Implementación de estrategia de control de acceso para plataformas IoT basadas en microservicios

María Katherine Plazas Olaya

Tesis de maestría presentada para optar al título de Magíster en Ingeniería

Tutor
José Edinson Aedo Cobo, Doctor (PhD)

Universidad de Antioquia
Facultad de Ingeniería
Maestría en Ingeniería
Medellín, Antioquia, Colombia
2023

Cita	M. K. Plazas Olaya [1]
Referencia Estilo IEEE (2020)	[1] M. K. Plazas Olaya, “Implementación de estrategia de control de acceso para plataformas IoT basadas en microservicios”, Tesis de maestría, Maestría en Ingeniería, Universidad de Antioquia, Medellín, Antioquia, Colombia, 2023.

Maestría en Ingeniería,

Grupo de Investigación Sistemas Embebidos e Inteligencia Computacional (SISTEMIC).

Centro de Documentación Ingeniería (CENDOI)

Repositorio Institucional: http://bibliotecadigital.udea.edu.co

Universidad de Antioquia - www.udea.edu.co

El contenido de esta obra corresponde al derecho de expresión de los autores y no compromete el pensamiento institucional de la Universidad de Antioquia ni desata su responsabilidad frente a terceros. Los autores asumen la responsabilidad por los derechos de autor y conexos.

Agradecimientos

Deseo expresar mi sincero agradecimiento a mi tutor, José Edinson Aedo Cobo, por su valiosa ayuda y apoyo en la realización de este trabajo de investigación. También quiero agradecer a los miembros del proyecto por su colaboración y contribución en la ejecución de este trabajo. Agradezco al grupo de investigación SISTEMIC por facilitarme la infraestructura necesaria para llevar a cabo la validación de los resultados de la investigación. Finalmente quiero agradecer al Fondo de Ciencia, Tecnología e Innovación (FCTeI) del Sistema General de Regalías (SGR), proyecto BPIN 2020000100044 "Administración inteligente de problemas de seguridad ciudadana a través de modelos y herramientas generadas a partir de plataformas para territorios inteligentes apoyadas por estrategias de participación ciudadana en la ciudad de Medellín", por el apoyo financiero para realizar este trabajo de investigación.

TABLA DE CONTENIDO

RESUMEN 10

ABSTRACT 11

I. INTRODUCCIÓN 12

II. OBJETIVOS 16

A. Objetivo general 16

B. Objetivos específicos 16

III. ESTADO DEL ARTE Y ESTRATEGIA DE SEGURIDAD PROPUESTA 17

A. Plataformas IoT basadas en microservicios 17

B. Vulnerabilidades en el contexto de plataformas IoT 19

C. Análisis de mecanismos de seguridad 23

1. Autenticación 24

2. Autorización 26

3. Detección de intrusos 28

D. Seguridad en la plataforma Snap4City 31

E. Estrategia de seguridad propuesta 34

IV. METODOLOGÍA 36

A. Implementación del escenario de prueba 36

1) Máquina víctima 37

2) Extractor de tráfico de red 38

3) Detector de ataques 38

4) Máquinas atacantes 39

B. Construcción de los modelos de aprendizaje automático 39

1) Exploración y selección de datos de entrenamiento 39

a) Base de datos BoT-IoT [58] 40

b) Base de datos UNSW-NB15 [62] 42
c) Generación de una base de datos propia 42
2) Preprocesamiento de datos 45
a) Características categóricas 45
b) Corrección de etiquetas 46
c) Eliminación de características 47
d) Estandarización de los datos 47
3) Entrenamiento y evaluación de los modelos 48
C. Desarrollo e implementación del sistema de detección de ataques en tiempo real 51
1) Extractor de tráfico 51
2) Detector On-line de ataques basado en microservicios 53
a) Microservicio de preprocesamiento 55
b) Microservicio de predicción 58
V. RESULTADOS Y DISCUSIÓN 61
A. Resultados sobre el entrenamiento de los modelos 61
B. Resultados sobre la detección de ataque en tiempo real 65
VI. CONCLUSIONES 70
REFERENCIAS 73

LISTA DE TABLAS

TABLA I VULNERABILIDADES IDENTICIADAS 22
TABLA II AMENZAS DE SEGURIDAD EN EL CONTEXTO DE PLATAFORMAS IOT 22
TABLA III ARTÍCULOS SELECIONADOS PARA LA REVISIÓN DE MECANISMOS DE SEGURIDAD 31
TABLA IV MECANISMO DE SEGURIDAD EN SNAP4CITY 33
TABLA V CARACTERÍSTICAS DE LAS BASES DE DATOS 40
TABLA VI DISTRIBUCIÓN DE LOS REGISTROS EN EL SUBCONJUNTO DEL 5% DE LA BASE DE DATOS BOT-IOT 41
TABLA VII DISTRIBUCIÓN DE LOS REGISTROS DE LA BASE DE DATOS UNSW-NB15 42
TABLA VIII DISTRIBUCIÓN DE LOS REGISTROS DE LA BASE DE DATOS PROPIA 43
TABLA IX CARACTERÍSTICAS GENERADAS POR LA HERRAMIENTA ARGUS 44
TABLA X CARACTERÍSTICAS CALCULADAS A PARTIR DE LAS ENTREGADAS POR ARGUS 44
TABLA XI DESCRIPCIÓN DE LA VARIABLE CATEGÓRICA ESTADO DEL FLUJO 46
TABLA XII DESCRIPCIÓN DE LA VARIABLE CATEGÓRICA TIPO DE PROTOCOLO 46
TABLA XIII HIPERPARÁMETROS PARA CADA MODELO DE APRENDIZAJE AUTOMÁTICO 49
TABLA XIV REPRESENTACIÓN DE LA MATRIZ DE CONFUSIÓN 50
TABLA XV MEJORES HIPERPARÁMETROS PARA CADA MODELO EN LA DETECCIÓN DE ATAQUES 61
TABLA XVI MATRICES DE CONFUSIÓN PARA EL MODELO ÁRBOL DE DECISIÓN 62
TABLA XVII MATRICES DE CONFUSIÓN PARA EL MODELO BOSQUE ALEATORIO 62
TABLA XVIII MATRICES DE CONFUSIÓN PARA EL MODELO REGRESIÓN LOGÍSTICA 63
TABLA XIX MATRICES DE CONFUSIÓN PARA EL MODELO SVM 63
TABLA XX MÉTRICAS DE RENDIMIENTO PARA LOS DATOS DE ENTRENAMIENTO Y PRUEBA 63
TABLA XXI PARÁMETROS DE LAS HERRAMIENTAS DE ATAQUE 65

TABLA XXII MATRIZ DE CONFUSIÓN PARA EL MODELO ÁRBOLES DE DECISIÓN..66
TABLA XXIII MATRIZ DE CONFUSIÓN PARA EL MODELO BOSQUES ALEATORIOS 66
TABLA XXIV MATRIZ DE CONFUSIÓN PARA EL MODELO REGRESIÓN LOGÍSTICA 66
TABLA XXV MATRIZ DE CONFUSIÓN PARA EL MODELO SVM 67
TABLA XXVI MÉTRICAS DE RENDIMIENTO PARA CADA MODELO DE APRENDIZAJE AUTOMÁTICO 67

LISTA DE FIGURAS

Fig. 1 Diagrama de la infraestructura de red para la evaluación del sistema de detección de ataques en tiempo real 37
Fig. 2 Estructura del extractor de tráfico de red. 52
Fig. 3 Arquitectura del detector de ataques basada en microservicios 54
Fig. 4 Diagrama de las entidades en el microservicio de preprocesamiento 56
Fig. 5 Diagrama de las entidades en el microservicio de predicción 59
Fig. 6 Tiempo de entrenamiento de cada clasificador 64
Fig. 7 Tiempo de predicción de los datos de prueba para cada clasificador 64
Fig. 8 Tiempo promedio en calcular las predicciones de 100 registros 68

SIGLAS, ACRÓNIMOS Y ABREVIATURAS

API	Application Program Interface
DDoS	Distributed Denial of Service
DoS	Denial of Service
FN	False Negative
FP	False Positive
HIDS	Host-based Intrusion Detection System
HTTP	Hypertext Transfer Protocol
IDS	Intrusion Detection System
IoT	Internet of Things
JSON	JavaScript Object Notation
JWT	JSON Web Token
NIDS	Network-based Intrusion Detection System
SSO	Single Sign-On
SVM	Support Vector Machine
TN	True Negative
TP	True Positive

RESUMEN

Las plataformas IoT pueden tener vulnerabilidades que permiten a los atacantes interrumpir el funcionamiento normal del sistema y comprometer la seguridad de la información. En este trabajo de investigación se desarrolla un sistema de detección de ataques en redes, utilizando modelos de aprendizaje automático para analizar el flujo de datos de paquetes de red en busca de tráfico malicioso. El sistema implementa un extractor de tráfico de red en tiempo real para capturar la información que viaja por la red y extraer los atributos que describen a los paquetes de red. Además, cuenta con un detector de ataques, que está diseñado bajo una arquitectura de microservicios que ejecuta cuatro modelos de aprendizaje automático (árboles de decisión, bosques aleatorios, regresión logística y máquinas de soporte vectorial) que han sido previamente entrenados mediante tres bases de datos (BoT-IoT, UNSW-NB15 y una base de datos propia), la cuales contienen patrones de ataques. También se construye un escenario de prueba para evaluar el rendimiento del sistema de detección, ejecutando ataques hacia una máquina víctima que ejecuta la plataforma IoT. La evaluación del sistema de detección se realiza mediante métricas de rendimiento como el accuracy, el recall, la precisión y el F1-score. Los resultados muestran que el modelo de árboles de decisión tiene el mejor desempeño, con un valor F1 del 98.08%, precisión del 99.25%, sensibilidad del 96.96% y exactitud del 99.62%. Además, se mide el tiempo de predicción de un ataque en la red, el cual es de 0.36 segundos desde la captura de 100 datos hasta la recepción de la predicción.

Palabras clave — **detección de ataques, aprendizaje automático, tráfico de red, IoT, arquitectura de microservicios, ciudades inteligentes, seguridad**

ABSTRACT

IoT platforms can have vulnerabilities that allow attackers to disrupt the system's regular operation and compromise information security. This research work develops a network attack detection system using machine learning models to analyze the flow of network packet data for malicious traffic. The system implements a real-time network traffic extractor to capture information traveling through the network and extract attributes that describe network packets. Additionally, it features an attack detector designed under a microservices architecture, executing four machine learning models (decision trees, random forests, logistic regression, and support vector machines) previously trained on three databases (BoT-IoT, UNSW-NB15 and a custom database) containing attack patterns. A test scenario is constructed to evaluate the performance of the detection system by launching attacks on a victim machine running the IoT platform. The detection system is evaluated using performance metrics such as accuracy, recall, precision, and F1-score. The results show that the decision tree model performs the best, with an F1-score of 98.08%, precision of 99.25%, recall of 96.96%, and accuracy of 99.62%. The prediction time for a network attack is 0.36 seconds, from capturing 100 data packets to receiving the prediction.

Keywords — **attack detection, machine learning, network traffic, IoT, microservices architecture, smart cities, security.**

I. INTRODUCCIÓN

Recientemente, las ciudades inteligentes (Smart Cities) han surgido como un modelo para abordar la planificación y desarrollo de las ciudades [1]. En general, se busca crear un entorno de innovación abierto impulsado por los ciudadanos y desplegar plataformas tecnológicas que permitan el ofrecimiento de nuevos servicios, denominados “inteligentes”, con el objetivo de lograr una administración más participativa y eficiente de la ciudad [2]. Estos nuevos servicios, emplean la información para apoyar la toma de decisiones y brindar mejores funcionalidades que generen mayor valor e incrementen la calidad de vida de los habitantes.

Uno de los componentes más importantes en las plataformas para ciudades inteligentes son las redes IoT. Estas redes tienen como propósito recopilar datos en tiempo real desde diferentes fuentes (dispositivos) con el propósito de automatizar procedimientos, aumentar la eficacia y simplificar la toma de decisiones. En el contexto de las ciudades inteligentes, las redes IoT ayudan a mejorar la gestión de infraestructuras urbanas, como el alumbrado público, el tráfico, la gestión de residuos y el suministro de agua, entre otros, buscando que las ciudades sean más eficientes y sostenibles. También en los procesos de gobernanza y participación ciudadana pueden ayudar a brindar servicios más inteligentes, permitiendo capturar y procesar información del entorno importante para la toma de decisiones y la actuación sobre la ciudad, ayudando de esta manera a mejorar la calidad de vida de sus habitantes. En general, las plataformas IoT suelen estructurarse en diversas capas de gestión de información, como la infraestructura, de datos, de red, de analítica y de aplicación, entre otras [3].

Debido a la diversidad de dominios que pueden existir en una ciudad y dependiendo de su tamaño, las plataformas para ciudades inteligentes pueden tornarse bastante complejas. De esta manera, es necesario que, en su implementación, se utilicen arquitecturas flexibles que puedan adaptarse a los nuevos requisitos funcionales y no funcionales en constante evolución dentro de los diferentes entornos [2]. Un requerimiento fundamental, por lo tanto, es su capacidad de escalabilidad para el manejo fácil del creciente número de usuarios, la integración de nuevos dispositivos (sensores), nuevas bases de datos y de nuevos servicios que se requieran en una ciudad inteligente [4].

De esta manera, las plataformas han evolucionado de arquitecturas monolíticas, donde todas las funcionalidades están integradas en una sola aplicación, a arquitecturas distribuidas basadas en microservicios [3], [5]. Este enfoque ha surgido para construir y mantener sistemas a gran escala, facilitando el escalamiento automático, el balanceo de carga y mejorando su tolerancia a fallas. En una arquitectura basada en microservicios, la plataforma se divide en pequeños servicios independientes y autónomos, cada uno con su propia lógica y funcionalidad [6]. Cada microservicio puede ser desarrollado, desplegado y escalado de forma independiente, lo cual facilita la evolución y actualización de la plataforma.

De esta manera, la adopción de una arquitectura basada en microservicios ofrece varios beneficios en el contexto de las ciudades inteligentes. Permite una mayor agilidad en el desarrollo y despliegue de nuevas funcionalidades, ya que cada microservicio puede ser desarrollado y mejorado de forma independiente [7]. Facilita la integración de tecnologías y sistemas heterogéneos, ya que cada microservicio puede estar especializado en una tarea específica y utilizar los protocolos y estándares adecuados para la comunicación entre diferentes componentes de la plataforma, como dispositivos IoT, sistemas de gestión urbana, fuentes de datos externas y aplicaciones de terceros [6]. Adicionalmente, facilita la escalabilidad de la plataforma, ya que permite escalar servicios específicos sin afectar el rendimiento global de la plataforma, adaptándose a la demanda y garantizando una respuesta eficiente en situaciones de alta carga [4].

Por otra parte, las plataformas basadas en microservicios, así como tienen grandes beneficios, requieren de la atención de nuevos retos que surgen por su estructura distribuida, como los relacionados con la gestión de la seguridad. Entre los posibles problemas se consideran, por ejemplo, los asociados a la vulnerabilidad de los contenedores, ya que es usual realizar el despliegue de los microservicios distribuidos usando este recurso. Estas vulnerabilidades pueden estar relacionadas con ataques de denegación de servicio (DoS), imágenes maliciosas que contienen código malicioso, que pueden permitir a los atacantes acceder a la red y al sistema de archivos, lo que podría llevar a la eliminación o modificación de datos, interrupción de servicios o incluso la infiltración en otros contenedores o sistemas conectados [8]–[10].

Otro problema asociado a la seguridad está vinculado a la protección de los datos. Puesto que los microservicios se comunican a través de APIs, es importante garantizar la seguridad de la información durante su transmisión. Si estas comunicaciones no están adecuadamente protegidas, existe el riesgo de que los datos sean interceptados por terceros no autorizados, lo que comprometería la confidencialidad y la integridad de la información. Por lo tanto, es fundamental implementar medidas de seguridad sólidas, como de cifrado y autenticación, para salvaguardar la privacidad y la seguridad de los datos en todo momento [8], [11]. Además, en una arquitectura de microservicios, los datos están distribuidos en múltiples bases de datos y sistemas, lo cual introduce nuevos desafíos en la protección y gestión de la información sensible. En este contexto, es necesario garantizar la seguridad en cada punto de almacenamiento y comunicación [8].

Adicionalmente, es importante considerar los problemas de seguridad que pueden surgir en los dispositivos conectados a las redes IoT dentro del contexto de los microservicios. Estos dispositivos, como sensores y actuadores, pueden constituir puntos de vulnerabilidad si no cuentan con los recursos necesarios para implementar algoritmos de autenticación y cifrados efectivos. Especial consideración debe tenerse cuando se despliegan con capacidad de procesamiento en el borde. En muchos casos, los dispositivos IoT tienen recursos limitados, como baja capacidad de procesamiento y memoria, lo que dificulta la implementación de técnicas avanzadas de seguridad, incrementa la vulnerabilidad a ataques maliciosos, como ataques de suplantación de identidad o ataques de intercepción de datos [8], [12].

Este trabajo se desarrolla en el marco de un proyecto de investigación, (“Administración inteligente de problemas de seguridad ciudadana a través de modelos y herramientas generadas a partir de plataformas para territorios inteligentes apoyadas por estrategias de participación ciudadana en la ciudad de Medellín” BPIN 202000010004), donde se desplegará una plataforma basada en microservicios para ciudades inteligentes, orientada inicialmente a aplicaciones relacionadas al dominio de la seguridad ciudadana. En este contexto, el trabajo de investigación se enfoca en proponer e implementar una estrategia, que permita gestionar la seguridad en una plataforma IoT basada en microservicios, mediante la implementación de mecanismos de control de acceso, teniendo en cuenta las vulnerabilidades en estas plataformas. La estrategia de seguridad

se enfoca en la detección de ataques en tiempo real en las redes, para la identificación y prevención temprana de intentos de acceso no autorizados.

La evaluación de los sistemas de detección de ataques en tiempo real se lleva a cabo utilizando métricas de rendimiento como la exactitud (accuracy), la sensibilidad (recall), la precisión y el valor F1 (F1-score) aplicadas a los cuatros modelos de aprendizaje automático utilizados, los cuales son árboles de decisión (decision trees), bosques aleatorios (random forests), regresión logística (logistic regression) y máquinas de soporte vectorial (support vector machines, SVM), obteniendo un sistema capaz de detectar ataques en un 98% de las veces.

Este trabajo de investigación se ha estructurado de la siguiente forma: en la sección II se presentan el objetivo general y los específicos definidos en la propuesta de investigación. En la sección III se presenta el estado del arte sobre la seguridad informática en el contexto de plataformas IoT para cuidades inteligentes, donde se abordan conceptos relevantes, se identifican vulnerabilidades y amenazas, principales mecanismos de seguridad y se define la estrategia de seguridad que se va a desarrollar. La sección IV describe la metodología implementada para llevar a cabo la estrategia de seguridad en un entorno controlado. En la sección V se presentan los resultados obtenidos y su discusión. A continuación, en la sección VI, se presentan las conclusiones y los trabajos futuros. Por último, se incluye la bibliografía utilizada en el desarrollo del trabajo de investigación.

Finalmente, como resultado de este trabajo, se presentó en la conferencia ICECCME 2023, Tenerife, Islas Canarias, España, el artículo titulado: “Machine learning based models for detecting attacks in IoT systems”, el cual fue evaluado y aceptado para su publicación.

II. OBJETIVOS

A. Objetivo general

Desarrollar una estrategia que permita gestionar la seguridad en una plataforma IoT para ciudades inteligentes basada en microservicios, mediante la implementación de mecanismos y políticas eficaces para el control de acceso, en el entorno de los microservicios críticos (que se definirán en el proyecto en el cual se enmarca esta investigación), a partir de la identificación de un conjunto de vulnerabilidades reportadas en la literatura para este tipo de plataformas.

B. Objetivos específicos

- Identificar las vulnerabilidades que se consideran para la definición de la estrategia, mediante un análisis del estado del arte y necesidades establecidas dentro del proyecto en el cual se contextualiza esta investigación.
- Seleccionar mecanismos que permitan solucionar las vulnerabilidades considerando un estudio comparativo de diferentes técnicas reportadas en la literatura.
- Desarrollar y verificar la estrategia propuesta en un ambiente controlado considerando el entorno de desarrollo definido para el proyecto.

III. ESTADO DEL ARTE Y ESTRATEGIA DE SEGURIDAD PROPUESTA

En esta sección, inicialmente se proporciona una visión general sobre la arquitectura de microservicios en el contexto de las plataformas IoT para ciudades inteligentes, resaltando sus ventajas y presentando a la plataforma Snap4City como un ejemplo relevante. Además, se realiza una revisión de la literatura para identificar las principales vulnerabilidades y amenazas en este contexto, lo cual es fundamental para definir la estrategia de seguridad. Se exploran las soluciones propuestas en la literatura para mitigar las vulnerabilidades de seguridad identificadas y se presentan los mecanismos de seguridad incorporados en Snap4City [13]. Finalmente, se describe la estrategia de seguridad propuesta, basada en la revisión de la literatura previamente realizada.

A. Plataformas IoT basadas en microservicios

Las plataformas IoT para ciudades inteligentes son sistemas integrados que permiten recopilar, gestionar y analizar datos generados por una amplia gama de dispositivos y sensores distribuidos en una ciudad. Estos dispositivos incluyen sensores ambientales, sistemas para monitorización del transporte inteligente, del consumo de energía, de la infraestructura de comunicación, entre muchos otros.

Como se mencionó previamente, estas plataformas han evolucionado a la adopción de arquitecturas distribuidas basadas en microservicios [14], [15]. Las cuales se basan en la idea de descomponer una aplicación en componentes más pequeños y autónomos, conocidos como microservicios. Cada microservicio se enfoca en una tarea específica y se comunica con otros microservicios a través de interfaces bien definidas, como APIs, lo que permite mayor flexibilidad y agilidad en el desarrollo y despliegue de aplicaciones [14].

La arquitectura de microservicios ofrece diversas ventajas, debido a que los microservicios pueden ser desarrollados y desplegados de forma independiente, lo que facilita la introducción de mejoras y actualizaciones sin afectar a todo el sistema. Adicionalmente, la arquitectura de microservicios es altamente escalable, ya que cada microservicio puede ser escalado de forma

independiente según las necesidades de carga de trabajo, lo que garantiza un rendimiento óptimo incluso en entornos de alto tráfico.

Otra ventaja de esta arquitectura es la facilidad de reutilizar componentes. Los microservicios se diseñan para ser independientes y modulares, lo que permite su reutilización en diferentes aplicaciones y contextos. Esto acelera el desarrollo y reduce la duplicación de esfuerzos.

Un ecosistema IoT para ciudades inteligentes que adopta una arquitectura de microservicio, se puede organizar en diferentes capas. En el trabajo propuesto por García y otros [16], se plantea una arquitectura que consta de las siguientes capas:

- **Sensores, actuadores y borde:** Capa compuesta por dispositivos que se encargan de recopilar datos del entorno, ejecutar acciones físicas en respuesta a las decisiones tomadas por la plataforma y realizar tareas de procesamiento y filtrado de datos antes de enviarlos a la plataforma para su posterior análisis y uso.
- **Interconexión:** Esta capa tiene la función de establecer la comunicación entre distintos componentes encargados de generar información y aquellos que la utilizan. Su objetivo es facilitar el intercambio de datos de manera eficiente y efectiva dentro del sistema.
- **Análisis de datos:** Esta capa está compuesta por varios servicios que se encargan de procesar y analizar los datos recopilados con la finalidad de adquirir información relevante para la toma de decisiones.
- **Gestión de la plataforma:** Esta capa sirve para administrar y controlar los diferentes aspectos de la plataforma, como la configuración de los servicios, la autenticación de usuarios, la gestión de permisos y roles, y otras funcionalidades relacionadas con la administración y el funcionamiento general de la plataforma. Además, puede incluir el componente API Gateway, el cual es un microservicio que actúa como un punto de entrada centralizado para todas las solicitudes de API de los clientes. El API Gateway proporciona funciones de seguridad, enrutamiento y transformación de solicitudes y respuestas, simplificando la gestión y el control de las interacciones entre las aplicaciones y los servicios de la plataforma.
- **Interfaz de usuario:** o capa de aplicaciones que permiten a los usuarios interactuar con la plataforma y acceder a sus funcionalidades.

En el marco del proyecto de investigación al cual está asociado este trabajo, se ha venido trabajando con la plataforma Snap4City [13], la cual está basada en microservicios. En este contexto, este trabajo de investigación se enfoca en implementar una estrategia, que permita gestionar la seguridad en plataformas con este tipo de arquitecturas.

Snap4City [17] fue desarrollada en la Universidad de Florencia, Italia, por un equipo de investigadores y varios expertos en diferentes áreas. Se trata de una plataforma de IoT y ciudades inteligentes que recopila, gestiona y analiza datos en tiempo real provenientes de diferentes sensores y fuentes. Ofrece herramientas y servicios para visualizar datos, desarrollar aplicaciones y tomar decisiones basadas en datos. Snap4City se viene utilizando en varias ciudades desplegando diversas aplicaciones, como en la ciudad de Helsinki, en el ámbito ambiental para monitorizar la calidad del aire en la ciudad [18], en la ciudad de Florence, en la gestión de emergencias para generar alertas sobre eventos críticos que ocurren en la ciudad [4] y en el área de movilidad y transporte analizando la evolución del tráfico vehicular cuando ocurren eventos inesperados [19].

B. Vulnerabilidades en el contexto de plataformas IoT

En este apartado se lleva a cabo un análisis de las principales vulnerabilidades que se pueden presentar en las plataformas IoT basadas en microservicios, reportadas en la literatura. Esta información resulta fundamental para identificar los puntos débiles de los sistemas IoT y establecer las bases necesarias para implementar medidas de seguridad efectivas.

Para la realizar la búsqueda se utilizan metabuscadores especializados y generales como IEEE y Scopus, considerando publicaciones desde el año 2017 hasta el 2022. Se emplean las siguientes palabras claves en las cadenas de búsqueda: “IoT”, “security”, “vulnerability”, “platform” y “microservice”. Además, solo se consideran artículos publicados en revistas y conferencias.

Después de seleccionar las publicaciones que abordan el objetivo de la búsqueda, se han identificado las siguientes vulnerabilidades:

- **Control de acceso deficiente:** Se refiere a la falta de una adecuada gestión de los mecanismos de autorización y autenticación en el sistema. Esto significa que existe una debilidad en el control y administración de quién tiene acceso a los recursos y servicios dentro del sistema. Por ejemplo, un atacante podría explotar esta vulnerabilidad para acceder a recursos o servicios para los cuales no tiene privilegios, o incluso obtener permisos más elevados de lo que le corresponde inicialmente. Esto puede resultar en la exposición de datos confidenciales, en la realización de acciones no permitidas o el acceso a información crítica del sistema [13], [20], [21].

  Además, el control de acceso deficiente puede dar lugar a que los recursos del sistema se agoten debido a solicitudes no autorizadas o maliciosas, lo cual puede causar una interrupción de los servicios o una degradación del rendimiento.

  Por otra parte, cuando los mecanismos de autenticación no son suficientemente sólidos, existe la posibilidad de que se incluyan nodos maliciosos falsificados en la red, lo que compromete la integridad de los datos y pone en riesgo la seguridad de los dispositivos IoT y las comunicaciones en la red [22].
- **Falta de cifrado:** Se refiere a la ausencia o la implementación incorrecta de mecanismos de cifrado en la transmisión y almacenamiento de datos. El cifrado es un proceso que convierte los datos en un formato ilegible para personas no autorizadas, lo que brinda protección a la confidencialidad y la integridad de la información.

  En el contexto de sistemas IoT basados en microservicios, donde existe un alto flujo de comunicaciones entre los distintos componentes, es importante implementar cifrado de extremo a extremo para evitar de riesgos de exposición, robo o manipulación de los datos. El cifrado de extremo a extremo implica cifrar los datos en su punto de origen, transmitirlos de manera segura a través de la red y descifrarlos solo en su destino final. Esto garantiza que los datos estén protegidos en todas las etapas de su recorrido [22]–[25].
- **Configuración inadecuada de seguridad:** esta vulnerabilidad se refiere a la falta de una correcta configuración y gestión de los componentes de seguridad del sistema. Implica el uso de contraseñas débiles, permisos excesivos o configuraciones predeterminadas que no se han ajustado adecuadamente para proteger el sistema contra posibles amenazas.

  Por ejemplo, si se utiliza una contraseña débil o predeterminada para acceder a un microservicio, un atacante podría intentar adivinar o realizar ataques de fuerza bruta para

obtener acceso no autorizado. Del mismo modo, si los permisos de acceso no se han configurado correctamente, un atacante podría obtener privilegios elevados y realizar acciones no permitidas [13], [23].

- **Componentes obsoletos y vulnerables:** Se refiere a la práctica de incorporar en una aplicación librerías, frameworks u otros módulos de software que han sido identificados como inseguros debido a fallos de seguridad previamente documentados y conocidos. Estos componentes pueden estar desactualizados y no haber recibido actualizaciones de seguridad para corregir las vulnerabilidades conocidas.

  Cuando se utilizan componentes obsoletos y vulnerables, se aumenta significativamente el riesgo de que los atacantes puedan explotar esas vulnerabilidades y comprometer la seguridad del sistema. Estas vulnerabilidades pueden permitir a los atacantes realizar acciones no autorizadas, como la ejecución de código malicioso, la divulgación de información confidencial o la interrupción del funcionamiento normal del sistema [13], [22], [24].
- **Seguridad física deficiente:** se refiere a la falta de medidas adecuadas para proteger los dispositivos IoT que funcionan de manera autónoma, en el borde, en ambientes sin supervisión. El acceso físico no autorizado a estos dispositivos o infraestructuras puede dar lugar a la manipulación de datos o el robo de información, lo que comprometería la integridad y la confiabilidad de la información procesada por el sistema. Además, existe el riesgo de que un atacante intente replicar el firmware para crear dispositivos falsos y utilizarlos para realizar actividades maliciosas [22], [24].
- **Registro y monitoreo insuficientes:** Se refiere a la falta de registros detallados y a un monitoreo adecuado de las actividades en el sistema. Esto puede tener un impacto negativo en la disponibilidad, integridad y confidencialidad de los sistemas.

  En el contexto de sistemas IoT, donde los componentes están distribuidos y se comunican entre sí, la ausencia de registros adecuados dificulta la capacidad de rastrear y auditar las transacciones y comunicaciones entre los diferentes dispositivos IoT, microservicios o componentes que integran el sistema. Esto complica la detección de actividades maliciosas o inusuales, el seguimiento de cambios no autorizados en los dispositivos IoT y la identificación de posibles puntos débiles en el sistema.

  El registro y monitoreo adecuados permiten obtener una visibilidad completa de las actividades en el sistema, lo que facilita la detección temprana de comportamientos sospechosos o ataques,

así como el análisis forense posterior para comprender las causas y tomar medidas correctivas [13], [22]–[24].

A continuación, en la TABLA I se muestra cómo las vulnerabilidades identificadas afectan los tres ejes de seguridad de la información: confidencialidad, que se refiere a la protección de la información contra accesos no autorizados; integridad, que asegura que la información no se altere de manera no autorizada; y disponibilidad, que garantiza que los recursos y servicios estén disponibles cuando se necesiten.

TABLA I
VULNERABILIDADES IDENTICIADAS

Vulnerabilidad	Confidencialidad	Integridad	Disponibilidad
Control de acceso deficiente	x	x	x
Falta de cifrado	x	x	
Configuración inadecuada de seguridad	x	x	
Componentes obsoletos y vulnerables	x		x
Seguridad física deficiente	x	x	x
Registro y monitoreo insuficientes	x	x	x

Por otra parte, en un entorno cada vez más digitalizado, es importante reconocer las amenazas cibernéticas que pueden comprometer la seguridad de la información. Estas amenazas adoptan diversas formas y buscan activamente explotar vulnerabilidades en los sistemas. Para fortalecer la seguridad en las plataformas IoT, es fundamental comprender y abordar las amenazas para garantizar la protección adecuada del sistema. En este contexto, la TABLA II presenta las amenazas identificadas en la literatura junto con las vulnerabilidades asociadas a cada una de ellas.

TABLA II
AMENZAS DE SEGURIDAD EN EL CONTEXTO DE PLATAFORMAS IOT

Amenaza	Descripción	Vulnerabilidad
Cambio de firmware	Modificar el software de un dispositivo para obtener control no autorizado o realizar acciones de maliciosas [26].	Seguridad física deficiente Falta de cifrado
Denegación de servicio (DoS)	Consiste en inundar un sistema o recurso con una gran cantidad de tráfico o solicitudes con el objetivo de agotar los recursos disponibles y hacer que el servicio sea inaccesible para los usuarios legítimos [25].	Registro y monitoreo insuficientes

Fuerza Bruta	Intentos para descifrar contraseñas o claves probando todas las combinaciones posibles o utilizando una lista de palabras comunes [27].	Componentes obsoletos y vulnerables Control de acceso deficiente
Man-in-the-Middle (MITM)	El atacante se posiciona entre dos partes que están comunicándose para interceptar los paquetes de información que se envían entre ellos. De esta manera, el atacante puede leer, modificar o incluso bloquear la información transmitida, sin que las partes involucradas lo sepan [24].	Falta de cifrado Control de acceso deficiente
Intrusión de red	Un atacante logra acceder y comprometer la seguridad de una red, ganando acceso no autorizado a los sistemas y recursos de esta [28].	Control de acceso deficiente Falta de monitoreo
Recopilación de información	Actividad de búsqueda para obtener datos valiosos sobre un sistema o red objetivo mediante la exploración y el análisis de su infraestructura y servicios expuestos, con el objetivo de planificar ataques cibernéticos más avanzados [29].	Configuración inadecuada de seguridad Control de acceso deficiente
Robo de información	El atacante tiene acceso no autorizado a una máquina para obtener a datos confidenciales o robar credenciales [29].	Control de acceso deficiente
Spoofing	Un atacante falsifica o suplanta la identidad de un dispositivo, dirección IP o entidad para engañar a otros dispositivos o usuarios y obtener acceso no autorizado a la red [30].	Control de acceso deficiente

Luego de examinar las posibles vulnerabilidades que podrían manifestarse en las plataformas IoT basadas en microservicios y tomando como referencia el análisis efectuado por la OWASP (The Open Web Application Security Project) [31], el cual señala que para el año 2021, el control de acceso ocupa el primer puesto entre los diez principales riesgos de seguridad, se ha tomado la determinación de seleccionar esta área como el foco central para la estrategia de seguridad propuesta en este estudio. A continuación, se realiza una búsqueda en la literatura con el objetivo de identificar los mecanismos de control de acceso utilizados actualmente.

C. Análisis de mecanismos de seguridad

En la literatura se proponen diferentes mecanismos de seguridad para hacer frente a las vulnerabilidades y amenazas identificadas anteriormente. En este apartado de ha dado prioridad a

análisis de propuestas que abordan las amenazas asociadas al control de acceso, debido su importancia en los sistemas IoT [31].

Para la realizar la búsqueda se utilizan metabuscadores especializados y generales como IEEE, Scopus y Google Shcolar, considerando publicaciones desde el año 2017 hasta el 2022. Se emplean las siguientes palabras claves en las cadenas de búsqueda: “IoT”, “security”, “microservice” o “access control”.

Como resultado de esta revisión se han seleccionado un total de 13 publicaciones, en donde se ha identificado que el control de acceso se puede dividir en tres componentes principales: autenticación, autorización y detección de intrusiones. La autenticación verifica la identidad del usuario. La autorización establece los permisos y privilegios que se otorgan a los usuarios, y la detección de intrusiones supervisa y detecta intentos no autorizados de acceder a los recursos. Estos tres componentes deben trabajar en conjunto para garantizar que los recursos no sean accedidos o utilizados de manera ilegal.

1. *Autenticación*

Existen diferentes técnicas para asegurar que un usuario de un servicio es auténtico o es quien dice ser. De acuerdo con la revisión de la literatura, se encuentra que las técnicas más usadas para la autenticación son [32]–[35]:

- **OpenID** es un protocolo de autenticación en línea que permite a los usuarios verificar su identidad en diversos sitios web mediante un único conjunto de credenciales. Proporciona una forma segura y conveniente de verificar la identidad del usuario sin la necesidad de crear nuevas credenciales.
- **JSON Web Token (JWT)** es un formato compacto y seguro para transmitir información entre partes en forma de objetos JSON. Se utiliza ampliamente como mecanismo de autenticación en aplicaciones web y API. Contiene información verificable sobre la identidad y los privilegios del usuario en un formato firmado y encriptado.

- **Single Sing-On (SSO)** es un método que permite a los usuarios acceder a múltiples aplicaciones y servicios con una única credencial de inicio de sesión. Con SSO, los usuarios no necesitan recordar múltiples conjuntos de credenciales, lo que mejora la experiencia del usuario y facilita la administración de accesos.
- Los **certificados** son documentos electrónicos emitidos por una autoridad de certificación confiable que vinculan la identidad de un usuario con su clave pública. Los certificados se utilizan para autenticar la identidad de un usuario y establecer comunicaciones seguras en entornos en línea.

Una solución para la autenticación en las plataformas IoT basadas en microservicios es la autenticación basada en certificados. Pahl y Donini [32] proponen una solución que permite a los nodos distribuidos verificar las propiedades de seguridad localmente, mediante la implementación de un microservicio en el nodo que gestiona de manera autónoma certificados basados en X.509. Este mecanismo permite implementar un buen sistema de autenticación, sin embargo, los dispositivos de la red IoT requieren una mayor capacidad de procesamiento para tal fin.

En el estudio realizado por Ghosh y otros [33] se proponen un esquema de autenticación de extremo a extremo para una plataforma IoT basada en microservicios, donde se combinan distintos métodos de autenticación. Para la comunicación entre los dispositivos IoT y el Gateway usan el sistema de cifrado AES, para la comunicación entre el Gateway y la plataforma implementan el uso de certificados asimétricos y para la comunicación entre microservicios se hace uso de la autenticación mediante JWT. Con esto consiguen transmitir la información desde los dispositivos hasta la plataforma IoT asegurando su privacidad e integridad, sin embargo, esta técnica incrementa la latencia del flujo de los datos a través de la red.

Adicionalmente, uno de los desafíos en los sistemas basados en microservicios es garantizar una respuesta rápida debido a que todos los microservicios requieren algún tipo de autenticación. En este contexto, Yang y su equipo [34] proponen usar JWT en combinación con una técnica autenticación jerárquica que consiste en dos niveles de jerarquía. Para autenticación después del API Gateway se utiliza el nivel bajo el cual solo requiere del token de autenticación, mientras que, para autenticación de clientes externos además de obtener el token, se realiza una validación para

asegurarse que no se encuentre en una lista de token anormales. Como resultado se obtiene una reducción de hasta el 50% en los tiempos de respuesta.

Otro mecanismo de autenticación es el inicio de sesión único (SSO Single-Sing-On), el cual consiste en un microservicio de autenticación central que los demás microservicios pueden utilizar. Un problema de este mecanismo es que, si deja de estar disponible debido a un error, todas aplicaciones que utilizan este servicio se vuelven inaccesibles. Para mitigar este problema Swapnoneel y otros [35] aplican la tecnología *blockchain*. Gracias a su arquitectura descentralizada, *blockchain* permite que la información se almacene y actualicen de forma distribuida, lo que lo hace ideal para garantizar transacciones confiables. El uso de *blockchain* agrega sobrecarga al sistema, pero mitiga el riesgo de falla.

2. *Autorización*

La autorización es un componente esencial del control de acceso en sistemas de seguridad. Se encarga de determinar quién tiene permiso para acceder y utilizar los recursos de un sistema. En la literatura, se han identificado varios mecanismos de autorización ampliamente utilizados.

Uno de ellos es OAuth 2.0, un protocolo de autorización ampliamente adoptado en aplicaciones web y móviles. OAuth 2.0 permite que los usuarios autoricen a terceros para acceder a sus recursos en un sistema sin compartir sus credenciales de acceso. Utiliza tokens de acceso para garantizar la seguridad durante el intercambio de información entre las partes involucradas [36], [37].

Otro mecanismo común es el control de acceso basado en roles [36]. En este enfoque, los usuarios se agrupan en roles y se les asignan permisos específicos según su función en el sistema. Esto simplifica la gestión de permisos al permitir la asignación de roles a los usuarios en lugar de otorgar permisos individuales.

Además, existe el control de acceso basado en capacidades [38]. En este modelo, se asignan capacidades específicas a los usuarios, lo que les permite acceder a recursos específicos. Estas

capacidades pueden incluir operaciones como lectura, escritura, eliminación, entre otras. El control de acceso basado en capacidades ofrece una gran flexibilidad al permitir un control preciso sobre qué usuarios pueden acceder a qué recursos.

Para brindar mayor seguridad en una arquitectura de microservicios Sussi y otros [36] implementan control de acceso basado en roles junto con el estándar OAuth 2.0, con la combinación de estos mecanismos se espera tener un sistema de autenticación y autorización más seguro y simple, ya que la información que lleva el token de acceso se minimiza al colocar solo el rol del usuario en lugar de especificar todos los permisos y recursos a los que tiene acceso. Este enfoque puede mejorar los tiempos de respuesta en la autenticación y autorización de los usuarios.

Los sistemas distribuidos deben proporcionar muchos puntos de acceso para los usuarios y las otras partes de la aplicación, generando muchos procesos de autenticación y autorización. Además, el número y la ubicación de estos puntos de acceso cambian constantemente durante el tiempo de ejecución, lo que implica nuevos desafíos en el campo de la seguridad y la gestión. Banati y otros [37] proporcionan una solución que impide que los microservicios manejen credenciales de usuario y el proceso de autorización, para ello crean un microservicio orquestador que es capaz de autenticar a los usuarios, administrar su autorización, emitir y retirar los tokens y determinar los roles de los usuarios. Esta solución puede funcionar en entornos distribuidos escalables, sin almacenar información de los usuarios. Además, dado que el orquestador está separado del resto del sistema, puede funcionar incluso si la arquitectura de microservicios cambia.

Una novedosa estrategia a la hora de brindar seguridad en plataformas IoT basadas en microservicios es hacer uso de la tecnología blockchain para ofrecer mecanismos de control de acceso flexible. En este contexto, Xu y otros [38] crean una red blockchain privada en un sistema inteligente de seguridad pública, para asegurar el acceso a los datos entre diferentes proveedores de servicios y entidades, otorgando autenticación de identidad y control de acceso a través de un contrato inteligente que se implementa en la red blockchain. La tecnología blockchain aplicada al control de acceso mostró una sobrecarga computacional y una sobrecarga en la comunicación, sin embargo, parece ser prometedora ya que pudieron aplicar de manera efectiva la autenticación de

identidad y la política de control de acceso en una red de vigilancia inteligente distribuida basada en IoT.

3. Detección de intrusos

Para prevenir el acceso no autorizado a un sistema es fundamental contar con mecanismos de seguridad efectivos que puedan detectar y contrarrestar las acciones de los atacantes. Uno de estos mecanismos es el sistema de detección de intrusos (IDS), el cual se encarga de monitorear y analizar las operaciones de una infraestructura computarizada en busca de actividades anómalas o intentos de ataque [39].

El objetivo principal del IDS es identificar las acciones maliciosas que podrían comprometer la seguridad de un sistema. Al aprovechar las debilidades en la arquitectura o configuración del sistema, los intrusos pueden evadir los mecanismos de autenticación y autorización establecidos. Es aquí donde el IDS desempeña un papel esencial en la prevención de acceso no autorizado a un sistema, al actuar como una línea de defensa adicional para detectar y alertar sobre estas actividades sospechosas.

Los sistemas de detección de intrusos pueden estar basados, según el entorno de monitoreo, en dos clases [24], [39]:

- **Basados en Host (HIDS):** Estos IDS se instalan en sistemas individuales y monitorean la actividad del sistema operativo, registros de eventos y cambios en los archivos para identificar posibles intrusiones. Son eficaces para detectar actividades maliciosas en un host específico.
- **Basados en red (NIDS):** Se enfocan en el tráfico de red y analizan los paquetes que circulan por la red en busca de actividades sospechosas o ataques.

En este sentido, varios estudios han propuesto diferentes enfoques para la detección de ataques en sistemas IoT. Por ejemplo, Doshi y otros [40] proponen un enfoque de detección de ataques de denegación de servicio distribuido (DDoS) en dispositivos de IoT utilizando técnicas de aprendizaje automático. En su investigación, recolectaron datos de tráfico de red de dispositivos IoT en un entorno controlado y aplicaron un enfoque basado en flujo para identificar patrones

normales y anómalos. Evaluaron cinco algoritmos de aprendizaje automático: árboles de decisión [41], bosques aleatorios [42], SVM, KDTree y redes neuronales. Los cinco algoritmos tuvieron una precisión de conjunto de prueba superior a 0.99. Sin embargo, es importante tener en cuenta que el estudio se basa en un conjunto de datos de tráfico de red limitado en términos de tamaño y diversidad de dispositivos IoT, lo que puede afectar la generalización de los resultados a otros escenarios y dispositivos. Además, no se abordó el desbalance de los datos en el análisis.

Leevy y otros [43] proponen un modelo de detección de ataques de robo de información en dispositivos IoT. Para esto, utilizaron la base de datos BoT-IoT [29], que simula un entorno realista de dispositivos IoT y ataques. Los autores se enfocaron específicamente en la detección de ataques de robo de información, que es una de las principales amenazas para los dispositivos IoT. Realizan entrenamientos y pruebas para ocho clasificadores tales como CatBoost [44], LightGBM [45], XGBoost [46], bosques aleatorios, árboles de decisión, regresión logística [47], Naive Bayes [47] y perceptrón en estructura multicapa [48]. Para evaluar los modelos se calculan dos métricas de rendimiento: el área bajo la curva ROC (AUC) y el área bajo la curva Precisión-Recall (AUPRC). Como resultado se tiene que los modelos con mejor desempeño son CatBoost, LightGBM y XGBoost siendo LightGBM el modelo con mejor desempeño tomando un puntaje de 0.99 tanto para AUC como para AUPRC. Por último, el estudio no consideró la utilización de técnicas de mejora de datos para abordar el desequilibrio de clases en el conjunto de datos utilizado.

En otro estudio, Zeeshan y otros [28] presentan la detección de ataques DoS y DDoS mediante el uso de técnicas de aprendizaje profundo. Los investigadores emplearon dos conjuntos de datos UNSW-NB15 [49] y BoT-IoT. En cuanto a la detección de intrusiones, se propone un enfoque basado en el análisis del protocolo de red utilizando una arquitectura de red neuronal profunda que consta de capas convolucionales y recurrentes. Esta arquitectura es capaz de clasificar el tráfico en tres categorías: no anómalo, ataques DoS y ataques DDoS. Los resultados muestran que el modelo logra una exactitud (accuracy) del 96,3% en el ajuste del modelo. Es importante destacar que, si bien el artículo presenta la matriz de confusión y la precisión del modelo, no se proporciona una interpretación detallada de otras métricas de rendimiento, como la precisión, el recall y el F1-socore. Sería relevante considerar estas métricas adicionales para obtener una evaluación más completa del desempeño del modelo propuesto.

En el estudio realizado por Liu y sus colegas [50], se aplicaron algoritmos de aprendizaje automático para detectar anomalías en sistemas IoT. Específicamente, se entrenaron modelos de clasificación utilizando el conjunto de datos de intrusión de red IoT [51], el cual está disponible públicamente y es diseñado para simular un entorno IoT. En el estudio, se utilizaron varios clasificadores, incluyendo regresión logística, SVM, K-vecinos más cercanos (KNN), bosques aleatorios y XGBoost. El rendimiento de los modelos se evalúa mediante la matriz de confusión y métricas como el accuracy, F1-score, recall y tiempo de ejecución. El modelo de bosques aleatorios arrojó los puntajes métricos más altos, pero requiere de un mayor esfuerzo computacional. El segundo modelo con el accuracy más alto fue el de KNN, con un valor del 99%, y un tiempo de ejecución de aproximadamente 2 minutos.

En la investigación desarrollada por Mirsky y otros [52], se propone un sistema de detección de intrusos en redes IoT que utiliza redes neuronales y autoencoders para la detección eficiente de patrones anómalos en el tráfico de red. Este sistema permite monitorizar y detectar anomalías estadísticas en el tráfico reciente, utilizando autoencoders para obtener una representación compacta de los datos. Cada autoencoder se encarga de detectar anomalías relacionadas con un aspecto específico del comportamiento de la red. Además, se destaca que este sistema puede ser implementado en dispositivos de baja potencia como la Raspberry Pi, lo que lo hace práctico y económico.

Por otro lado, Popoola y otros [53] utilizan técnicas de aprendizaje automático profundo para reducir la dimensionalidad de las características del tráfico de red IoT. Mediante esta reducción, se construye un clasificador basado en aprendizaje profundo que puede ser biclase o multiclase. La evaluación del modelo se realiza utilizando la base de datos BoT-IoT, y se selecciona el optimizador adecuado utilizando la métrica de coeficiente de correlación de Matthews (MCC), especialmente diseñada para datos altamente desequilibrados. Los resultados muestran una reducción significativa del espacio de memoria necesario para el almacenamiento de datos de tráfico de red a gran escala, así como un rendimiento de clasificación superior al 93%.

En la TABLA III, se presenta un resumen de los trabajos analizados anteriormente, detallando los autores, la categoría relacionada con el control de acceso y el mecanismo de

seguridad utilizado. Esta tabla proporciona una visión general de las diversas fuentes y los enfoques empleados para abordar la seguridad en el control de acceso.

TABLA III
ARTÍCULOS SELECIONADOS PARA LA REVISIÓN DE MECANISMOS DE SEGURIDAD

Ref.	**Año**	**Categoría**	**Mecanismos de seguridad**
[32]	2018	Autenticación	Certificados X.509
[33]	2022	Autenticación	Certificados JWT
[34]	2021	Autenticación	JWT Autenticación jerárquica
[35]	2021	Autenticación	SSO Blockchain
[36]	2019	Autorización	Control de acceso basado en roles Oauth 2.0
[37]	2018	Autenticación Autorización	OpenID SSO Oauth 2.0 JWT
[38]	2019	Autenticación Autorización	Blockchain Contratos inteligentes Control de acceso basado en capacidades
[40]	2018	Detección de intrusos	Técnicas tradicionales Redes neuronales
[43]	2021	Detección de intrusos	Técnicas tradicionales Redes neuronales
[28]	2022	Detección de intrusos	Aprendizaje profundo
[50]	2020	Detección de intrusos	Técnicas tradicionales
[52]	2018	Detección de intrusos	Redes neuronales
[53]	2021	Detección de intrusos	Aprendizaje profundo

D. Seguridad en la plataforma Snap4City

Como se mencionó previamente, la seguridad y la gestión de la privacidad de los datos presentan crecientes desafíos en la actualidad. Estos desafíos van más allá de simplemente establecer conexiones seguras, ya que también implican definir reglas claras para la gestión y acceso a los datos. En esta sección se realiza una revisión de los mecanismos de seguridad que se emplean en la plataforma Snap4City previamente al diseño de la estrategia de seguridad.

La plataforma Snap4City cumple con el Reglamento General de Protección de Datos (GDPR) de la Unión Europea [5], lo que garantiza la privacidad y seguridad de los datos, los dashboards, los dispositivos IoT, las aplicaciones IoT, la información personal, las analíticas de datos y los procesos. Además, la plataforma permite a los usuarios transferir dichos elementos, de los cuales son propietarios, a otros usuarios que utilizan la plataforma.

En un estudio realizado por Badii y otros [13], se abordan los problemas de privacidad y seguridad relacionados con el diseño de la plataforma Snap4City. Los autores presentan los requisitos principales que deben cumplir las plataformas IoT para ciudades inteligentes, abordan las vulnerabilidades en seguridad que estas plataformas deben evitar y detallan los mecanismos de seguridad implementados en la infraestructura de Snap4City.

La plataforma utiliza el módulo de inicio de sesión único (SSO) basado en el protocolo OpenIDConnect para verificar y distribuir la identidad del usuario. OpenIDConnect es un protocolo basado en OAuth 2.0 que ofrece un sistema de autenticación seguro y unificado para todos los módulos de Snap4City. Mediante OpenIDConnect, se enriquece la autenticación al incluir la identificación del usuario. Esto garantiza que los usuarios puedan acceder de manera segura y sin problemas a los diferentes componentes de la plataforma Snap4City.

Todas las comunicaciones entre los distintos módulos de Snap4City se realizan a través del protocolo SSL/TLS (Secure Sockets Layer/Transport Layer Security). Esto asegura que la transmisión de datos se mantenga confidencial y protegida. Además, se utilizan tokens de acceso en forma de JWT como mecanismo para compartir información confidencial de forma temporal, lo que permite que el sistema SSO funcione de manera segura entre los diferentes módulos.

El Registro de Usuarios en Snap4City se administra parcialmente a través de un servicio de información de directorio distribuido basado en el protocolo LDAP (Lightweight Directory Access Protocol). Este módulo LDAP solo es accesible desde una subred interna privada, lo que facilita su protección contra posibles ataques. Además, los usuarios tienen la capacidad de revisar y actualizar su información, así como solicitar la eliminación completa de sus datos de acuerdo con los requisitos del Reglamento General de Protección de Datos (GDPR).

Para otorgar acceso a las funcionalidades de Snap4City, se utilizan roles como Manager, AreaManager, ToolAdmin y RootAdmin para clasificar a los usuarios según su nivel de confianza y responsabilidad. Además, cada organización puede establecer sus propios grupos y definir roles personalizados según sus necesidades.

Dado que la mayoría de los IoT Brokers no admiten autenticación nativa ni protección de canales, Snap4City cuenta con un IoT Firewall para garantizar una transmisión de datos segura entre los dispositivos IoT y los dispositivos de borde (edge) con la plataforma. El IoT Firewall establece una autenticación mutua y una conexión segura mediante HTTPS, utilizando los protocolos OMA NGSI Ver.1 y Ver.2 (Open Mobile Appliance, Next Generation Service Interfaces). Además, el firewall de IoT admite otros mecanismos de autenticación, como tokens de acceso, pares de claves y autenticación básica a través de HTTPS. A continuación, en la TABLA IV se muestra un resumen de los principales mecanismos de seguridad utilizados en Snap4City.

TABLA IV
MECANISMO DE SEGURIDAD EN SNAP4CITY

Aspecto de seguridad	**Mecanismo de seguridad**
Autenticación de usuarios	SSO + OpenIDConnect + OAuth 2.0
Gestión de usuarios	LDAP
Autorización de usuarios	Basada en roles
Comunicación entre módulos	SSL/TLS + JWT
Protección IoT	Firewall IoT: HTTPS, token de acceso, parejas de claves

En conclusión, la plataforma IoT esta respalda por mecanismos de autenticación y autorización, sin embargo, no cuenta con un sistema que permita detectar ataques del tipo de denegación de servicio, que puedan agotar los recursos de la plataforma. Adicionalmente, la plataforma se instala dentro de las instalaciones de la Universidad de Antioquia, donde puede ser víctima de ataques internos o externos. Por lo tanto, se requiere complementar la seguridad con un sistema de detección de intrusos que permita alertar en tiempo real posibles actividades de ataques dirigidos hacia el sistema IoT.

E. Estrategia de seguridad propuesta

Con base en los resultados de la búsqueda bibliográfica sobre las principales vulnerabilidades y amenazas de las plataformas IoT, así como a la identificación de los mecanismos de seguridad con los que cuenta la plataforma seleccionada en este trabajo de investigación, Snap4City, se llega a la conclusión que la estrategia de seguridad a desarrollar es un sistema de detección de intrusos. Este sistema se considera un mecanismo de defensa adicional que complementa las estrategias de seguridad ya implementadas con el fin de garantizar la protección y seguridad integral de la plataforma.

Como se mencionó, la plataforma IoT puede estar expuesta a diversas amenazas y ataques cibernéticos, como el acceso no autorizado, DoS, la inyección de código malicioso y el robo de información sensible. Un sistema de detección de intrusos permite identificar y responder proactivamente a estos incidentes, proporcionando una capa adicional de protección.

Por otra parte, a pesar de tener medidas de autenticación y autorización sólidas, siempre existe la posibilidad de que un usuario malicioso logre evadir estas barreras y acceda a la plataforma. Un sistema de detección de intrusiones complementa el control de acceso al monitorear activamente el tráfico de la red e identificar patrones o actividades sospechosas que podrían indicar un acceso no autorizado.

El sistema de detección de intrusos basado en red debe contar con la capacidad de detectar ataques en tiempo real, mediante el análisis de patrones de actividades maliciosas conocidas, principalmente los ataques DoS, utilizando modelos de aprendizaje automático entrenados. Para implementar esta estrategia de seguridad, se seguirán los siguientes pasos:

- **Creación de un escenario de prueba:** Se desarrollará un entorno de prueba para validar el funcionamiento del IDS. Esto permitirá simular diferentes tipos de ataques y evaluar la efectividad del sistema en la detección de ataques.
- **Construcción de modelos de aprendizaje automático:** Se entrenarán modelos de aprendizaje automático utilizando conjuntos de datos relevantes que contengan ejemplos de actividades

maliciosas. Estos modelos principalmente identificarán patrones en el flujo de datos en la red para detectar posibles ataques.

- **Desarrollo e implementación del sistema de detección de ataques:** Con base en los modelos de aprendizaje automático entrenados, se procederá al desarrollo del sistema de detección de ataques. Este sistema analizará en tiempo real el tráfico de red, identificará comportamientos de actividades maliciosas.
- **Verificación del sistema propuesto:** En el escenario de prueba se simularán ataques para determinar la eficacia del sistema en la detección de ataques en tiempo real.

IV. METODOLOGÍA

La detección en tiempo real de ataques puede considerarse como una medida de seguridad dentro de la estrategia de control de acceso a los recursos o datos. Esta técnica es necesaria para identificar posibles amenazas y permitir la toma de decisiones tempranas para permitir o denegar el acceso al sistema IoT a usuarios y dispositivos. Al permitir la detección temprana de intentos de accesos no autorizados, se puede tomar acciones correctivas antes de que se produzcan daños en la red. De esta manera, la detección en tiempo real de ataques se convierte en una herramienta clave en la prevención y mitigación de riesgos en la seguridad del sistema.

Para el desarrollo de la estrategia de seguridad se consideraron tres fases: inicialmente se implementó un escenario de prueba con el fin de disponer los recursos necesarios para validar los modelos de aprendizaje automático, mediante la estimación de un conjunto de métricas de rendimiento; luego se seleccionaron y construyeron los modelos de aprendizaje automático a evaluar y finalmente se desarrolló el sistema de detección de ataques en tiempo real para una red IoT.

A. Implementación del escenario de prueba

Para el escenario de prueba se diseñó una infraestructura de red, como se ilustra en la Fig. 1, el cual permite el despliegue del sistema de detección ataques monitoreando el tráfico de red en tiempo real. La estructura de la red consta de varios componentes claves: una máquina víctima que actúa como objetivo de los atacantes, un extractor de tráfico de red, un detector de ataques y varios dispositivos que generan ataques de manera programada o aleatoria.

Este escenario de prueba puede ser considerado una representación real de la aplicación IoT plateada en el proyecto en el cual se enmarca este trabajo, ya que cuenta con la plataforma IoT Snap4City, una interfaz de usuario para la visualización de datos y un flujo de datos simulados que se asemeja al flujo real.

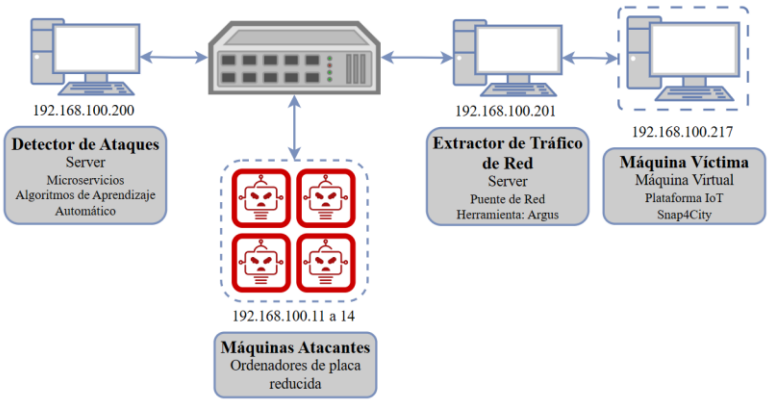

Fig. 1 Diagrama de la infraestructura de red para la evaluación del sistema de detección de ataques en tiempo real

A continuación, se detallan los distintos componentes que integran la infraestructura de red.

1) Máquina víctima

La máquina víctima se utiliza como objetivo para los atacantes en el escenario de prueba y en ella se despliega la plataforma IoT basada en microservicios Snap4City para simular un entorno realista del sistema IoT. La máquina virtual cuenta con una memoria RAM de 8GB, 8 CPUs, 200GB de almacenamiento en disco virtual y dos adaptadores de red.

En las aplicaciones IoT existen diversos tipos de nodos cuyo periodo de trasmisión de datos puede variar desde segundos a días, para generar un buen flujo de datos normales en esta máquina se simulan nodos IoT que transmiten datos aleatorios cada 5 y 10 segundos. Estos datos se pueden visualizar a través de una interfaz de usuario. Además, desde un script desarrollado en Python, se realizan solicitudes continuas a microservicios desplegados en otro servidor mediante una API REST. Para realizar estas solicitudes HTTP, se utiliza la biblioteca `request` de Python.

El algoritmo comienza definiendo varias variables, como las URLs de los endpoints a llamar, los valores necesarios para la autenticación y los headers necesarios para cada solicitud. A continuación, se inicia un bucle infinito que realiza una secuencia de solicitudes HTTP a los diferentes endpoints. La primera solicitud es una petición HTTP POST para autenticarse y obtener un token de acceso. La segunda solicitud HTTP GET se utiliza para realizar una verificación de conexión a un microservicio. Luego, se realiza una tercera solicitud HTTP GET que utiliza el token

de acceso obtenido en la primera solicitud para recibir una lista de usuarios registrados. Este proceso se repite continuamente dentro del bucle cada 30 segundos.

2) Extractor de tráfico de red

El extractor de tráfico de red se encarga de analizar y vigilar el flujo de información en una red para obtener información útil sobre los paquetes de datos que circulan en ella. Esta información puede incluir características tales como la dirección IP de origen y destino, el protocolo utilizado, el tamaño de los paquetes y la hora en que se envían y reciben. La herramienta utilizada para realizar el monitoreo es Open Argus [54], la cual permite la captura y el análisis de datos en tiempo real.

El servidor donde se instala el extractor de tráfico no sólo cumple la función de analizar el tráfico de red, sino que también sirve como intermediario para capturar los paquetes de datos que se envían y reciben desde la máquina víctima. Esto permite obtener información valiosa y relevante sobre el tráfico de red. En otras palabras, el servidor actúa como un puente de red para obtener los datos que necesita para analizar el tráfico en tiempo real. Este servidor utiliza el sistema operativo Ubuntu 18.04.6 LTS, tiene una memoria RAM de 192GB y utiliza una CPU Intel(R) Xeon(R) 2.00GHz con 88 CPU(S).

3) Detector de ataques

Este componente que se encarga de identificar patrones de tráfico de red que podrían estar asociados a ataques informáticos. Para ello, utiliza técnicas de análisis de patrones para detectar comportamientos sospechosos en el tráfico de red que sean característicos de un ataque a través de modelos de aprendizaje automático. Dichos modelos son entrenados utilizando conjuntos de datos que contienen patrones de tráfico normal y de tráfico de ataque.

El detector de ataques se desarrolló utilizando una arquitectura basada en microservicios empleando contenedores. De esta manera, se obtiene una mayor flexibilidad y escalabilidad en el despliegue de la herramienta. Los microservicios se despliegan en un servidor que tiene un sistema

operativo Ubuntu 20.04.4 LTS, cuenta con una memoria RAM de 15GB, una arquitectura x86_64 y utiliza una CPU Intel(R) Xeon(R) 2.00GHz con 4 CPU(S).

4) Máquinas atacantes

Las máquinas atacantes son un conjunto de ordenadores de placa reducida que se usan para realizar ataques de DoS a la máquina objetivo. En particular, se ha elegido el modelo Khadas VIM3, que cuenta con el sistema operativo Ubuntu 18.04.6 LTS, una memoria RAM LPDDR4/4X de 4GB y utiliza una arquitectura Big-Little de 6 cores: Quad core ARM Cortex-A73 de 2.2GHz y dual core Cortex-A53 de 1.8GHz.

Cada uno de estos sistemas está equipado con herramientas de ataque, tales como Hping3 [55], GoldenEye [56] y Hulk [57], las cuales son de código abierto y se utilizan para generar grandes cantidades de tráfico de red con el objetivo de afectar el rendimiento del sistema objetivo.

B. Construcción de los modelos de aprendizaje automático

Para desarrollar los modelos de aprendizaje automático para la detección de ataques se sigue un proceso que consta de tres fases principales: exploración y selección de datos de entrenamiento, preprocesamiento de los datos y entrenamiento y evaluación de los modelos. Para la construcción de los modelos se utilizaron las librerías Numpy, Pandas y Scikit-Learn en el lenguaje de programación Python.

1) Exploración y selección de datos de entrenamiento

Como se mencionó previamente, el detector de ataques utiliza técnicas de aprendizaje automático para analizar el tráfico de una red y determinar si es flujo de red normal o generado por un ataque. Para que el sistema pueda realizar esta tarea, primero debe ser entrenado con un conjunto de datos de red (una base de datos) con patrones de tráfico normales y bajo ataque.

Existen varios conjuntos de datos a disposición del público que pueden ser utilizados en los sistemas de detección de intrusos, cada uno con características distintas, como se muestra en la TABLA V.

TABLA V
CARACTERÍSTICAS DE LAS BASES DE DATOS

Parámetro	KDD99	UNSW-NB15	CICIDS2017	BOT-IOT
Número de registros	4.898.431	2.540.044	2.830.743	73.370.443
Familias de ataques	4	9	6	4
Número de atributos	42	49	80	46
Tráfico realista	No	Sí	Sí	Sí
Tráfico en el contexto IoT	No	No	No	Sí
Datos Etiquetados	Sí	Sí	Sí	Sí

De acuerdo con esta información se ha decidido utilizar tres bases de datos para el entrenamiento de los modelos. En primer lugar, se utiliza BoT-IoT ya que cubre dispositivos IoT en el tráfico de paquetes, está formado por una buena cantidad de registros, cuenta con diversos tipos de ataques y es uno de los conjuntos de datos más destacado y actualizado para detectar ataques en redes de internet de las cosas. UNSW-NB15 para enriquecer el conjunto de datos debido a que BoT-IoT contiene pocos registros de tráfico normal. Y una base de datos propia construida en el escenario de prueba, la cual contiene el comportamiento del tráfico normal de la arquitectura de red construida y patrones de ataques de DoS. A continuación, se describen cada una de estas bases de datos:

En los apartados siguientes, se describen las bases de datos utilizadas para el entrenamiento de los modelos:

a) Base de datos BoT-IoT [58]

Es un conjunto de datos diseñado para el análisis de tráfico de red en el contexto del internet de las cosas. El entorno de prueba de BoT-IoT es un escenario que simula el comportamiento de dispositivos IoT en una red. Este banco de prueba está conformado por máquinas virtuales atacantes que se encargan de ejecutar acciones maliciosas, como ataques de DoS, DDoS, recopilación y robo de información. También se incluyen máquinas virtuales tanto normales como

vulnerables para simular clientes de red que realizan solicitudes. Además, cuenta con un servidor Ubuntu para la implementación de una serie de servicios y simulación de sensores, en donde se incluyen escenario IoT tales como una estación meteorológica, un refrigerador inteligente, luces activadas por movimiento, una puerta de garaje controlada remotamente y un termostato inteligente.

El banco de prueba cuenta con un firewall para controlar y supervisar el tráfico de red (generado en el banco de pruebas) que sirve para facilitar el proceso de etiquetado de los datos. Además, se tiene una máquina virtual Ubuntu configurada para escuchar todo el tráfico de red que fluye a través de la red simulada. Se utiliza la herramienta Tshark [59] para capturar tanto los paquetes de red normales como los de ataque. Para generar una cantidad masiva de tráfico de red normal, se utilizó la herramienta Ostinato [60] debido a su flexibilidad para generar tráfico benigno realista con IPs y puertos específicos.

La base de datos BoT-IoT está compuesta por varios conjuntos y subconjuntos, los cuales difieren en tamaño, formato y número de características. Para el desarrollo de este trabajo, se ha seleccionado un subconjunto que representa el 5% de la base de datos original. Esta elección se debe a que este subconjunto es más pequeño y manejable, lo que facilita su uso en el análisis y detección de ataques en dispositivos IoT [61]. Este conjunto de datos se compone de 4 archivos CSV con un tamaño total de 970 MB y cuenta con más de 3 millones de registros que incluyen 43 características y tres funciones de etiqueta. Una de las funciones de etiquetas permite clasificar el tráfico de red como normal o ataque [58]. En la TABLA VI se presenta la distribución de los registros en el conjunto de datos BoT-IoT. Este es un conjunto de datos enfocado en almacenar principalmente registros de ataque, en su mayoría son ataques DoS, los cuales generan una gran cantidad de solicitudes.

TABLA VI
DISTRIBUCIÓN DE LOS REGISTROS EN EL SUBCONJUNTO DEL 5% DE LA BASE DE DATOS BOT-IOT

Clase	**Número de registros**
Normal	643
Ataque	3667879
Total	**3668522**

b) Base de datos UNSW-NB15 [62]

Es otra base de datos diseñada para evaluación y prueba de los sistemas de detección de intrusos o ataques en redes. Para crear esta base de datos, se utilizó la herramienta llamada IXIA PerfectStorm para generar un híbrido de actividades normales modernas reales y comportamientos de ataque contemporáneos sintéticos que se actualizan continuamente a partir de un diccionario público de vulnerabilidades y exposiciones de seguridad informática conocidas [49]. El tráfico de red generado en la red se captura utilizando una herramienta llamada Tcpdump, y se procesa para crear características confiables utilizando otras herramientas como Argus y Bro-IDS.

Durante el periodo de simulación de 16 y 15 horas se capturó información del tráfico que circula por la red y se almacena en archivos Pcap, el cual tiene una cantidad total de 100GB. El conjunto de datos contiene más de dos millones de registros que se almacenan en 4 archivos CSV con un tamaño total de 559 MB [62]. Cada registro incluye 47 características que contienen información como la hora de inicio y finalización de la conexión, las direcciones IP y los puertos de origen y destino, la cantidad de paquetes y bytes transferidos, así como las tasas de direcciones IP promedio en 100 conexiones. Además, cada registro esta etiquetado con dos funciones distintas: una es binaria y se utiliza para identificar el tráfico como normal o de ataque mientras que la otra es multiclase y contiene nueve categorías de ataques. La TABLA VII presenta la distribución de los registros en el conjunto de datos UNSW-NB15.

TABLA VII
DISTRIBUCIÓN DE LOS REGISTROS DE LA BASE DE DATOS UNSW-NB15

Clase	**Número de registros**
Normal	2218764
Ataque	321283
Total	**2540047**

c) Generación de una base de datos propia

Mediante la infraestructura de red explicada en la sección IV-A se construyó una base de datos propia, almacenando la información de los paquetes de red que circulan en el escenario de prueba.

La herramienta Argus fue utilizada para llevar a cabo una auditoría, en tiempo real, del flujo de paquetes que se captura y transmite a una de las interfaces de red del sistema. Este flujo de paquetes incluye tanto actividades normales como de posibles ataques, lo que permite monitorear y analizar el tráfico de red en busca de patrones o comportamientos sospechosos. El tráfico capturado se escribe en un formato binario de Argus, y luego con la misma herramienta se extraen las características que describen a los paquetes de red. Los datos son almacenados en una base de datos MongoDB y luego son extraídos en un archivo CSV, alcanzado un tamaño de 1,25GB, obteniendo finalmente 3595500 de registros. En la TABLA VIII se detalla la distribución de los datos. Los datos se etiquetan ya que se tienen identificadas las máquinas atacantes.

TABLA VIII
DISTRIBUCIÓN DE LOS REGISTROS DE LA BASE DE DATOS PROPIA

Clase	**Número de registros**
Normal	341423
Ataque	3254077
Total	**3595500**

Con el fin de seleccionar las características a utilizar, se identificaron las características comunes presentes en las bases de datos BoT-IoT y UNSW-NB15, siguiendo las definiciones dadas por los autores. Las características similares seleccionadas también son extraídas en la base de datos propia, resultando en un total de 27 características basadas en el flujo de red, las cuales se utilizaron en la construcción de modelos. Se utilizó una clasificación binaria para determinar si existe un ataque o no en la infraestructura de red. De estas características, 15 son proporcionadas directamente por la herramienta Argus (ver TABLA IX), mientras que las 12 características restantes se calculan a partir de las proporcionadas por Argus (ver TABLA X), utilizando expresiones específicas. Estas características adicionales suministran información más detallada sobre el comportamiento del tráfico de red, lo que permite una mejor identificación de posibles amenazas en la red. La TABLA IX presenta las características del flujo de red entregadas por Argus, mientras que en la TABLA X se muestran las características calculadas a partir de estas características de flujo de red.

TABLA IX
CARACTERÍSTICAS GENERADAS POR LA HERRAMIENTA ARGUS

Categoría	Característica	Descripción
Información de tiempo	stime	Tiempo de inicio del flujo
	ltime	Tiempo de la última actividad del flujo
	dur	Duración del flujo
Información de protocolo	proto	Protocolo utilizado en el flujo
Información de direcciones y puertos	saddr	Dirección IP de origen
	daddr	Dirección IP de destino
	sport	Puerto de origen
	dport	Puerto de destino
Información de tráfico	pkts	Cantidad de paquetes
	bytes	Cantidad de bytes
	spkts	Cantidad de paquetes desde la dirección IP de origen
	dpkts	Cantidad de paquetes desde la dirección IP de destino
	sbytes	Cantidad de bytes desde la dirección IP de origen
	dbytes	Cantidad de bytes desde la dirección IP de destino
Información de estado	state	Estado del flujo

TABLA X
CARACTERÍSTICAS CALCULADAS A PARTIR DE LAS ENTREGADAS POR ARGUS

Categoría	Característica	Descripción
Estadísticas de conexión	TnBPSrcIP	Número total de bytes por IP origen en 100 conexiones
	TnBPDstIP	Número total de bytes por IP destino en 100 conexiones
	TnP_PSrcIP	Número total de paquetes por IP origen en 100 conexiones
	TnP_PDstIP	Número total de paquetes por IP destino en 100 conexiones
	TnP_PerProto	Número total de paquetes por protocolo en 100 conexiones
	TnP_Per_Dport	Número total de paquetes por puerto de destino en 100 conexiones
	AR_P_Proto_P_SrcIP	Tasa promedio por protocolo por IP origen en 100 conexiones (Calculado por paquetes/duración)
	AR_P_Proto_P_DstIP	Tasa promedio por protocolo por IP destino en 100 conexiones
	N_IN_Conn_P_SrcIP	Número de conexiones entrantes por IP origen en 100 conexiones
	N_IN_Conn_P_DstIP	Número de conexiones entrantes por IP destino en 100 conexiones
	AR_P_Proto_P_Sport	Tasa promedio por protocolo por puerto de origen en 100 conexiones
	AR_P_Proto_P_Dport	Tasa promedio por protocolo por puerto de destino en 100 conexiones

Un análisis de las tres bases de datos previamente mencionadas mostró que existía una distribución desigual entre los registros de tráfico normal y de ataques. En BoT-IoT, el 0,018% de los registros son tráfico normal, en UNSW-NB15 el 87,26% son tráfico normal, y en la base de datos propia se tiene un 9,49% de tráfico normal. Con el fin de balancear las muestras para el entrenamiento y prueba, se construyó una base de datos balanceada mediante una submuestra con una distribución equitativa de datos. En el caso de BoT-IoT, se toman todos los registros de tráfico normal y se seleccionan 200.000 registros aleatorios de tráfico etiquetados como ataques. Para UNSW-NB15, solo se toman 199.357 registros de tráfico normal. En el conjunto de datos propio, se toman todos los registros de tráfico normal y se seleccionan 341.423 registros aleatorios de tráfico de ataque. De esta manera, se logra una distribución equilibrada en la base de datos, obteniendo un total de 541.423 registros para el tráfico normal y 541.423 registros para el tráfico de ataque.

2) Preprocesamiento de datos

Con el fin de adecuar los datos para construir los modelos mediante aprendizaje automático, se llevó a cabo un proceso de adecuación de los datos. Este proceso se aplicó a las bases de datos utilizadas, con el fin de mejorar su calidad. A continuación, se detalla cada una de las acciones de preprocesamiento implementadas en los datos:

a) Características categóricas

En el conjunto de datos considerando existen dos variables categóricas que son relevantes para la detección de posibles ataques en la red: el tipo de protocolo utilizado en el flujo ($proto$) y el estado del flujo ($state$). Ambas variables han sido transformadas a valores enteros para su uso en los algoritmos de aprendizaje automático. La variable $state$ cuenta con 16 categorías diferentes, que se muestran en la TABLA XI, mientras que la variable $proto$ se compone de 5 categorías, que se detallan en la TABLA XII.

TABLA XI
DESCRIPCIÓN DE LA VARIABLE CATEGÓRICA ESTADO DEL FLUJO

state	**Valor entero**	**Descripción**
CON	0	Establecimiento de conexión
INT	1	Servicio interno al host, no necesita conexión
FIN	2	Finalización de la conexión
NRS	3	Estado de red no válido
RST	4	Reinicio de la conexión
URP	5	Mensaje urgente enviado
REQ	6	Petición enviada
ACC	7	Aceptación de la conexión
TST	8	Prueba de conexión
ECO	9	Eco del mensaje
MAS	10	Segmento largo del mensaje
CLO	11	Cierre de conexión
TXD	12	Transmisión de datos
ECR	13	Eco de respuesta
URN	14	Mensaje urgente recibido
Otro	15	Para representar otro tipo de estados no incluidos

TABLA XII
DESCRIPCIÓN DE LA VARIABLE CATEGÓRICA TIPO DE PROTOCOLO

proto	**Valor entero**	**Descripción**
ARP	0	Protocolo de resolución de direcciones
TCP	1	Protocolo de control de transmisión
UDP	2	Protocolo de datagramas de usuario
ICMP	3	Protocolo de mensajes de control de Internet
Otro	4	Para representar otro tipo de protocolos no incluidos

b) Corrección de etiquetas

De acuerdo con una revisión y análisis sobre la base datos BoT-IoT realizada por Peterson et al. [61], se ha encontrado que el protocolo ARP no contribuye a las acciones de ataque reportadas en dicha base de datos. Sin embargo, se han identificado instancias en las que se utiliza el protocolo ARP y se les ha asignado erróneamente la etiqueta de ataque. Con el objetivo de corregir esta situación, se ha decido cambiar la etiqueta de dichas instancias de forma que todas queden

etiquetadas como tráfico normal y no como ataques. De esta manera, se puede mejorar la calidad de la base de datos y su uso apropiado para la detección de posibles ataques.

c) Eliminación de características

En las bases de datos utilizadas para la detección de posibles ataques en la red, algunas características no son útiles para la creación de modelos de predicción, ya que están diseñadas para identificar la fuente y el destino del paquete y no aportan información generalizable [43]. Para mejorar la calidad de los datos y reducir el ruido innecesario, se elimina la información de dirección y puerto tanto del origen como del destino. Esto reduce la cantidad de variables y puede aumentar la precisión del modelo de predicción, así como facilitar su interpretación.

d) Estandarización de los datos

La normalización de los datos es una técnica común en la preparación de datos para su uso en modelos de aprendizaje automático. A menudo, las características o variables de un conjunto de datos tienen escalas de valores muy diferentes y comportamientos variados. Para estandarizar los datos de entrada, buscando que las características tengan la misma importancia, se estableció que estas deben tener una distribución normal con media 0 y desviación estándar 1, lo cual se puede calcular utilizando la ecuación (1).

$$z = \frac{x - \mu}{\sigma} \tag{1}$$

En esta ecuación, z es el valor estandarizado, x es el valor original de la característica o variable, μ es la media de la variable y σ es la desviación estándar de la variable en el conjunto de datos. De esta manera, se resta la media de la variable de cada valor de la variable original y luego divide el resultado por la desviación estándar de la variable.

La implementación de la estandarización de los datos de entrada se realiza utilizando la función StandardScaler de la biblioteca Scikit-learn (Sklearn) de Python [63]. Esta función automatiza el proceso de estandarización de los datos y se encarga de calcular la media y la

desviación estándar de cada variable del conjunto de datos. Luego, aplica la fórmula para estandarizar los datos y retornar un conjunto de datos normalizados.

3) Entrenamiento y evaluación de los modelos

En el proceso de construcción del modelo de detección de ataques en sistemas IoT, de acuerdo con el estado del arte realizado en la sección III-C-3, se seleccionan cuatro modelos tradicionales de clasificación de aprendizaje automático supervisado: árboles de decisión, bosques aleatorios, regresión logística y SVM. Según lo encontrado en la literatura, son los más utilizados en clasificación offline (fuera de línea). Sin embargo, en este trabajo se quiere comprobar su comportamiento en un escenario en línea, usando la integración de varias bases de datos para el entrenamiento de los modelos. Además, estos modelos fueron escogidos con el propósito de tener un punto de referencia para la comparación. Para implementar estos modelos, se utilizó la librería *Sklearn* y sus instancias *RandomForestClassifier* [64], *DecisionTreeClassifier* [65] y *SGDClassifier* [66].

Para el entrenamiento se utilizó una submuestra equilibrada de 1.082.846 registros, dividida en 70% para el entrenamiento (*train*) y un 30% para la evaluación y prueba de los modelos (*test*). Durante el proceso de la construcción de los modelos mediante aprendizaje automático, se prestó atención a los hiperparámetros, que son configuraciones ajustables que afectan el comportamiento y la eficacia del modelo. Una elección adecuada de los hiperparámetros es fundamental para maximizar la precisión y generalización del modelo.

Por lo tanto, se utilizó una técnica de validación cruzada para refinar los hiperparámetros del modelo. Esta técnica consiste en dividir los datos de entrenamiento en varias particiones o folds y entrenar el modelo en uno de ellos mientras se evalúa su rendimiento en los *folds* restantes. En este caso, se utilizó una validación cruzada de *3-fold*. Esto significa que los datos se dividen en tres subconjuntos. Por otra parte, se utilizó la métrica de exactitud (accuracy) para evaluar el rendimiento del modelo y seleccionar los mejores hiperparámetros. En la TABLA XIII se describen los hipermarámetros utilizados en los modelos y los diferentes valores que pueden tener.

TABLA XIII
HIPERPARÁMETROS PARA CADA MODELO DE APRENDIZAJE AUTOMÁTICO

Modelo	Hiperparámetro	Descripción	Valores
Árboles de Decisión	max_depth	Controla la profundidad máxima del árbol	$8, 10, 12, 14$
	min_smaple_split	Establece el número mínimo de muestras necesarias para dividir un nodo interno	$2, 4, 6$
Bosques Aleatorios	$n\_estimators$	Determina el número de árboles que se deben construir	$100, 200, 300$
	max_depth	Controla la profundidad máxima de cada árbol	$20, 22, 24$
Regresión Logística	$alpha$	Establece el parámetro de regularización	$10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}$ $10^{-2}, 10^{-1}, 1, 10$
	$penalty$	Especifica la norma de regularización	$l1, l2$
SVM	$alpha$	Establece el parámetro de regularización	$10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}$ $10^{-1}, 1, 10, 100$
	$penalty$	Especifica la norma de regularización	$l1, l2$

Una técnica para encontrar los mejores hiperparámetros para cada modelo es la utilización de GridSearch, la cual permite automatizar la búsqueda de los mejores valores para los hiperparámetros. En este trabajo, se utilizó *GridSearchCV* de la librería *Sklearn*, que emplea validación cruzada para evaluar el rendimiento de cada combinación de hiperparámetros. Para cada modelo se definieron diferentes combinaciones de hiperparámetros y se evaluaron utilizando como métrica el accuracy para seleccionar los mejores valores.

Con los mejores hiperparámetros obtenidos con el procedimiento descrito, se pueden entrenar los modelos con la combinación óptima de parámetros. Una vez entrenados, se llevan a cabo las predicciones con los datos de prueba y se evalúa la precisión, y la generalización de los modelos.

Finalmente, para evaluar el rendimiento de los modelos, se utilizan las métricas: matriz de confusión, la exactitud o accuracy, la sensibilidad o recall, la precisión y el valor F1 o F1-score. Adicionalmente, se calcula el tiempo que tarda en realizar las predicciones para los datos de prueba.

La matriz de confusión proporciona información sobre el rendimiento de los modelos de clasificación como se muestra en la TABLA XIV.

TABLA XIV
REPRESENTACIÓN DE LA MATRIZ DE CONFUSIÓN

		Predicción	
		Negativa	Positiva
Real	Negativa	Verdaderos Negativos (TN)	Falsos Positivos (FP)
	Positiva	Falsos Negativos (FN)	Verdaderos Positivos (TP)

El accuracy mide el número de observaciones predichas correctamente respecto al total de observaciones. Se calcula mediante la ecuación 2.

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2}$$

El recall es la proporción de casos positivos que el modelo predijo correctamente en relación con todos los casos positivos reales. Se calcula mediante la ecuación 3.

$$reacall = \frac{TP}{TP + FN} \tag{3}$$

La precisión es la proporción de casos positivos predichos correctamente por el modelo en relación con todos los casos clasificados como positivos. Se calcula mediante la ecuación 4.

$$precision = \frac{TP}{TP + FP} \tag{4}$$

El F1-score se calcula como la media armónica de la precisión y el recall, y se define mediante ecuación 5.

$$F1 = 2 * \frac{presicion * recall}{presicion + recall} \tag{5}$$

C. Desarrollo e implementación del sistema de detección de ataques en tiempo real

En esta fase se cuenta con dos componentes fundamentales para el monitoreo y detección de ataques en el tráfico de red. El primer componente es el extractor de tráfico, el cual realiza una auditoría en tiempo real de los flujos del tráfico que circulan por la red. Y el segundo componente es el detector On-line de ataques, el cual utiliza modelos de aprendizaje automático para analizar y evaluar los flujos de tráfico que han sido recolectados previamente por el extractor de tráfico.

1) Extractor de tráfico

Como se mencionó previamente, este componente es esencial para la captura y análisis de los paquetes que transitan por una interfaz de red. Su función principal es extraer las características relevantes que describen a los paquetes que llegan, para crear registros de datos con los atributos necesarios para los modelos de aprendizaje automático. Para llevar a cabo esta tarea, se utilizó la herramienta de código abierto Open Argus, que permite la captura y análisis de paquetes de red y proporciona información detallada sobre las transacciones que detecta.

Open Argus organiza los paquetes de red en datos de flujo de red periódicos, que son útiles para el análisis del comportamiento de la red. La herramienta es capaz de realizar un seguimiento de flujo bidireccional, es decir, captura los paquetes que se transmiten en ambas direcciones entre un dispositivo de origen y un dispositivo de destino. Para identificar un flujo de red, Argus utiliza un conjunto de cinco valores, como la dirección IP de origen y destino, el puerto de origen y destino y el tipo de protocolo.

Una vez recolectados los datos de flujo de red por la herramienta Argus, mediante un script elaborado en Python son enviados al detector de ataques. Este script garantiza una transferencia de datos eficiente y segura. En Fig. 2 se muestra un esquema del extractor de tráfico de red. Usando la herramienta Argus, se capturan todos los paquetes que pasan por una interfaz de red del sistema. Posteriormente, la misma herramienta analiza los paquetes y para cada registro de datos, se extraen 15 características que son recibidas por el script de Python.

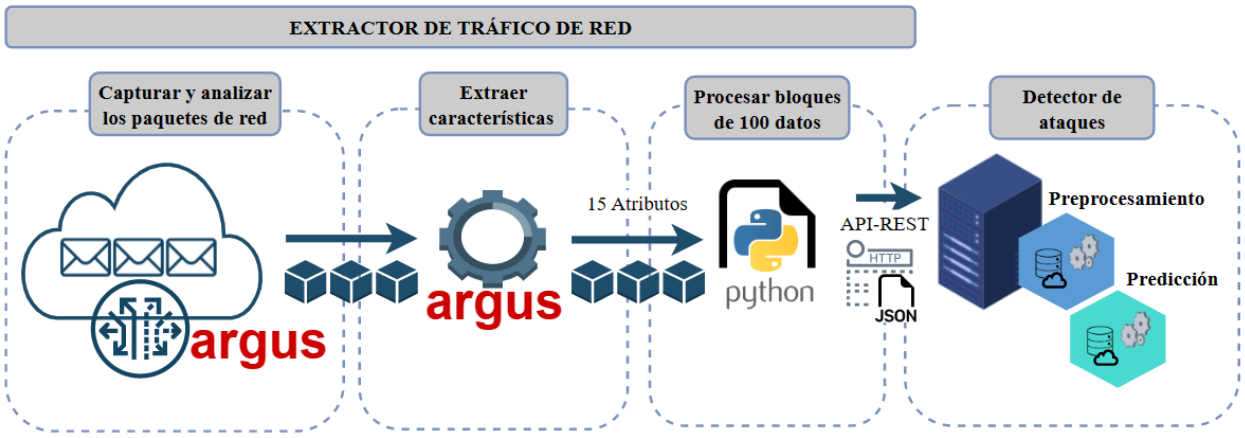

Fig. 2 Estructura del extractor de tráfico de red.

Este script utiliza la librería asyncio para implementar un proceso asíncrono que lee los datos desde la entrada estándar y los procesa en bloques de 100 registros, ya que algunas características de las bases de datos utilizadas se calcularon agrupando los datos en conjunto de 100 registros. Cada bloque de 100 registros se envía a los microservicios de preprocesamiento y predicción para su análisis, guardando el tiempo que tarda el servicio en procesar el bloque con los 100 registros a través de una API. El proceso se repite indefinidamente, leyendo continuamente los datos de entrada y procesándolos hasta que se interrumpe el proceso.

El programa tiene tres funciones principales: get_stream_reader, $send\_api\_request$, y $main$:

La función get_stream_reader se utiliza para crear una conexión de entrada/salida entre dos procesos. Esta función crea un objeto de la clase StreamReader del módulo asyncio para leer los datos del flujo de la entrada estándar y devuelve un apuntador a dicho objeto que es usado posteriormente en la función $main$.

La función $send\_api\_request$ se ejecuta de manera concurrente y recibe como dato de entrada una cola de solicitudes. Se utiliza el módulo aiohttp para realizar solicitudes HTTP a una API de forma asíncrona, por la cual se envían los datos que se encuentran en cola, a través de una solicitud GET para ser procesados por los microservicios de preprocesamiento y predicción.

Además, la función realiza una solicitud POST a otra API para enviar el tiempo que tardaron los microservicios en procesar los datos.

La función $main$ comienza llamando la función get_steam_reader para crear el objeto que permite leer los datos de la entrada estándar correspondientes a los datos obtenidos por la herramienta Argus. Luego se inicializa la cola de solicitudes y se utiliza la función send_api_request como una tarea en segundo plano para procesar dicha cola. Posteriormente, se inicia un bucle infinito que lee los datos desde la entrada estándar y los va almacenando en un dataframe y cuando ha acumulado 100 datos, el dataframe es colocado en la cola de solicitudes.

2) Detector On-line de ataques basado en microservicios

El detector de ataques se basa en una arquitectura de microservicios, que se muestra en la Fig. 3. Los microservicios fueron desarrollados usando *Flask*, un *framework* ligero y flexible de Python que se utiliza comúnmente para desarrollar microservicios. Esta tecnología fue seleccionada por su capacidad para crear aplicaciones de manera rápida y eficiente. Para lograr una comunicación con los microservicios y su integración en la arquitectura general del sistema, se han desarrollado API's REST, las cuales son ampliamente utilizadas para construir servicios web escalables y fáciles de mantener. La comunicación entre los microservicios se realiza a través de Kafka, un broker de mensajería asincrónica que garantiza un procesamiento rápido y eficiente de los mensajes en el backend.

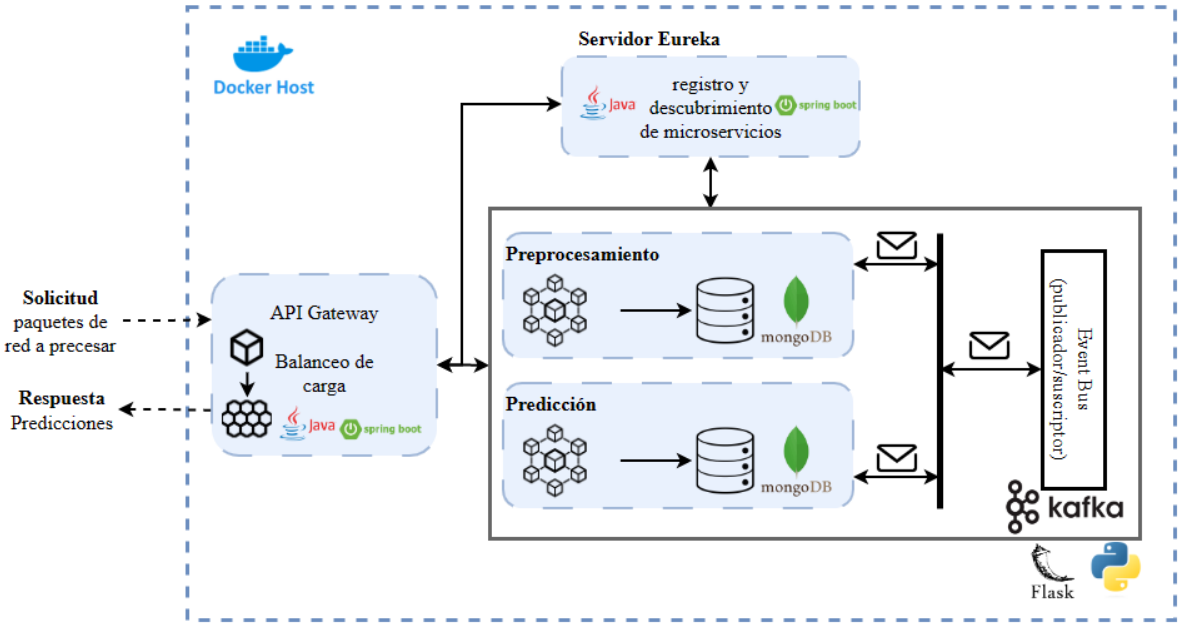

Fig. 3 Arquitectura del detector de ataques basada en microservicios

Cada microservicio se conecta a una base de datos no relacional MongoDB, lo que permite un almacenamiento de datos eficiente y escalable. Además, cada microservicio es puesto en ejecución mediante virtualización en contenedores Docker, lo que permite su fácil despliegue y gestión, además de garantizar un mayor aislamiento y seguridad.

Los microservicios se registran en un servidor de descubrimiento (Eureka), que actúa como un punto de entrada único y centralizado para las APIs. De esta manera, el servidor de descubrimiento puede ofrecer información actualizada sobre los microservicios disponibles, lo que simplifica su gestión y configuración.

Una vez que los microservicios se han registrado en el servidor de descubrimiento, el API Gateway se encarga de enrutar las solicitudes de los clientes al microservicio correspondiente. El API Gateway es responsable de proteger la arquitectura del sistema al actuar como un punto de entrada único para todas las solicitudes externas, y puede aplicar políticas de seguridad, como autenticación y autorización. Además, existe un balanceador de carga que se encarga de distribuir el tráfico entre los diferentes microservicios que proporcionan el mismo servicio. Esto permite que el sistema sea escalable y tolerante a fallos, ya que, si un microservicio falla, el balanceador de carga puede redirigir el tráfico a otros microservicios que proporcionan el mismo servicio.

El proceso de detección de ataques comienza una vez que el extractor de tráfico de red envía los datos que ha recolectado. A continuación, estos datos son procesados por algoritmos de aprendizaje automático para analizar el tráfico y clasificarlo como normal o de ataque. De esta forma, se pueden identificar patrones y comportamientos anómalos en el tráfico, lo que permite detectar posibles amenazas o intrusiones.

El detector de ataques se compone de dos microservicios principales:

a) Microservicio de preprocesamiento

Este microservicio se encarga de la preparación de los datos del tráfico de red antes de ser enviados a los modelos de predicción. Recibe datos en formato JSON en bloques de 100, y aplica una serie de transformaciones para llevar a cabo las acciones de preprocesamiento de los datos, descritas previamente en la sección IV-B. Además, calcula las características adicionales descritas en la TABLA X. Los datos son estandarizados de la misma manera que los datos de entrenamiento en la fase de construcción de los modelos. Una vez que los datos son procesados, se serializan en formato JSON y se envían a través del broker Kafka al microservicio de predicción, donde son publicados en el tópico `data-preprocessing`. Luego, el microservicio de predicción analiza los datos utilizando los modelos de aprendizaje automático para determinar si se trata de una actividad normal o un ataque. La respuesta enviada por el microservicio contiene los resultados de la clasificación y el tiempo de predicción para cada modelo.

Los datos y las predicciones generadas por los diferentes modelos de aprendizaje automático se almacenan en una base de datos MongoDB llamada `preprocesamiento`. En la Fig. 4 se presentan las entidades y los atributos que se desarrollaron en el microservicio de preprocesamiento, donde se puede observar la colección principal llamada `data`, que almacena las características que representan a los paquetes de red, las predicciones de los diferentes modelos, el tiempo de predicción en procesar 100 registros para cada modelo y la etiqueta real de los registros.

Además, para almacenar archivos en MongoDB se utiliza una colección llamada `fs.files` y para almacenar sus chunks se utiliza la colección `fs.chunks`. En la colección `fs.files` se guarda el archivo en sí mismo, mientras que en la colección `fs.chunks` se guardan los chunks o fragmentos de ese archivo. En particular, la colección `fs.files` se utiliza para almacenar el archivo que contiene los parámetros de estandarización necesarios para preprocesar los nuevos datos.

Fig. 4 Diagrama de las entidades en el microservicio de preprocesamiento

A continuación, se detallan los endpoints disponibles en este microservicio:

- Almacenar los parámetros de estandarización:

  POST

  `<ip>:<port>/seguridad/preprocesamiento/save/param/standardization`

  Body: form-data (Files)

  `param: archivo de estandarización (.plk)`

  Respuesta:

  Almacenamiento correcto

  ```
  Status: 200
  Mesanje: Save successfully
  ```

- Preprocesamiento y predicción de 100 registro de datos:

  GET

  ```
  <ip>:<port>/seguridad/preprocesamiento/data/standardization/<param1>
  /<param2>/<param3>
  ```

  Los tres últimos valores la API corresponde a los parámetros utilizados en el momento de las acciones de ataque, en caso de no ser necesario enviar un valor de 0.

  Body: Raw (JSON)

  ```
  {
    "stime":{"0":1667596695.270251,"1":1667596695.414449,…,"99":16675
    96716.776453},
    "proto":{"0":"tcp","1":"tcp",…,"99":"udp"},
    "saddr":{"0":"192.168.100.200","1":"192.168.100.200",…,"99":"192.
    168.100.201"},
    "sport":{"0":43098,"1":54344,…,"99":59093},
    "daddr":{"0":"192.168.100.201","1":"192.168.100.217",…,"99":"200.
    24.19.252"},
    "dport":{"0":22,"1":80,…,"99":53 },
    "pkts":{"0":2,"1":3,…,"99":2},
    "bytes": {"0":288,"1":388,…,"99":244},
    "state": {"0":"CON","1":"CON",…,"99":"CON"},
    "ltime":{"0":1667596695.270511,"1":1667596695.414707,…,"99":16675
    96716.77742},
    "dur":{"0":0.00026,"1":0.000258,…,"99":0.000967},
    "spkts":{"0":1,"1":1,…,"99":1},
    "dpkts":{"0":1,"1":2,…,"99":1},
    "sbytes": {"0":66,"1":66,…,"99":93},
    "dbytes": {"0":222,"1":322,…,"99":151}
  }
  ```

  Respuesta:

  Valores de predicción y tiempo de predicción

  ```
  Status: 200 OK
  Body: JSON
   [
     [1,1,1,…,1,0.002948045],
     [1,1,0,…,0,0.002532482],
  ```

```
    [0,0,1,…,1,0.031775236],
    [1,0,0,…,1,0.003280401]
]
```

Se recibe una lista con las predicciones de los cuatros modelos en el siguiente orden árboles de decisión, regresión logística, bosques aleatorios y SVM. Cada posición de la lista contiene un vector con los resultados de la predicción del modelo correspondiente, donde 1 representa tráfico de ataque y 0 tráfico normal, al final del vector se encuentra el tiempo que tarda el modelo en entregar los resultados.

b) Microservicio de predicción

Este microservicio está diseñado para realizar el proceso de clasificación del tráfico de red utilizando los diferentes modelos de aprendizaje automático que han sido previamente entrenados. Cuenta con una conexión al broker Kafka, que es el medio por donde recibe los datos para realizar las predicciones. Este microservicio está suscrito al tópico `data-preprocessing`, el cual es producido por el microservicio de preprocesamiento. Cada vez que se publican los datos en ese tópico, el microservicio de predicción los convierte en un dataframe y luego los utiliza en los modelos de aprendizaje automático para realizar el análisis correspondiente (predicción). Los modelos responden con las predicciones y también se calcula el tiempo que tarda cada modelo en obtener los resultados de dichas predicciones. Finalmente, se construye una lista con las predicciones de los cuatro modelos y el tiempo que tarda en calcularlas. Esta lista es publicada en el tópico `data-prediction` serializando los datos en formato JSON. El microservicio de preprocesamiento esta suscrito a este tópico, lo que le permite recibir los datos y utilizarlos en su funcionamiento. Además, es importante tener en cuenta que, para utilizar este servicio, es necesario tener instalada la librería scikit-learn y tener acceso a los modelos previamente entrenados.

El microservicio de predicción cuenta con una conexión a una base de datos MongoDB llamada `prediccion`. En la Fig. 5 se pueden observar las entidades y los atributos que se desarrollaron en este microservicio. Se tiene una colección para almacenar el tiempo que tarda desde la captura de los 100 datos hasta obtener las predicciones dadas por los modelos de aprendizaje automático denomina `time_argus`. Además, para almacenar los cuatro modelos de

predicción en MongoDB se utiliza una colección llamada `fs.files` y para almacenar sus chunks se utiliza la colección `fs.chunks`.

Fig. 5 Diagrama de las entidades en el microservicio de predicción

A continuación, se describen los diferentes endpoints desarrollados para este microservicio:

- Almacenar el modelo árboles de decisión

  POST

  `<ip>:<port>/seguridad/prediccion/save/model/dt`

  Body: form-data (Files)

  `model: archivo del modelo de clasificación (.plk)`

  Respuesta:

  Almacenamiento correcto

  ```
  Status: 200
  Mesanje: Update successfully
  ```

- Almacenar el modelo regresión logística

  POST

  `<ip>:<port>/seguridad/prediccion/save/model/lr`

  Body: form-data (Files)

  `model: archivo del modelo de clasificación (.plk)`

  Respuesta:

  Almacenamiento correcto

  ```
  Status: 200
  Mesanje: Update successfully
  ```

- Almacenar el modelo bosques aleatorios

  POST

  `<ip>:<port>/seguridad/prediccion/save/model/rf`

  Body: form-data (Files)

  `model: archivo del modelo de clasificación (.plk)`

  Respuesta:

  Almacenamiento correcto

  ```
  Status: 200
  Mesanje: Update successfully
  ```

- Almacenar el modelo SVM

  POST

  `<ip>:<port>/seguridad/prediccion/save/model/svm-linear`

  Body: form-data (Files)

  `model: archivo del modelo de clasificación (.plk)`

  Respuesta:

  Almacenamiento correcto

  ```
  Status: 200
  Mesanje: Update successfully
  ```

- Almacenar el tiempo que tarda en responder los microservicios al procesar 100 registros.

  POST

  `<ip>:<port>/seguridad/prediccion/save/time/argus/<time>`

  El parámetro `<time>` de la API corresponde al tiempo desde la captura de 100 registros hasta obtener la respuesta con las predicciones de los datos.

  Respuesta:

  Almacenamiento correcto

  ```
  Status: 200
  Body: True
  ```

V. RESULTADOS Y DISCUSIÓN

En esta sección se presentan los resultados y análisis divididos en dos etapas. En la primera etapa, se detallan los resultados obtenidos en la fase de entrenamiento de los modelos de aprendizaje automático. Se muestra la selección de los mejores hiperparámetros, las métricas de rendimiento tanto para los datos de entrenamiento y como para los de prueba y el tiempo de predicción y entrenamiento. En la segunda etapa, se presentan los resultados asociados a la detección de ataques en tiempo real en un escenario controlado. Se calculan las métricas de rendimiento de los cuatro modelos y el tiempo de predicción.

A. Resultados sobre el entrenamiento de los modelos

Durante la fase de entrenamiento de los modelos para la detección de ataques, se realiza una optimización de los hiperparámetros de cada modelo con el fin de obtener el mejor desempeño posible. Este proceso implica ajustar los valores de los parámetros de los modelos y evaluar el impacto de estos cambios en una métrica de evaluación. Como se mencionó previamente, en este caso, la métrica seleccionada fue el accuracy. La TABLA XV muestra los valores óptimos de los hiperparámetros estimados para cada uno de los modelos: árboles de decisión, bosques aleatorios, regresión logística y SVM. En general, todos los modelos presentan un alto rendimiento en su entrenamiento con una exactitud superior al 96%. El árbol de decisión y el bosque aleatorio son los modelos con el mejor desempeño, logrando una exactitud superior al 99%.

TABLA XV
MEJORES HIPERPARÁMETROS PARA CADA MODELO EN LA DETECCIÓN DE ATAQUES

Modelo	**Hiperparámetro**	**Valor**	**Accuracy**
Árboles de Decisión	max_depth	14	0.9998
	min_smaple_split	2	
Bosques Aleatorios	$n\_estimators$	100	0.9999
	max_depth	20	
Regresión Logística	$alpha$	10^{-4}	0.9689
	$penalty$	$l2$	
SVM	$alpha$	10^{-3}	0.9696
	$penalty$	$l2$	

Después de haber seleccionado los mejores hiperparámetros para cada uno de los modelos, se procede a realizar el entrenamiento utilizando los datos de entrenamiento. Durante el entrenamiento, se ajustan los pesos de los modelos para minimizar la función de pérdida y mejorar su capacidad de generalización. Una vez entrenados los modelos, es fundamental evaluar su rendimiento para determinar su efectividad en la detección de ataque. Para ello, se utilizan las métricas: el accuracy, la precisión, el recall, el F1-score y la matriz de confusión. Las métricas de rendimiento se calculan tanto para los datos de entrenamiento como para los datos de prueba. La evaluación del modelo con los datos de entrenamiento es importante para medir su capacidad de ajustarse a los datos usados en la fase de entrenamiento, mientras que la evaluación en los datos de prueba mide la capacidad de generalización del modelo a datos no vistos previamente. A continuación, de la TABLA XVI a la TABLA XIX se muestran las matrices de confusión de los diferentes modelos en ambos conjuntos de datos.

TABLA XVI
MATRICES DE CONFUSIÓN PARA EL MODELO ÁRBOL DE DECISIÓN

Árboles de decisión		**Entrenamiento**		**Prueba**	
		Predicción		**Predicción**	
		Tráfico normal	Tráfico de ataque	Tráfico normal	Tráfico de ataque
Real	Tráfico normal	379565	0	161854	4
	Tráfico de ataque	109	378318	43	162953

TABLA XVII
MATRICES DE CONFUSIÓN PARA EL MODELO BOSQUE ALEATORIO

Bosques aleatorios		**Entrenamiento**		**Prueba**	
		Predicción		**Predicción**	
		Tráfico normal	Tráfico de ataque	Tráfico normal	Tráfico de ataque
Real	Tráfico normal	379565	0	161858	0
	Tráfico de ataque	0	378427	4	162992

TABLA XVIII
MATRICES DE CONFUSIÓN PARA EL MODELO REGRESIÓN LOGÍSTICA

Regresión logística		**Entrenamiento**		**Prueba**	
		Predicción		**Predicción**	
		Tráfico normal	Tráfico de ataque	Tráfico normal	Tráfico de ataque
Real	Tráfico normal	363212	16353	154903	6955
	Tráfico de ataque	7253	371174	3091	159905

TABLA XIX
MATRICES DE CONFUSIÓN PARA EL MODELO SVM

SVM		**Entrenamiento**		**Prueba**	
		Predicción		**Predicción**	
		Tráfico normal	Tráfico de ataque	Tráfico normal	Tráfico de ataque
Real	Tráfico normal	362670	16895	154655	7203
	Tráfico de ataque	6203	372224	2625	160371

En la TABLA XX, se detalla el rendimiento obtenido con los modelos de detección de ataques, tanto para los datos de entrenamiento como para los de prueba. Se pueden observar las métricas de evaluación para cada modelo, lo que permite comparar su efectividad en la detección de ataques.

TABLA XX
MÉTRICAS DE RENDIMIENTO PARA LOS DATOS DE ENTRENAMIENTO Y PRUEBA

Datos de entrenamiento				
Modelo	**Accuracy**	**Precisión**	**Recall**	**F1-score**
Arboles de decisión	0.999856	1	0.999712	0.999856
Bosques Aleatorios	1	1	1	1
Regresión Logística	0.968857	0.957802	0.980874	0.969181
SVM	0.969527	0.956581	0.983608	0.969907
Datos de prueba				
Modelo	**Accuracy**	**Precisión**	**Recall**	**F1-score**
Arboles de decisión	0.999855	0.999975	0.999736	0.999856
Bosques Aleatorios	0.999988	1	0.999975	0.999988
Regresión Logística	0.969075	0.958318	0.981036	0.969033
SVM	0.969746	0.957016	0.983895	0.970270

Además, en la Fig. 6 se detalla el tiempo de entrenamiento medido para cada modelo de aprendizaje automático y en la Fig. 7 se muestra el tiempo que tarda cada uno de los modelos o clasificadores en calcular las predicciones de los datos de prueba. Estos resultados, permiten evaluar la eficiencia computacional de cada modelo.

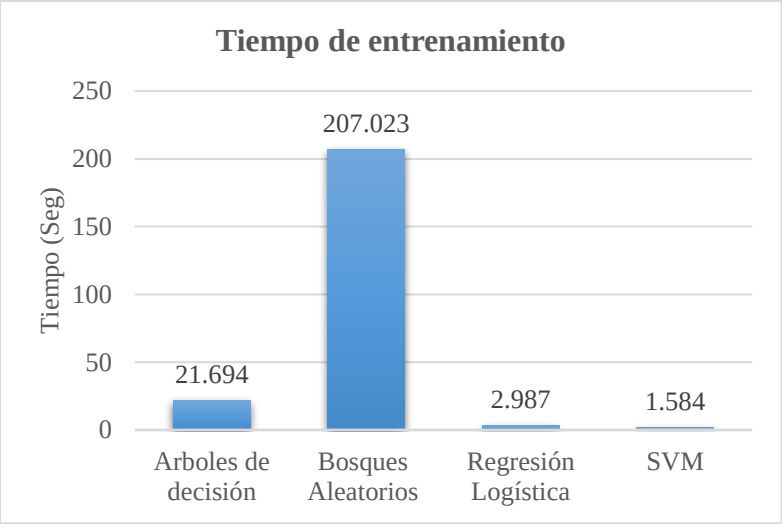

Fig. 6 Tiempo de entrenamiento de cada clasificador

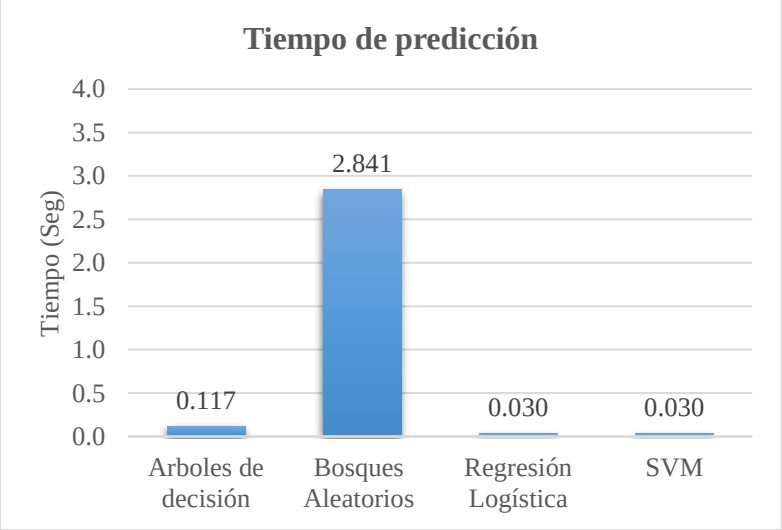

Fig. 7 Tiempo de predicción de los datos de prueba para cada clasificador

De los resultados obtenidos, se puede concluir que los modelos de árboles de decisión y bosques aleatorios muestran una mayor eficacia en la detección de paquetes maliciosos en una red. Sin embargo, el modelo de bosques aleatorios requiere de un tiempo de predicción significativamente mayor en comparación con los otros modelos, lo que sería una desventaja, por ejemplo, para dispositivos de borde que generalmente cuentan con recurso de procesamiento limitados. Por lo tanto, en términos de eficiencia computacional, se concluye que el modelo de árboles de decisión tiene un desempeño adecuado para detectar paquetes maliciosos de manera efectiva y rápida. Este modelo puede ser útil en aplicaciones que requieran detección de ataques en tiempo real, ya que permite procesar los datos de manera eficiente y proporciona buenos resultados en corto tiempo.

B. Resultados sobre la detección de ataque en tiempo real

Para evaluar la capacidad de detección en tiempo real de cada uno de los clasificadores previamente entrenados, se utiliza el escenario de prueba descrito en la sección IV-A, en el cual se simulan actividades maliciosas ejecutadas desde las máquinas atacantes hacia la máquina objetivo o víctima. Se realizó un experimento, durante un periodo de seis días, donde se llevaron a cabo ataques DoS, con variaciones en los parámetros de ejecución de las herramientas de ataque. La TABLA XXI muestra las configuraciones de los parámetros de cada herramienta de ataque y los valores que pueden tomar durante el periodo de la prueba.

TABLA XXI
PARÁMETROS DE LAS HERRAMIENTAS DE ATAQUE

Herramienta de Ataque	**Parámetro**	**Descripción**	**Valor**
Hping3 [55]	`--fast`	Enviar paquetes de red a una velocidad de 10 paquetes por segundo.	-
	`--faster`	Enviar paquetes de red a una velocidad de 100 paquetes por segundo.	-
	`--flood`	Enviar paquetes de red lo más rápido posible, sin tener en cuenta la tasa de transmisión de la red.	-

GoldenEye [56]	`-s`	Número de sockets simultáneos que se crean para realizar conexiones con la máquina objetivo.	10, 50
	`-m`	Método HTTP para usar en las solicitudes.	`random, get, post`
Hulk [57]	-	Se manejan las configuraciones por defecto de la herramienta.	-

Después de llevar a cabo los experimentos y las predicciones, los resultados son almacenados en una base de datos para su posterior análisis y evaluación del desempeño del sistema de detección de ataques en tiempo real. Se obtuvo un total de 3.026.300 registros, los cuales se utilizaron para calcular las métricas de rendimiento de los modelos de aprendizaje automático. Las matrices de confusión obtenidas para cada clasificador se presentan desde la TABLA XXII a la TABLA XXV. Estas matrices permiten evaluar el desempeño de los clasificadores en términos de su capacidad para predecir correctamente el tráfico de ataque y el tráfico normal.

TABLA XXII
MATRIZ DE CONFUSIÓN PARA EL MODELO ÁRBOLES DE DECISIÓN

Árboles de decisión		**Predicción**	
		Tráfico normal	Tráfico de ataque
Real	Tráfico normal	148990	9516
	Tráfico de ataque	1753	2866041

TABLA XXIII
MATRIZ DE CONFUSIÓN PARA EL MODELO BOSQUES ALEATORIOS

Bosques aleatorios		**Predicción**	
		Tráfico normal	Tráfico de ataque
Real	Tráfico normal	131334	27172
	Tráfico de ataque	1055	2866739

TABLA XXIV
MATRIZ DE CONFUSIÓN PARA EL MODELO REGRESIÓN LOGÍSTICA

Regresión logística		**Predicción**	
		Tráfico normal	Tráfico de ataque
Real	Tráfico normal	138135	20371
	Tráfico de ataque	50952	2816842

TABLA XXV
MATRIZ DE CONFUSIÓN PARA EL MODELO SVM

SVM		**Predicción**	
		Tráfico normal	Tráfico de ataque
Real	Tráfico normal	136834	21672
	Tráfico de ataque	11188	2856606

Los resultados del rendimiento, obtenidos de cada uno de los modelos de detección de ataques en tiempo real, se encuentran disponible en la TABLA XXVI. Dicha tabla muestra las métricas de desempeño para cada modelo, como el accuracy, la precisión asociada a cada clase (0 y 1) y su precisión promedio, el recall asociado a cada clase (0 y 1) y su recall promedio, así como el F1-score asociado a cada clase (0 y 1) y su F1-score promedio. La clase 0 representa el tráfico normal, mientras que la clase 1 representa el tráfico con ataque.

TABLA XXVI
MÉTRICAS DE RENDIMIENTO PARA CADA MODELO DE APRENDIZAJE AUTOMÁTICO

Métrica	**Árboles de decisión**	**Bosques aleatorios**	**Regresión Logística**	**SVM**
Accuracy	0.996276	0.990673	0.976432	0.989142
Precision_1	0.996691	0.990611	0.99282	0.99247
Precision_0	0.988371	0.992031	0.730537	0.924417
Precision_avg	0.992531	0.991321	0.861678	0.958444
Recall_1	0.999389	0.999632	0.982233	0.996099
Recall_0	0.939964	0.828574	0.871481	0.863273
Recall_avg	0.969677	0.914103	0.926857	0.929686
F1-score_1	0.998038	0.995101	0.987498	0.994281
F1-score_0	0.96356	0.902965	0.794809	0.892799
F1-score_avg	0.980799	0.949033	0.891154	0.94354

Adicionalmente, la Fig. 8 muestra el tiempo promedio que cada modelo requiere para calcular las predicciones de 100 registros. Se observa que el modelo de bosques aleatorios requiere una mayor capacidad de procesamiento, debido a que tarda más tiempo en procesar los 100 registros en comparación con los otros modelos. Esto se debe a la estructura del modelo, que utiliza múltiples árboles de decisión para realizar la predicción, lo que implica una mayor complejidad computacional.

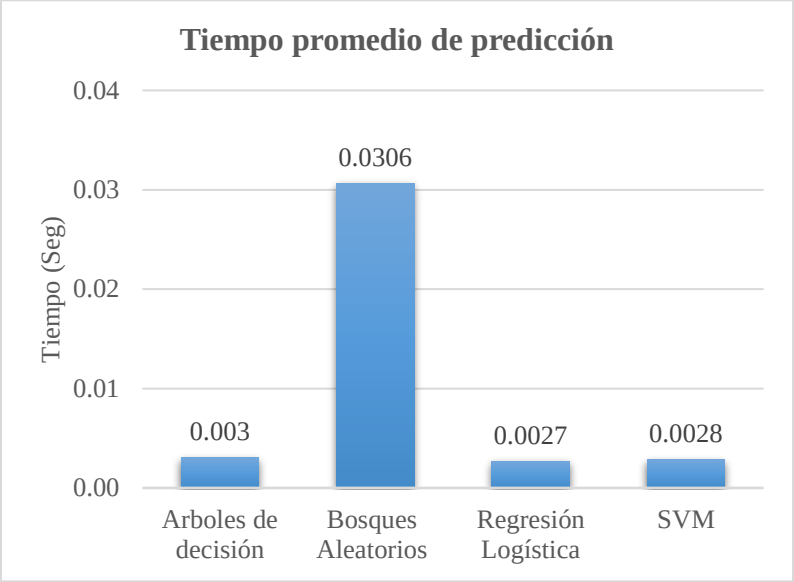

Fig. 8 Tiempo promedio en calcular las predicciones de 100 registros

Los resultados obtenidos muestran que la detección de ataques en tiempo real se puede realizar de manera efectiva, ya que presenta un alto rendimiento en la identificación de tráfico malicioso en cortos periodos de tiempo y ambiente reales de ejecución. Los modelos de árboles de decisión y bosques aleatorios tienen un rendimiento superior en comparación con los modelos de regresión logística y SVM. La métrica F1-score promedio es una medida combinada entre el recall y la precisión, la cual da una mejor idea del comportamiento del modelo. En particular, el modelo de árboles de decisión muestra el mejor desempeño en términos de F1-score promedio con un rendimiento de 98,08%.

Además, para estimar el tiempo de respuesta de la predicción de los paquetes de red, se toma como tiempo inicial el momento, que se inicia desde la captura de 100 registros y hasta el momento en que se recibe la respuesta por parte de los microservicios de detección de ataques. Se obtuvo un tiempo promedio (usando la infraestructura descrita en un apartado anterior) de 0,82 segundos y un tiempo mínimo de 0,37 segundos.

Es de anotar que cuando el flujo de paquetes en la red disminuye, el tiempo de respuesta se ve afectado por la espera de los 100 paquetes que se deben analizar, sin embargo, un flujo de datos pequeños también puede implicar un tráfico normal de la red.

En resumen, los resultados obtenidos demuestran que el sistema de detección de ataques basado en técnicas de aprendizaje automático es una estrategia efectiva para la detección de tráfico malicioso en tiempo real en un entorno de red. La combinación de estas técnicas con la arquitectura de microservicios contribuye a un procesamiento rápido y eficiente de los datos. Este sistema puede ser una herramienta muy útil para mejorar la seguridad en línea de plataformas IoT, ya que permite la detección temprana de amenazas y una respuesta rápida y eficiente ante ellas.

En comparación con otros trabajos encontrados en el estado del arte, la mayoría de ellos obtiene un rendimiento del 99%, sin embargo, estos modelos solo trabajan fuera de línea. El sistema de detección de ataques basado en microservicios desarrollado tiene un rendimiento fuera de línea similar a los trabajos referenciados, con la ventaja de poder proporcionar una detección en línea. También, permite monitorear distintos puntos en la red utilizando pocos recursos computacionales ya solo es necesario desplegar extractores de tráfico de red en los puntos deseados.

La estrategia implementada es un complemento a la seguridad de la plataforma IoT, que se desplegará en el proyecto en el que se enmarca este trabajo, ya que permite tener un sistema de defensa que detecta posibles ataques en tiempo real brindando la posibilidad de tomar acciones correctivas en poco tiempo.

VI. CONCLUSIONES

Este trabajo ha logrado alcanzar los objetivos planteados, en cuanto al desarrollo de una estrategia para gestionar la seguridad en una plataforma IoT para ciudades inteligentes basada en microservicios. A través de una revisión de la literatura, se han identificado las vulnerabilidades relevantes en el contexto de estas plataformas, las cuales representan riesgos potenciales para la seguridad del sistema. Entre estas vulnerabilidades se encuentran el control acceso deficientes, la falta de cifrado, las configuraciones de seguridad inadecuadas, el uso de componentes obsoletos y vulnerables, la seguridad física deficiente y el registro y monitoreo insuficientes.

Adicionalmente, se ha realizado un análisis de los diferentes mecanismos de seguridad reportados en la literatura para solucionar vulnerabilidades identificadas. Se ha encontrado que para abordar la vulnerabilidad asociada al control de acceso en una plataforma IoT para ciudades inteligentes es necesario contar con mecanismos de autorización y autenticación eficientes. Sin embargo, también es fundamental contar con mecanismos de seguridad efectivos que puedan detectar y contrarrestar las acciones de los atacantes, uno de estos mecanismos es la detección de intrusos.

De esta manera, se ha desarrollado una estrategia de seguridad que consiste en el diseño e implementación de un IDS, capaz de analizar el tráfico de red en tiempo real para detectar ataques en un entorno controlado. El IDS utiliza modelos de aprendizaje automático para la detección de actividades maliciosas en el sistema IoT. Se implementaron cuatros modelos de aprendizaje automático: arboles de decisión, bosques aleatorios, regresión logística y SVM, los cuales fueron entrenados con bases de datos (BoT-IoT, UNSW-NB15 y una base de datos propia) que contienen patrones de ataques conocidos.

La eficacia del sistema en la detección en tiempo real ha sido evaluada utilizando métricas de rendimiento asociadas a los modelos de predicción, como el accuracy, la precisión, el recall y el F1-score. El modelo de árboles de decisión ha demostrado un alto porcentaje de eficacia, alcanzando valores del 98,08% para la métrica de F1-score, 99,25% para la precisión y 96,96% para el recall. Esto indica su capacidad para identificar tráfico malicioso en un entorno realista,

brindando una mayor seguridad a la red y aumentando la confianza en el sistema de detección de ataques.

En contraste con otras investigaciones encontradas en la revisión del estado del arte sobre la detección de intruso, la mayoría de ellas logra un rendimiento del 99%. Sin embargo, es relevante señalar que estos modelos operan únicamente en modo offline. En comparación, el sistema de detección de ataques basado en microservicios que se ha desarrollado tiene un rendimiento similar en el ámbito offline con los estudios analizados. La singularidad del sistema radica en su capacidad para brindar detección en tiempo real, ofreciendo la posibilidad de tomar acciones correctivas rápidamente.

Además, se ha medido el tiempo de predicción de un ataque en la red, el cual ha sido de 0,36 segundos desde la captura de 100 datos hasta la recepción de la predicción. Esta rápida detección permite una respuesta temprana ante posibles atacantes, lo que a su vez facilita la adopción de medidas oportunas para proteger la red y prevenir daños. En conjunto, estos resultados demuestran la efectividad y la eficiencia del IDS implementado en la detección y mitigación de ataques en un entorno IoT.

Adicionalmente, el sistema de detección de ataques puede ser implementado en cualquier punto de la infraestructura. Esto se logra gracias a que el detector de ataques ha sido desarrollado siguiendo una arquitectura basada en microservicios. Solo se requiere desplegar el extractor de tráfico en el área de interés y transmitir la información a los microservicios del detector de ataques. Este enfoque incrementa la cobertura de seguridad y fortalece la protección de la plataforma IoT para ciudades inteligentes frente a posibles amenazas.

Como trabajos futuros, se pueden considerar las siguientes ideas:

Integración de un modelo multiclase: Para mejorar la capacidad de protección del sistema contra intrusiones, se puede integrar un modelo multiclase. Este modelo no solo detectará si ha ocurrido un ataque, sino que también sea capaz de identificar y especificar el tipo de ataque. Al

proporcionar información más detallada sobre las amenazas detectadas, el sistema estará mejor preparado para defenderse contra diversos tipos de intrusiones.

Implementación en tiempo real de estrategias de mitigación: Para fortalecer la robustez y seguridad del sistema, es importante implementar estrategias de mitigación en tiempo real. Esto significa que el sistema no solo debe detectar y notificar sobre los ataques, sino también tomar acciones inmediatas y automatizadas para contrarrestarlos. La implementación en tiempo real de estrategias de mitigación permitirá que el sistema responda de manera rápida y efectiva para reducir el impacto de los ataques.

Exploración de otros modelos de aprendizaje automático: Es recomendable explorar y probar modelos de aprendizaje profundo (deep learning). Los modelos de aprendizaje profundo, como las redes neuronales profundas, han demostrado tener un gran potencial en diversas áreas de aplicación, incluida la detección de ataques y anomalías en sistemas de seguridad. Esta exploración permitirá evaluar el rendimiento y la eficacia de los modelos de aprendizaje profundo en comparación con los enfoques tradicionales, brindando así una visión más completa de las capacidades y limitaciones de diferentes enfoques de detección de ataques.

REFERENCIAS

[1] M. Krämer, S. Frese, and A. Kuijper, "Implementing secure applications in smart city clouds using microservices," *Future Generation Computer Systems*, vol. 99, pp. 308–320, Oct. 2019, doi: 10.1016/J.FUTURE.2019.04.042.

[2] Y. A. Prasetyo and Suhardi, "Microservice Platform for Smart City: Concepts, Services and Technology," *2018 International Conference on Information Technology Systems and Innovation, ICITSI 2018 - Proceedings*, pp. 358–363, Jul. 2018, doi: 10.1109/ICITSI.2018.8695927.

[3] T. Panagiotakopoulos, D. P. Vlachos, T. V. Bakalakos, A. Kanavos, and A. Kameas, "A FIWARE-based IoT Framework for Smart Water Distribution Management," *IISA 2021 - 12th International Conference on Information, Intelligence, Systems and Applications*, Jul. 2021, doi: 10.1109/IISA52424.2021.9555509.

[4] C. Badii, P. Bellini, A. Difino, P. Nesi, G. Pantaleo, and M. Paolucci, "MicroServices Suite for Smart City Applications," *Sensors 2019, Vol. 19, Page 4798*, vol. 19, no. 21, pp. 1–32, Nov. 2019, doi: 10.3390/S19214798.

[5] N. Mitolo, P. Nesi, G. Pantaleo, and M. Paolucci, "Snap4City Platform to Speed Up Policies," *Green Energy and Technology*, pp. 103–114, 2021, doi: 10.1007/978-3-030-57764-3_7.

[6] A. Nebel, "Arquitectura de microservicios para plataformas de integración," Universidad de la República (Uruguay)., Uruguay, 2019. Accessed: Oct. 30, 2021. [Online]. Available: https://hdl.handle.net/20.500.12008/20586

[7] Chris. Richardson, *Microservices patterns: with examples in Java*. Manning Publications, 2018. Accessed: Oct. 24, 2022. [Online]. Available: https://learning.oreilly.com/library/view/microservices-patterns/9781617294549/

[8] M. Driss, D. Hasan, W. Boulila, and J. Ahmad, "Microservices in IoT Security: Current Solutions, Research Challenges, and Future Directions," *Procedia Comput Sci*, vol. 192, pp. 2385–2395, Jan. 2021, doi: 10.1016/J.PROCS.2021.09.007.

[9] S. Sultan, I. Ahmad, and T. Dimitriou, "Container security: Issues, challenges, and the road ahead," *IEEE Access*, vol. 7, pp. 52976–52996, 2019, doi: 10.1109/ACCESS.2019.2911732.

[10] W. S. Shameem Ahamed, P. Zavarsky, and B. Swar, "Security Audit of Docker Container Images in Cloud Architecture," *ICSCCC 2021 - International Conference on Secure Cyber Computing and Communications*, pp. 202–207, May 2021, doi: 10.1109/ICSCCC51823.2021.9478100.

[11] A. K. Chitturi and P. Swarnalatha, "Exploration of Various Cloud Security Challenges and Threats," *Advances in Intelligent Systems and Computing*, vol. 1057, pp. 891–899, 2020, doi: 10.1007/978-981-15-0184-5_76.

[12] M. Tavana, V. Hajipour, and S. Oveisi, "IoT-based enterprise resource planning: Challenges, open issues, applications, architecture, and future research directions," *Internet of Things (Netherlands)*, vol. 11, Sep. 2020, doi: 10.1016/J.IOT.2020.100262.

[13] C. Badii, P. Bellini, A. Difino, and P. Nesi, "Smart city IoT platform respecting GDPR privacy and security aspects," *IEEE Access*, vol. 8, pp. 23601–23623, 2020, doi: 10.1109/ACCESS.2020.2968741.

[14] C. Lai, F. Boi, A. Buschettu, and R. Caboni, "IoT and microservice architecture for multimobility in a smart city," *Proceedings - 2019 International Conference on Future Internet of Things and Cloud, FiCloud 2019*, pp. 238–242, Aug. 2019, doi: 10.1109/FICLOUD.2019.00040.

[15] D. Salikhov, K. Khanda, K. Gusmanov, M. Mazzara, and N. Mavridis, "Microservice-based IoT for smart buildings," *Proceedings - 31st IEEE International Conference on Advanced Information Networking and Applications Workshops, WAINA 2017*, pp. 302–308, May 2017, doi: 10.1109/WAINA.2017.77.

[16] L. G. García Morales, J. A. Vergara Tejada, R. A. Velásquez Vélez, and J. E. Aedo Cobo, "Selecting a Data Platform to Support the Development of Applications Oriented to Improve Citizen Security," *Congreso Internacional de Ingeniería - IC EXPOI 2022*, pp. 533–540, Oct. 2022.

[17] "Snap4City: Smart aNalytic APp builder for sentient Cities and IOT | Snap4City." https://www.snap4city.org/drupal/node/1 (accessed May 14, 2023).

[18] C. Badii *et al.*, "Real-Time Automatic Air Pollution Services from IOT Data Network," *Proc IEEE Symp Comput Commun*, vol. 2020-July, Jul. 2020, doi: 10.1109/ISCC50000.2020.9219580.

[19] P. Bellini, S. Bilotta, A. L. I. Palesi, P. Nesi, and G. Pantaleo, "Vehicular Traffic Flow Reconstruction Analysis to Mitigate Scenarios with Large City Changes," *IEEE Access*, vol. 10, pp. 131061–131075, 2022, doi: 10.1109/ACCESS.2022.3229183.

[20] D. Lu, D. Huang, A. Walenstein, and D. Medhi, "A Secure Microservice Framework for IoT," *Proceedings - 11th IEEE International Symposium on Service-Oriented System Engineering, SOSE 2017*, pp. 9–18, Jun. 2017, doi: 10.1109/SOSE.2017.27.

[21] C. Vorakulpipat, E. Rattanalerdnusorn, P. Thaenkaew, and H. Dang Hai, "Recent challenges, trends, and concerns related to IoT security: An evolutionary study," *International Conference on Advanced Communication Technology, ICACT*, vol. 2018-February, pp. 405–410, Mar. 2018, doi: 10.23919/ICACT.2018.8323774.

[22] N. Neshenko, E. Bou-Harb, J. Crichigno, G. Kaddoum, and N. Ghani, "Demystifying IoT Security: An Exhaustive Survey on IoT Vulnerabilities and a First Empirical Look on Internet-Scale IoT Exploitations," *IEEE Communications Surveys and Tutorials*, vol. 21, no. 3, pp. 2702–2733, 2019, doi: 10.1109/COMST.2019.2910750.

[23] O. Diaz, M. Munoz, and J. Mejia, "Responsive infrastructure with cybersecurity for automated high availability DevSecOps processes," *2019 8th International Conference on Software Process Improvement, CIMPS 2019 - Applications in Software Engineering*, Oct. 2019, doi: 10.1109/CIMPS49236.2019.9082439.

[24] K. Kimani, V. Oduol, and K. Langat, "Cyber security challenges for IoT-based smart grid networks," *International Journal of Critical Infrastructure Protection*, vol. 25, pp. 36–49, Jun. 2019, doi: 10.1016/J.IJCIP.2019.01.001.

[25] D. E. Kouicem, A. Bouabdallah, and H. Lakhlef, "Internet of things security: A top-down survey," *Computer Networks*, vol. 141, pp. 199–221, Aug. 2018, doi: 10.1016/J.COMNET.2018.03.012.

[26] I. Stellios, P. Kotzanikolaou, M. Psarakis, C. Alcaraz, and J. Lopez, "A survey of iot-enabled cyberattacks: Assessing attack paths to critical infrastructures and services," *IEEE Communications Surveys and Tutorials*, vol. 20, no. 4, pp. 3453–3495, Oct. 2018, doi: 10.1109/COMST.2018.2855563.

[27] C. Faircloth, G. Hartzell, N. Callahan, and S. Bhunia, "A Study on Brute Force Attack on T-Mobile Leading to SIM-Hijacking and Identity-Theft," *2022 IEEE World AI IoT Congress, AIIoT 2022*, pp. 501–507, 2022, doi: 10.1109/AIIOT54504.2022.9817175.

[28] M. Zeeshan *et al.*, "Protocol-Based Deep Intrusion Detection for DoS and DDoS Attacks Using UNSW-NB15 and Bot-IoT Data-Sets," *IEEE Access*, vol. 10, pp. 2269–2283, 2022, doi: 10.1109/ACCESS.2021.3137201.

[29] N. Koroniotis, N. Moustafa, E. Sitnikova, and B. Turnbull, "Towards the Development of Realistic Botnet Dataset in the Internet of Things for Network Forensic Analytics: Bot-IoT Dataset," *Future Generation Computer Systems*, vol. 100, pp. 779–796, Nov. 2018, doi: 10.48550/arxiv.1811.00701.

[30] P. Rojas, S. Alahmadi, and M. Bayoumi, "Physical Layer Security for IoT Communications - A Survey," *7th IEEE World Forum on Internet of Things, WF-IoT 2021*, pp. 95–100, Jun. 2021, doi: 10.1109/WF-IOT51360.2021.9595025.

[31] "OWASP Top Ten Web Application Security Risks | OWASP." https://owasp.org/www-project-top-ten/ (accessed Apr. 15, 2022).

[32] M. O. Pahl and L. Donini, "Securing IoT microservices with certificates," *IEEE/IFIP Network Operations and Management Symposium: Cognitive Management in a Cyber World, NOMS 2018*, pp. 1–5, Jul. 2018, doi: 10.1109/NOMS.2018.8406189.

[33] A. Ghosh, A. Mukherjee, and S. Misra, "SEGA: Secured Edge Gateway Microservices Architecture for IIoT-Based Machine Monitoring," *IEEE Trans Industr Inform*, vol. 18, no. 3, pp. 1949–1956, Mar. 2022, doi: 10.1109/TII.2021.3102158.

[34] J. Yang, H. Hou, H. Li, and Q. Zhu, "User Fast Authentication Method Based on Microservices," *Proceedings of 2021 IEEE International Conference on Power Electronics, Computer Applications, ICPECA 2021*, pp. 93–98, Jan. 2021, doi: 10.1109/ICPECA51329.2021.9362656.

[35] R. Swapnoneel, M. Sam, and M. Debajyoti, "On Application of Blockchain to Enhance Single Sign-On (SSO) Systems | IEEE Conference Publication | IEEE Xplore," *2021 IEEE 20th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pp. 1191–1195, 2021, doi: 10.1109/TrustCom53373.2021.00161.

[36] Sussi, R. M. Negara, and Z. Triartono, "Implementation of role-based access control on OAuth 2.0 as authentication and authorization system," *International Conference on Electrical Engineering, Computer Science and Informatics (EECSI)*, pp. 259–263, Sep. 2019, doi: 10.23919/EECSI48112.2019.8977061.

[37] A. Banati, E. Kail, K. Karoczkai, and M. Kozlovszky, "Authentication and authorization orchestrator for microservice-based software architectures," *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics, MIPRO 2018 - Proceedings*, pp. 1180–1184, Jun. 2018, doi: 10.23919/MIPRO.2018.8400214.

[38] R. Xu, S. Y. Nikouei, Y. Chen, E. Blasch, and A. Aved, "BlendMAS: A blockchain-enabled decentralized microservices architecture for smart public safety," *Proceedings - 2019 2nd IEEE International Conference on Blockchain, Blockchain 2019*, pp. 564–571, Jul. 2019, doi: 10.1109/BLOCKCHAIN.2019.00082.

[39] S. Anwar *et al.*, "From Intrusion Detection to an Intrusion Response System: Fundamentals, Requirements, and Future Directions," *Algorithms 2017, Vol. 10, Page 39*, vol. 10, no. 2, p. 39, Mar. 2017, doi: 10.3390/A10020039.

[40] R. Doshi, N. Apthorpe, and N. Feamster, "Machine learning DDoS detection for consumer internet of things devices," *Proceedings - 2018 IEEE Symposium on Security and Privacy Workshops, SPW 2018*, pp. 29–35, Aug. 2018, doi: 10.1109/SPW.2018.00013.

[41] Bharadwaj, K. B. Prakash, and G. R. Kanagachidambaresan, "Pattern Recognition and Machine Learning," *EAI/Springer Innovations in Communication and Computing*, pp. 105–144, 2021, doi: 10.1007/978-3-030-57077-4_11/FIGURES/51.

[42] T. M. Khoshgoftaar, M. Golawala, and J. Van Hulse, "An empirical study of learning from imbalanced data using random forest," *Proceedings - International Conference on Tools with Artificial Intelligence, ICTAI*, vol. 2, pp. 310–317, 2007, doi: 10.1109/ICTAI.2007.46.

[43] J. L. Leevy, J. Hancock, T. M. Khoshgoftaar, and J. Peterson, "Detecting Information Theft Attacks in the Bot-IoT Dataset," *Proceedings - 20th IEEE International Conference on Machine Learning and Applications, ICMLA 2021*, pp. 807–812, 2021, doi: 10.1109/ICMLA52953.2021.00133.

[44] A. V. Dorogush, V. Ershov, and A. G. Yandex, "CatBoost: gradient boosting with categorical features support," Oct. 2018, doi: 10.48550/arxiv.1810.11363.

[45] G. Ke *et al.*, "LightGBM: A Highly Efficient Gradient Boosting Decision Tree," *Adv Neural Inf Process Syst*, vol. 30, 2017, Accessed: Feb. 07, 2023. [Online]. Available: https://github.com/Microsoft/LightGBM.

[46] T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, vol. 13-17-August-2016, pp. 785–794, Mar. 2016, doi: 10.1145/2939672.2939785.

[47] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, 2nd ed. in Springer Series in Statistics. New York, NY: Springer New York, 2009. doi: 10.1007/978-0-387-84858-7.

[48] J. Rynkiewicz, "Asymptotic statistics for multilayer perceptron with ReLU hidden units," *Neurocomputing*, vol. 342, pp. 16–23, May 2019, doi: 10.1016/J.NEUCOM.2018.11.097.

[49] N. Moustafa and J. Slay, "UNSW-NB15: A comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)," *2015 Military Communications and Information Systems Conference, MilCIS 2015 - Proceedings*, Dec. 2015, doi: 10.1109/MILCIS.2015.7348942.

[50] Z. Liu, N. Thapa, A. Shaver, K. Roy, X. Yuan, and S. Khorsandroo, "Anomaly detection on Iot network intrusion using machine learning," *2020 International Conference on Artificial Intelligence, Big Data, Computing and Data Communication Systems, icABCD 2020 - Proceedings*, Aug. 2020, doi: 10.1109/ICABCD49160.2020.9183842.

[51] Hyunjae Kang, Dong Hyun Ahn, Gyung Min Lee, Jeong Do Yoo, Kyung Ho Park, and Huy Kang Kim, "IoT network intrusion dataset | IEEE DataPort," Sep. 2019. https://ieee-dataport.org/open-access/iot-network-intrusion-dataset (accessed Mar. 17, 2023).

[52] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, "Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection," Feb. 2018, doi: 10.48550/arxiv.1802.09089.

[53] S. I. Popoola, B. Adebisi, M. Hammoudeh, G. Gui, and H. Gacanin, "Hybrid Deep Learning for Botnet Attack Detection in the Internet-of-Things Networks," *IEEE Internet Things J*, vol. 8, no. 6, pp. 4944–4956, Mar. 2021, doi: 10.1109/JIOT.2020.3034156.

[54] "openargus - Home." https://openargus.org/ (accessed Jan. 17, 2023).

[55] "hping3 | Kali Linux Tools." https://www.kali.org/tools/hping3/#tool-documentation (accessed Feb. 07, 2023).

[56] "GitHub - jseidl/GoldenEye: GoldenEye Layer 7 (KeepAlive+NoCache) DoS Test Tool." https://github.com/jseidl/GoldenEye (accessed Feb. 07, 2023).

[57] "GitHub - grafov/hulk: HULK DoS tool ported to Go with some additional features." https://github.com/grafov/hulk (accessed Feb. 07, 2023).

[58] "The Bot-IoT Dataset | UNSW Research." https://research.unsw.edu.au/projects/bot-iot-dataset (accessed May 11, 2023).

[59] "Tshark | tshark.dev." https://tshark.dev/ (accessed May 18, 2023).

[60] "Ostinato Traffic Generator for Network Engineers." https://ostinato.org/ (accessed May 18, 2023).

[61] J. M. Peterson, J. L. Leevy, and T. M. Khoshgoftaar, "A Review and Analysis of the Bot-IoT Dataset," *Proceedings - 15th IEEE International Conference on Service-Oriented System Engineering, SOSE 2021*, pp. 20–27, Aug. 2021, doi: 10.1109/SOSE52839.2021.00007.

[62] "The UNSW-NB15 Dataset | UNSW Research." https://research.unsw.edu.au/projects/unsw-nb15-dataset (accessed May 11, 2023).

[63] "scikit-learn: machine learning in Python — scikit-learn 1.1.2 documentation." https://scikit-learn.org/stable/ (accessed Aug. 07, 2022).

[64] "sklearn.ensemble.RandomForestClassifier — scikit-learn 1.1.2 documentation." https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html#sklearn.ensemble.RandomForestClassifier (accessed Aug. 07, 2022).

[65] "sklearn.tree.DecisionTreeClassifier — documentación de scikit-learn 1.1.2." https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html (accessed Aug. 07, 2022).

[66] "sklearn.linear_model.SGDClassifier — scikit-learn 1.1.2 documentation." https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html#sklearn.linear_model.SGDClassifier (accessed Aug. 07, 2022).