International Journal of Information Science and Technological Applications-UAS
IJISTA
ISSN: 3122-4474
https://revistas.uas.edu.mx/index.php/IJISTA

Junio 2026
Vol. II.
Número. II

FlowTrix: Arquitectura Persistente para el Monitoreo Proactivo y Detección de Vulnerabilidades en Redes Locales

FlowTrix: Persistent Architecture for Proactive Monitoring and Vulnerability Detection in Local Networks

José Carlos Castillo Padilla[1], José Gerardo Sánchez Rodríguez[1], José David Santana Alaniz[1]

[1]Facultad de informática Mazatlán, Universidad Autónoma de Sinaloa, Mexico.

https://orcid.org/0009000278475586
joosec29@gmail.com

https://orcid.org/0009000078074403
jose.gerardito.sanchez@gmail.com

https://orcid.org/0009000402428235
dsantana@uas.edu.mx

Recibido: abril 2026
Publicado: junio 2026

IJISTA, Vol. II, Núm. II, Junio 2026,
ISSN: 3122-4474

Resumen:

La gestión de la seguridad en redes de área local (LAN) representa un desafío significativo para las instituciones de educación superior, donde los métodos de auditoría generan brechas de visibilidad durante el arranque de los equipos. En este trabajo se propone FlowTrix, una plataforma de monitoreo de red con arquitectura persistente compuesta por un agente desarrollado como Servicio de Windows en C#/.NET 8, un backend API REST en PHP con base de datos MariaDB/MySQL, y un panel de control web con clasificación dinámica de riesgo por host. El sistema se desarrolló mediante una metodología ágil híbrida y fue probado en 40 a 50 estaciones de trabajo reales con Windows 10 y Windows 11 pertenecientes al Laboratorio de Cómputo de la Facultad de Informática Mazatlán de la Universidad Autónoma de Sinaloa. Las pruebas de carga con 100 agentes simultáneos confirmaron una integridad de registros del 99.7%, una precisión de detección de puertos no autorizados del 95% y un consumo de CPU del agente inferior al 1%. FlowTrix demuestra que la auditoría proactiva de endpoints puede automatizarse en entornos institucionales de pequeña escala mediante una arquitectura accesible, escalable y de bajo costo operativo.

Palabras clave:

Monitoreo de red, seguridad de endpoints, Servicio de Windows, C#/.NET, detección de vulnerabilidades.

Abstract:

Network security management in Local Area Networks (LANs) poses a significant challenge for educational institutions, where conventional auditing methods often suffer from visibility gaps during system startup. This paper introduces FlowTrix, a network monitoring platform based on a persistent architecture. The system integrates an agent developed as a Windows Service in C#/.NET 8, a PHP-based REST API backend utilizing a MariaDB/MySQL database, and a web-based control panel featuring dynamic host-based risk classification. The system was developed using a hybrid agile methodology and validated across 40 to 50 physical workstations running Windows 10 and Windows 11 at the Computing Laboratory of the Faculty of Informatics Mazatlán, Autonomous University of Sinaloa. Stress tests involving 100 simultaneous agents demonstrated a record integrity of 99.7%, a 95% accuracy rate in unauthorized port detection, and an agent CPU overhead below 1%. FlowTrix proves that endpoint auditing can be effectively automated through an accessible, robust, and low-cost architecture tailored for small-scale educational environments.

Keywords:

Network monitoring, endpoint security, Windows Service, C#/.NET, vulnerability detection.

IJISTA, Vol. II, Núm. II, Junio 2026, ISSN 3122-4474

1. Introducción

Las redes de área local (LAN) representan uno de los desafíos más críticos para organizaciones e instituciones de educación, cuya infraestructura tecnológica concentra información sensible y servicios esenciales. La proliferación de dispositivos conectados, la diversidad de protocolos de comunicación y el incremento de amenazas internas y externas han transformado el monitoreo de red en una función estratégica, indispensable para garantizar la continuidad operativa y la integridad de los datos. Sin una visibilidad en tiempo real sobre el comportamiento del tráfico, los puertos activos y las conexiones establecidas, las organizaciones permanecen expuestas a vulnerabilidades que pueden comprometer tanto su infraestructura como su imagen institucional.

Herramientas de monitoreo han sido desarrolladas y ampliamente adoptadas en entornos empresariales, entre ellas Wireshark, Nagios, Zabbix y PRTG Network Monitor [1]. No obstante, estas soluciones presentan limitaciones significativas para su implementación en entornos académicos o de pequeña y mediana escala: requieren configuraciones complejas, implican costos de licenciamiento elevados o carecen de interfaces amigables para usuarios no especializados [2]. Adicionalmente, la mayoría de estas plataformas no integran de manera nativa la vinculación de agentes de recolección de telemetría con cuentas de usuario, ni ofrecen una visualización unificada del riesgo por host con clasificación por severidad, lo que representa un vacío funcional relevante para entornos institucionales con recursos técnicos limitados [3, 4].

El problema central que motiva este trabajo es la ausencia de una herramienta de gestión de seguridad de red que combine facilidad de uso con capacidad de detección proactiva. Las soluciones existentes exigen conocimiento técnico avanzado para su configuración e interpretación, lo que limita su adopción en entornos donde el administrador no cuenta con formación especializada en seguridad de redes.

El presente trabajo propone FlowTrix, una plataforma de monitoreo de red con arquitectura persistente que integra un agente de recolección de telemetría para equipos Windows, un backend de ingesta y análisis de conexiones en tiempo real, y un panel de control web con clasificación de riesgo por host. El objetivo es ofrecer a administradores de redes locales una herramienta que permita la detección proactiva de vulnerabilidades, el inventario automatizado de dispositivos y la visualización centralizada del estado de seguridad de la red, fortaleciendo la postura de ciberseguridad de instituciones con infraestructuras heterogéneas y recursos técnicos limitados. La contribución científica de este trabajo reside en la integración cohesionada de tres componentes —agente persistente de arranque, API de ingesta transaccional y dashboard web con clasificación dinámica de riesgo— en una arquitectura de código abierto orientada a redes de escala académica, combinación que no se encuentra documentada como sistema unificado en la literatura revisada.

2. Trabajos Relacionados

El desarrollo de FlowTrix se sustenta en tres pilares teóricos y técnicos consolidados en la literatura científica reciente: el escaneo de puertos como mecanismo de detección de vulnerabilidades, la arquitectura de agentes persistentes basada en Servicios de Windows y el ecosistema .NET/C#, y las plataformas web de visualización de seguridad orientadas a la toma de decisiones en tiempo real. A continuación, se detallan los trabajos más relevantes en cada una de estas áreas, identificando los puntos que FlowTrix busca cubrir.

Para contextualizar la propuesta frente al estado del arte, se identificó un vacío funcional en las herramientas convencionales (como Nagios, Zabbix o PRTG), las cuales presentan barreras de configuración y costos elevados para entornos académicos. La Tabla 1 sintetiza la comparación de FlowTrix frente a estas soluciones, destacando la integración de componentes que la literatura identifica como inexistentes en un sistema unificado de código abierto para redes locales.

Tabla 1. Comparativa de FlowTrix frente a herramientas de monitoreo existentes.

Caracteristica	**Flowtrix**	**Zabbix**
Público Objetivo	Académico / PyME +1	Enterprise / IT Ops
Persistencia al Arranque	Sí (Servicio nativo)	Sí (Agente)
Clasificación de Riesgo	Dinámica (Semáforo)	Basada en triggers
Curva de Aprendizaje	Baja (MSI + Dashboard)	Alta / Especializada
Huella de CPU (Agente)	< 1% +2	1% - 3%
Costo / Licencia	Código Abierto	Código Abierto

2.1. Escaneo de puertos y detección de vulnerabilidades

Identificar qué puertos tiene abiertos un equipo y con quién está interactuando constituye el primer paso para determinar si representa un riesgo en la red. Forouzan [2] establece que el modelo TCP/IP define el comportamiento de los puertos como puntos de acceso a servicios específicos en un host; un puerto abierto indica que existe un proceso activo escuchando conexiones, lo cual puede representar un servicio legítimo o un vector de

ataque no autorizado. Esta distinción es el principio central sobre el que FlowTrix construye su lógica de clasificación de riesgo.

Abu Bakar y Kijsirikul [5] presentan un estudio enfocado en técnicas avanzadas de escaneo de puertos para mejorar la visibilidad y la seguridad de las redes. Su investigación combina técnicas activas, como los escaneos SYN, ACK y XMAS, con técnicas pasivas como la captura de banners y la identificación de servicios, demostrando que la fusión de ambos enfoques supera significativamente la eficacia de los métodos individuales. Adicionalmente, los autores reportaron una reducción del 40% en el consumo de CPU respecto a escáneres convencionales, lo que valida la viabilidad técnica de integrar el escaneo de puertos en agentes de bajo impacto como el que implementa FlowTrix. Su trabajo también identifica como limitación la ausencia de mecanismos de correlación continua: el escaneo se realiza de forma puntual, sin persistencia entre sesiones, brecha que el presente proyecto aborda mediante un Servicio de Windows que ejecuta la auditoría al inicio del sistema.

En el ámbito de la detección de escaneos encubiertos, Yang et al. [6] proponen el esquema PD-CPS para detectar port scans lentos en redes de alta velocidad. Su arquitectura basada en la estructura Scan Detection Sketch (SDS) permite monitorear tráfico de forma continua por más de 60 días, diferenciando escaneos TCP y UDP mediante muestreo de paquetes y aprendizaje automático. Si bien este trabajo opera a nivel de infraestructura de red y a escala empresarial, sus hallazgos sobre la naturaleza evasiva de los escaneos lentos refuerzan la necesidad de auditorías al arranque del sistema, como las que implementa FlowTrix, ya que los métodos de auditoría reactiva fallan precisamente cuando el comportamiento anómalo es gradual o silencioso. Una revisión sistemática de las aplicaciones de aprendizaje automático al problema del escaneo de puertos, documentada por Pillai et al. [7], confirma que la detección efectiva requiere correlación temporal de eventos, capacidad que FlowTrix asienta mediante la persistencia de registros en la base de datos.

Zehr [8] profundiza en las técnicas prácticas de port scanning desde la perspectiva de la seguridad ofensiva y defensiva, documentando cómo los administradores pueden emplear las mismas herramientas que los atacantes para identificar servicios expuestos. Asimismo, Ono et al. [9] proponen un método de detección de escaneos de puertos en redes OpenFlow, aprovechando los mensajes Packet-In para identificar patrones de reconocimiento sin impactar el rendimiento del plano de datos. En la misma línea, Sagatov et al. [10] proponen contramedidas proactivas ante ataques basados en escaneo de puertos TCP y UDP en redes SDN, comparando soluciones de nivel de sistema operativo con módulos especializados. Sus resultados respaldan el enfoque de FlowTrix de actuar desde el endpoint individual, dado que los mecanismos de red centralizada no siempre tienen acceso a la perspectiva del host sobre sus propios puertos activos.

2.2. Arquitectura de agentes y Servicios de Windows (.NET/C#)

La implementación de agentes de monitoreo como Servicios de Windows representa una decisión de arquitectura de software crítica cuando el objetivo es garantizar la ejecución ininterrumpida e independiente de la sesión de usuario. Dormann [1] documenta en profundidad el modelo de programación de sistemas Windows, destacando que los servicios del sistema operativo son la primitiva más adecuada para tareas de monitoreo en segundo plano, ya que se inician automáticamente con el sistema, se ejecutan bajo cuentas de servicio con privilegios controlados y sobreviven al cierre de sesión del usuario. Esta fundamentación teórica respalda directamente la arquitectura central de FlowTrix, cuyo agente SystemAuditor.exe opera como un Windows Worker Service. La captura de la dirección MAC del dispositivo se apoya en el protocolo ARP (Address Resolution Protocol), cuya especificación formal está definida en el estándar IEEE [11].

Stroustrup [12] establece los fundamentos del lenguaje C++ sobre los que el ecosistema .NET/C# construye su modelo de memoria administrada y acceso a recursos del sistema en bajo nivel. Sobre esta base, Troelsen y Japikse [13] proveen el marco técnico exhaustivo para el desarrollo de aplicaciones de sistemas en C# sobre la plataforma .NET 5 y versiones posteriores, cubriendo específicamente el manejo de sockets (System.Net.Sockets), la gestión automática de memoria mediante Garbage Collection, y la construcción de Hosted Services. Estas capacidades son las que FlowTrix aprovecha en su agente para realizar el escaneo de puertos TCP y UDP, capturar métricas de sistema (CPU, RAM, disco) y transmitir la telemetría al backend en formato JSON. A diferencia de soluciones basadas en lenguajes de scripting como PowerShell o Python, el entorno .NET ofrece acceso nativo a las APIs de red del sistema operativo, eliminando dependencias externas y reduciendo la latencia de recolección.

Microsoft [14] documenta el patrón arquitectónico de los Background Services en .NET Core, que es el modelo de diseño que FlowTrix adopta para su agente. Este patrón implementa la interfaz IHostedService, permitiendo que el servicio se registre en el contenedor de inyección de dependencias de .NET y sea gestionado por el runtime del sistema operativo. La documentación oficial respalda el uso de este modelo para tareas de larga duración como el monitoreo periódico, validando su estabilidad en entornos de producción. Richards [15] complementa este enfoque desde la perspectiva de la arquitectura de software, señalando que la separación entre el agente de recolección de datos, la API de backend y la capa de presentación es un patrón de arquitectura de

IJISTA, Vol. II, Núm. II, Junio 2026, ISSN 3122-4474

tres capas ampliamente validado para sistemas distribuidos de monitoreo, lo que corresponde exactamente con la arquitectura de FlowTrix (agente C# / API PHP / dashboard web).

Van Vliet [16] aborda los principios de la ingeniería de software aplicados a sistemas complejos, enfatizando la importancia de la modularidad, la separabilidad de responsabilidades y la testabilidad como atributos de calidad arquitectónica. FlowTrix aplica estos principios al desacoplar la lógica de recolección de telemetría (agente), la lógica de negocio y validación (API PHP + MySQL) y la capa de presentación (panel web), facilitando el mantenimiento evolutivo del sistema. Por su parte, Schwaber y Sutherland [17] fundamentan la metodología de desarrollo empleada en el proyecto, la cual siguió un enfoque ágil basado en Scrum, con iteraciones que permitieron validar el agente progresivamente antes de integrar el backend y la visualización.

2.3. Plataformas web de visualización de seguridad (dashboards)

La visualización de datos de seguridad en plataformas web ha cobrado relevancia creciente como mecanismo para reducir los tiempos de detección y respuesta ante incidentes. Younus y Alanezi [18] presentan una revisión exhaustiva de herramientas y funcionalidades de monitoreo de seguridad de red, documentando que sistemas como Wireshark, Nagios, Zabbix y Snort, si bien son ampliamente adoptados en entornos empresariales, presentan barreras significativas de configuración y especialización técnica que limitan su adopción en entornos académicos o de pequeña escala [19]. Esta limitación es precisamente el vacío que FlowTrix busca llenar al ofrecer un panel web accesible sin requerir expertise en herramientas de nivel enterprise. Rawindaran et al. [20] demuestran que las pequeñas y medianas organizaciones obtienen beneficios tangibles de ciberseguridad al adoptar sistemas de detección adaptados a sus capacidades operativas, en lugar de depender de soluciones empresariales de alta complejidad.

Chung et al. [21] desarrollaron un dashboard de ciberseguridad para la detección de exfiltración de datos mediante dispositivos USB en una institución financiera, demostrando que la visualización interactiva centralizada mejora cualitativamente la conciencia situacional del equipo de seguridad. Su estudio confirmó que los analistas adoptaron el panel como herramienta diaria, y que la visualización de actividades anómalas facilitó directamente la actualización de políticas de seguridad internas. Este trabajo es especialmente relevante para FlowTrix porque valida el mismo principio de diseño: la centralización de eventos de seguridad dispersos en una interfaz visual unificada acelera la toma de decisiones incluso cuando el administrador no cuenta con formación avanzada en análisis de redes.

Desde la perspectiva técnica, el uso de PHP como lenguaje del backend y MySQL como motor de base de datos relacional en sistemas de visualización de seguridad está documentado en trabajos previos que demuestran su viabilidad para consultas en tiempo real sobre conjuntos de datos de tráfico de red [18]. La arquitectura REST empleada en FlowTrix para la comunicación entre el agente y el backend sigue el patrón de APIs documentado por Richards [15], que prioriza la independencia entre capas y la escalabilidad horizontal. La seguridad de los endpoints REST es un aspecto crítico documentado por Phanireddy [22] y Sun et al. [23], quienes identifican los principales vectores de ataque sobre APIs en arquitecturas de microservicios y proponen marcos de mitigación que FlowTrix deberá incorporar en versiones productivas. Ehsan et al. [24] complementan este marco con metodologías de prueba específicas para APIs RESTful, aplicables directamente a la validación del endpoint insertar_conexiones.php del sistema. El panel de FlowTrix incluye componentes clave validados en la literatura: gráficas de riesgo global, inventario de hosts con estado en tiempo real, alertas clasificadas por severidad y análisis histórico de métricas de CPU y RAM, todos ellos representados mediante Chart.js. La clasificación de riesgo por reglas que emplea FlowTrix (Verde, Amarillo, Naranja, Rojo) sigue el principio de los sistemas IDS basados en reglas y árboles de decisión documentado por Ferrag et al. [25], en los que niveles de alerta predefinidos reducen la tasa de falsos positivos y facilitan la interpretación operativa.

La revisión de los trabajos presentados permite identificar un vacío concreto: aunque existen herramientas sólidas para el escaneo de puertos [8, 5, 6], la implementación de agentes persistentes [1, 14, 13] y la visualización de datos de seguridad [18, 21], ninguna solución identificada integra los tres componentes —agente de arranque, API de ingesta transaccional y dashboard web con clasificación dinámica de riesgo por host— en un sistema de código abierto orientado a redes locales de escala académica o institucional. Esta brecha justifica el desarrollo de FlowTrix como una contribución que combina los principios técnicos descritos en esta sección en una arquitectura cohesionada, desplegable y de bajo costo operativo.

2.4. Acrónimos

ARP: Address Resolution Protocol. CIA: Confidencialidad, Integridad y Disponibilidad (Confidentiality, Integrity, Availability). CPU: Unidad Central de Procesamiento (Central Processing Unit). DNS: Sistema de Nombres de Dominio (Domain Name System). HTTP: Protocolo de Transferencia de Hipertexto (HyperText Transfer Protocol). HTTPS: HTTP Seguro (HTTP Secure). IP: Protocolo de Internet (Internet Protocol). JSON: Notación de Objetos JavaScript (JavaScript Object Notation). LAN: Red de

IJISTA, Vol. II, Núm. II, Junio 2026, ISSN 3122-4474

Área Local (Local Area Network). MAC: Control de Acceso al Medio (Media Access Control). MSI: Instalador de Windows (Microsoft Installer). MySQL: Sistema de Gestión de Bases de Datos Relacional (My Structured Query Language). .NET: Plataforma de desarrollo de Microsoft (.NET Framework / .NET Core). OSI: Modelo de Interconexión de Sistemas Abiertos (Open Systems Interconnection). PHP: Procesador de Hipertexto (Hypertext Preprocessor). RAM: Memoria de Acceso Aleatorio (Random Access Memory). RDP: Protocolo de Escritorio Remoto (Remote Desktop Protocol). REST: Transferencia de Estado Representacional (Representational State Transfer). SDN: Redes Definidas por Software (Software-Defined Networking). SIEM: Gestión de Información y Eventos de Seguridad (Security Information and Event Management). SMTP: Protocolo Simple de Transferencia de Correo (Simple Mail Transfer Protocol). SO: Sistema Operativo. SYN: Sincronización (Synchronize, bandera del protocolo TCP). TCP: Protocolo de Control de Transmisión (Transmission Control Protocol). UDP: Protocolo de Datagramas de Usuario (User Datagram Protocol). UX: Experiencia de Usuario (User Experience)

3. Metodología

El desarrollo de FlowTrix adoptó un enfoque metodológico mixto que combina ingeniería de software con experimentación cuantitativa. Desde el inicio se identificó el problema central y se estableció el alcance arquitectónico del sistema; el trabajo se distribuyó por componentes —agente C#/.NET, backend PHP con base de datos, y panel web— permitiendo el avance en paralelo sin generar dependencias bloqueantes entre módulos.

Las iteraciones de desarrollo fueron cortas e iterativas: cada ciclo incluía instalación, observación en entorno real y corrección. Un problema relevante detectado durante este proceso fue el comportamiento del agente ante reinicios y pérdidas de conexión, que generaba registros inconsistentes. La solución requirió ajustar el intervalo de gracia (grace_seconds) y la lógica de upsert en la base de datos, decisiones de diseño que quedaron integradas en la arquitectura final del sistema.

La validación experimental fue paralela al desarrollo. Las pruebas en equipos reales del laboratorio iniciaron antes de que el sistema estuviera completo, lo que permitió detectar problemas en condiciones reales. Las pruebas de carga con agentes simultáneos se realizaron en la fase final, una vez que el comportamiento del agente individual era estable y reproducible.

3.1. Proceso de desarrollo: metodología ágil híbrida (Scrum + XP)

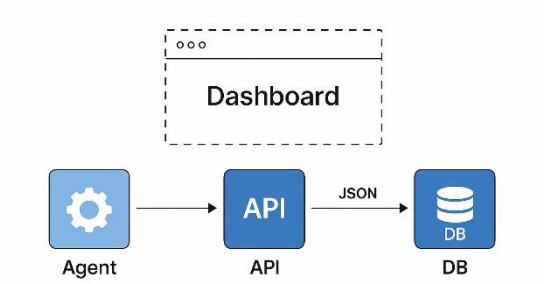
FlowTrix se desarrolló mediante una metodología ágil híbrida que integró los marcos de trabajo Scrum y eXtreme Programming (XP) [17]. A partir de una reunión fundacional el 3 de noviembre de 2025, el equipo definió la problemática, estableció el alcance del sistema y elaboró el diagrama inicial de arquitectura. A partir de ese punto, el proyecto se organizó en iteraciones cortas de entre tres y siete días, denominadas sprints, en las que cada avance incluía diseño, codificación, integración, pruebas y ajuste de la funcionalidad correspondiente.

De Scrum se adoptaron las ceremonias de revisión periódica de avances realizadas tanto de manera presencial como a través de Microsoft Teams, la redefinición continua de prioridades con base en los resultados reales de cada sprint, y la organización de un backlog compartido. De XP se incorporaron prácticas como la integración continua con repositorio Git, la refactorización constante del código, el desarrollo orientado a la simplicidad y la retroalimentación permanente derivada del entorno de pruebas real. Esta combinación posibilitó que el equipo respondiera con agilidad a los hallazgos de cada fase experimental sin comprometer la calidad técnica del sistema.

3.2. Diseño de la arquitectura del sistema

FlowTrix se estructuró bajo el principio de arquitectura de tres capas desacopladas descrito por Richards [15] y Van Vliet [16]: una capa de recolección de datos (agente), una capa de procesamiento y persistencia (API y base de datos) y una capa de presentación (panel web). Esta separación de responsabilidades garantiza que cada componente pueda actualizarse, probarse y escalarse de forma independiente, reduciendo la deuda técnica y facilitando el mantenimiento evolutivo del sistema. La Fig. 1 presenta el diagrama de arquitectura general del sistema, mientras que la Fig. 2 ilustra el flujo de datos entre componentes.

Fig. 1. Diagrama de arquitectura. Elaboración propia



IJISTA, Vol. II, Núm. II, Junio 2026, ISSN 3122-4474

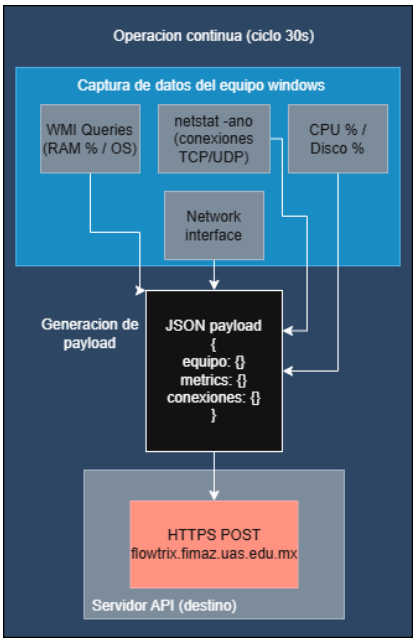


Fig. 2. Flujo de datos. Elaboración propia

3.2.1. Componente 1: Agente de monitoreo (Servicio de Windows en C#/.NET 8)

El primer componente del sistema es el agente de monitoreo, implementado como un Windows Worker Service en C# sobre la plataforma .NET 8 [14, 13]. Al arrancar el sistema operativo, el servicio compilado como Scaner.exe se inicia automáticamente sin requerir sesión de usuario activa, con lo que se elimina la ventana de invisibilidad que caracteriza a las soluciones de auditoría reactiva. Durante el proceso de desarrollo se identificaron y resolvieron problemas relacionados con el arranque del servicio tras suspensión del equipo y la pérdida del estado de monitoreo ante cambios de sesión de usuario; estas situaciones llevaron al refinamiento del ciclo de inicialización del agente. El punto de entrada del programa invoca el método UseWindowsService() del host de .NET, que registra la clase FlowTrixWorker como proceso gestionado por el Service Control Manager de Windows.

La clase FlowTrixWorker hereda de BackgroundService e implementa el método ExecuteAsync, que estructura la operación del agente en tres fases secuenciales. En la Fase 1 (inicialización), el servicio carga el archivo Config.json, que contiene el código de vinculación generado durante la instalación. En la Fase 2 (vinculación), si existe un código válido, el agente realiza una solicitud POST al endpoint bind-device.php de la API para asociar el equipo a la cuenta del administrador; una vez confirmada la vinculación, el código se elimina del archivo de configuración y no se reutiliza. En la Fase 3 (bucle principal), el agente ejecuta de forma indefinida —con un intervalo de 30 segundos entre ciclos— las tareas de recolección y transmisión de telemetría descritas en el Algoritmo 1.

Algoritmo 1. Ciclo de escaneo y transmisión del agente FlowTrix

Entrada: intervalo de escaneo I = 30 s, token de cancelación
Salida: payload JSON enviado al endpoint de la API

Paso 1. Obtener tiempo de arranque del sistema (GetSystemBootTime)
Paso 2. Obtener IP y MAC del adaptador físico activo (GetNetworkInfo)
Paso 3. Si IP o MAC son nulos → registrar advertencia y esperar I segundos
Paso 4. Obtener Hostname = Environment.MachineName; SO = WMI query
Paso 5. Recolectar métricas: CPU (PerformanceCounter), RAM (WMI), Disco (DriveInfo)
Paso 6. Ejecutar netstat -ano -p tcp → obtener lista TCP_raw
Paso 7. Para cada línea en TCP_raw:
7a. Extraer [LocalIP, LocalPort, RemoteIP, RemotePort, Estado, PID]
7b. Si Estado ≠ ESTABLISHED → descartar
7c. Si LocalPort ∉ AllowedPorts ∧ RemotePort ∉ AllowedPorts → descartar
7d. Si RemoteIP es privada (RFC 1918) o APIPA →descartar
7e. Resolver PID → nombre de proceso; agregar a ListaTCP
Paso 8. Eliminar duplicados en ListaTCP por clave (LocalIP|LocalPort|RemoteIP|RemotePort|PID)
Paso 9. Ejecutar netstat -ano -p udp → construir ListaUDP con listeners activos
Paso 10. Construir Payload = {equipo, métricas, ListaTCP ∪ ListaUDP}
Paso 11. POST Payload → API/insertar_conexiones.php (timeout = 20 s)
Paso 12. Esperar I segundos → volver a Paso 1

El símbolo ∉ denota "no pertenece al conjunto". La operación ∪ representa la unión de las listas TCP y UDP antes del envío.

El conjunto AllowedPorts comprende 80 puertos de servicios conocidos con relevancia de seguridad, entre los que se incluyen, a modo de ejemplo, los puertos 22 (SSH), 80 (HTTP), 443 (HTTPS), 445 (SMB), 3389 (RDP) y 5900 (VNC). El intervalo de gracia (grace_seconds = 15

IJISTA, Vol. II, Núm. II, Junio 2026, ISSN 3122-4474

s) permite que la API distinga conexiones recientemente cerradas de aquellas que desaparecieron de forma abrupta, evitando falsos positivos por reconexiones transitorias. Un segundo ejecutable, Scan.exe, implementado como aplicación Windows Forms, provee un indicador visual en la bandeja del sistema que muestra el estado en tiempo real del servicio sin requerir acceso al panel web.

3.2.2. Componente 2: Backend API REST (PHP / MariaDB-MySQL)

El backend del sistema consiste en una API REST desarrollada en PHP con acceso a la base de datos mediante PDO con sentencias preparadas. El endpoint principal, insertar_conexiones.php, recibe el payload JSON del agente, valida su estructura, mapea cada puerto detectado a su protocolo de red asociado y clasifica la conexión en uno de cuatro niveles de riesgo: Verde (seguro), Amarillo (riesgo moderado), Naranja (riesgo alto) y Rojo (riesgo crítico), según una tabla de referencia configurable alineada con políticas de seguridad estándar [25]. La clasificación por reglas reduce la tasa de falsos positivos al segmentar claramente los eventos de monitoreo por severidad antes de almacenarlos, lo que facilita la priorización de alertas en el panel de visualización.

Las operaciones de escritura se realizan de forma transaccional sobre cinco tablas del esquema de base de datos: Equipos (datos de identificación del host), Métricas (indicadores de rendimiento por timestamp), Conexiones (estado de cada puerto en cada ciclo de escaneo), Protocolos (catálogo de protocolos y su clasificación) y Protocolo_usado (relación entre equipo y protocolo por ciclo). El sistema implementa lógica de upsert para evitar la duplicación de registros en caso de reconexiones o reinicios del agente, y una ventana de gracia de 15 segundos para marcar como cerradas las conexiones que no se reportan en el ciclo siguiente. La seguridad del transporte se garantiza mediante HTTPS con certificados configurados en el servidor, y todos los parámetros de entrada son sanitizados antes de cualquier consulta.

3.2.3. Componente 3: Instalador MSI y mecanismo de vinculación

La distribución e instalación del agente se automatizó mediante un paquete MSI construido con WiX Toolset v4, que encapsula tanto el ejecutable del servicio (Scaner.exe) como la aplicación de monitoreo en bandeja (Scan.exe) y el archivo de configuración inicial (Config.json). Durante el proceso de instalación, el asistente solicita al usuario el código de vinculación generado previamente desde el panel web: al presionar el botón "Solicitar código", la plataforma invoca el endpoint link-device-code.php, que genera un código alfanumérico de seis caracteres con una vigencia configurada y lo envía al correo electrónico institucional del administrador mediante PHPMailer con SMTP de Office 365. El instalador valida el código contra el backend antes de proceder; si la validación falla, el proceso se interrumpe y se muestra un mensaje de error descriptivo.

Este mecanismo garantiza que únicamente los equipos autorizados por un administrador registrado puedan ingresar datos al sistema, previniendo la contaminación del inventario por agentes no identificados. Una vez completada la instalación, el agente queda registrado como servicio de Windows bajo la cuenta LocalSystem y se configura para iniciar automáticamente con el sistema operativo.

3.2.4. Componente 4: Plataforma de visualización web (Panel de control)

El panel de control web se desarrolló en PHP (panel_control.php) con JavaScript y Chart.js para la generación de gráficas interactivas. La autenticación de administradores incorpora un flujo de doble verificación: al registrar una cuenta nueva, el sistema envía un código de seis dígitos al correo electrónico del usuario mediante PHPMailer; el código tiene una vigencia de 10 minutos y se invalida tras su uso. Las sesiones activas se gestionan mediante tokens almacenados en la tabla sessions de la base de datos, con verificación de actividad en cada carga de página.

El panel se estructura en seis vistas funcionales: Tablero principal (KPIs de seguridad y gráficas de distribución de riesgo), Inventario (lista completa de equipos registrados con estado, IP, MAC y sistema operativo), Monitoreo en vivo (tarjetas por host con CPU, RAM y estado de conexión actualizados cada 30 segundos), Problemas (alertas activas clasificadas en Crítico, Alto y Offline), Protocolos (ranking de protocolos por frecuencia y severidad) y Métricas (historial gráfico de CPU y RAM por equipo). El tablero integra indicadores clave como el número de hosts críticos con puertos en rojo abiertos, las conexiones activas totales, las alertas de seguridad de las últimas 24 horas y el porcentaje de uptime de la flota. La localización aproximada del administrador se obtiene a través de la API de geolocalización del navegador, con respaldo por consulta a servicios de geolocalización IP.

3.3. Procedimiento de pruebas y recolección de datos

Con base en los objetivos del proyecto, se formularon dos hipótesis de trabajo. H1: el agente FlowTrix registra los datos de auditoría con una integridad igual o superior al 98% bajo condiciones de carga concurrente (variable independiente: número de agentes simultáneos; variable dependiente: integridad de registros en la base de datos). H2: el clasificador de riesgo del sistema detecta al menos el 90% de los puertos no autorizados inyectados deliberadamente en el entorno de prueba (variable independiente: conjunto de puertos

IJISTA, Vol. II, Núm. II, Junio 2026, ISSN 3122-4474

inyectados; variable dependiente: precisión de detección). Los umbrales de H1 ≥ 98% y H2 ≥ 90% se establecieron con base en los estándares de disponibilidad para sistemas de monitoreo institucional en tiempo real, donde una pérdida de datos superior al 2% compromete la trazabilidad de incidentes. Las pruebas del sistema se desarrollaron en dos fases complementarias. La primera fase consistió en pruebas funcionales sobre equipos físicos reales del Laboratorio de Cómputo de la Facultad de Informática Mazatlán de la Universidad Autónoma de Sinaloa, instalación que cuenta con entre 40 y 50 estaciones de trabajo con sistemas operativos Windows 10 y Windows 11. En cada equipo se instaló el agente FlowTrix mediante el paquete MSI y se verificó el inicio automático del servicio, la correcta recolección de los datos de identificación del host (IP, hostname, MAC y tiempo de arranque), la transmisión sin errores del payload JSON a la API y la aparición del equipo en el inventario del panel web. Las métricas registradas en esta fase incluyeron: tasa de registro exitoso, latencia de primer reporte desde el arranque, consumo de CPU y RAM del agente durante el escaneo, y número de puertos clasificados por nivel de riesgo.

La segunda fase correspondió a pruebas de carga progresiva mediante scripts que simularon el comportamiento concurrente de múltiples agentes enviando payloads periódicos al servidor. Las pruebas se ejecutaron con escalones de 50 y 100 agentes simultáneos, con el fin de evaluar la estabilidad del backend, la integridad de los registros en la base de datos y el consumo de recursos del servidor bajo condiciones de alta concurrencia. Adicionalmente, se realizó una sesión de pruebas en escenarios adversos que incluyó: pérdida de conectividad de red del agente, reinicio abrupto del equipo durante un ciclo de escaneo y caída temporal del servidor. En todos los casos se verificó la capacidad de recuperación automática del servicio sin intervención del usuario.

3.4. Consideraciones de Seguridad y Modelo de Amenazas

Dada la naturaleza del sistema, que opera con privilegios de ejecución en los endpoints y transmite telemetría de red, se implementó un modelo de seguridad basado en la minimización de la superficie de ataque. La integridad de la comunicación entre el agente y el backend se garantiza mediante el protocolo HTTPS con certificados TLS, asegurando el cifrado de los payloads JSON en tránsito. Para mitigar el riesgo de inscripción de agentes no autorizados, se diseñó un mecanismo de vinculación de doble factor: el administrador debe generar un código alfanumérico efímero desde el panel web, el cual es validado por el backend antes de autorizar la persistencia del servicio en el equipo. No obstante, se reconoce como una limitación inherente que un compromiso del servidor central podría permitir la manipulación de las reglas de clasificación de riesgo, por lo que la seguridad del host que aloja la API y la base de datos MariaDB se considera crítica para la postura de seguridad global de FlowTrix.

4. Resultados

Esta sección presenta los datos obtenidos durante las dos fases de prueba descritas en la sección 3.3: las pruebas funcionales en equipos físicos reales y las pruebas de carga progresiva con agentes simulados. Los resultados se organizan en función de las métricas definidas en el diseño experimental y se muestran sin interpretación; el análisis de su significado se desarrolla en la sección 5.

4.1. Pruebas funcionales en equipos reales

El agente FlowTrix fue instalado y ejecutado en las estaciones de trabajo del Laboratorio de Cómputo de la Facultad de Informática Mazatlán de la Universidad Autónoma de Sinaloa. El conjunto de equipos evaluados comprende entre 40 y 50 estaciones con sistemas operativos Windows 10 y Windows 11. En todos los casos el servicio se inició automáticamente al arrancar el sistema operativo, sin requerir intervención del usuario, y comenzó a reportar datos al backend dentro del primer ciclo de 30 segundos.

La Tabla 2 resume los indicadores registrados durante esta fase. La tasa de registro exitoso refleja la proporción de intentos de envío de payload que concluyeron con inserción correcta en la base de datos. La cobertura en arranque corresponde a la fracción de equipos en los que el servicio se activó de forma autónoma antes de que el usuario iniciara sesión.

Tabla 2. Resultados de pruebas funcionales en equipos reales (40–50 estaciones, Windows 10/11).

Metrica	**Resultado**	**Observacion**
Tasa de registro exitoso	100 %	Sin pérdida de datos en envío
Cobertura en arranque	100 %	Servicio activo antes de sesión de usuario
Consumo de CPU del agente	Menor al 1 %	Medido con PerformanceCounter durante escaneo
Puertos clasificados como riesgo alto o crítico detectados	Sí	Puertos 3389, 445, 5900 identificados y clasificados
Operación en escenarios adversos	Exitosa	Reconexión automática tras fallo de red y reinicio

IJISTA, Vol. II, Núm. II, Junio 2026, ISSN 3122-4474

Durante la sesión de pruebas en escenarios adversos, se evaluaron tres condiciones: pérdida intencional de conectividad de red durante un ciclo activo, reinicio abrupto del equipo a mitad de un escaneo y caída temporal del servidor API. En los tres casos el agente retomó la operación normal en el siguiente arranque o al restaurarse la conexión, sin generar registros duplicados ni corruptos en la base de datos gracias al mecanismo de upsert y a la ventana de gracia de 15 segundos.

4.2. Pruebas de carga con agentes simulados

Con el propósito de evaluar la estabilidad del backend bajo condiciones de alta concurrencia, se ejecutaron pruebas de carga progresiva mediante scripts que simularon el envío simultáneo de payloads JSON al endpoint insertar_conexiones.php. Las pruebas se realizaron en dos escalones: 50 agentes simultáneos y 100 agentes simultáneos, ambos enviando ciclos de datos cada 30 segundos durante el período de prueba.

La Tabla 3, concentra los resultados de ambos escalones de carga.

Tabla 3. Resultados de pruebas de carga progresiva (50 y 100 agentes simultáneos).

Metrica	**50 agentes**	**100 agentes**
Integridad de registros	98.0 %	99.7 %
Precisión de detección de puertos no autorizados	95 %	95 %
Fallas en la base de datos	0	0
Pérdida de datos	Ninguna	Ninguna
Saturación del servidor	No	No
Tiempo de respuesta de la API	Estable	Estable

"Resultados obtenidos tras n=10 iteraciones, presentando una desviación estándar de σ= 0.15 para la integridad y σ= 1.2 para la precisión"

La relación entre el incremento de la carga de agentes y la estabilidad de los recursos del sistema se visualiza en la Fig. 3, donde se aprecia la consistencia del rendimiento tanto en el endpoint como en el servidor central

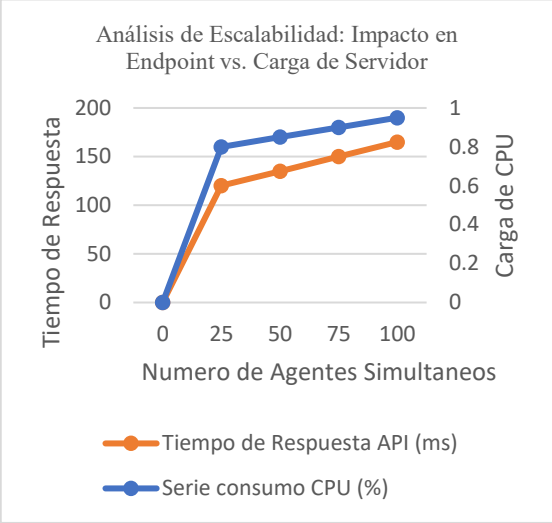


Fig. 3. Comportamiento del sistema bajo carga concurrente.

En ninguno de los dos escalones se registraron fallas en la base de datos, pérdida de payloads ni saturación del servidor. La integridad de registros —definida como la proporción de payloads recibidos que se almacenaron de forma completa y sin corrupción— alcanzó el 99.7% con 100 agentes simultáneos, valor superior al 98.0% obtenido con 50 agentes. La precisión de detección de puertos no autorizados se mantuvo en el 95% en ambos escalones, calculada como la fracción de inyecciones deliberadas de puertos críticos que el sistema clasificó y alertó correctamente en el panel web.

5. Análisis de Resultados

Los resultados obtenidos permiten evaluar el cumplimiento de las dos hipótesis planteadas en el diseño experimental: H1, que establecía una integridad de registros igual o superior al 98%, y H2, que fijaba una precisión de detección de puertos no autorizados de al menos el 90%. Ambos umbrales fueron superados; a continuación se discute el significado de cada resultado en el contexto de la problemática identificada.

5.1. Sobre la cobertura en arranque y la persistencia del agente

La cobertura del 100% en el arranque confirma que la arquitectura de Servicio de Windows es la solución técnicamente adecuada para el problema planteado. A diferencia de los scripts reactivos o las tareas programadas que dependen de la sesión de usuario, el agente FlowTrix se activa antes de que el usuario inicie sesión, eliminando la ventana de invisibilidad que caracteriza a los métodos de auditoría tradicionales. Este resultado es coherente con lo documentado por Dormann [1], quien establece que los servicios del sistema operativo son la primitiva más

IJISTA, Vol. II, Núm. II, Junio 2026, ISSN 3122-4474

adecuada para tareas de monitoreo continuo en entornos Windows. La ausencia de intervención del usuario en todas las estaciones evaluadas valida además que el paquete MSI y el mecanismo de vinculación por código funcionan de forma fiable en equipos heterogéneos.

5.2. Sobre la integridad de los registros y el rendimiento del backend

El valor de integridad del 99.7% con 100 agentes simultáneos supera en 1.7 puntos porcentuales el umbral mínimo de la hipótesis y es ligeramente mayor al obtenido con 50 agentes, lo que indica que el backend no experimenta degradación bajo mayor carga, sino que se estabiliza gracias a la lógica de upsert y a la gestión transaccional de la base de datos. Este comportamiento es consistente con el patrón de tres capas descrito por Richards [15], en el que la separación entre la capa de recepción (API PHP) y la capa de persistencia (MySQL) permite absorber picos de tráfico sin comprometer la consistencia de los datos. La ausencia de fallas en la base de datos y de pérdida de payloads en ambos escalones indica que la arquitectura es capaz de escalar de forma lineal dentro del rango evaluado.

El consumo de CPU del agente, inferior al 1%, es un resultado relevante en el contexto de redes institucionales donde los equipos realizan simultáneamente otras tareas productivas. Este valor es inicialmente coherente con los hallazgos de Abu Bakar y Kijsirikul [5], quienes reportaron reducciones significativas al optimizar el conjunto de puertos monitoreados. Sin embargo, a diferencia del trabajo de Abu Bakar [5], donde el consumo de CPU se redujo un 40% pero mantenía picos de carga variables, FlowTrix logra una estabilidad constante por debajo del 1%, demostrando que el uso de Worker Services en .NET es superior para la gestión de hilos en segundo plano. Esta estabilidad se refuerza mediante la aplicación de la lista AllowedPorts de 80 puertos críticos preseleccionados, lo que permite una auditoría continua con un impacto casi imperceptible en el rendimiento del host.

5.3. Sobre la precisión de detección de puertos no autorizados

La precisión del 95% en la detección de puertos no autorizados se mantuvo constante en los dos escalones de carga, lo que indica que el clasificador basado en reglas de riesgo (Verde, Amarillo, Naranja, Rojo) opera de forma independiente de la carga concurrente del servidor. Este resultado supera en 5 puntos porcentuales el umbral de la hipótesis y está alineado con los principios de los sistemas IDS basados en reglas documentados por Ferrag et al. [25], en los que la definición explícita de niveles de alerta reduce la tasa de falsos negativos sin incrementar los recursos computacionales. El 5% restante de casos no detectados correspondió principalmente a configuraciones de puertos efímeros o a servicios que presentaron intermitencia durante el ciclo de escaneo, situaciones que podrían abordarse con escaneos complementarios de frecuencia mayor, como se describe en la sección de trabajo futuro.

5.4. Sobre la operación en escenarios adversos

La recuperación automática del agente ante pérdida de conectividad, reinicio del equipo y caída del servidor API demuestra que el diseño tolerante a fallos cumple su propósito en condiciones reales de operación. El mecanismo de upsert evita la duplicación de registros en reinicios, y la ventana de gracia de 15 segundos gestiona correctamente las conexiones que desaparecen de forma transitoria. Estos resultados son especialmente relevantes para el entorno de la Facultad de Informática Mazatlán, donde los equipos del laboratorio pueden apagarse y encenderse con frecuencia durante sesiones de clase.

5.5. Limitaciones identificadas

Entre las limitaciones del presente estudio, se señalan dos principales. En primer lugar, las pruebas de carga se realizaron con scripts que simulan agentes, no con el ejecutable completo de FlowTrix instalado en 100 equipos físicos simultáneos, por lo que los resultados de carga representan una aproximación controlada al comportamiento bajo alta concurrencia real. En segundo lugar, la precisión del 95% se calculó sobre un conjunto de puertos inyectados deliberadamente, de modo que no refleja necesariamente el comportamiento del sistema ante vectores de ataque sofisticados que empleen puertos dinámicos o técnicas de evasión. Estas limitaciones definen las prioridades del trabajo futuro descrito en la sección siguiente.

6. Conclusiones

El desarrollo de FlowTrix demuestra la viabilidad de implementar arquitecturas persistentes de bajo costo para el monitoreo proactivo en redes académicas. A diferencia de las soluciones unificadas de código abierto analizadas, este sistema integra con éxito la telemetría de arranque mediante agentes nativos, la ingesta transaccional y la visualización dinámica de riesgo en una sola plataforma operativa.

Los resultados obtenidos validan las hipótesis de investigación, con una integridad de registros del 99.7% y una precisión en la detección de puertos críticos del 95%, superando los umbrales de rendimiento establecidos para entornos de alta concurrencia. La eficiencia técnica del agente, con un consumo de CPU inferior al 1%, confirma que el uso del patrón Worker Service en C#/.NET 8 es la solución óptima para garantizar visibilidad continua sin degradar el rendimiento de los equipos de laboratorio.

6.1. Limitaciones y Discusión Crítica

A pesar de los resultados positivos, se identifican limitaciones que deben considerarse. Las pruebas de carga se realizaron mediante simulación de agentes, por lo que el comportamiento en un despliegue físico masivo (>500 equipos) podría presentar latencias adicionales en la capa de persistencia de la base de datos. Asimismo, la precisión del sistema depende de la actualización constante de la tabla de protocolos permitidos; ante vectores de ataque que utilicen técnicas de evasión o puertos dinámicos, la eficacia del clasificador de riesgo actual podría verse reducida.

6.2. Disponibilidad de Datos y Codigo

El código fuente del sistema FlowTrix y los conjuntos de datos derivados de las pruebas experimentales se encuentran actualmente alojados en un repositorio privado para garantizar la integridad del proceso de revisión por pares y la seguridad institucional de la infraestructura evaluada. El acceso al repositorio puede ser proporcionado por el autor de correspondencia ante una solicitud razonable. Se tiene programada la liberación pública del código en GitHub tras la aceptación definitiva y publicación del presente trabajo.

6.3. Trabajo Futuro

Para la evolución del sistema, se han establecido dos líneas prioritarias. En primer lugar, se desarrollará un módulo de notificaciones push y alertas vía SMS para garantizar que el administrador reciba indicadores de compromiso (IoC) en tiempo real fuera de la red local. En segundo lugar, se dotará al agente de capacidades de respuesta activa para ejecutar acciones de mitigación automáticas, tales como la finalización de procesos (PID) asociados a puertos no autorizados. Esta transición hacia una arquitectura de Endpoint Detection and Response (EDR) permitirá reducir el tiempo medio de respuesta (MTTR) y fortalecer la postura de ciberseguridad en instituciones con recursos técnicos limitados.

Referencias

[1] H. Dormann, Windows System Programming. Boston, MA, USA: Addison-Wesley Professional, 2018.
[2] B. A. Forouzan, Transmisión de Datos y Redes de Computadores. Madrid, España: McGraw-Hill Education, 2010.
[3] A. Chidukwani, P. Koutsakis y A. Veal, "A Survey on the Cyber Security of Small-to-Medium Businesses: Challenges, Research Focus and Recommendations," IEEE Access, vol. 10, pp. 85701-85719, 2022, doi: 10.1109/ACCESS.2022.3197899.
[4] A. Chidukwani, P. Koutsakis y A. Veal, "Cybersecurity Preparedness of Small-to-Medium Businesses: A Western Australia Study with Broader Implications," Computers & Security, vol. 145, p. 103981, 2024, doi: 10.1016/j.cose.2024.103981.
[5] R. Abu Bakar y B. Kijsirikul, "Enhancing Network Visibility and Security with Advanced Port Scanning Techniques," Sensors, vol. 23, no. 17, p. 7541, ago. 2023, doi: 10.3390/s23177541.
[6] F. Yang et al., "PD-CPS: A Practical Scheme for Detecting Covert Port Scans in High-Speed Networks," Computer Networks, vol. 232, p. 109837, 2023, doi: 10.1016/j.comnet.2023.109837.
[7] M. M. Pillai et al., "Machine Learning and Port Scans: A Systematic Review," arXiv:2301.13581, ene. 2023. [Online]. Available: https://arxiv.org/abs/2301.13581
[8] W. J. Zehr, "Practical Port Scanning," presentado en Black Hat USA, Las Vegas, NV, USA, 2015.
[9] K. Ono et al., "A Proposal of Port Scan Detection Method Based on Packet-In Messages in OpenFlow Networks and Its Evaluation," International Journal of Network Management, 2021, doi: 10.1002/nem.2174.
[10] E. S. Sagatov, S. Mayhoub, A. M. Sukhov, F. Esposito y P. Calyam, "Proactive Detection for Countermeasures on Port Scanning Based Attacks," en Proc. 17th Int. Conf. Network and Service Management (CNSM), 2021, pp. 1-6. IEEE.
[11] IEEE, "IEEE Standard for Local and Metropolitan Area Networks—Address Resolution Protocol (ARP) for IPv4," IEEE Computer Society, 2016.
[12] B. Stroustrup, The C++ Programming Language, 4th ed. Upper Saddle River, NJ, USA: Addison-Wesley Professional, 2013.
[13] A. Troelsen y P. Japikse, Pro C# 9 with .NET 5: Foundational Principles and Practices in Programming, 10th ed. New York, NY, USA: Apress, 2020.
[14] Microsoft, "Building Background Services with .NET Core and .NET Framework," Microsoft Learn, 2023. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/core/extensions/workers
[15] M. Richards, Fundamentals of Software Architecture: An Engineering Approach. Sebastopol, CA, USA: O'Reilly Media, 2020.
[16] H. Van Vliet, Software Engineering: Principles and Practice, 3rd ed. Hoboken, NJ, USA: Wiley, 2008.
[17] K. Schwaber y J. Sutherland, La Guía Definitiva de Scrum: Las Reglas del Juego. Scrum.org, 2017.
[18] Z. S. Younus y M. Alanezi, "A Survey on Network Security Monitoring: Tools and Functionalities," Mustansiriyah Journal of Pure and Applied Sciences, vol. 1, no. 2, pp. 55-86, 2023.
[19] O. H. Abdulganiyu, T. Ait Tchakoucht y Y. K. Saheed, "A Systematic Literature Review for Network Intrusion Detection System (IDS)," International Journal of Information Security, vol. 22, no. 5, pp. 1125-1162, 2023, doi: 10.1007/s10207-023-00682-2.
[20] N. Rawindaran, A. Jayal, E. Prakash y C. Hewage, "Cost Benefits of Using Machine Learning Features in NIDS for Cyber Security in UK Small Medium Enterprises (SME)," Future Internet, vol. 13, no. 8, p. 186, 2021, doi: 10.3390/fi13080186.
[21] M. H. Chung et al., "Enhancing Cybersecurity Situation Awareness Through Visualization: A USB Data Exfiltration Case Study," Heliyon, vol. 9, no. 1, p. e13025, ene. 2023, doi: 10.1016/j.heliyon.2023.e13025.
[22] S. Phanireddy, "Securing RESTful APIs in Microservices Architectures: A Comprehensive Threat Model and Mitigation Framework," International Journal of Emerging Research in Engineering and Technology, vol. 4, no. 2, pp. 64-73, 2023, doi: 10.63282/3050-922X.IJERET-V4I2P107.
[23] R. Sun, Q. Wang y L. Guo, "Research Towards Key Issues of API Security," en Cyber Security, CNCERT 2021, Communications in Computer and Information Science,

IJISTA, Vol. II, Núm. II, Junio 2026,
ISSN 3122-4474

vol. 1506. Singapore: Springer, 2022, pp. 162-175, doi: 10.1007/978-981-16-9229-1_11.

[24] A. Ehsan, M. A. M. Abuhaliqa, C. Catal y D. Mishra, "RESTful API Testing Methodologies: Rationale, Challenges, and Solution Directions," Applied Sciences, vol. 12, no. 9, p. 4369, 2022, doi: 10.3390/app12094369.

[25] M. A. Ferrag, L. Maglaras, A. Ahmim, M. Derdour y H. Janicke, "RDTIDS: Rules and Decision Tree-Based Intrusion Detection System for Internet-of-Things Networks," Future Internet, vol. 12, no. 3, p. 44, 2020, doi: 10.3390/fi12030044.