabilities** associated with the monitored technologies, out of a total of **7503 general CVEs** identified. Despite specific limitations, the system demonstrated high effectiveness. **CVE-Hunter** positions itself as a crucial first line of defense, empowering organizations to make informed decisions and respond agilely, thereby optimizing their cybersecurity posture.

Keywords— Vulnerabilities; Notification; Cybersecurity; Software Developer; CVE (Common Vulnerabilities and exposures).

GLOSARIO

CNA: Computer Network Attack.

CVE: Common Vulnerabilities and Exposures / Vulnerabilidades y Exposiciones Comunes.

NIST: National Institute of Standards and Technology.

MITRE CORPORATION: Organización sin fines de lucro que opera centros de investigación y desarrollo financiados con fondos federales en los Estados Unidos.

OSS: Open-Source Software / Software de código abierto.

BUG: Error o problema no esperado en un software o hardware.

JSON: JavaScript Object Notation.

BSON: Binary JSON.

NoSQL: Not Only Structured Query Language.

API: Application Programming Interface / Interfaz de Programación de Aplicaciones.

NVD: National Vulnerability Database o en español, Base de datos nacional de vulnerabilidades, es una base de datos mantenida por el *NIST*, que es utilizada de manera centralizada para almacenar información sobre CVEs.

SMTP: Simple Transfer Mail Protocol o Protocolo Simple de Transferencia de Correo. Se trata de un protocolo de comunicación que se utiliza para enviar y recibir mensajes de correo electrónico a través de Internet.

Token: Cadena de caracteres que tiene un significado coherente en cierto lenguaje de programación.

CVSS: Common Vulnerability Scoring System / Sistema Común de Puntuación de Vulnerabilidades.

ETL: *Extract, Transform, Load* / Extracción, Transformación y carga.

HTML: *HyperText Markup Language*

XML: *Extensible Markup Language*

FORTINET: Empresa multinacional de Estados Unidos, dedicada al desarrollo y la comercialización de software, dispositivos y servicios de ciberseguridad, como firewalls, antivirus, prevención de intrusiones y seguridad en dispositivos de usuario, entre otros.

CISCO: Empresa global dedicada a la fabricación, venta, mantenimiento y consultoría de equipos de telecomunicaciones.

Regex: *Regular Expresion* / Expresión regular.

CSV: *Comma-separated values* / Valores separados por coma.

Fastly: Proveedor estadounidense de servicios de computación en la nube.

HTTP: **Hypertext Transfer Protocol** es un protocolo de comunicación que permite la transferencia de información.

OWASP: **Open Web Application Security Project** es un proyecto open source dedicado a determinar y combatir las causas que hacen que el software sea inseguro. Este proporciona diferentes herramientas y frameworks, que facilitan a las organizaciones analizar sus sistemas de una manera segura. Esta guía, menciona que no existen los programas perfectos, ni una manera única de desarrollar software ya que siempre existirán riesgos que pueden afectar a las organizaciones.

Cron: Herramienta de programación de tareas en sistemas operativos tipo Unix. Permite programar la ejecución de comandos o scripts en momentos específicos, como cada hora, cada día, cada semana, etc.

ÍNDICE DE CONTENIDOS

RESUMEN	IV
ABSTRACT	IV
GLOSARIO	V
ÍNDICE DE FIGURAS	IX
ÍNDICE DE TABLAS	IX
INTRODUCCIÓN	1
CAPÍTULO 1: DEFINICIÓN DEL PROBLEMA	2
1.1 Contexto	2
1.2 Objetivos generales	3
1.3 Objetivos específicos	3
1.4 Árbol del problema	4
CAPÍTULO 2: MARCO CONCEPTUAL	5
2.1 <i>Common Vulnerabilities and Exposures (CVE)</i>	5
2.1.1 ¿Qué información contiene un CVE?	5
2.2 Qué dice la literatura sobre la ciberseguridad y los CVEs	7
2.2.1 Sistemas automatizados basados en bases de datos CVE	8
2.2.2 Aplicación de inteligencia artificial en la detección de vulnerabilidades	8
2.2.3 Pruebas de penetración	9
2.2.4 Plataformas Web	9
2.3 Enfoques existentes para la obtención de vulnerabilidades a través de CVEs .	9
2.3.1 Técnicas activas:	10
2.3.2 Técnicas pasivas	11
2.4 Conceptos Técnicos	13
2.4.1 Scraping de datos	13
2.4.2 ETL	13
2.4.3 Microservicios	14
2.4.4 Bróker de mensajería	15
2.4.5 Patrón adaptador	17
CAPÍTULO 3: PROPUESTA DE SOLUCIÓN	19
3.1 Descripción de <i>CVE-HUNTER</i>	19
3.2 Metodología utilizada	19
3.3 Arquitectura	20
3.4 API de Usuarios y API de Compañías	22
3.5 Parser	24
3.5.1 Modelo planificado para el CVE	24

3.5.2	ETLs	26
3.6	Monitor	27
3.6.1	CVEManagerForUser	28
3.6.2	Sender	32
3.6.3	Clase Monitor	33
3.7	Canales de comunicación	35
CAPÍTULO 4: VALIDACIÓN DE LA SOLUCIÓN		36
4.1	Entorno de trabajo	36
4.2	Alcance y aclaraciones	36
4.3	Criterios de éxito del sistema	37
4.4	Metodología de validación	38
4.5	Resultados	38
4.5.1	Validación de la Precisión de Detección	40
4.5.2	Monitoreo Continuo	42
4.5.3	Eficacia en la notificación a Usuarios	42
4.6	Limitaciones y problemas detectados	44
CAPÍTULO 5: CONCLUSIONES		46
5.1	Conclusiones generales	46
5.2	Conclusiones por objetivos específicos	47
5.3	Trabajos a futuros	48
5.3.1	Palabras finales	49
ANEXOS		50
5.1	Links asociados a los parsers	50
5.2	CVEs encontrados por el sistema en el período mencionado	50
5.3	Ejemplo plantillas	54
5.3.1	Template correo electrónico	54
5.3.2	Mensajes de Telegram	56
5.3.3	Mensajes de Slack	56
5.4	Trabajos adicionales	57
REFERENCIAS BIBLIOGRÁFICAS		61

ÍNDICE DE FIGURAS

1	Árbol del problema.	4
2	Ejemplo gráfico de ETL	14
3	Ejemplo de arquitectura basada en microservicios	14
4	Ejemplo básico de una cola en RabbitMQ.	16
5	Ejemplo básico de un <i>exchange</i> en RabbitMQ.	16
6	Ejemplo representativo del patrón adaptador	18
7	Ciclo de desarrollo.	19
8	Arquitectura.	21
9	Ejemplo de la colección de usuarios	22
10	Ejemplo de la colección de compañía	23
11	Modelo de CVE	25
12	Arquitectura de los parseadores.	27
13	Arquitectura de los parseadores.	28
14	Código del método <code>get_if_system_vulnerate</code>	29
15	Código del método <code>create_cve_for_user</code>	30
16	Código del método <code>cve_for_send</code>	31
17	Código de la clase <code>Sender</code>	32
18	Código del método <code>_extract_cpe_info</code>	33
19	Código del método <code>_compare_user_technologies</code>	34
20	Código del método <code>_process_user</code>	35
21	Cantidad de vulnerabilidades encontradas por tecnología	39
22	Distribución de vulnerabilidades por severidad	40
23	Ejemplos de la recepción de algunos correos	42
24	Encabezado y información de resumen en la notificación.	43
25	Cuerpo y información de CVEs	44
26	Ejemplo de templates de correo electrónico.	54
27	Ejemplo de templates de correo electrónico.	55
28	Plantilla de mensaje para CVE en telegram	56
29	Ejemplo de notificación de Slack	56
30	Interfaz del servicio de búsqueda de CVE (1).	57
31	Interfaz del servicio de búsqueda de CVE (2).	58
32	Ejemplo de informe ejecutivo (1).	59
33	Ejemplo de informe ejecutivo (2).	60

ÍNDICE DE TABLAS

1	Características del servidor	36
2	Conjunto de tecnologías entregadas por <i>Develrox</i>	37
3	Muestra de CVEs verificados manualmente y su precisión.	41
4	CVEs encontrados por tecnología Fuente: Elaboración Propia.	53

INTRODUCCIÓN

La ciberseguridad es la práctica de proteger a las personas, los sistemas y los datos frente a amenazas digitales, mediante el uso de tecnologías, procesos y políticas [Jonker *et al.*, 2025], con el objetivo de preservar la confidencialidad, integridad y disponibilidad de la información, conforme a los principios establecidos en la norma ISO/IEC 27001 [ISO, 2022]. En este contexto, el ciberdelito representa una de las principales amenazas, ya que involucra el uso de técnicas maliciosas para comprometer la seguridad de los sistemas [UNODC, 2020]. Los atacantes pueden obtener acceso a la información mediante diversos métodos, como ataques de fuerza bruta a contraseñas débiles, acciones malintencionadas de usuarios internos, o la explotación de vulnerabilidades. Este trabajo se enfoca en las vulnerabilidades, donde los delincuentes buscan aprovechar distintas amenazas que pueden afectar a los sistemas de las organizaciones, como software desactualizado, librerías vulnerables, errores de desarrollo o decisiones inadecuadas de infraestructura.

Common Vulnerabilities and Exposures o Vulnerabilidades y exposiciones comunes consiste en un sistema público de enumeración y clasificación de vulnerabilidades, donde se almacena información, referencia, criticidad, entre otros aspectos, acerca de las distintas vulnerabilidades que surgen a lo largo del tiempo. Estos registros son mantenidos por *The MITRE Corporation* desde el año 1999 y suelen ser almacenados por la **National Vulnerability Database** de Estados Unidos.

Este trabajo expone los enfoques existentes, tanto metodos como activos y pasivos, además de técnicas y herramientas utilizadas para la detección de vulnerabilidades. Se propone un producto de software, con un enfoque pasivo, que permite a los usuarios obtener información sobre la criticidad, descripción y métodos de solución acerca de vulnerabilidades utilizando distintas fuentes de información. Esto permite a los equipos tecnológicos complementar sus desarrollos con información de vulnerabilidades que pueden afectar sus sistemas.

Esta memoria se divide en 5 capítulos; **Capítulo 1: Definición del problema**, se definen los problemas que existen actualmente en el mundo de la tecnología en temas de ciberseguridad, y como estos afectan a las organizaciones. Se remarcan los objetivos generales y específicos que quiere realizar este trabajo de título. **Capítulo 2: Marco Conceptual**, se definen términos importantes y tecnologías utilizadas que serán fundamentales para desarrollar los demás capítulos, además de un estado del arte, con información de trabajos asociados que ayudan a las empresas en temas de seguridad. En el **Capítulo 3: Propuesta de solución**, se describe el método desarrollado, además de la construcción del software en sí, presentando las funcionalidades y características que este tiene, y que pueda aportar a las distintas empresas. Este software se valida en el **Capítulo 4: Validación de la solución**, donde a través de tecnologías entregadas por una empresa, se presentan los resultados del software. Finalmente se presenta el **Capítulo 5: Conclusiones**, el cual concluye el trabajo realizado, presentando conclusiones generales, conclusiones por objetivos, además de trabajos futuros.

CAPÍTULO 1

DEFINICIÓN DEL PROBLEMA

1.1. Contexto

En el ámbito de las tecnologías de la información, surgen continuamente propuestas innovadoras que pueden ayudar a resolver alguna problemática del mundo real, provocando muchas veces un desarrollo de *software* de por medio. Estos *softwares* son realizados en distintas tecnologías, como algún lenguaje de programación, un framework o librerías específicas. Pero estas herramientas a través del tiempo pueden presentar algunas fallas que pueden poner en peligro la ciberseguridad de las empresas. Debido a esto, cada vez los equipos de desarrollo están más preocupados por la ciberseguridad de sus *software*, ya que cada día se producen más y más ataques cibernéticos, debido a alguna vulnerabilidad específica en las tecnologías utilizadas. En todo momento, los cibercriminales buscan explotar algún bug¹ y obtener acceso no autorizado a sistemas y redes, buscando robar información privilegiada o obtener accesos no autorizados de las distintas entidades, provocando repercusiones negativas en ellas, como pérdidas económicas, daño a la imagen o incluso el quiebre de las mismas compañías.

Cada día se crean en internet más de 2.5 billones de bytes de datos, y se estima que para el 2025 la cantidad ascienda a 463 exabytes de datos por día. Esto ha provocado que cada vez se resguarde más la información, ya que el riesgo de explotación de estas ha provocado que las vulnerabilidades puedan ser aprovechadas por actores malintencionados llevando ataques cibernéticos, comprometiendo la seguridad de los sistemas y la información confidencial de las organizaciones. También se estima que para el año 2025, los costos producidos por el cibercrimen costará a las empresas a nivel global **10.5 billones de dolares**. Una empresa que tarda un mes en detectar una violación de datos cibernéticos está expuesta a pérdidas financieras significativas, que suelen ser a menudo en millones de dólares [Ambrissi, 2023].

Otro punto importante, es que según el **Foro económico mundial** una organización tarda aproximadamente 280 días en detectar que una vulnerabilidad está afectando a sus sistemas, pudiendo ya haber sido vulnerada por algún atacante hace un largo tiempo atrás. A causa de ello, se recomiendan diferentes estrategias para intentar mitigar las vulnerabilidades a tiempo, como mantener los sistemas actualizados, evitar acceso no autorizado de personal que no es de la organización, gestionar vulnerabilidades a través de diferentes softwares, entre otras [Entel Digital, 2023]. Esto destaca la importancia y la urgencia de la ciberseguridad, donde el tiempo es equivalente a dinero.

Finalmente, según el informe de seguridad anual de **Fastly**[Fastly, 2023], menciona que las organizaciones son víctimas en promedio a 46 ciberataques al año y tardan una media de

¹Bug: Es un error o problema no esperado en un software o hardware.

ocho meses en recuperarse totalmente de los efectos negativos de las brechas de seguridad. También en el informe se menciona que 2 de los 5 ataques más comunes, están relacionados a la explotación de vulnerabilidades en *softwares open sources* y ataques relacionados a aplicaciones web y *APIs*.

Esto junto con lo mencionado en el **Foro económico mundial**, confirma que siempre se está expuesto, y que puede existir desinformación acerca de las vulnerabilidades que van surgiendo en el día a día.

El término ***Common Vulnerabilities and exposures (CVE)*** o en español “Vulnerabilidades y Exposiciones Comunes”, consiste en un sistema estandarizado utilizado para identificar, definir y enumerar vulnerabilidades de seguridad informática, divulgándolas públicamente a través de distintas organizaciones alrededor de la web, con tal de que exista un registro por cada una de estas.

Para el caso de los desarrolladores, obtener información específica sobre algún *CVE* que afecte a su empresa puede resultar complejo, debido a que diariamente se registran cientos de vulnerabilidades siendo engorroso encontrar alguno que afecte a su software en específico. Además, puede que los equipos de desarrollo ni se enteren que exista una vulnerabilidad que los afecte, provocando ser vulnerables para algún atacante.

Para abordar esta problemática, existirá *CVE-HUNTER*, un producto de software encargado de recopilar datos de servicios oficiales, divulgando a través de canales de comunicación, como ***Telegram, Slack y Gmail***, los *CVEs* asociados a las tecnologías entregadas por los clientes, y así permitir que los equipos de desarrollo estén informados sobre las posibles vulnerabilidades que pueden estar afectando a sus softwares.

1.2. Objetivos generales

Desarrollar un sistema de software que permita el monitoreo, identificación y notificación automática de vulnerabilidades que puedan afectar la infraestructura tecnológica de las organizaciones, con la finalidad de informar oportunamente sobre los riesgos asociados en sus sistemas.

1.3. Objetivos específicos

1. Implementar un sistema de extracción de datos de *CVEs* desde diferentes fuentes, con el propósito de recolectar información relevante sobre las vulnerabilidades que se registran de forma continua.
2. Estructurar la información obtenida desde las distintas fuentes, y integrar en una base de datos unificada, con el fin de facilitar el filtrado de esta.

3. Crear un servicio de monitoreo, encargado de asociar la información obtenida de las distintas fuentes, y filtrar en función de las tecnologías utilizadas en los entornos objetivo, con el fin de notificar los CVEs que afectan a sus softwares.
4. Crear servicios de canales de comunicación, con el objetivo de informar sobre las vulnerabilidades detectadas en sus sistemas.
5. Evaluar la funcionalidad del sistema, a través de tecnologías dadas por una compañía, para validar la efectividad del sistema respecto del objetivo general.

1.4. Árbol del problema



Figura 1: Árbol del problema.
Fuente: Elaboración propia.

CAPÍTULO 2

MARCO CONCEPTUAL

En este capítulo se presentan los conceptos y términos fundamentales relacionados con la ciberseguridad y las vulnerabilidades. Se comenzará con la definición del estándar CVE y sus características, para luego analizar la literatura, metodologías aplicadas y enfoques actuales. Finalmente, se describirán las tecnologías y patrones arquitectónicos empleados en el desarrollo del sistema propuesto.

2.1. *Common Vulnerabilities and Exposures (CVE)*

El programa **CVE (Common Vulnerabilities and Exposures)**, denominado en español como **Vulnerabilidades y Exposiciones comunes**, constituye una iniciativa estandarizada cuyo propósito es identificar, catalogar y difundir de manera unificada las debilidades de seguridad en sistema de *software* y *firmware*. Gestionado por la **MITRE Corporation** desde su creación en 1999 y patrocinado por la **División Nacional de Ciberseguridad (NCS)** de Estados Unidos [CVE Program, 2023], este sistema surge como respuesta a la necesidad de **centralizar** la información sobre vulnerabilidades, evitando así la dispersión de criterios en la industria.

CVE busca establecer un lenguaje común para describir, referenciar y analizar fallos de seguridad, facilitando la interoperabilidad entre herramientas de ciberseguridad, bases de datos y equipos multidisciplinarios. Su estructura opera bajo el principio de unificar identificadores, lo que permite correlacionar reportes heterógeneos, optimizar la gestión de parches y fortalecer la colaboración entre entidades. Este depende de sistemas complementarios (como CVSS) para priorizar riesgos y cuantificar impactos. Gracias a esta estandarización, el programa se ha consolidado como un componente esencial en los procesos de gestión de vulnerabilidades, permitiendo a las organizaciones actuar de forma proactiva frente a amenazas emergentes y mejorar la resiliencia de sus infraestructuras tecnológicas.

La estandarización del CVE es respaldada por entidades como el *National Institute of Standards and Technology (NIST)*, que en conjunto con *MITRE Corporation*, se encargan del proceso de validación, clasificación y publicación de cada vulnerabilidad registrada [NIST, 2025].

2.1.1. ¿Qué información contiene un CVE?

Cada CVE incluye distintos elementos que permiten entender la naturaleza, criticidad, contexto y la forma de gestionar adecuadamente una vulnerabilidad. Entre los campos más relevantes destacan:

- **Identificador (ID):** Corresponde a un código único con el formato CVE-AÑO-NÚMERO, el cual permite referirse de manera inequívoca a una vulnerabilidad específica (por ejemplo, CVE-2022-24697).
- **Descripción:** Corresponde a un componente importante de un CVE, el cual proporciona un resumen del problema de seguridad, que suele incluir el tipo de vulnerabilidad, tecnologías afectadas, versiones y posibles impactos.
- **CVSS (Common Vulnerability Scoring System):** Es un sistema de puntuación que evalúa la gravedad de una vulnerabilidad. Su escala va de 0 a 10, el cual se define por un conjunto de métricas que ayudan a determinar su valor, donde valores más altos indican mayor severidad [FIRST, 2024]. Este se clasifica de la siguiente manera:
 - **CRITICAL (9.0 – 10.0):** Vulnerabilidades extremadamente severas que requieren atención inmediata.
 - **HIGH (7.0 – 8.9):** Fallas importantes con potencial de explotación significativa.
 - **MEDIUM (4.0 – 6.9):** Riesgos moderados con condiciones específicas de explotación.
 - **LOW (0.1 – 3.9):** Impacto limitado o difícil explotación.
 - **UNDEFINED:** Aún sin puntaje asignado.
- **Referencias:** Enlaces a fuentes externas como boletines de seguridad, bases de datos de fabricantes u otras plataformas que amplían la información técnica sobre el CVE.
- **CPE (Common Platform Enumeration):** Corresponde a una nomenclatura estandarizada utilizada para identificar y describir de manera precisa sistemas operativos, aplicaciones y dispositivos de hardware dentro del ámbito de las tecnologías de la información [National Institute of Standards and Technology, 2022b].

La versión más utilizada del estándar CPE es la 2.3, cuyo formato general es el siguiente:

`cpe:version_cpe:a:vendor:product:version`

- `cpe`: Prefijo que indica que es un CPE.
- `version`: Indica la versión del CPE. En la actualidad, nos encontramos en la versión 2.3.
- `a`: Indica que el producto es una aplicación.
- `vendor`: Nombre del proveedor del producto.
- `product`: Nombre del producto.
- `version`: Versión del producto.

Ejemplo de CPE:

`cpe:2.3:a:microsoft:internet_explorer:11.0.9600.16384:*:*:*:*:*:`

Aunque el estándar CVE es ampliamente reconocido como la fuente de referencia para la identificación de vulnerabilidades, su aplicación en entornos empresariales presenta importantes brechas. Un estudio realizado entre el 2019 al 2024 indicó que el 60 % de las brechas de seguridad se originó en vulnerabilidades ya conocidas y parcheables, mientras que casi el 80 % de los ataques en 2020 explotaron fallas con más de tres años de antigüedad [Omar Santos, 2025]. Esto sugiere que, a pesar de contar con un sistema de nominación estandarizado, muchas organizaciones no actualizan ni monitorean activamente los CVEs relevantes a su entorno. Además, el informe del 25 aniversario del Programa CVE resalta que existen sectores clave, como IoT e infraestructura, donde la cobertura y gestión de CVEs todavía es parcial [MITRE Corporation, 2024]. Por tanto, es necesario fomentar una adopción más proactiva y estructurada del estándar dentro de los procesos de ciberseguridad corporativos.

2.2. Qué dice la literatura sobre la ciberseguridad y los CVEs

En la actualidad, la tecnología ejerce un papel fundamental en el desarrollo de la sociedad, influyendo significativamente en la forma en que operan las organizaciones y se relacionan con su entorno. La transformación digital ha emergido como una fuerza clave que impulsa la adaptación constante de procesos, equipos y metodologías, con el fin de mantener la competitividad en un entorno global dinámico. Áreas como la inteligencia artificial, redes, robótica y el desarrollo de software destacan por su impacto en el mercado laboral, pero en todas ellas la ciberseguridad ocupa un lugar esencial.

Reconocer la creciente importancia de la ciberseguridad resulta imprescindible, dado que cualquier debilidad presente en aplicaciones, sensores o infraestructuras tecnológicas puede convertirse en una vía de acceso para ciberataques. Estas amenazas afectan no solo a individuos, sino también a empresas, instituciones y organizaciones de diversa índole [Briceño, 2020]. Frente al uso intensivo de tecnologías de la información y comunicación, se vuelve fundamental evaluar y fortalecer la seguridad de los datos. Para ello, se recomienda implementar procesos de diagnóstico que permitan establecer el estado actual de la seguridad y detectar posibles vulnerabilidades [Solarte *et al.*, 2015].

Antes de abordar los enfoques específicos para la detección y gestión de vulnerabilidades, es importante revisar algunos datos de lo que la literatura ha documentado sobre los distintos métodos utilizados en el ámbito de la ciberseguridad. A continuación, se presentan diversas estrategias que permiten identificar vulnerabilidades en sistemas informáticos, desde el uso de bases de datos estandarizadas hasta metodologías manuales, técnicas basadas en inteligencia artificial y herramientas de escaneo especializadas. Cada una de estas alternativas ofrece ventajas y limitaciones que deben ser consideradas al momento de diseñar mecanismos eficaces de protección.

2.2.1. Sistemas automatizados basados en bases de datos CVE

Los sistemas automatizados basados en bases de datos de vulnerabilidades como NVD y OVAL representan la columna vertebral de la detección moderna de amenazas. Estas plataformas permiten identificar, clasificar y priorizar vulnerabilidades mediante la correlación automatizada de metadatos técnicos, reduciendo la dependencia de intervención manual. Su eficacia radica en la capacidad para escanear configuraciones de sistemas y comparar componentes instalados con repositorios actualizados de amenazas conocidas, proporcionando alertas tempranas sobre posibles vectores de ataque [NetThreat Insight, 2025].

Estos sistemas suelen integrarse en flujos de trabajo continuos dentro de entornos corporativos, permitiendo ejecutar análisis programados, generar reportes personalizados y activar respuestas automáticas ante la detección de amenazas críticas. Si bien su implementación ha demostrado ser eficaz para mantener una vigilancia constante sobre los activos tecnológicos, su precisión depende en gran medida de la calidad y frecuencia de actualización de las bases de datos utilizadas. Asimismo, uno de los desafíos persistentes radica en la correcta interpretación del contexto de cada vulnerabilidad, ya que no todas representan el mismo nivel de riesgo en función de la configuración específica del sistema afectado [FasterCapital, 2025].

2.2.2. Aplicación de inteligencia artificial en la detección de vulnerabilidades

La incorporación de técnicas de inteligencia artificial en la ciberseguridad ha permitido el desarrollo de métodos avanzados para la identificación de CVEs. En particular, el aprendizaje automático (*Machine Learning*), el aprendizaje profundo (*Deep Learning*) y el procesamiento de lenguaje natural (*Natural Language Processing*, NLP) se han empleado para analizar descripciones textuales de vulnerabilidades, extraer patrones ocultos y predecir nuevas amenazas [Jeon y Kim, 2021]. Estos enfoques permiten procesar grandes volúmenes de datos provenientes de repositorios como la NVD, mejorando la velocidad y el alcance del análisis de amenazas.

Sin embargo, existen desafíos importantes que limitan su aplicabilidad práctica, entre ellos destacan la generación de sesgos en los algoritmos, la falta de explicabilidad en los modelos complejos y la necesidad de datos etiquetados para su entrenamiento [Kanakogi *et al.*, 2021]. Estas limitaciones dificultan su adopción en contextos sensibles o altamente regulados, donde la trazabilidad de las decisiones es fundamental.

En el siguiente artículo [K y Arun, 2017], se proponen técnicas de análisis de texto basadas en procesamiento de lenguaje natural, las cuales buscan mejorar la comprensión contextual de las descripciones de vulnerabilidades. No obstante, los autores advierten que estas soluciones requieren aún validación rigurosa en entornos productivos para garantizar su eficacia. A pesar de estas limitaciones, la inteligencia artificial continúa consolidándose como un componente estratégico en el ámbito de la ciberseguridad, dado su potencial para complementar enfoques tradicionales, automatizar tareas complejas y detectar amenazas emergentes que

podrían pasar inadvertidas por los métodos convencionales.

2.2.3. Pruebas de penetración

Las pruebas de penetración, o pentesting, constituyen una metodología utilizada en ciberseguridad para evaluar la resistencia de un sistema frente a ataques reales. Consiste en simular intrusiones controladas con el fin de identificar y explotar vulnerabilidades que podrían comprometer la confidencialidad, integridad o disponibilidad de los recursos [Krutz y Vines, 2007] [Henry, Kevin M., 2007].

Existen distintos enfoques según el nivel de acceso a la información: las pruebas de caja negra, sin conocimiento previo del sistema; caja blanca, con acceso completo a su arquitectura y configuración; y caja gris, con información parcial [Briceño, 2020]. Esta práctica puede realizarse de forma manual, con análisis detallados realizados por especialistas, o apoyarse en herramientas automatizadas para agilizar ciertos procesos.

A pesar de su utilidad para descubrir fallos críticos que podrían no ser detectados por métodos convencionales, el pentesting presenta como desventaja el tiempo y la experiencia técnica que requiere su correcta ejecución. No obstante, su valor radica en la precisión de sus hallazgos y en su capacidad para reproducir escenarios reales de ataque. Guías como las del proyecto OWASP proporcionan marcos de referencia ampliamente adoptados en este tipo de evaluaciones [OWASP Foundation, 2024].

2.2.4. Plataformas Web

Si bien existen múltiples plataformas para consultar vulnerabilidades, muchas de ellas no logran notificar de manera directa al usuario sobre las amenazas que afectan sus sistemas. La CVE API del NIST [National Institute of Standards and Technology, 2022a], por ejemplo, ofrece información detallada pero presenta un modelo de datos complejo. En contraste, la plataforma del **MITRE** [The MITRE Corporation, 2023] brinda acceso histórico, aunque su frecuencia de actualización es limitada, lo cual genera información redundante. Además, diversos servicios externos no oficiales carecen de automatización, lo que obliga a realizar búsquedas manuales poco eficientes.

2.3. Enfoques existentes para la obtención de vulnerabilidades a través de CVEs

Una vez revisadas algunas metodologías utilizadas para gestionar las vulnerabilidades, resulta pertinente abordar ciertos enfoques presentados en la literatura especializada. En este

contexto, se describen dos enfoques generales: **pasivo** y **activo**. El objetivo es establecer una distinción entre las distintas metodologías y herramientas empleadas para su detección, así como identificar las ventajas y desventajas asociadas a cada uno.

2.3.1. Técnicas activas:

Las técnicas de evaluación activa de vulnerabilidades corresponden a aquellas que implican realizar pruebas directas sobre los dispositivos o sistemas, con el propósito de identificar posibles puntos débiles [Pham y Dang, 2018]. A partir del análisis de las respuestas obtenidas desde los sistemas conectados, es posible evaluar la presencia de debilidades. Esta evaluación puede llevarse a cabo de forma manual, como parte de una prueba de penetración, o mediante el uso de herramientas y servicios automatizados [Plasman Dries, 2023]. Algunas de las formas en que se aplica este tipo de enfoque se describen a continuación:

- **Escanear puertos:** Consiste en enviar paquetes de red a los puertos de un dispositivo para identificar los servicios disponibles, con el objetivo de detectar servicios potencialmente vulnerables.
- **Realizar inyecciones SQL y HTML:** Consiste en introducir consultas SQL o código HTML maliciosos en una aplicación web, con el fin de identificar vulnerabilidades en la manipulación de datos o en la presentación de contenido.
- **Monitorear el tráfico de red:** Consiste en analizar los flujos de datos en la red para detectar anomalías, patrones sospechosos o intentos de intrusión.
- **Enviar cargas útiles maliciosas:** Consiste en proporcionar al sistema entradas especialmente diseñadas para evaluar su comportamiento frente a datos maliciosos y detectar posibles fallas de seguridad.

Herramientas representativas para la evaluación activa de vulnerabilidades

- **UdecEscaner:** Herramienta diseñada para detectar y evaluar vulnerabilidades en equipos de red mediante la identificación de servicios. Permite determinar los nombres y versiones de los servicios activos en un dispositivo, y buscar vulnerabilidades asociadas en la *National Vulnerability Database (NVD)*. Su uso en diversos entornos de prueba ha demostrado efectividad para identificar debilidades en sistemas informáticos, contribuyendo a facilitar la mitigación de riesgos por parte de los administradores de tecnologías de la información [Franco *et al.*, 2013].
- **SV-AF (Security Vulnerability Analysis Framework):** Propuesta orientada a establecer enlaces de trazabilidad bidireccionales entre el código fuente de un proyecto y las vulnerabilidades reportadas en los componentes utilizados. Esta trazabilidad permite

identificar qué proyectos se ven afectados por vulnerabilidades registradas en bases como NVD, lo que mejora la precisión del análisis de impacto. Además, su integración con prácticas ágiles permite reforzar la extracción y seguimiento de requisitos de seguridad [Alqahtani *et al.*, 2017].

- **Nessus:** Herramienta ampliamente utilizada en la industria de la ciberseguridad para escanear vulnerabilidades en sistemas y redes. Permite analizar una variedad de dispositivos incluidos routers, firewalls y servidores, con el fin de detectar configuraciones incorrectas, parches ausentes o software desactualizado. Nessus se encuentra disponible en versiones gratuitas y de pago, ofreciendo funcionalidades avanzadas y soporte profesional a través de la empresa Tenable²³.

Para finalizar, las técnicas de enfoque activa son ideales para identificar vulnerabilidades explotables con alta precisión. Su principal ventaja es que permiten validar riesgos de seguridad de manera directa, simulando ataques reales. Sin embargo, al ser intrusivas, requieren permisos especiales y pueden afectar sistemas en producción si no se implementan cuidadosamente, provocando por ejemplo un ataque DDos. Son más adecuadas para entornos controlados, pruebas de penetración (Pentesting) y etapas de desarrollo de software.

2.3.2. Técnicas pasivas

Las técnicas de evaluación pasiva de vulnerabilidades se centran en comparar las características específicas de los sistemas con bases de datos de vulnerabilidades conocidas, como la NVD. Este enfoque no requiere interacción directa con los dispositivos evaluados, ya que se fundamenta en recopilar y correlacionar información para identificar posibles debilidades [Pham y Dang, 2018]. A diferencia de las técnicas activas, las pasivas resultan no intrusivas, evitando riesgos asociados a la ejecución de pruebas directas sobre los sistemas.

A continuación, se describen algunas herramientas representativas que implementan este tipo de técnicas:

- **P0f (Passive Operating System Fingerprinting):** Herramienta de detección pasiva utilizada para identificar sistemas operativos y aplicaciones en la red. Analiza los encabezados de los paquetes para inferir características del sistema remoto, como el sistema operativo y la versión del software.
- **PVS (Passive Vulnerability Scanner):** Herramienta diseñada para identificar vulnerabilidades sin generar tráfico activo en la red. A diferencia de los escáneres tradicionales, PVS analiza de manera continua el tráfico que circula por la red, extrayendo información sobre los sistemas, y comparándola con bases de datos de seguridad como la NVD.

²<https://es-la.tenable.com/>

³<https://www.techtarget.com/searchnetworking/definition/Nessus>

- **ShoVAT:** Herramienta basada en el motor de búsqueda Shodan⁴, orientada a evaluar servicios expuestos públicamente. Utiliza información recopilada por Shodan para detectar posibles vulnerabilidades en los servicios accesibles desde la red.

Este tipo de técnicas resulta útil para llevar a cabo evaluaciones históricas de vulnerabilidades a lo largo del ciclo de vida de un sistema, sin necesidad de afectar su operación. Permiten monitorear de manera continua los riesgos existentes, facilitando la identificación de amenazas potenciales sin poner en riesgo la disponibilidad de los servicios.

Entre sus principales ventajas se encuentra la posibilidad de ser aplicadas en entornos productivos sin generar interrupciones. No obstante, su efectividad depende en gran medida de la disponibilidad y actualización de las fuentes de información externas, y no permiten confirmar si una vulnerabilidad es efectivamente explotable en el contexto particular del sistema analizado [Plasman Dries, 2023].

⁴Motor especializado en identificar dispositivos conectados a internet.

2.4. Conceptos Técnicos

En esta sección se presentan los conceptos técnicos fundamentales utilizados en la implementación del sistema. Su incorporación permite contextualizar las decisiones de diseño adoptadas durante el desarrollo.

2.4.1. Scraping de datos

Técnica utilizada por un programa informático encargada de extraer datos del resultado generado por otro programa. Es comúnmente aplicado a la extracción de datos desde sitios web, especialmente cuando no existen APIs públicas o interfaces formales de consulta. La utilización de bots de extracción, tratan de conseguir datos de los sitios, independiente de los intentos que se hagan para limitar el acceso [Cloudflare, 2022].

El proceso de scraping web es bastante simple, aunque la implementación pueda ser compleja. Este suele ocurrir en tres pasos:

1. El fragmento de código utilizado, para extraer la información, que llamamos bot de extracción, envía una solicitud *HTTP GET* a un sitio web específico.
2. Cuando el sitio web responde, el scraper analiza el documento **HTML** para buscar un patrón de datos específico.
3. Una vez se hayan extraído los datos, se convierte a cualquier formato específico proyectado por el autor del bot scraper.

Esta técnica se relaciona estrechamente con el proceso mencionado en la sección 2.4.2, ya que se utiliza la misma terminología para la obtención de datos, es decir, extracción, transformación y carga.

2.4.2. ETL

Un **ETL** por sus siglas Extracción, transformación y carga, es el proceso consistente en combinar datos de diferentes orígenes en un gran repositorio central llamado almacenamiento de datos. **ETL** utiliza un conjunto de reglas para limpiar y organizar datos en bruto y prepararlos para el almacenamiento, el análisis de datos y el machine learning (ML). Puede abordar necesidades de inteligencia empresarial específicas mediante análisis de datos, como la predicción del resultado de decisiones empresariales, la generación de informes y paneles, la reducción de la ineficacia operativa y más [Amazon Web Services, 2024].

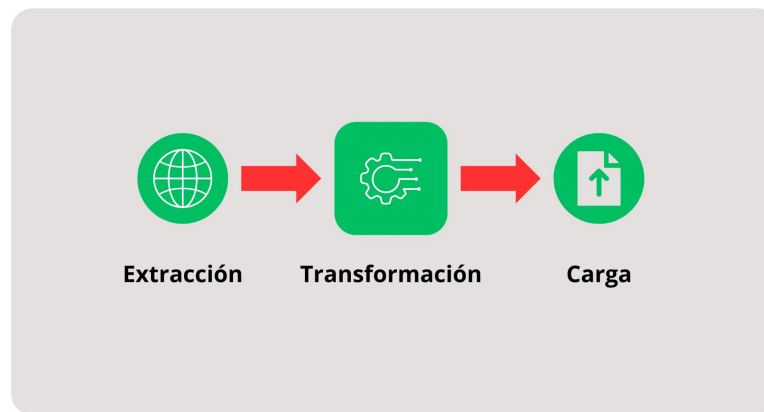


Figura 2: Ejemplo gráfico de ETL
Fuente: Elaboración propia.

2.4.3. Microservicios

Los microservicios, corresponden a un tipo de arquitectura, donde una aplicación o software, es desarrollada como una colección de servicios, lo que permite un desarrollo, despliegue y mantenibilidad independiente [Microsoft, 2023].

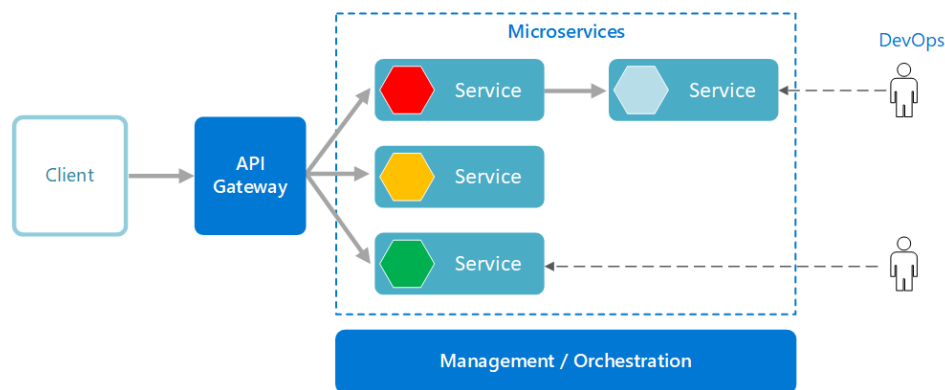


Figura 3: Ejemplo de arquitectura basada en microservicios
Fuente: [Microsoft, 2023].

Características

- Los microservicios son pequeños e independientes, y están acoplados de forma imprecisa. Un único equipo reducido de programadores puede escribir y mantener un servicio.
- Cada servicio es un código base independiente, que puede administrarse por un equipo de desarrollo pequeño.

- Un equipo puede actualizar un servicio existente sin tener que volver a generar e implementar toda la aplicación.

La arquitectura de microservicio hace que las aplicaciones sean más fáciles de escalar y rápidas de desarrollar. Esto permite la innovación y acelera el tiempo de comercialización de las nuevas características.

2.4.4. Bróker de mensajería

Un *bróker* de mensajería es un software que permite la comunicación entre aplicaciones, sistemas y servicios, facilitando el intercambio de mensajes entre ellos. Este intermediario actúa transportando mensajes entre diferentes protocolos, lo que permite que sistemas heterogéneos, desarrollados en distintos lenguajes o plataformas, puedan interoperar de manera efectiva.

Los *brókers* de mensajería forman parte de soluciones conocidas como *middleware* orientado a mensajes (*Message-Oriented Middleware*, MOM). Este tipo de *middleware* proporciona un mecanismo estandarizado para gestionar el flujo de datos entre los componentes de una aplicación, permitiendo a los desarrolladores centrarse en la lógica del negocio sin preocuparse por la complejidad de la comunicación entre servicios.

Entre sus funcionalidades, los *brókers* pueden validar, almacenar, enrutar y entregar mensajes a los destinos adecuados. De este modo, los emisores pueden enviar mensajes sin necesidad de conocer la ubicación, el estado o la cantidad de receptores, lo cual favorece el desacoplamiento entre componentes.

Para asegurar una entrega fiable, los *brókers* suelen utilizar estructuras llamadas *colas de mensajes*, donde se almacenan los mensajes de forma temporal hasta que son procesados por los consumidores correspondientes [IBM Cloud Education, 2021].

Modelos de distribución de mensajes

Dentro del marco conceptual, se reconocen dos patrones principales de distribución de mensajes en los sistemas de mensajería:

- **Mensajería punto a punto:** Este modelo establece una relación uno a uno entre el productor y el consumidor. El mensaje se envía a una cola específica y es consumido por un único receptor. Cada mensaje es procesado una sola vez. Este enfoque resulta adecuado cuando se necesita una respuesta individual por cada solicitud.

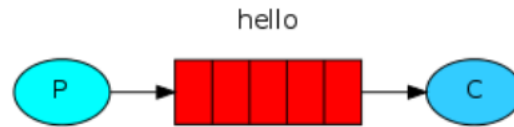


Figura 4: Ejemplo básico de una cola en RabbitMQ.

Fuente: [RabbitMQ, 2024].

P: Productor, C: Consumidor.

- **Mensajería de publicación/suscripción:** En este patrón, el productor publica mensajes en un *topic* o canal, al cual se pueden suscribir múltiples receptores. Todos los mensajes enviados a ese canal son distribuidos a cada uno de los suscriptores, estableciendo así una relación de uno a muchos.

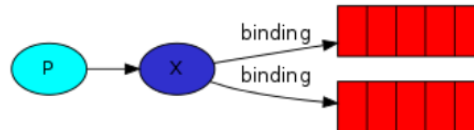


Figura 5: Ejemplo básico de un *exchange* en RabbitMQ.

Fuente: [RabbitMQ, 2024].

P: Productor, X: Exchange.

Conceptos clave en sistemas de mensajería

A continuación, se presentan los elementos fundamentales que componen un sistema de mensajería:

- **Producer (Productor):** Componente encargado de generar y enviar mensajes al sistema.
- **Consumer (Consumidor):** Componente que recibe los mensajes desde una cola y los procesa.
- **Queue (Cola):** Estructura donde se almacenan los mensajes de manera temporal hasta ser consumidos.
- **Exchange (Intercambio):** Elemento que recibe los mensajes desde el productor y los distribuye a las colas correspondientes, según reglas de enrutamiento definidas.

Los *exchanges*, contienen características y elementos que pueden clasificarse de la siguiente manera:

- **Binding key:** Es relación entre un *exchange* y una cola.
- **Routing Key:** Es la dirección que utiliza el *exchange* para determinar a que cola debe ir un mensaje.
- **Direct:** Enruta los mensajes a colas cuya *binding key* coincida exactamente con la *routing key* del mensaje.
- **Fanout:** Distribuye los mensajes a todas las colas asociadas al *exchange*, sin importar la *routing key*.
- **Topic:** Permite el enrutamiento mediante patrones, facilitando la suscripción a temas específicos utilizando comodines.
- **Headers:** Utiliza los valores de los encabezados del mensaje para determinar su destino.

RabbitMQ

Entre los brókers de mensajería más utilizados se encuentran *Apache Kafka*, *RabbitMQ*, *MQTT* y *Apache ActiveMQ*. Para el desarrollo de este trabajo, se ha seleccionado *RabbitMQ* debido a su solidez, amplia adopción, documentación disponible y compatibilidad con el protocolo AMQP (*Advanced Message Queuing Protocol*).

RabbitMQ es un sistema de mensajería de código abierto que permite la comunicación asincrónica entre distintos componentes de una arquitectura distribuida. Los mensajes pueden transportar información diversa, como eventos, instrucciones o datos, y ser entregado de manera confiable entre productores y consumidores [RabbitMQ, 2025].

2.4.5. Patrón adaptador

Los patrones de diseño son un juego de herramientas que brindan soluciones a problemas habituales en el diseño de software. Definen un lenguaje común que ayuda a los equipos a comunicarse con más eficiencia. En este contexto, el patrón adaptador es un patrón de diseño estructural, que permite que interfaces incompatibles, logren trabajar juntas, sin modificar el código fuente [Refactoring Guru, 2020a].

De acuerdo con el ejemplo presentado en [Refactoring Guru, 2020b], se plantea el desarrollo de una aplicación bursátil que recolecta información desde diversas fuentes en formato XML, con el objetivo de presentarla al usuario bajo ciertos requisitos. En una etapa posterior, se decide mejorar el sistema incorporando una nueva biblioteca de análisis desarrollada por un tercero. No obstante, dicha biblioteca solo admite datos en formato JSON, lo que genera una incompatibilidad entre formatos.

Para resolver esta situación, se propone la implementación de un adaptador, entendido como un objeto cuya función es transformar la interfaz de un componente para que pueda ser

utilizada por otro. Este adaptador actúa como un intermediario que envuelve al objeto original, ocultando la complejidad de la conversión que ocurre internamente. De este modo, el objeto adaptado puede operar de manera transparente dentro del sistema, sin necesidad de conocer la existencia del adaptador ni realizar modificaciones en su estructura original.

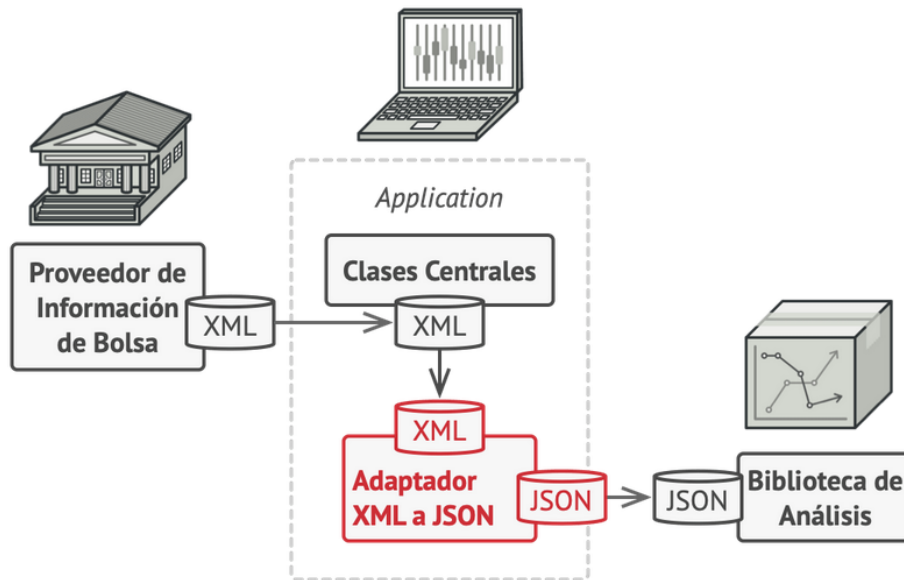


Figura 6: Ejemplo representativo del patrón adaptador
Fuente: [Refactoring Guru, 2020b].

CAPÍTULO 3

PROPUESTA DE SOLUCIÓN

3.1. Descripción de CVE-HUNTER

CVE-Hunter es un sistema de monitoreo pasivo de vulnerabilidades que automatiza la recolección de registros CVE desde fuentes oficiales. Su principal funcionalidad es analizar las tecnologías declaradas por los usuarios y generar notificaciones automatizadas cuando se identifiquen vulnerabilidades relevantes asociadas a ellas.

El sistema estará configurado para el envío de notificaciones a través de tres posibles canales de comunicación, como *Telegram*, *Slack* y *Correo electrónico*. El usuario que utilice el sistema tendrá la flexibilidad de elegir el canal de comunicación que mejor se adapte a su flujo de trabajo.

3.2. Metodología utilizada

CVE-Hunter fue desarrollado siguiendo un enfoque flexible y pragmático, en el que los requisitos se definían de forma progresiva según las necesidades que iban surgiendo. Las funcionalidades se implementaron de manera incremental, organizando el desarrollo en iteraciones cortas que permitieran adaptarse a los cambios y validar avances de forma continua.

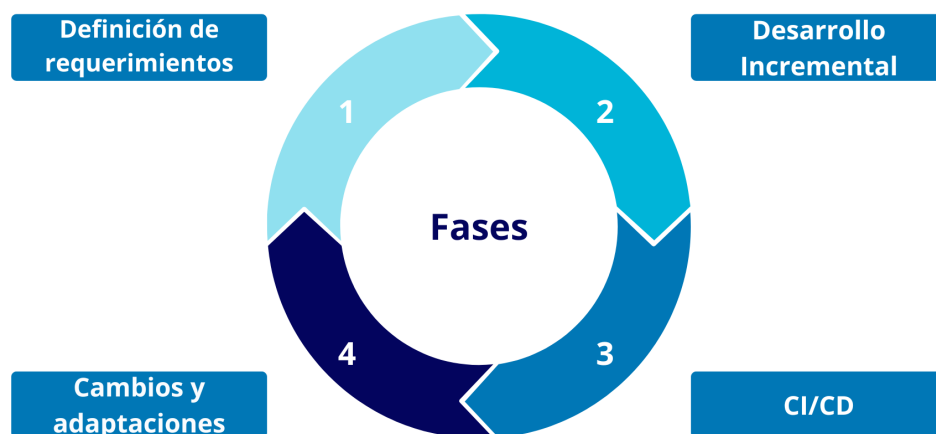


Figura 7: Ciclo de desarrollo.
Fuente: Elaboración propia.

Descripción de las fases:

1. Definir requisitos ad-hoc:
 - Los requerimientos se priorizaban según urgencia o criticidad.
 - No se estructuraron *sprints* formales, pero sí iteraciones naturales basadas en tareas completadas.
2. Desarrollar funcionalidad incremental:
 - Cada funcionalidad se desarrollaba en ciclos cortos, permitiendo ajustes según retroalimentación.
3. Implementar *CI/CD*:
 - Se utilizaron prácticas de **CI/CD** para garantizar actualizaciones rápidas y estables.
4. Adaptarse ante cambios emergentes:
 - El desarrollo se adaptaba ante nuevas características o hallazgos.

La construcción del sistema contempla cinco iteraciones, cada una centrada en los objetivos definidos. La primera iteración consiste en definir la arquitectura a utilizar, junto con la configuración de la base de datos. Posteriormente, la segunda iteración se enfoca en crear dos servicios *REST*, encargados de la gestión de usuarios y empresas respectivamente. La tercera iteración se dedica a construir un servicio de obtención de vulnerabilidades a través de diferentes fuentes de información, así como a la estructuración de dicha información en un formato común. Prosiguiendo, la cuarta iteración se centra en crear un servicio de monitoreo, cuya función es enlazar la información desde los distintos servicios y enviar esta información al usuario. Finalmente, la quinta iteración se orienta en crear un servicio encargado de la comunicación con los usuarios a través de diversos canales.

3.3. Arquitectura

Para desarrollar el sistema, se optó por adoptar una arquitectura basada en microservicios, con el fin de garantizar escalabilidad, flexibilidad y sostenibilidad a largo plazo. Si bien enfoques como el modelo monolítico pueden satisfacer los requerimientos funcionales iniciales, presentan limitaciones importantes en cuanto a escalabilidad, mantenibilidad y tolerancia a fallos.

Implementar una arquitectura de microservicios permite dividir el sistema en componentes independientes, capaces de desarrollarse, desplegarse y escalarse de forma autónoma [Red Hat Enterprise, 2023]. Esta característica contribuye a mejorar la disponibilidad general del sistema, ya que, ante una falla, únicamente el servicio afectado se ve comprometido, mientras que el resto de los componentes continúa operativo.

Dado lo anterior, se considera que este patrón arquitectónico se alinea adecuadamente con los objetivos del proyecto, al ofrecer una base técnica sólida que facilite su evolución futura.

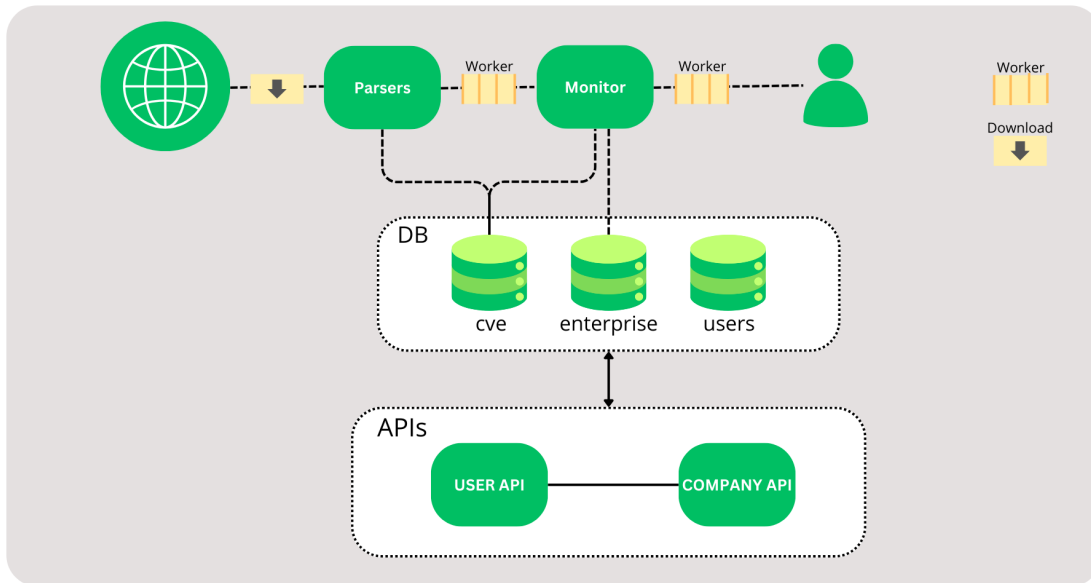


Figura 8: Arquitectura.
Fuente: Elaboración propia.

Como se ilustra en la Figura 8, la arquitectura propuesta se compone de cinco servicios principales: los módulos de *Parsers*, *Monitor*, *Canales de comunicación*, junto con una **API de Usuarios** y una **API de Compañías**. Estos servicios se comunican entre sí a través del sistema de mensajería *RabbitMQ*, lo cual permite establecer una integración desacoplada, eficiente y resiliente.

Adicionalmente, se definió una base de datos de tipo **NoSQL**, específicamente *MongoDB*, seleccionada por su compatibilidad con los formatos *JSON* y *BSON*, así como por su alto rendimiento en el manejo de datos no estructurados o semiestructurados. Esta decisión permite aprovechar capacidades avanzadas de consulta y ofrece flexibilidad en el modelado de datos, características clave para satisfacer los requerimientos del sistema [MongoDB, 2025].

3.4. API de Usuarios y API de Compañías

La **API de Usuarios** y **API de Compañías** son dos servicios dedicados a la administración y gestión de los usuarios y las empresas dentro del sistema. Ambas *APIs* fueron implementadas siguiendo los principios **CRUD**, permitiendo la creación, lectura, actualización y eliminación de registros. Estas fueron desarrolladas en **Go** utilizando el *framework* **Gin**, facilitando la gestión eficiente de usuarios y compañías mediante peticiones *RESTful* [Manabu, Gin Contributors, 2024].

```
1  {
2      "id": 1,
3      "name": "Jhon",
4      "lastname": "Doe",
5      "role": "analist",
6      "email": "jhon.doe@mail.com",
7      "company_id": 1
8  }
```

Figura 9: Ejemplo de la colección de usuarios
Fuente: Elaboración propia.

```
1  {
2      "id": 1,
3      "name": "Doe's Company",
4      "communication": [
5          {
6              "type": "email",
7              "use": true,
8              "setting": "jhon.operator@mail.com"
9          },
10     ],
11     "technologies": [
12         {
13             "name": "forticamera",
14             "version": "3.4.1",
15             "vendor": "fortinet"
16         },
17     ]
18     "cves": [
19         {
20             "date": "2024-06-10",
21             "cves": [
22                 {
23                     "id": "CVE-2024-1234",
24                     "technology": "forticamera",
25                     "version": "3.4.1",
26                     "critically": "high",
27                     "status": "open",
28                     "description": [
29                         {
30                             "lang": "en",
31                             "value": "FortiCamera 3.4.1 is
32                                 vulnerable to a remote code
33                                 execution vulnerability in the
34                                 web interface."
35                         }
36                     ],
37                 }
38             ],
39         },
40     ],
41 }
```

Figura 10: Ejemplo de la colección de compañía
Fuente: Elaboración propia.

3.5. Parser

Los parsers están conformados por múltiples *ETLs* (Extracción, Transformación y Carga), encargados de recolectar información desde cuatro fuentes clave como **FORTINET**, **CISCO**, **MITRE** y **NIST**. Para ello, se utilizan técnicas de web scraping que permiten extraer los datos directamente desde los sitios web correspondientes. Estos módulos fueron desarrollados en Python, haciendo uso de librerías como BeautifulSoup y Scrapy, las cuales facilitan la lectura y navegación del **DOM** de las páginas web [Enrika Pavlovskytė, 2023]. La información obtenida es posteriormente procesada y almacenada en una base de datos *NoSQL*, organizada en tres colecciones distintas, que reflejan las fuentes desde las cuales fue recopilada.

3.5.1. Modelo planificado para el CVE

Dado que las fuentes utilizadas para obtener los datos de los CVEs presentan estructuras heterogéneas, resulta necesario unificar su representación mediante un formato común de almacenamiento. Considerando el uso de *MongoDB* como sistema de base de datos, se optó por definir una estructura orientada a documentos en formato *JSON*, dado que este permite representar la información de manera simple y flexible. Además, su amplia adopción en entornos web lo convierte en una alternativa adecuada para facilitar la integración y manipulación de datos dentro del sistema.

Al definir los datos a almacenar, se procuró priorizar la inclusión de aquellos elementos que resultan más relevantes y recurrentes en los registros de CVE. Asimismo, con miras a una posible expansión futura del sistema, se consideró conveniente incorporar la mayor cantidad de información potencialmente útil para enriquecer la solución propuesta. Dado que los datos provienen de múltiples fuentes con estructuras diferentes, fue necesario identificar tanto los elementos comunes como aquellos atributos exclusivos de cada fuente, con el objetivo de construir una estructura de datos unificada, coherente y extensible.

Todas las fuentes consideradas contienen los atributos principales asociados a los CVE, tales como el identificador, la descripción, el CPE, el puntaje CVSS, y las fechas de publicación y modificación de la vulnerabilidad. No obstante, la fuente proporcionada por *NIST* incorpora información adicional de alto valor, como métricas de estado, referencias externas, entre otros elementos relevantes. Por esta razón, se optó por definir una estructura común de datos basada en el formato utilizado por *NIST*, con el propósito de estandarizar los registros recopilados. A continuación, se presenta el documento resultante.

Figura 11: Modelo de CVE
Fuente: Elaboración propia.

3.5.2. ETLs

Debido a la cantidad de formatos presentes en las fuentes utilizadas, tales como HTML, XML y JSON, se consideró necesario implementar un conjunto de procesos basados en el patrón adaptador, con el objetivo de normalizar los datos y almacenarlos en la estructura presentada anteriormente 11.

En el caso de las fuentes **FORTINET** y **CISCO**, la información se encuentra estructurada en formato **HTML**, lo que requiere aplicar técnicas de *web scraping* para extraer los datos relevantes. Para abordar esta tarea, se seleccionó el framework *Scrapy*, debido a su capacidad para acceder al contenido de las etiquetas HTML mediante selectores CSS, expresiones *XPath* y otras herramientas integradas que facilitan la automatización del proceso de recolección.

La fuente **MITRE** publica un archivo en formato **XML**, el cual se actualiza diariamente, ya sea mediante la incorporación de nuevos (CVEs) o con la modificación del estado de registros existentes. Para procesar esta información, se seleccionó la librería *Beautiful Soup*, que permite navegar y extraer los nodos XML relevantes de manera eficiente. Los datos obtenidos son posteriormente transformados al formato previamente definido, con el propósito de integrarlos al sistema de almacenamiento.

Por otro lado, la fuente proporcionada por **NIST**, expone la información a través de una **API pública**, permitiendo acceder directamente a los datos estructurados en formato **JSON**, eliminando la necesidad de realizar transformaciones complejas como en los casos anteriores.

Estos *ETLs* permiten procesar cada fuente de forma independiente, con el fin de asegurar que toda la información extraída se ajuste al modelo común definido previamente para los documentos **CVE** almacenados en la base de datos.

Finalmente, los procesos *ETL* se ejecutan de forma programada cada hora, con el objetivo de realizar la extracción continua de información y garantizar que la base de datos se mantenga actualizada con los datos más recientes disponibles. Adicionalmente, cada 24 horas se realiza la comunicación de los **CVEs** a los usuarios, con el fin de mantenerlos informados sobre las vulnerabilidades que afectan a sus tecnologías. El envío de esta información se lleva a cabo mediante *RabbitMQ*, configurado bajo el patrón *publish/subscribe*, donde los *parsers* actúan como emisores de los mensajes y el servicio *monitor* como suscriptor, utilizando un *topic* para el enrutamiento de los datos [RabbitMQ, 2024]. Esta configuración permite establecer una comunicación eficiente y escalable entre los distintos componentes del sistema, facilitando su integración y el procesamiento distribuido de la información.

La arquitectura correspondiente al servicio parser queda de la siguiente manera:

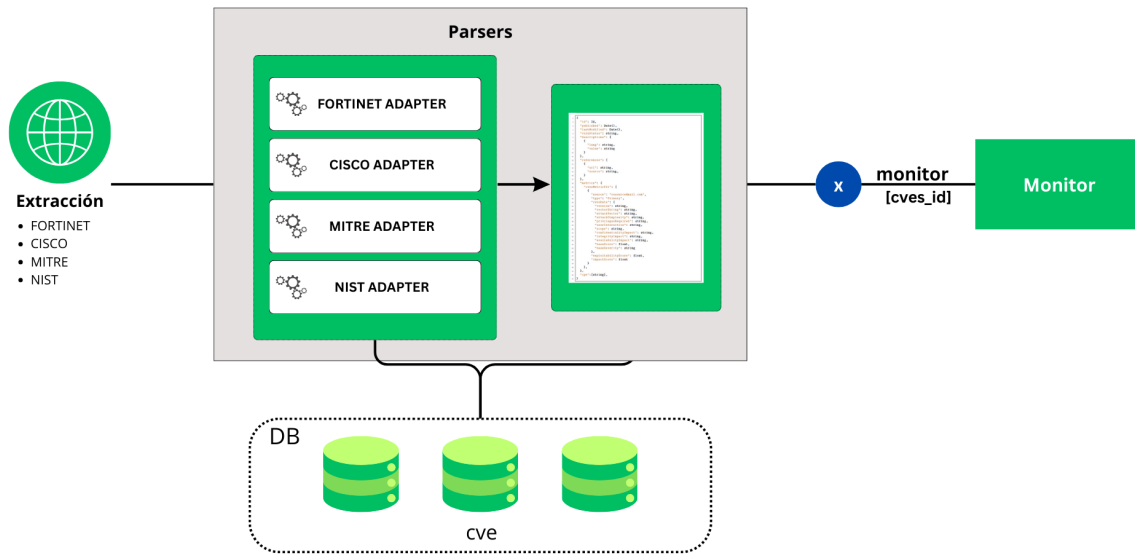


Figura 12: Arquitectura de los parseadores.
Fuente: Elaboración propia.

En el Anexo 5.1 se presentan los distintos formatos a través de los cuales se obtiene la información relacionada con los CVEs.

3.6. Monitor

Este servicio tiene como propósito gestionar la información proveniente del componente *Parser*. Para ello, recibe una lista de identificadores de **CVE** a través de **RabbitMQ**, los cuales son posteriormente procesados y comparados con las tecnologías registradas por cada compañía. En caso de detectar coincidencias, los datos relevantes son almacenados en la base de datos para su posterior análisis y consulta.

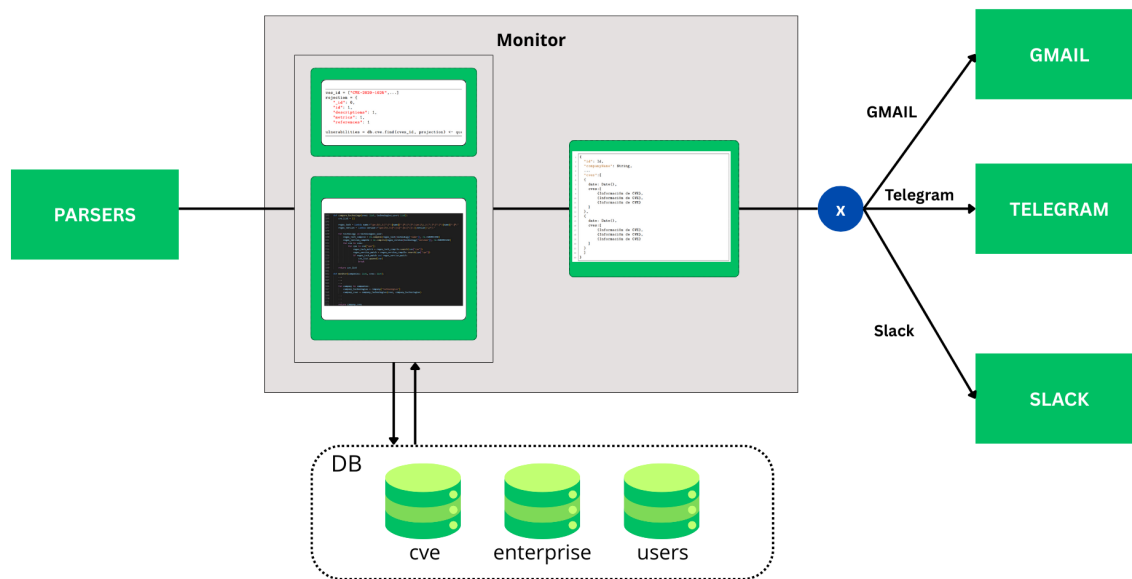


Figura 13: Arquitectura de los parseadores.
Fuente: Elaboración propia.

Para su realización, se disponen con 3 clases principales, encargada del manejo de la información, Monitor, CVEManagerForUser y Sender.

3.6.1. CVEManagerForUser

Clase encargada de gestionar y verificar la información de los CVEs para cada usuario, proporcionando una estructura para su almacenamiento en la base de datos, una estructura para el envío de información a los canales de comunicación, y validando si es que la tecnología del usuario es vulnerable. Esta clase recibe como parámetros un CVE, y una tecnología asociada a la compañía, además contiene tres métodos principales que serán explicados a continuación:

- **get_if_system_vulnerable():** Método encargado de verificar si la tecnología de una compañía es vulnerable, mediante la comparación entre la versión de la tecnología y la versión afectada por la vulnerabilidad. En caso de que la tecnología sea vulnerable, el método retorna un valor booleano *True*; de lo contrario, retorna *False*.

```
1 def get_if_system_vulnerate(self, cpe: Dict[str, Any]) -> bool:
2
3     if not cpe or not isinstance(cpe, dict):
4         self.logger.warning("Invalid CPE data provided")
5         return False
6
7     version_str = self.technology["version"]
8
9     try:
10         current_ver = version.parse(version_str)
11         if not self._check_version_lower_bounds(current_ver, cpe):
12             return False
13
14         if not self._check_version_upper_bounds(current_ver, cpe):
15             return False
16
17         return True
18
19     except version.InvalidVersion:
20         if version_str == "*":
21             return True
22
23         self.logger.warning(f"Invalid version format: {version_str}")
24
25         return False
26     except Exception as e:
27         self.logger
28             .error(f"Error comparing versions: {e}", exc_info=True)
29         return False
```

Figura 14: Código del método `get_if_system_vulnerate`

Fuente: Elaboración propia.

- `create_cve_for_user()`: Método encargado de crear la estructura para el almacenamiento de los CVEs en la colección del usuario. Esta valida a través de ciertas *keys*, si la información obtenida por el CVE y la tecnología es correcta, y en caso de serlo, retorna un *diccionario* con la información respectiva.

```
1 def create_cve_for_user(self) -> Optional[Dict[str, Any]]:
2     required_cve_keys = ["id", "descriptions"]
3     required_tech_keys = ["name", "version"]
4
5     if not self._validate_required_keys(required_cve_keys,
6         required_tech_keys):
7         return None
8
9     try:
10         return {
11             "id": self.cve["id"],
12             "technology": self.technology["name"],
13             "version": self.technology["version"],
14             "critically": self.cve["criticality"],
15             "status": self.cve["status"],
16             "description": self.cve["descriptions"],
17             "lastModified": self.cve["lastModified"],
18             "published": self.cve["published"],
19             "POC": {
20                 "available": True,
21                 "urls": self.cve["references"]
22             },
23         }
24     except Exception as e:
25         self.logger.error(f"Error creating CVE for user: {e}", exc_info=
            True)
26         return None
```

Figura 15: Código del método create_cve_for_user

Fuente: Elaboración propia.

- **cve_for_send()**: Método encargado de crear la estructura para el envío de información a los canales de comunicación.

```
1 def cve_for_send(self) -> Optional[Dict[str, Any]]:
2
3     required_cve_keys = ["id", "descriptions", "references"]
4     required_tech_keys = ["name", "version"]
5
6     if not self._validate_required_keys(
7         required_cve_keys,
8         required_tech_keys):
9         return None
10
11     try:
12         cve_sender = {
13             "cveId": self.cve["id"],
14             "descriptions": self._get_first_description()
15         }
16
17         complexity, score = Cve.get_cvss(self.cve)
18
19         return {
20             "cve": cve_sender,
21             "technology": self.technology,
22             "complexity": complexity,
23             "score": score,
24             "references": self.cve["references"]
25         }
26
27     except Exception as e:
28         self.logger
29             .error(f"Error preparing CVE for sending: {e}", exc_info=True
30             )
31         return None
```

Figura 16: Código del método *cve_for_send*
Fuente: Elaboración propia.

3.6.2. Sender

Clase encargada de enviar la información a los canales de comunicación, conectándose directamente a *RabbitMQ*. Los métodos `setup` y `setup_exchange` corresponden a la configuración encargada de la conexión al bróker, en cambio, el método `publish_topic` es el encargado de enviar la información a los canales de comunicación, conectandose al exchange que distribuye la información a las *routing keys* respectivas, redirigiendo la información al topic correspondiente [RabbitMQ, 2024]. El Sender actúa como el emisor de los mensajes al servicio de comunicación respectivo.

```
1 class Sender:
2
3     def __init__(self, hostname: str, username: str, password: str):
4         self.__channel = None
5         self.__connection = None
6         self.__exchange = None
7         self.setup(hostname, username, password)
8
9     def setup(self, hostname: str, username: str, password: str):
10        credentials = pika
11            .PlainCredentials(username=username, password=password)
12
13        self.__connection = pika
14            .BlockingConnection(pika.ConnectionParameters(
15                host=hostname, credentials=credentials))
16
17        self.__channel = self.__connection.channel()
18
19    def setup_exchange(self, **kwargs):
20        self.__channel.exchange_declare(**kwargs)
21
22    def publish_topic(self, exchange_name: str, routing_key: str, data):
23
24        logging
25            .info(f'RABBITMQ, Sending data to exchange {exchange_name}
26                with routing key: {routing_key}')
27
28        self.__channel.basic_publish(exchange=exchange_name,
29                                    routing_key=routing_key,
30                                    body=data)
31
32        self.__connection.close()
```

Figura 17: Código de la clase Sender
Fuente: Elaboración propia.

- **total1**, asociado a la cantidad de **vulnerabilidades**.
- **cves**, que consisten en una lista de vulnerabilidades obtenidas durante el día.
- **users**, corresponde a los usuarios del sistema.
- **source**, asociado a la fuente de donde se obtuvo la información.

Este método itera sobre los *CPEs* asociados a cada *CVE* y, mediante el método estático `_extract_cpe_info` 18, extrae el proveedor, la tecnología y la versión de la vulnerabilidad utilizando una expresión regular. En caso de no encontrar coincidencias, dicho método retorna el valor `None` para cada uno de los campos.

Figura 18: Código del método `_extract_cpe_info`
Fuente: Elaboración propia.

```
1 def _compare_user_technologies(self, technologies: List[Dict]) -> Tuple[
2     List[Dict], List[Dict]]:
3     cves_user = []
4     data_send = []
5
6     user_tech_map = {
7         (tech["vendor"].lower(), tech["name"].lower()): tech
8     }
9
10    for cve in self.cves:
11        if not cve.get("cpes"):
12            continue
13
14        for cpes in cve["cpes"]:
15            for cpe in cpes:
16                vendor, technology_cve, version_cpe = self.
17                    _extract_cpe_info(cpe["criteria"])
18                if not all([vendor, technology_cve, version_cpe]):
19                    continue
20
21                user_tech = user_tech_map.get((vendor.lower(),
22                    technology_cve.lower()))
23                if not user_tech:
24                    continue
25
26                version_tech_user = user_tech["version"]
27                cve_manager = CVEManagerForUser(cve, user_tech)
28
29                if version_tech_user == "*" or version_cpe in ["*", "-"]:
30                    if cve_manager.get_if_system_vulnerable(cpe):
31                        cves_user.append(cve_manager.create_cve_for_user
32                            ())
33                        data_send.append(cve_manager.cve_for_send())
34
35    return cves_user, data_send
```

Figura 19: Código del método `_compare_user_technologies`
Fuente: Elaboración propia.

Luego, a través de la clase `CVEManagerForUser`, se encarga de verificar si la tecnología del usuario es vulnerable o no, y en caso de serlo, almacena la información en la colección de la compañía en la lista `cves_user`, y crea la estructura para el envío de información a los canales de comunicación en la lista `data_send`. Finalmente, a través del método `_process_user`, se envía la información a la base de datos y a los canales de comunicación respectivos, utilizando la clase `Sender`.

```
1 def _process_user(self, user: Dict[str, Any]) -> None:
2     """Procesa un usuario individual"""
3     cves_user, data_send = self._compare_user_technologies(user["
4         technology"])
5
6     if not cves_user:
7         return
8
9     self._update_user_cves(user, cves_user)
10    self._send_user_notifications(user, data_send)
```

Figura 20: Código del método `_process_user`
Fuente: Elaboración propia.

3.7. Canales de comunicación

Los canales de comunicación corresponden a servicios diseñados para facilitar la interacción entre el sistema y los usuarios. Su propósito es informar, de manera automatizada, sobre los CVEs registrados durante el día que puedan estar afectando a las tecnologías utilizadas por cada empresa. Estos canales son configurables, permitiendo adaptarse a los medios de comunicación preferidos por los usuarios. Actualmente, se encuentran disponibles tres opciones, *Telegram* [Telegram, 2024], *Slack* [Slack API Contributors, 2024] y correo electrónico.

Para el envío de notificaciones por correo electrónico, se implementó el protocolo *SMTP* (*Simple Mail Transfer Protocol*), estándar utilizado para enviar correo electrónicos a través de internet [Klensin, 2008]. Por su parte, los canales de *Telegram* y *Slack* fueron desarrollados mediante las librerías oficiales que cada plataforma proporciona, permitiendo así una integración directa y eficiente con el sistema.

Estos servicios reciben, a través de RabbitMQ, los CVEs que afectan a las tecnologías de la empresa, junto con la información de los usuarios que deben ser notificados. La gestión de los mensajes se realiza aprovechando las ventajas que ofrece RabbitMQ, el cual sigue el enfoque *FIFO* (*First In, First Out*), permitiendo establecer una comunicación confiable y eficiente. Se optó por utilizar estas tecnologías debido a su fácil implementación en los sistemas existentes y a su amplia adopción en el mercado.

CAPÍTULO 4

VALIDACIÓN DE LA SOLUCIÓN

En el capítulo anterior, se abarcaron 4 objetivos específicos definidos en nuestra definición del problema, los cuales estaban enfocados en las definiciones de diseño y creación de nuestro software. Este capítulo se centra en la validación funcional del sistema **CVE-Hunter**, contrastando su desempeño con el objetivo específico de evaluar su precisión, continuidad operativa y eficacia en la notificación de vulnerabilidades reales.

Para llevar a cabo el proceso de validación del software, se contó con la colaboración de la empresa **Develrox**, especializada en consultoría en seguridad de la información. Esta organización facilitó los recursos necesarios para la validación, incluyendo infraestructura tecnológica y apoyo profesional. El equipo estuvo conformado por dos especialistas, un ingeniero senior con el rol de *Pentester*, encargado de la revisión del código, y Esteban Vaquero, líder técnico y consultor en ciberseguridad, quien brindó asesoría estratégica durante el proceso.

4.1. Entorno de trabajo

El entorno de trabajo consistió en un servidor dedicado con las siguientes características:

Sistema operativo	Ubuntu 22.04.2 LTS
Almacenamiento	256Gb SSD
Ram	32Gb
Procesador	Intel(R) Core(TM) i7-4790K CPU @ 4.00GHz
Arquitectura	x86_64

Tabla 1: Características del servidor
Fuente: Elaboración Propia.

4.2. Alcance y aclaraciones

- *CVE-Hunter* no es un gestor de vulnerabilidades, ya que este proceso puede ser complicado de desarrollar, debido a que la mitigación de los CVEs dependen de varios factores, los cuales pueden llevar mucho tiempo de resolución, como de los parches desarrollados por las fuentes oficiales, o también del soporte que tienen las tecnologías vulneradas.
- Para validar el sistema, se llevó a cabo un proceso de estudio y evaluación entre el **14 de mayo de 2025** y el **15 de junio del mismo año**, con el objetivo de analizar la

capacidad del sistema para detectar CVEs reales utilizando el conjunto de tecnologías que se lista a continuación 2. La selección de estas tecnologías busca representar los sistemas críticos y de uso frecuente en entornos corporativos, lo que asegura que los resultados de la validación sean pertinentes para escenarios de seguridad reales. Para **Develrox**, se hace necesario esta selección, ya que al ser una empresa dedicada a la consultoría de seguridad, uno de los puntos de análisis suelen estar en servidores y herramientas de código abierto.

Tecnología	Versión	Proveedor
Exchange server	Todas las versiones	Microsoft
Secure Email And Web Manager	14.0	Cisco
Linux Kernel	Todas las versiones	Linux
Windows Server 2022	Todas las versiones	Microsoft
Windows Server 2019	Todas las versiones	Microsoft
Http Server	2.3	Apache
Kylin	3.1.0	Apache
Vm ware player	Todas las versiones	VM Ware
Fortios	7.4.2	Fortinet
Open ssh	9.2	openbsd
phpmailer project	6.4.1	phpmailer
OpenVPN	2.6.6	OpenVPN

Tabla 2: Conjunto de tecnologías entregadas por *Develrox*
Fuente: Elaboración Propia.

- Es importante aclarar que *Develrox* no utilizaba algunos canales de comunicación, como *Slack* y *Telegram*, por lo que se optó utilizar *Gmail* como canal de comunicación para la notificación de vulnerabilidades. De igual manera, en los anexos, se evidencia el envío de información del sistema a través de los dos canales mencionados 5.3.

4.3. Criterios de éxito del sistema

Para considerar que el sistema ha sido validado exitosamente y que cumple con el objetivo general propuesto, se establecieron los siguientes criterios de evaluación durante el período de estudio:

1. **Precisión y relevancia en la detección de CVEs:** El sistema debe ser capaz de identificar vulnerabilidades que afecten efectivamente a las tecnologías y versiones monitoreadas. Con el fin de validar esta capacidad, se realizará una revisión manual sobre una muestra aleatoria de las CVEs detectadas, verificando que la correlación entre vulnerabilidad y tecnología sea precisa y consistente con la información pública disponible.

2. **Monitoreo continuo:** El sistema debe ejecutar sus procesos de extracción, análisis y notificación de forma automática y periódica, sin necesidad de intervención humana durante el período de validación.
3. **Eficacia en las notificaciones a usuarios:** Las alertas generadas por el sistema deben ser claras, completas y entregadas de manera constante a través del canal configurado, permitiendo a los usuarios comprender rápidamente los riesgos detectados.

4.4. Metodología de validación

Para evaluar el cumplimiento de los criterios, se implementó la siguiente metodología:

1. Se ejecutó el sistema de forma continua durante el período de estudio definido. La automatización se realizó mediante una tarea programada (*cron job*), encargada de ejecutar el *script* principal del sistema diariamente.
2. Se monitorearon las alertas generadas por el sistema y se registraron las vulnerabilidades detectadas.
3. Se realizaron verificaciones aleatorias de la precisión de las vulnerabilidades reportadas, contrastándolas con la National Vulnerability Database (NVD) y validando su aplicabilidad con el equipo de ciberseguridad de Develrox cuando fue necesario.
4. Se confirmó la recepción y el formato adecuado de las notificaciones enviadas a través del canal de comunicación configurado, asegurando la correcta entrega y comprensión de la información reportada.

4.5. Resultados

Durante el período de tiempo mencionado, el sistema detectó un total de **7503** vulnerabilidades, de las cuales **242** estaban asociadas a las tecnologías entregadas por Develrox. Estas detecciones permitieron validar la funcionalidad del sistema al identificar amenazas relevantes para el entorno analizado.

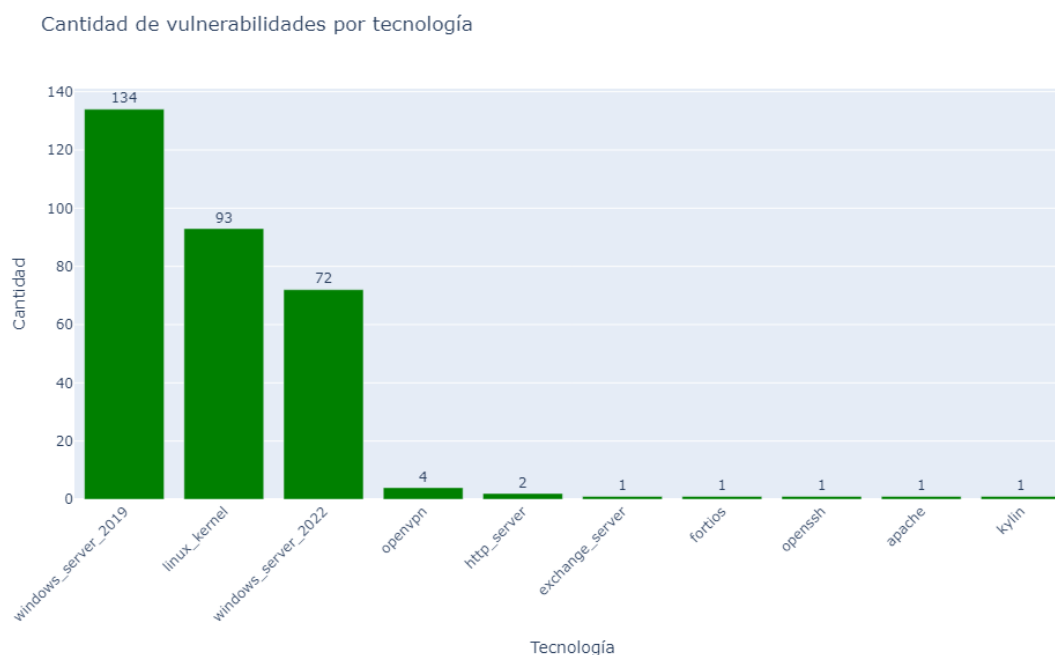


Figura 21: Cantidad de vulnerabilidades encontradas por tecnología
Fuente: Elaboración propia.

Analizando el gráfico 21, se evidencia que las tecnologías con mayor cantidad de vulnerabilidades son *Windows_server_2019*, *Windows_server_2022*, *Linux_kernel*. Esta tendencia se explica porque se realizó una cobertura amplia de vulnerabilidades, recopilando CVEs correspondientes a todas las versiones disponibles de dichas tecnologías. Además, es importante señalar que para las tecnologías de *Microsoft*, existían vulnerabilidades que compartían el mismo **identificador**. En contraste, las demás tecnologías presentaron una menor cantidad de CVEs, ya que tenían asociada una versión específica para su detección. En algunos casos no se detectaron vulnerabilidades, ya que durante el período de análisis, no se publicaron CVEs asociados a esas tecnologías y versiones.

Distribución de Severidad de Vulnerabilidades

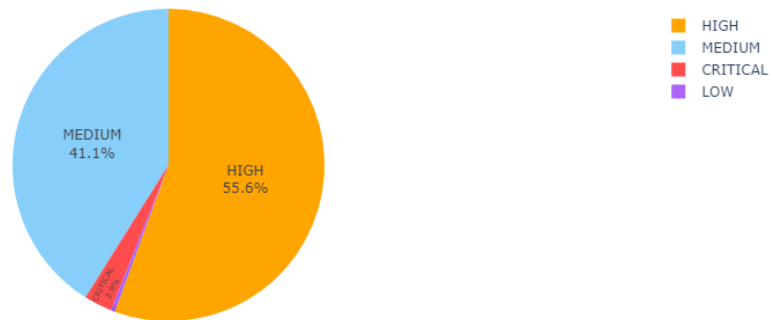


Figura 22: Distribución de vulnerabilidades por severidad
Fuente: Elaboración propia.

En la Figura 22, y conforme a los niveles de severidad definidos por la **CVSS 2.1.1**, se evidencia que la mayoría de las vulnerabilidades detectadas presentan una criticidad **ALTA (HIGH)**. Esto implica que se trata de fallos con un elevado potencial de impacto sobre la seguridad de los sistemas afectados, especialmente si no son abordados oportunamente. Este resultado respalda la capacidad del sistema para identificar amenazas relevantes y con alto grado de severidad, lo que refuerza su valor como herramienta de apoyo de vulnerabilidades.

4.5.1. Validación de la Precisión de Detección

Para evaluar la precisión con la que el sistema identifica y asocia las vulnerabilidades a las tecnologías correspondientes, se llevó a cabo una verificación manual sobre una muestra aleatoria de 12 CVEs, seleccionados entre los 242 casos detectados como relevantes para la empresa **Develrox**. La muestra fue construida mediante un proceso de **muestreo estratificado**, seleccionando un CVE representativo por cada tecnología identificada, y complementando la selección con dos CVEs adicionales correspondientes a las tecnologías con mayor volumen de detecciones: *Windows_server_2019*, *Windows_server_2022* y *linux_kernel*.

Para cada uno de los CVEs seleccionados, se consultó la *National Vulnerability Database* con el objetivo de verificar tanto la existencia de la vulnerabilidad como la correspondencia entre la tecnología y la versión afectada, según la información publicada por la fuente oficial.

Como resultado de este análisis, se constató que **10 de las 12 vulnerabilidades fueron clasificadas correctamente**, lo que representa una **precisión del 83.3 %**. En los dos casos restantes (CVE-2024-41057 y CVE-2024-56655, ambos relacionados con *Linux Kernel*), no se pudo establecer una coincidencia exacta de versiones, debido a que la información proporcionada

por la NVD se presenta con rangos de versiones específicos que, en ocasiones, no se alinean con la clasificación más general utilizada por el sistema. No obstante, en ambos casos la asociación a la familia tecnológica fue correcta.

Este proceso de validación permite corroborar que el sistema cuenta con una alta capacidad para identificar y asociar con precisión las vulnerabilidades más relevantes, aun cuando la granularidad de la información varíe entre las distintas fuentes de datos.

La Tabla 3 resume los resultados de la verificación manual para la muestra seleccionada.

CVE-ID	Tecnología/Versión (Sistema)	Versiones Afectadas (NVD)	Concordancia
CVE-2024-41057	Linux Kernel v*	[5.17 a 6.1.101], [6.2 a 6.6.42], [6.7 a 6.9.11]	Parcial
CVE-2024-56655	Linux Kernel v*	[6.6.61 a 6.6.67], [6.11.8 a 6.12.6]	Parcial
CVE-2019-1023	Windows Server 2019 v*	Todas las versiones	Correcto
CVE-2019-0941	Windows Server 2019 v*	Todas las versiones	Correcto
CVE-2022-34703	Windows Server 2022 v*	Todas las versiones	Correcto
CVE-2025-29835	Windows Server 2022 v*	Todas las versiones antes de la 10.0.20348.3692	Correcto
CVE-2021-40438	Http Server v2.3.0	Todas las versiones hasta la 2.4.48 incluyendola	Correcto
CVE-2022-24697	Apache Kylin v3.1.0	[2.0.0 a 2.6.6], [3.0.0 a 3.1.2], [4.0.0 a 4.0.1]	Correcto
CVE-2023-48795	Open SSH v9.2.0	Todas las versiones antes de la 9.6	Correcto
CVE-2023-46849	Open VPN v2.6.6	[2.6.0 a 2.6.6], [2.11.0 a 2.11.3]	Correcto
CVE-2022-34692	Exchange Server v*	Todas las versiones	Correcto
CVE-2025-47295	Fortios v7.4.2	[6.4.0 a 7.0.15], [7.2.0 a 7.2.8], [7.4.0 a 7.4.4]	Correcto

Tabla 3: Muestra de CVEs verificados manualmente y su precisión.

Fuente: Elaboración propia.

4.5.2. Monitoreo Continuo

Como se indica en el segundo criterio de éxito 2, el monitoreo continuo fue implementado mediante el uso de *crontab*, una herramienta propia de sistemas basados en Unix que permite programar la ejecución automática de tareas [The Linux man-pages project, 2024]. En este contexto, se configuraron dos funciones del servicio *Parser*, una destinada a recolectar los CVEs más recientes y otra encargada de enviar dicha información al servicio *Monitor*.

Esta configuración permite que el sistema opere de manera autónoma, sin requerir intervención manual. Las tareas se ejecutan de forma periódica y precisa, a través de un mecanismo liviano con un consumo mínimo de recursos.

La automatización del proceso no solo optimiza la eficiencia operativa, sino que también garantiza un monitoreo constante de nuevas amenazas, lo que refuerza la capacidad del sistema para responder de forma oportuna ante la aparición de vulnerabilidades emergentes.

De	Cualquier momento	Contiene archivos adjuntos	Para	No leídos	Búsqueda avanzada
☐				Más recientes	1-50 de muchas
☐	☆	noreply-cve	Daily Report	Reporte del día 2 nuevas vulnerabilidades que podrían afectar la seguridad de su empresa 2 tie...	10 jun
☐	☆	noreply-cve	Daily Report	Reporte del día 2 nuevas vulnerabilidades que podrían afectar la seguridad de su empresa 2 tie...	8 jun
☐	☆	noreply-cve	Daily Report	Reporte del día 1 nuevas vulnerabilidades que podrían afectar la seguridad de su empresa 1 tien...	7 jun
☐	☆	noreply-cve	Daily Report	Reporte del día 61 nuevas vulnerabilidades que podrían afectar la seguridad de su empresa 3 tie...	5 jun
☐	☆	noreply-cve	Daily Report	Reporte del día 15 nuevas vulnerabilidades que podrían afectar la seguridad de su empresa 15 ti...	3 jun
☐	☆	noreply-cve	Daily Report	Reporte del día 21 nuevas vulnerabilidades que podrían afectar la seguridad de su empresa 18 ti...	31 may
☐	☆	noreply-cve	Daily Report	Reporte del día 41 nuevas vulnerabilidades que podrían afectar la seguridad de su empresa 2 tie...	30 may
☐	☆	noreply-cve	Daily Report	Reporte del día 6 nuevas vulnerabilidades que podrían afectar la seguridad de su empresa 5 tie...	29 may
☐	☆	noreply-cve	Recibidos Daily Report	Reporte del día 25 nuevas vulnerabilidades que podrían afectar la seguridad de su e...	28 may
☐	☆	noreply-cve	Recibidos Daily Report	Reporte del día 24 nuevas vulnerabilidades que podrían afectar la seguridad de su e...	27 may

Figura 23: Ejemplos de la recepción de algunos correos
Fuente: Elaboración propia.

4.5.3. Eficacia en la notificación a Usuarios

Debido a la alta cantidad de vulnerabilidades registradas, se optó por validar la funcionalidad del sistema de notificaciones utilizando un caso representativo ocurrido el día **6 de junio de 2025**. En dicha fecha, el sistema identificó dos nuevas vulnerabilidades e informó al usuario que sus tecnologías *http_server v2.3.0* y *windows_server_2019 v** podían estar expuestas. La notificación enviada por correo electrónico se estructuró en dos secciones principales, las cuales se detallan a continuación:

■ **Sección 1:**

En la imagen 24, se visualiza información resumida acerca de las nuevas vulnerabilidades detectadas, incluyendo la cantidad de CVEs identificados y su nivel de severidad, destacando aquellas clasificados como **Alta** o **Crítica**. Además, se incorpora una recomendación dirigida al usuario, orientada a la adopción de medidas preventivas que permitan mitigar eventuales riesgos de explotación en los sistemas afectados.



Figura 24: Encabezado y información de resumen en la notificación.

Fuente: Elaboración propia.

■ **Sección 2:**

En esta sección se presenta información detallada sobre cada CVE detectado, destacando los elementos más relevantes, como la descripción de la vulnerabilidad, el puntaje CVSS, la tecnología afectada y el identificador correspondiente. Asimismo, se incluye un enlace de referencia que redirige directamente a la fuente oficial del CVE, en caso de que el usuario requiera consultar información complementaria.

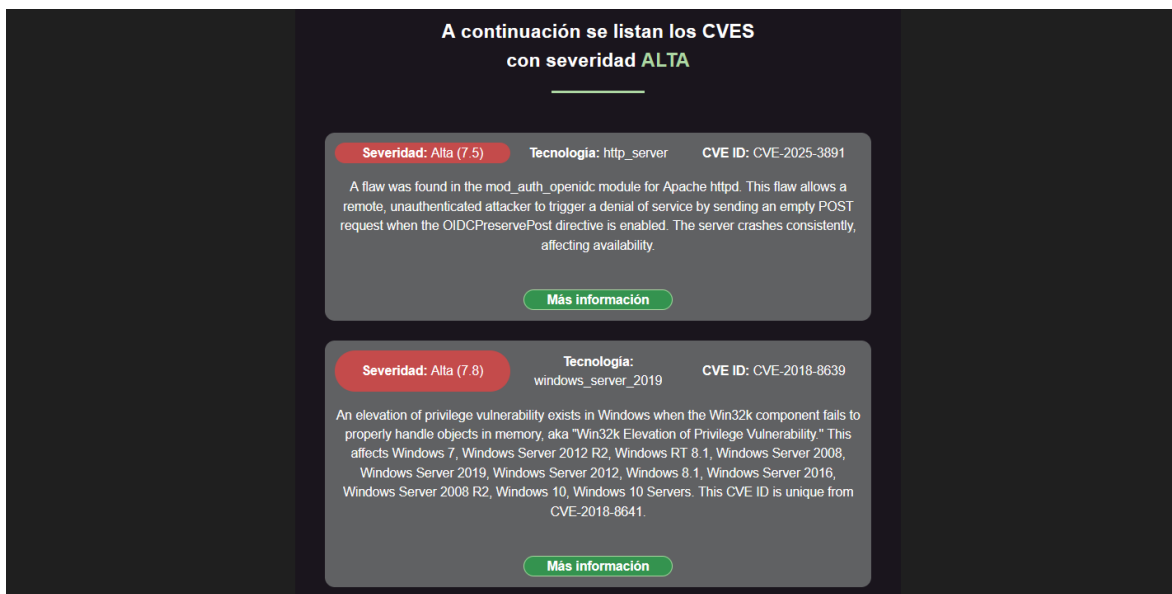


Figura 25: Cuerpo y información de CVEs

Fuente: Elaboración propia.

Este mecanismo de notificación facilitó al equipo la identificación de riesgos asociados a cada tecnología, proporcionando de manera clara y accesible la información sobre las vulnerabilidades que afectaban a sus sistemas. Gracias a este enfoque, se evitó la necesidad de realizar búsquedas manuales o recurrir a métodos activos de recopilación, optimizando así el tiempo de detección.

Según Esteban Vaquero de Develrox, *“Un sistema de notificación como este permite tener de manera rápida las posibles vulnerabilidades que surgen durante el día, obteniendo directamente desde fuentes oficiales las vulnerabilidades que afectan a nuestros sistemas, teniendo información de mitigación y el estado real en el cual se encuentra la vulnerabilidad.”*

4.6. Limitaciones y problemas detectados

A pesar del adecuado funcionamiento del software en la detección de **CVEs**, se identificaron algunas limitaciones durante el período de validación, las cuales se detallan a continuación:

- En ciertas ocasiones, no se detectaron **CVEs** debido a que su estado se encontraba como **Awaiting** o **Esperando**, lo que indica que el identificador ha sido reservado, pero sus detalles completos aún no han sido publicados. Esta situación impide que algunas versiones potencialmente vulnerables estén disponibles en las bases de datos al momento del análisis. Para mitigar esta limitación, se recomienda considerar dichos

estados durante la recopilación de datos, implementando un monitoreo constante de su evolución y un método de evaluación que analice las descripciones mediante contención de caracteres.

- El envío de información no se realiza en tiempo real, sino que ocurre una vez al día. Esto se debe a que la recolección y el análisis de vulnerabilidades requieren múltiples consultas a diversas fuentes, algunas de las cuales imponen restricciones de uso o límites de acceso. Además, ejecutar el análisis de forma continua podría incrementar considerablemente la carga computacional, comprometiendo así la eficiencia del sistema. Para abordar esta situación, se propone implementar un mecanismo de **priorización y notificación incremental**, que permita el envío inmediato de vulnerabilidades clasificadas con criticidad alta o crítica. Complementariamente, se sugiere un mecanismo para la **gestión eficiente de recursos**, incorporando soluciones como balanceadores de carga y rotación de APIs.
- Se detectó una duplicación en las notificaciones de ciertas vulnerabilidades, atribuida a que algunas tecnologías, como *Windows Server*, compartían identificadores de CVE. Este comportamiento corresponde a un error en el código, cuya mitigación se implementó mediante una mejora en los filtros aplicados durante el procesamiento de la información.
- Si bien no constituye un problema funcional, se identificó una limitante en la validación de los canales de comunicación como **Telegram** y **Slack**, la cual se realizó de forma independiente a la empresa mediante pruebas locales, dado que dichos canales no son utilizados por Develrox.

CAPÍTULO 5

CONCLUSIONES

La implementación de herramientas destinadas a la detección de vulnerabilidades puede reducir significativamente los riesgos de ciberseguridad a los que se enfrentan las organizaciones. Estas herramientas permiten identificar vulnerabilidades de manera oportuna, previniendo la obsolescencia del software, la desactualización tecnológica, la pérdida de información sensible, la fuga de clientes y las consiguientes pérdidas económicas. Al entregar alertas tempranas, estos sistemas fortalecen la capacidad de respuesta y la toma de decisiones en materia de seguridad.

Si bien la cultura de la ciberseguridad ha crecido notablemente en nuestro tiempos, es necesario que se desarrolle en todas las áreas respectivas, ya que a medida que la tecnología avanza, también lo hacen los riesgos de ciberataques, lo que subraya la importancia de una educación continua y robusta en este ámbito.

En este contexto, la utilización de *Common vulnerabilities and Exposures (CVE)* resulta de gran valor, debido a que su estandarización de basa en otros estándares reconocidos como CVSS (*Common vulnerability Scoring System*), CWE (*Common Weakness Enumeration*), CPE (*Common Platform Enumeration*), lo que permite ofrecer una cantidad exhaustiva y estructurada de información sobre cada vulnerabilidad.

5.1. Conclusiones generales

Debido a lo anterior, el sistema de notificación *CVE-Hunter* demuestra ser una herramienta valiosa para asistir a los equipos de desarrollo al informar oportunamente sobre CVEs emergentes en sus sistemas. Al centralizar y entregar información relevante, facilita la toma de decisiones de mitigación de forma temprana y efectiva, pudiendo contribuir significativamente a la ciberseguridad de las organizaciones.

Si bien durante su validación *CVE-Hunter* presentó ciertas limitaciones, como la no detección de algunos CVEs debido a su estado (ej., “awaiting” o reservados) o la restricción a canales de comunicación específicos no utilizados por la empresa colaboradora, el sistema demostró ser altamente efectivo en la mayoría de los casos. Esto valida su concepto y funcionalidad principal.

Para mitigar la limitación relacionada con la detección de CVEs por su estado o especificidad de versión, una mejora clave sería ampliar el alcance de las métricas de obtención de vulnerabilidades, esto implicaría no solo verificar la vulnerabilidad a través del CPE, sino también considerar el estado del CVE o la contención de caracteres clave mediante algoritmos de procesamiento de lenguaje natural en la descripción. Esta aproximación permitiría una

identificación más granular y exhaustiva.

Finalmente, *CVE-HUNTER* es recomendable trabajar como complemento a otras herramientas enfocadas en la detección de vulnerabilidades, tanto pasivas como activas, sirviendo como primera línea de defensa en la alerta y detección de posibles vulnerabilidades que pueden afectar a las organizaciones en el futuro.

5.2. Conclusiones por objetivos específicos

- 1. Implementar un sistema de extracción de datos de CVEs desde diferentes fuentes, con el propósito de recolectar información relevante sobre las vulnerabilidades que se registran de forma continua.**

Se implementó el *Parser3.5*, un sistema encargado de la recolección de información desde fuentes como, *National Institute of Standards and Technology (NIST)*, *MITRE*, *Fortinet* y *Cisco*. Este servicio se realizó a través de distintos métodos de *Web Scraping*, permitiendo a través del patrón de diseño adaptador, almacenar la información en un formato común.

- 2. Estructurar la información obtenida desde las distintas fuentes, y integrar en una base de datos unificada, con el fin de facilitar el filtrado de esta.**

Se creó una estructura de información única 11, basada en la fuente de información del *NIST*, que nos permitió crear una estructura común de la información que venía desde distintas fuentes utilizando el patrón adaptador como se mencionó anteriormente.

- 3. Crear un servicio de monitoreo, encargado de asociar la información obtenida de las distintas fuentes, y filtrar en función de las tecnologías utilizadas en los entornos objetivo, con el fin de notificar los CVEs que afectan a sus softwares.**

Se implementó el servicio *Monitor3.6*, encargado de recibir la información recopilada por el servicio parser, y asociarla con la información de las tecnologías que eran utilizadas por el usuario. Nosotros lo definimos como el corazón del sistema, ya que organizaba y estructuraba la información final para el usuario.

- 4. Crear servicios de canales de comunicación, con el objetivo de informar sobre las vulnerabilidades detectadas en sus sistemas.**

Se implementaron tres servicios encargados de comunicar al usuario la información recolectada por el sistema 3.7. A través de *Telegram*, *Slack* y *Gmail*, se notificaba a los usuarios sobre las vulnerabilidades que estaban afectando a sus sistemas. Si bien, solo se pudo realizar pruebas oficiales con el correo electrónico, se realizaron pruebas locales con los otros canales.

- 5. Evaluar la funcionalidad del sistema, a través de tecnologías dadas por una compañía, para validar la efectividad del sistema respecto del objetivo general.**

En el capítulo de validación 3.7 se presenta el trabajo realizado en conjunto con la empresa *Develrox* para evaluar el sistema, utilizando un conjunto de tecnologías que fueron proporcionadas por la misma organización. Esta colaboración permitió comprender con mayor claridad el alcance real del proyecto y evidenciar el nivel de exposición a vulnerabilidades que puede existir durante el desarrollo de software. La validación se llevó a cabo mediante un proceso de revisión aleatoria estratificada, en el cual se seleccionaron distintas vulnerabilidades (CVEs) detectadas en los sistemas de la empresa, y se contrastaron con la base de datos oficial **NVD**, con el objetivo de verificar la precisión de las alertas generadas por el sistema.

5.3. Trabajos a futuros

Gracias a los datos almacenados por el desarrollo, se pueden realizar diferentes funcionalidades que le entregarán mucho más valor al software. Estas funcionalidades pueden consistir en nuevos servicios con distintas características, que serán mencionadas a continuación.

1. Creación de Dashboard de gestión

Esta característica puede consistir en distintas vistas, donde los usuarios puedan interactuar directamente con el sistema. a través de él un usuario, dependiendo de los permisos de accesos que tenga, gestionará las vulnerabilidades que vayan surgiendo en el sistema, asignando tickets a otros usuarios, para que estos se encargen de mitigar y clasificar los **CVEs** que afectan a sus sistemas. Esto provocará una mejor comprensión y control sobre las vulnerabilidades por parte de los clientes.

2. Generación de estadísticas

Consistiría en un servicio encargado de analizar los datos generados por los **Parsers** y **Monitor**. A través de ellos, se analizaría información asociada a las tecnologías utilizadas por el usuario que suelen verse más afectada, como estadísticas asociadas al *score* de la vulnerabilidad, tiempo promedio de mitigación, estadísticas con enfoque temporal, etc. Además de estadísticas dedicadas a las vulnerabilidades con enfoque más general, es decir, no solamente estadísticas de vulnerabilidades presentada en los sistemas del usuario.

3. Generación de API Callbacks

Esta característica consistirá de diferentes *API Callbacks* para que los usuarios a través de sus canales de comunicación, puedan configurar y consumir este sistema, recibiendo información importante sobre sus vulnerabilidades, de manera más personalizada.

Finalmente, se podrían implementar más servicios encargados de la mensajería, donde los usuarios serán los encargados de configurar los canales de comunicación que ellos estimen necesarios, y activarlos cuando sea necesario entregándole un mayor control al usuario sobre sus sistemas y vulnerabilidades.

4. Creación de agentes

Generación de agentes *GPTs*, encargados de la automatización del proceso de recolección de información, interacción con usuarios y automatización en la generación de informes ejecutivos de ***Common Vulnerability and Exposures***.

5.3.1. Palabras finales

La presente tesis representó un desafío considerable. Si bien se comprendía el concepto de CVE, el continuo descubrimiento de nuevas características generaba, en ciertos momentos, confusión durante la construcción del *software*. La claridad se alcanzó después de diversas reuniones con el personal de **Develro**x, quienes proporcionaron explicaciones detalladas y clarificaron el significado y la importancia de cada atributo de un CVE.

La gestión del proyecto resultó ser un factor crítico, permitiendo una mejora progresiva en las estimaciones para las nuevas funcionalidades a implementar. La dirección de la construcción del *software* propició un aprendizaje significativo en áreas como el trabajo con microservicios, la adopción de buenas prácticas, la aplicación de patrones de diseño y la comprensión de reglas de negocio.

Existe un considerable potencial para expandir este *software* más allá de un simple sistema de notificaciones. El desarrollo futuro apunta a un crecimiento a gran escala, transformándolo en una herramienta integral que asista en la gestión y mitigación de vulnerabilidades mediante diversas funcionalidades, tal como se mencionó previamente en la sección de trabajos futuros 5.3.

ANEXOS

5.1. Links asociados a los parsers

A continuación se presentan los links de las fuentes de información de donde se obtuvieron los CVEs.

MITRE: Corresponde a un XML del MITRE. A esta referencia, se le puede cambiar la fecha del archivo.

NVD API: Corresponde a una API del NIST, la cual permite obtener información de vulnerabilidades, a través de distintos *query params*.

XML Fortinet: Archivo XML asociado a fortinet.

Web Fortinet: Web de CVEs de fortinet.

Web Cisco: Web de CVEs de cisco.

5.2. CVEs encontrados por el sistema en el período mencionado

Tecnología y versión	CVE Asociado
Exchange server v*	['CVE-2022-34692']
Secure Email And Web Manager v14.0	-
Http Server v2.3	['CVE-2021-40438', 'CVE-2025-3891']
Apache Kylin 3.1.0	['CVE-2022-24697']
Vm ware player	-
Fortios v7.4.2	['CVE-2025-47295']
Open ssh v9.2	['CVE-2023-48795']
phpmailer project v6.4.1	-
OpenVPN v2.6.6	['CVE-2025-2704']

Linux Kernel v*	['CVE-2025-37803', 'CVE-2024-23848', 'CVE-2025-37800', 'CVE-2025-37802', 'CVE-2024-26739', 'CVE-2024-38577', 'CVE-2025-3942', 'CVE-2024-41057', 'CVE-2021-47037', 'CVE-2024-53209', 'CVE-2022-41218', 'CVE-2023-46838', 'CVE-2022-48735', 'CVE-2022-49168', 'CVE-2022-42717', 'CVE-2024-36903', 'CVE-2024-26952', 'CVE-2024-46751', 'CVE-2024-43863', 'CVE-2024-23851', 'CVE-2022-35721', 'CVE-2025-1330', 'CVE-2021-28715', 'CVE-2024-35790', 'CVE-2022-3303', 'CVE-2022-41674', 'CVE-2024-27018', 'CVE-2022-42721', 'CVE-2025-1329', 'CVE-2025-37801', 'CVE-2024-50083', 'CVE-2024-26783', 'CVE-2024-38667', 'CVE-2024-26618', 'CVE-2023-53025', 'CVE-2022-48425', 'CVE-2025-33136', 'CVE-2022-40748', 'CVE-2022-40277', 'CVE-2022-48740', 'CVE-2024-38664', 'CVE-2024-39479', 'CVE-2022-22423', 'CVE-2022-42719', 'CVE-2024-53203', 'CVE-2024-50203', 'CVE-2025-3937', 'CVE-2024-22705', 'CVE-2023-34324', 'CVE-2024-8207', 'CVE-2024-22099', 'CVE-2022-49063', 'CVE-2025-3943', 'CVE-2024-43840', 'CVE-2025-1331', 'CVE-2022-3103', 'CVE-2024-42093', 'CVE-2025-3940', 'CVE-2025-33138', 'CVE-2022-42720', 'CVE-2025-22062', 'CVE-2023-6270', 'CVE-2024-56655', 'CVE-2024-39277', 'CVE-2025-3938', 'CVE-2022-40274', 'CVE-2024-39291', 'CVE-2024-0340', 'CVE-2024-23849', 'CVE-2022-41222', 'CVE-2022-41848', 'CVE-2022-48733', 'CVE-2024-26807', 'CVE-2025-37805', 'CVE-2025-3939', 'CVE-2025-34028', 'CVE-2023-44466', 'CVE-2024-56758', 'CVE-2024-38541', 'CVE-2024-36477', 'CVE-2024-56662', 'CVE-2024-36941', 'CVE-2023-50431', 'CVE-2024-38581', 'CVE-2025-33137', 'CVE-2025-3928', 'CVE-2022-2778', 'CVE-2023-4387', 'CVE-2024-36927', 'CVE-2025-3944']
-----------------	--

Windows Server 2022 v*	['CVE-2022-35763', 'CVE-2022-35771', 'CVE-2025-29839', 'CVE-2022-34709', 'CVE-2025-27468', 'CVE-2025-29959', 'CVE-2022-34705', 'CVE-2022-35769', 'CVE-2022-35765', 'CVE-2025-29831', 'CVE-2025-32709', 'CVE-2022-34707', 'CVE-2025-24054', 'CVE-2022-35766', 'CVE-2022-34706', 'CVE-2025-29837', 'CVE-2025-30394', 'CVE-2025-29968', 'CVE-2025-29963', 'CVE-2025-29830', 'CVE-2025-32706', 'CVE-2022-35762', 'CVE-2025-30388', 'CVE-2022-34714', 'CVE-2025-29964', 'CVE-2025-29974', 'CVE-2025-29841', 'CVE-2025-21224', 'CVE-2025-29842', 'CVE-2025-29840', 'CVE-2025-29957', 'CVE-2025-29962', 'CVE-2025-29956', 'CVE-2025-26677', 'CVE-2022-34712', 'CVE-2025-29833', 'CVE-2022-34696', 'CVE-2022-35768', 'CVE-2022-35767', 'CVE-2022-34715', 'CVE-2022-35761', 'CVE-2023-35622', 'CVE-2025-29976', 'CVE-2025-29954', 'CVE-2022-34708', 'CVE-2022-34702', 'CVE-2025-29967', 'CVE-2024-21309', 'CVE-2025-29829', 'CVE-2025-29832', 'CVE-2025-24985', 'CVE-2025-29958', 'CVE-2025-24063', 'CVE-2022-34701', 'CVE-2025-29835', 'CVE-2025-29961', 'CVE-2025-30400', 'CVE-2022-34703', 'CVE-2025-27488', 'CVE-2025-29966', 'CVE-2025-32701', 'CVE-2025-29969', 'CVE-2022-34691', 'CVE-2022-34699', 'CVE-2022-35764', 'CVE-2025-29960', 'CVE-2025-29836', 'CVE-2025-30385', 'CVE-2025-30397', 'CVE-2022-34710']
------------------------	--

Windows Server 2019 v*	['CVE-2019-1065', 'CVE-2019-0998', 'CVE-2025-29831', 'CVE-2018-8639', 'CVE-2025-29837', 'CVE-2022-35762', 'CVE-2025-30388', 'CVE-2022-34714', 'CVE-2025-29964', 'CVE-2025-29840', 'CVE-2019-1040', 'CVE-2019-1019', 'CVE-2022-34696', 'CVE-2023-35622', 'CVE-2025-29976', 'CVE-2019-1012', 'CVE-2022-34702', 'CVE-2025-29967', 'CVE-2019-1024', 'CVE-2025-29832', 'CVE-2019-1014', 'CVE-2025-29961', 'CVE-2022-34703', 'CVE-2019-0888', 'CVE-2019-0722', 'CVE-2025-29969', 'CVE-2022-35764', 'CVE-2019-1003', 'CVE-2019-0711', 'CVE-2025-29836', 'CVE-2025-30385', 'CVE-2019-1038', 'CVE-2019-0948', 'CVE-2019-1005', 'CVE-2019-1080', 'CVE-2019-1041', 'CVE-2025-32709', 'CVE-2022-35766', 'CVE-2025-24054', 'CVE-2019-0992', 'CVE-2022-34706', 'CVE-2025-29968', 'CVE-2019-0941', 'CVE-2025-32706', 'CVE-2019-1026', 'CVE-2019-1081', 'CVE-2019-1028', 'CVE-2025-29957', 'CVE-2025-29962', 'CVE-2019-1018', 'CVE-2025-32707', 'CVE-2022-35768', 'CVE-2019-0620', 'CVE-2019-0909', 'CVE-2019-1025', 'CVE-2025-24063', 'CVE-2019-1044', 'CVE-2025-24985', 'CVE-2019-0989', 'CVE-2019-1043', 'CVE-2019-0983', 'CVE-2019-0984', 'CVE-2019-1053', 'CVE-2019-1021', 'CVE-2025-29966', 'CVE-2025-32701', 'CVE-2022-34691', 'CVE-2022-34699', 'CVE-2019-1010', 'CVE-2019-1039', 'CVE-2022-35763', 'CVE-2025-29839', 'CVE-2019-0988', 'CVE-2019-0906', 'CVE-2022-34705', 'CVE-2022-34707', 'CVE-2019-0972', 'CVE-2025-29974', 'CVE-2019-0905', 'CVE-2019-0973', 'CVE-2025-29842', 'CVE-2019-1027', 'CVE-2025-26677', 'CVE-2025-29833', 'CVE-2019-1052', 'CVE-2019-0943', 'CVE-2025-29954', 'CVE-2019-0993', 'CVE-2025-29958', 'CVE-2019-0713', 'CVE-2019-1007', 'CVE-2025-30400', 'CVE-2019-0959', 'CVE-2025-27488', 'CVE-2019-1051', 'CVE-2019-1064', 'CVE-2025-29960', 'CVE-2019-1017', 'CVE-2022-34710', 'CVE-2022-35771', 'CVE-2019-1046', 'CVE-2022-34709', 'CVE-2019-0908', 'CVE-2025-27468', 'CVE-2025-29959', 'CVE-2019-1054', 'CVE-2022-35769', 'CVE-2022-35765', 'CVE-2019-0904', 'CVE-2019-1069', 'CVE-2025-30394', 'CVE-2025-29963', 'CVE-2019-1055', 'CVE-2025-29830', 'CVE-2019-0991', 'CVE-2025-29956', 'CVE-2019-0920', 'CVE-2019-1023', 'CVE-2019-1050', 'CVE-2022-35767', 'CVE-2022-35761', 'CVE-2022-34708', 'CVE-2025-29829', 'CVE-2019-0907', 'CVE-2022-34701', 'CVE-2025-29835', 'CVE-2019-0990', 'CVE-2019-0986', 'CVE-2019-0974', 'CVE-2019-0710', 'CVE-2019-1022', 'CVE-2025-30397']
------------------------	---

Tabla 4: CVEs encontrados por tecnología

Fuente: Elaboración Propia.

5.3. Ejemplo plantillas

5.3.1. Template correo electrónico



Figura 26: Ejemplo de templates de correo electrónico.
Fuente: Elaboración propia.

A continuación se listan los CVES con severidad **ALTA**

Severidad: Crítica (9.8)

Tecnología: apache

CVE ID: CVE-2022-24706

In Apache CouchDB prior to 3.2.2, an attacker can access an improperly secured default installation without authenticating and gain admin privileges. The CouchDB documentation has always made recommendations for properly securing an installation, including recommending using a firewall in front of all CouchDB installations.

[Más información](#)

Severidad: Crítica (9.8)

Tecnología: apache

CVE ID: CVE-2022-47966

Multiple Zoho ManageEngine on-premise products, such as ServiceDesk Plus through 14003, allow remote code execution due to use of Apache Santuario xmlsec (aka XML Security for Java) 1.4.1, because the xmlsec XSLT features, by design in that version, make the application responsible for certain security protections, and the ManageEngine applications did not provide those protections. This affects Access Manager Plus before 4308, Active Directory 360 before 4310, ADAudit Plus before 7081, ADManager Plus before 7162, ADSelfService Plus before 6211, Analytics Plus before 5150, Application Control Plus before 10.1.2220.18, Asset Explorer before 6983, Browser Security Plus before 11.1.2238.6, Device Control Plus before 10.1.2220.18, Endpoint Central before 10.1.2228.11, Endpoint Central MSP before 10.1.2228.11, Endpoint DLP before 10.1.2137.6, Key Manager Plus before 6401, OS Deployer before 1.1.2243.1, PAM 360 before 5713, Password Manager Pro before 12124, Patch Manager Plus before 10.1.2220.18, Remote Access Plus before 10.1.2228.11, Remote Monitoring and Management (RMM) before 10.1.41, ServiceDesk Plus before 14004, ServiceDesk Plus MSP before 13001, SupportCenter Plus before 11026, and Vulnerability Manager Plus before 10.1.2220.18. Exploitation is only possible if SAML SSO has ever been configured for a product (for some products, exploitation requires that SAML SSO is currently active).

Figura 27: Ejemplo de templates de correo electrónico.
Fuente: Elaboración propia.

5.3.2. Mensajes de Telegram

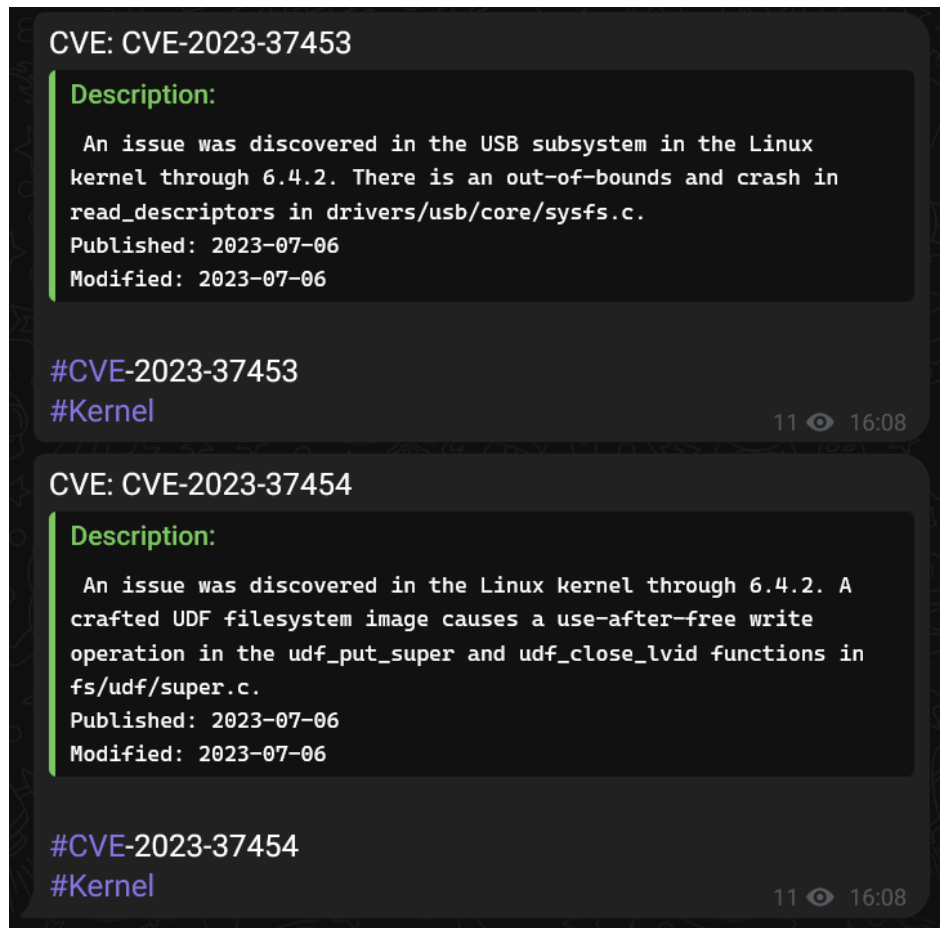


Figura 28: Plantilla de mensaje para CVE en telegram
Fuente: Elaboración propia

5.3.3. Mensajes de Slack

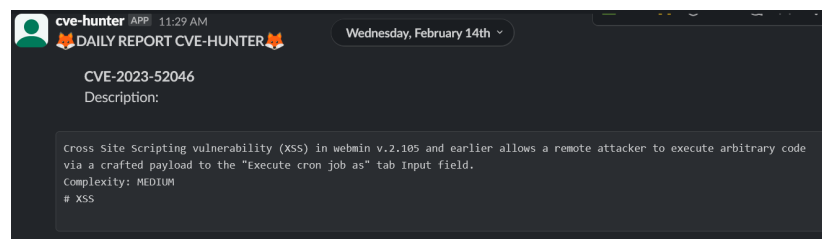


Figura 29: Ejemplo de notificación de Slack
Fuente: Elaboración Propia.

5.4. Trabajos adicionales

Durante el desarrollo del sistema, se implementaron funcionalidades complementarias que, si bien no fueron consideradas dentro de los objetivos específicos de esta memoria, contribuyeron a enriquecer el valor global de la solución propuesta. Estas características fueron concebidas como extensiones que amplían las capacidades del sistema en contextos prácticos.

Una de las funcionalidades desarrolladas corresponde a un servicio de búsqueda de vulnerabilidades CVE, el cual permite realizar consultas utilizando el nombre de una tecnología o mediante archivos de administración de dependencias, tales como *requirements.txt*, *package.json* o *composer.json*.



Figura 30: Interfaz del servicio de búsqueda de CVE (1).
Fuente: Elaboración propia.



Figura 31: Interfaz del servicio de búsqueda de CVE (2).
Fuente: Elaboración propia.

Adicionalmente, se desarrolló un servicio de generación de informes ejecutivos en formato *PDF*, orientado a facilitar a los equipos de desarrollo la evaluación de los riesgos potenciales que podrían estar presentes en sus sistemas. Estos informes consolidan la información de las vulnerabilidades detectadas, permitiendo una visualización clara y estructurada.

Reporte Mensual CVE-HUNTER para la compañía DEVELROX

En el presente documento se detalla información de las vulnerabilidades que pudieron haber afectado la ciberseguridad de sus sistemas. Dependiendo de la criticidad, se debe realizar las acciones pertinentes para mitigarlas. Es importante que si se detalla alguna vulnerabilidad crítica o alta, esta se intente resolver lo más rápido posible, debido a que suelen ser fáciles de explotar.

Recordar que vulnerabilidades con severidad **Crítica**, no necesariamente requieren actividad humana para ser explotadas. Obtener permisos de administrador suelen ser sencillo para los atacantes, provocando disrupciones en los sistemas.

Se recomienda **siempre tener los sistemas actualizados**, en la mayoría de los casos suele ser la mejor manera de protegerse contra alguna amenaza.

Durante los últimos 30 días, sus sistemas han presentado 201 vulnerabilidades.

A continuación, se presenta la cantidad total de **CVEs** por tecnología.

Nombre tecnología	Cantidad de CVEs	Porcentaje
java	11	5.47%
php	106	52.74%
cisco	19	9.45%
microsoft	44	21.89%
javascript	18	8.96%
python	3	1.49%
typescript	0	0.0%

Porcentaje de severidad relacionado a la cantidad de vulnerabilidades

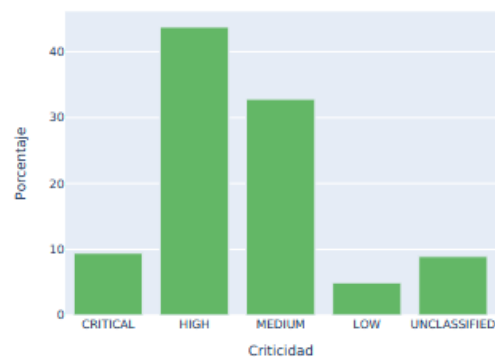


Figura 32: Ejemplo de informe ejecutivo (1).
Fuente: Elaboración propia.

	CRÍTICOS	ALTOS	MEDIO	BAJO	NO CLASIFICADO
Porcentajes	9.45%	43.78%	32.84%	4.98%	8.96%
Cantidad	19	88	66	10	18

A continuación, se muestran el Top de CVEs con más alta criticidad

Id	CVE-2024-6960		
Technology	java		
Metrics			
Base score	7.5	Base severity	HIGH
Descriptions			
The H2O machine learning platform uses "Iced" classes as the primary means of moving Java Objects around the cluster. The Iced format supports inclusion of serialized Java objects. When a model is deserialized, any class is allowed to be deserialized (no class whitelist). An attacker can construct a crafted Iced model that uses Java gadgets and leads to arbitrary code execution when imported to the H2O platform.			

Id	CVE-2024-6957		
Technology	php		
Metrics			
Base score	7.3	Base severity	HIGH
Descriptions			
A vulnerability classified as critical has been found in itsourcecode University Management System 1.0. This affects an unknown part of the file functions.php of the component Login. The manipulation of the argument username leads to sql injection. It is possible to initiate the attack remotely. The exploit has been disclosed to the public and may be used. The associated identifier of this vulnerability is VDB-272079.			

Id	CVE-2024-6945		
Technology	php		
Metrics			
Base score	6.3	Base severity	MEDIUM
Descriptions			
A vulnerability was found in Flute CMS 0.2.2.4-alpha. It has been classified as critical. This affects an unknown part of the file app/Core/Http/Controllers/Profile/ImagesController.php of the component Avatar Upload Page. The manipulation of the argument avatar leads to unrestricted upload. It is possible to initiate the attack remotely. The exploit has been disclosed to the public and may be used. The associated identifier of this vulnerability is VDB-272067.			

Id	CVE-2024-6953		
Technology	php		
Metrics			
Base score	6.3	Base severity	MEDIUM
Descriptions			
A vulnerability was found in itsourcecode Tailoring Management System 1.0 and classified as critical. This issue affects some unknown processing of the file sms.php. The manipulation of the argument customer leads to sql injection. The attack may be initiated remotely. The exploit has been disclosed to the public and may be used. The associated identifier of this vulnerability is VDB-272075.			

Figura 33: Ejemplo de informe ejecutivo (2).

Fuente: Elaboración propia.

REFERENCIAS BIBLIOGRÁFICAS

- [Cloudflare, 2022] Cloudflare (2022). ¿Qué es el scraping de datos? <https://www.cloudflare.com/es-es/learning/bots/what-is-data-scraping/>.
- [Alqahtani et al., 2017] Alqahtani, S. S., Eghan, E. E., y Rilling, J. (2017). SV-AF - a security vulnerability analysis framework. En *Proceedings of the 27th IEEE International Symposium on Software Reliability Engineering (ISSRE)*, pp. 219–229.
- [Amazon Web Services, 2024] Amazon Web Services (2024). ¿Qué es ETL (Extract, Transform, Load)? <https://aws.amazon.com/es/what-is/etl/>.
- [Ambrissi, 2023] Ambrissi, R. D. P. (2023). Perdidas por ciberataques podrían alcanzar los 10 billones de dólares en 2025.
- [Briceño, 2020] Briceño, E. (2020). *Planificación y ejecución de evaluaciones de seguridad informática desde un enfoque de ethical hacking*. 3Ciencias.
- [CVE Program, 2023] CVE Program (2023). About the cve program. <https://www.cve.org/About/Overview#AbouttheCVEProgram>.
- [Enrika Pavlovskytė, 2023] Enrika Pavlovskytė (2023). Scrapy vs. BeautifulSoup: A Comparison of Web Scraping Tools. <https://oxylabs.io/blog/scrapy-vs-beautifulsoup>.
- [Entel Digital, 2023] Entel Digital (2023). Cómo analizar la ciberseguridad de tu empresa y su vulnerabilidad. <https://enteldigital.cl/blog/como-analizar-la-ciberseguridad-de-tu-empresa-y-su-vulnerabilidad>.
- [FasterCapital, 2025] FasterCapital (2025). Challenges and limitations of vulnerability scanning. <https://fastercapital.com/topics/challenges-and-limitations-of-vulnerability-scanning.html>.
- [Fastly, 2023] Fastly (2023). The race to Adapt. *Global Security Research Report*, (10):2–12.
- [FIRST, 2024] FIRST (2024). Common Vulnerability Scoring System SIG. <https://www.first.org/cvss/>.
- [Franco et al., 2013] Franco, D. A., Perea, J. L., y Tovar, L. C. (2013). Herramienta para la Detección de Vulnerabilidades basada en la Identificación de Servicios. *Información tecnológica*, 24:13 – 22.
- [Henry, Kevin M., 2007] Henry, Kevin M. (2007). *Pruebas de Penetración: Protección de Redes y Sistemas*. Gobierno de TI Ltd.
- [IBM Cloud Education, 2021] IBM Cloud Education (2021). ¿Qué son los brókeres de mensajería? <https://www.ibm.com/mx-es/topics/message-brokers>.

- [ISO, 2022] ISO, I. O. f. S. (2022). ISO/IEC 27001:2022 — Information security, cybersecurity and privacy protection — Information security management systems — Requirements. <https://www.iso.org/standard/82875.html>. Norma internacional que establece los requisitos para un sistema de gestión de seguridad de la información (SGSI).
- [Jeon y Kim, 2021] Jeon, S. y Kim, H. K. (2021). Autovas: An automated vulnerability analysis system with a deep learning approach. *Computers Security*, 106:102308.
- [Jonker et al., 2025] Jonker, A., Lindemulder, G., y Kosinski, M. (2025). What is Cybersecurity. <https://www.ibm.com/think/topics/cybersecurity>.
- [K y Arun, 2017] .K, V. y Arun, C. (2017). Automated risk identification using nlp in cloud based development environments. *Journal of Ambient Intelligence and Humanized Computing*, pp. 1–13.
- [Kanakogi et al., 2021] Kanakogi, K., Washizaki, H., Fukazawa, Y., Ogata, S., Okubo, T., Kato, T., Kanuka, H., Hazeyama, A., y Yoshioka, N. (2021). Tracing cve vulnerability information to capec attack patterns using natural language processing techniques. *Information*, 12:298.
- [Klensin, 2008] Klensin, D. J. C. (2008). Simple Mail Transfer Protocol. RFC 5321.
- [Krutz y Vines, 2007] Krutz, R. L. y Vines, R. D. (2007). *The CISSP Prep Guide*, capítulo Chapter 6: Operations Security, pp. 317–318. John Wiley & Sons.
- [Manabu, Gin Contributors, 2024] Manabu, Gin Contributors (2024). Gin web framework. <https://gin-gonic.com>.
- [Microsoft, 2023] Microsoft (2023). Microservices architecture style. <https://learn.microsoft.com/es-es/azure/architecture/guide/architecture-styles/microservices>.
- [MITRE Corporation, 2024] MITRE Corporation (2024). CVE Program 25th Anniversary Report. <https://www.cve.org/Resources/Media/Cve25YearsAnniversaryReport.pdf>.
- [MongoDB, 2025] MongoDB (2025). Advantages of MongoDB. <https://www.mongodb.com/resources/compare/advantages-of-mongodb>.
- [National Institute of Standards and Technology, 2022a] National Institute of Standards and Technology (2022a). National Vulnerability Database. <https://nvd.nist.gov/developers/vulnerabilities>.
- [National Institute of Standards and Technology, 2022b] National Institute of Standards and Technology (2022b). Official Common Platform Enumeration (CPE) Dictionary. <https://nvd.nist.gov/products/cpe>.
- [NetThreat Insight, 2025] NetThreat Insight (2025). Automated vulnerability management. <https://www.netthreatinsight.com/automated-vulnerability-management/>.

- [NIST, 2025] NIST, N. V. D. (2025). CVE Assignment and Vetting Process. <https://nvd.nist.gov/general/cve-process>.
- [Omar Santos, 2025] Omar Santos (2025). Evolution of Cybersecurity Vulnerability Management (2019-2024). <https://aisecuritychronicles.org/evolution-of-cybersecurity-vulnerability-management-2019-2024-fec8326331dd>.
- [OWASP Foundation, 2024] OWASP Foundation (2024). OWASP Top Ten. <https://owasp.org/www-project-top-ten/>.
- [Pham y Dang, 2018] Pham, V. y Dang, T. (2018). Cvexplorer: Multidimensional visualization for common vulnerabilities and exposures. En *2018 IEEE International Conference on Big Data (Big Data)*, pp. 1296-1301.
- [Plasman Dries, 2023] Plasman Dries (2023). Active or passive vulnerability management - what's the difference and which one is better?
- [RabbitMQ, 2024] RabbitMQ (2024). Tutorial five: Topics - rabbitmq. <https://www.rabbitmq.com/tutorials/tutorial-five-python>.
- [RabbitMQ, 2025] RabbitMQ (2025). Rabbitmq: One broker to queue them all. <https://www.rabbitmq.com/>.
- [Red Hat Enterprise, 2023] Red Hat Enterprise (2023). ¿Que son los Microservicios? <https://www.redhat.com/es/topics/microservices/what-are-microservices>.
- [Refactoring Guru, 2020a] Refactoring Guru (2020a). Patrones de Diseño. <https://refactoring.guru/es/design-patterns>.
- [Refactoring Guru, 2020b] Refactoring Guru (2020b). Patrón Adaptador. <https://refactoring.guru/es/design-patterns/adapter>. Parte de: *Patrones de diseño*. Refactoring.Guru.
- [Slack API Contributors, 2024] Slack API Contributors (2024). Python Slack SDK: Slack Developer Kit for Python. <https://github.com/slackapi/python-slack-sdk>.
- [Solarte et al., 2015] Solarte, F. N. S., Rosero, E. R. E., y del Carmen Benavides, M. (2015). Metodología de análisis y evaluación de riesgos aplicados a la seguridad informática y de información bajo la norma iso/iec 27001. *Revista Tecnológica-ESPOL*, 28(5).
- [Telegram, 2024] Telegram (2024). From BotFather to 'Hello World'. <https://core.telegram.org/bots/tutorial>.
- [The Linux man-pages project, 2024] The Linux man-pages project (2024). crontab(5) — Linux manual page. <https://man7.org/linux/man-pages/man5/crontab.5.html>.
- [The MITRE Corporation, 2023] The MITRE Corporation (2023). Mitre Corporation CVE. <https://cve.mitre.org/>.

[UNODC, 2020] UNODC (2020). Cybercrime in Brief. <https://www.unodc.org/e4j/en/cybercrime/module-1/key-issues/cybercrime-in-brief.html>. Módulo UNODC "Cybercrime in brief".

Acta de validación del sistema CVE-Hunter

Fecha	7 de Julio del 2025
Plataforma	Google Meet
Participantes	- Camilo Duque - Esteban Vaquero

Objetivo:

Certificar la validación del sistema **CVE-HUNTER**, analizando el objetivo del proyecto y los criterios de éxito definidos, presentados a continuación: 1) Precisión y relevancia en la detección de *CVEs*. 2) Monitoreo continuo de nuevas vulnerabilidades. 3) Eficacia en la notificación de usuarios.

Resultados:

El sistema presento un total de 242 vulnerabilidades asociadas a las tecnologías proporcionadas por **Develrox**. Viendo los resultados por los criterios de éxitos, tenemos lo siguiente:

- 1) **Precisión y relevancia en la detección de *CVEs*:** La medición de la precisión y relevancia se llevó a cabo mediante un análisis aleatorio y estratificado, alcanzando un **83% de precisión**.
- 2) **Monitoreo continuo de nuevas vulnerabilidades:** El monitoreo se desarrollo mediante un cron programado para que se ejecutara cada hora para la detección de vulnerabilidades y cada 24 horas se realizaba el envío de *CVEs*.
- 3) **Eficacia en la notificación:** Se analizó una notificación aleatoriamente, verificando que la notificación fuera clara y explicativa, entregando información relevante de cada *CVE*.

Limitaciones y problemas detectados:

- Se identificaron *CVEs* que se encontraban en estado **reservado** que no estaban siendo notificados.
- Duplicación en algunos mensajes de *CVEs*.

- El envío de *CVEs* se realizó una vez al día, debido a que se realizan múltiples consultas a muchos sistemas, por lo que pueden existir bloqueos de por medio.

Finalmente, por este medio, yo, **Esteban Vaquero**, en mi calidad de **Consultor en Ciberseguridad**, en la empresa **Develrox**, certifico que el sistema **CVE-Hunter** demuestra un funcionamiento positivo, cumpliendo con sus objetivos principales como herramienta de monitoreo y notificación acerca de vulnerabilidades a los usuarios.

X 

Esteban Vaquero
Consultor en Ciberseguridad