ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA Y SISTEMAS DE TELECOMUNICACIÓN

PROYECTO FIN DE GRADO

TÍTULO: Sistema automatizado de detección de ataques en redes IoT

AUTOR: Adrián Camargo Belmar

TITULACIÓN: Grado en Ingeniería Telemática

TUTOR: Pedro Castillejo Parrilla

DEPARTAMENTO: Ingeniería Telemática y Electrónica

VºBº

Miembros del Tribunal Calificador:

PRESIDENTE: Juana Sendra Pons

TUTOR: Pedro Castillejo Parrilla

SECRETARIO: Esther Gago García

Fecha de lectura: 2 de julio de 2021

Calificación:

El Secretario,

"A todas las personas que encontré en el camino

y pusieron su granito de arena para que todo esto sea posible"

Resumen

Sistema automatizado para la detección de ataques en redes IoT

El Internet de las Cosas o IoT (*Internet of Things*) hace referencia a la interconexión de objetos cotidianos a través de Internet. Estos objetos pueden ser desde relojes inteligentes hasta sensores distribuidos en una ciudad o en un hogar. Los dispositivos conectados a la red transmiten y reciben información de forma constante y, en numerosas ocasiones, los datos manejados son de naturaleza sensible. Por este motivo, la seguridad es un requisito imprescindible en las comunicaciones.

Los dispositivos IoT se caracterizan por sus limitadas capacidades: memoria, batería, capacidad de cálculo, etc. Debido a ello, los sistemas desarrollados sobre plataformas IoT deben ser lo más eficientes posibles. El desarrollo de aplicaciones basadas en mecanismos de aprendizaje automático requiere, por regla general, grandes capacidades para su ejecución. Por tanto, la sinergia entre el aprendizaje automático y las redes de dispositivos IoT es todo un reto.

Los sistemas de seguridad para plataformas IoT pueden ser implementados a nivel de red o a nivel de nodo. El propósito de este Proyecto de Fin de Grado es llevar a cabo el diseño y la implementación de un sistema de detección de ataques basado en mecanismos de aprendizaje automático. Por ello, el proyecto aborda la implementación de la solución de seguridad a nivel de red y trata de demostrar la viabilidad de dichos mecanismos sobre plataformas IoT.

La inteligencia artificial o, concretamente, el aprendizaje automático trata de generar un modelo a través del aprendizaje. Dicho aprendizaje se produce mediante el ajuste de los parámetros del modelo en un proceso iterativo. Existen diferentes paradigmas de aprendizaje: aprendizaje supervisado, aprendizaje no supervisado y aprendizaje por refuerzo. Debido a la naturaleza del problema de seguridad a resolver, en el proyecto se implementa un mecanismo de aprendizaje automático supervisado que aprende del funcionamiento normal de la red para analizar los paquetes que en ella se intercambian y determinar si se está produciendo o no un ataque.

Para lograr el objetivo principal del proyecto fue un requisito imprescindible el estudio del marco tecnológico y de los antecedentes. Se analizaron las diferentes plataformas IoT con el fin de familiarizarse con las limitaciones de los dispositivos, los posibles ataques que pueden ocurrir en las redes de dispositivos IoT y soluciones a ellos basadas en mecanismos de aprendizaje automático. Además, se estudiaron distintos entornos de simulación para el desarrollo del proyecto.

Una vez obtenida la información necesaria, se diseñó la solución de seguridad y se llevó a cabo su implementación. Se integró el sistema automatizado de detección de ataques con un sistema de simulación de redes IoT y se realizaron las pruebas correspondientes demostrando la viabilidad del uso de mecanismos de aprendizaje automático sobre plataformas IoT simuladas con un rendimiento satisfactorio.

Como conclusión, durante el Proyecto de Fin de Grado se ha desarrollado un sistema automatizado de detección de ataques en redes de dispositivos IoT integrado en un entorno de simulación de redes. La solución de seguridad es capaz de aprender del comportamiento normal de la red y detectar si se está produciendo un ataque en la red independientemente de la información que es transmitida e independientemente de la topología de la red en la que dicha información viaja.

Abstract

Automated attack detection system in IoT networks

The Internet of Things or IoT refers to the interconnection of everyday objects through the Internet. These objects can be from smart watches to sensors distributed in a city or in a home. Devices connected to the network are constantly transmitting and receiving information and, in many cases, the data handled is of a sensitive nature. For this reason, security is an essential requirement in communications.

IoT devices are characterized by their limited capacities: memory, battery, computing power, etc. Due to this, systems developed on IoT platforms must be as efficient as possible. The development of applications based on machine learning mechanisms requires, as a general rule, large capacities for their execution. Therefore, the synergy between machine learning and IoT device networks is a challenge.

Security systems for IoT platforms can be implemented at the network level or at the node level. The purpose of this Final Degree Project is to carry out the design and implementation of an attack detection system based on machine learning mechanisms. For this reason, the project approaches the implementation of the security solution at the network level and tries to demonstrate the viability of these mechanisms on IoT platforms.

Artificial intelligence or, specifically, machine learning tries to generate a model through learning. This learning occurs by adjusting the parameters of the model in an iterative process. There are different learning paradigms: supervised learning, unsupervised learning, and reinforcement learning. Due to the nature of the security problem to be solved, the project implements a supervised machine learning mechanism that learns from the normal operation of the network to analyze the packets that are exchanged there and determine whether or not an attack is taking place.

To achieve the main objective of the project, a study of the technological framework and background was an essential requirement. Different IoT platforms were analyzed in order to become familiar with the limitations of the devices, the possible attacks that can occur in the networks of IoT devices and solutions to them based on machine learning mechanisms. In addition, different simulation environments were studied for the development of the project.

Once the necessary information was obtained, the security solution was designed and implemented. The automated attack detection system was integrated with an IoT network simulation system and the corresponding tests were carried out demonstrating the viability of using machine learning mechanisms on simulated IoT platforms with satisfactory performance.

In conclusion, during the Final Degree Project, an automated system for detecting attacks in networks of IoT devices integrated in a network simulation environment has been developed. The security solution is able to learn from the normal behavior of the network and detecting if an attack is taking place on the network regardless of the information that is transmitted and regardless of the topology of the network in which said information travels.

Índice de contenidos

Índice de figuras

Índice de tablas

Índice de fragmentos de código

Lista de acrónimos

ACK	*Acknowledgement*
CPU	*Central Processing Unit*
DDoS	*Distributed Denial of Service*
DoS	*Denial of Service*
DT	*Decision Tree*
IEEE	*Institute of Electrical and Electronics Engineers*
IoT	*Internet of Things*
IP	*Internet Protocol*
MAC	*Medium Access Control*
MITM	*Man In The Middle*
ML	*Machine Learning*
NED	*NEtwork Description*
RAM	*Random Access Memory*
SVM	*Support Vector Machine*
UDP	*User Datagram Protocol*
UML	*Unified Modeling Language*
WSN	*Wireless Sensor Network*
XML	*eXtensible Markup Language*

1. Introducción

Los avances en las redes están provocando una evolución hacia una sociedad completamente conectada. Ya no sólo los ordenadores están interconectados entre sí, en la actualidad se busca interconectar objetos cotidianos a través de Internet. Esta tecnología es conocida como el Internet de las Cosas o IoT (*Internet of Things*) y el principal objetivo de este Proyecto de Fin de Grado es investigar y aprender acerca de esta.

El desarrollo de las redes IoT tiene su origen en las aplicaciones domóticas. La domótica consiste en aplicar tecnologías de control y automatización a la vivienda con el objetivo de facilitar la vida de las personas o aumentar la seguridad de los hogares. Las redes de dispositivos IoT se han extendido por numerosos sectores, siendo aplicables en prácticamente cualquier ámbito que haga uso de sensores y/o actuadores: viviendas inteligentes, ciudades inteligentes, salud inteligente, agricultura inteligente, etc.

Los dispositivos IoT en la mayor parte de las ocasiones son dispositivos electrónicos con capacidades limitadas pues son habitualmente implementados en objetos cotidianos como, por ejemplo, una lavadora o un sensor de temperatura. A la hora de diseñar un sistema relacionado con dispositivos del Internet de las Cosas es imprescindible reflexionar sobre las limitaciones que presentan: memoria, capacidad de cálculo, batería, etc. Además, una de las características principales de este tipo de redes es su naturaleza inalámbrica. Los dispositivos IoT se comunican entre sí a través del aire, lo cual puede conllevar retos en el ámbito de la seguridad de la información que intercambian.

El intercambio de información a través de redes inalámbricas supone tener en cuenta posibles amenazas. Existen multitud de ataques sobre redes IoT para los cuales se han desarrollado estrategias con el objetivo de proteger la información intercambiada. El propósito de este proyecto es presentar una solución de seguridad para redes de dispositivos IoT, independiente de la topología de la red y de la información que en ella se transmite, haciendo uso de mecanismos de aprendizaje automático con el fin de diseñar un sistema automatizado de detección de ataques.

1.1. Objetivos

El objetivo principal del proyecto consiste en el diseño e implementación de un sistema de seguridad en una red de dispositivos IoT usando mecanismo de aprendizaje automático. Este sistema tratará de detectar si se está produciendo un ataque en la red de forma automática analizando el tráfico de paquetes intercambiado en ella.

Para lograr cumplir el objetivo principal, se han establecido objetivos específicos para facilitar alcanzarlo:

- Viabilidad

  El primer objetivo específico consiste en demostrar la viabilidad de integrar soluciones de seguridad basadas en mecanismos de aprendizaje automáticos a redes de dispositivos IoT como, por ejemplo, redes de sensores.

- Simulación

  El segundo objetivo específico establecido trata de lograr simular redes IoT genéricas en las cuales integrar el sistema automatizado de detección de ataques propuesto. Además, este segundo objetivo específico permite demostrar la independencia del sistema con la topología de red y con la información transportada en los paquetes que en ella se transmiten.

- Rendimiento

  El último objetivo específico establecido consiste en la validación de la solución de seguridad propuesta mediante la realización de pruebas de rendimiento. Estas pruebas serán llevadas a cabo mediante la simulación de las respectivas redes de dispositivos IoT con el sistema automatizado de detección de ataques integrado.

1.2. Estructura del documento

En este apartado se expondrá la estructura de la memoria del proyecto con el fin de proporcionar una visión global del documento.

La memoria del Proyecto de Fin de Grado está organizada en capítulos y a continuación se explicará el contenido de cada uno de ellos.

- Capítulo 1. Introducción

  El primer capítulo de la memoria del proyecto consiste en la *Introducción*. En él se presenta una breve descripción del marco tecnológico en el cual se desarrollará la propuesta. Además, se muestra el objetivo principal del proyecto junto con los objetivos específicos que facilitarán lograr cumplirlo y la estructura del documento.

- Capítulo 2. Estado del arte

  El capítulo número dos trata del *Estado del arte* y trata de exponer el marco tecnológico y los antecedentes del proyecto. En primer lugar, se expone la definición de la tecnología central de la propuesta. Tras ello, se muestra el estudio de distintas plataformas IoT existentes que pueden ser encontradas en el mercado. Una vez expuestas las diferentes opciones de plataformas IoT, se definen y se analizan los posibles ataques que pueden tener lugar en redes IoT. Conociendo las potenciales amenazas que pueden ocurrir, seguidamente se estudian soluciones de seguridad en el ámbito de las redes de dispositivos IoT basadas en mecanismos de aprendizaje automático tras una breve descripción de sus paradigmas. Por último, se muestran diferentes herramientas de simulación de redes IoT con las que trabajar en la propuesta.

- Capítulo 3. Diseño de la solución propuesta

  El tercer capítulo se denomina *Diseño de la solución propuesta* y en él se realiza una descripción del diseño del proyecto. Primero se exponen las especificaciones y las restricciones que conlleva el desarrollo de la solución propuesta junto con el estudio del presupuesto de la misma. Después se lleva a cabo la descripción del escenario de estudio y se expone el diseño detallado del sistema apoyado sobre los distintos diagramas UML

(*Unified Modeling Language*). Finalmente, se estudia el mecanismo de aprendizaje automático escogido para ser implementado en el sistema automatizado de detección de ataques.

- Capítulo 4. Implementación

  El cuarto capítulo de la memoria del proyecto consiste en la *Implementación*. En este capítulo se explica el desarrollo de la solución de seguridad propuesta empezando por una descripción detallada de los escenarios de aplicación. Seguidamente, se muestra la implementación del mecanismo de aprendizaje automático escogido y se explica de forma profunda. Y, finalmente, se expone la integración del sistema de detección de ataques con los escenarios diseñados en el entorno de simulación.

- Capítulo 5. Validación y resultados

  La exposición del conjunto de pruebas realizado para el análisis del rendimiento del sistema desarrollado se lleva a cabo en el capítulo cinco, *Validación y resultados*. En él se describen las pruebas de validación con las que se comprobará el rendimiento de la solución propuesta y las diferentes métricas utilizadas para ello junto con los resultados obtenidos en ellas.

- Capítulo 6. Conclusiones y trabajos futuros

  Por último, el sexto capítulo consiste en la exposición de las *Conclusiones y trabajos futuros*. Primero se analiza el conjunto el proyecto y se extraen las conclusiones del estudio. Y finalmente, se proponen trabajos futuros sobre los que continuar la investigación y el desarrollo.

2. Estado del arte

El concepto de Internet de las Cosas o IoT es definido por el IEEE (*Intitute of Electrical and Electronics Engineers*) como "Un grupo de infraestructuras que interconectan los objetos conectados y permiten su gestión, minería de datos y el acceso a los datos que generan" siendo dichos objetos "Sensores y/o actuadores que realizan una función específica y que pueden comunicarse con otros equipos" [1].

La información intercambiada entre los dispositivos interconectados en la red puede ser, en ocasiones, de naturaleza sensible. Por este motivo, la seguridad es un requisito imprescindible en las comunicaciones. Existen diversos ciberataques en redes IoT para los cuales se implementan servicios de seguridad con el objeto de preservar el correcto funcionamiento de la red o la información que se intercambia en ella.

En este apartado se presentarán, en primer lugar, las distintas plataformas que soportan esta tecnología. En segundo lugar, se estudiarán los diversos ciberataques específicos de este tipo de redes y, tras ello, las soluciones de seguridad basadas en aprendizaje automático que existen para hacer frente a los intentos de vulnerar las políticas de seguridad de los sistemas. Finalmente, se realizará una comparación entre distintos simuladores que permiten emular redes IoT con el objetivo de implementar soluciones de seguridad y comprobar su rendimiento.

2.1. Plataformas IoT

Una plataforma IoT gestiona la conectividad de los dispositivos y facilita la recopilación de datos en un sistema IoT. Además, conecta los diferentes componentes, asegurando un flujo de comunicación ininterrumpido entre ellos. Dichos elementos son denominados nodos y poseen componentes básicos como CPU (*Central Processing Unit*), memoria RAM (*Random Access Memory*), memoria Flash o módulo de comunicación, además de los respectivos sensores y actuadores.

Es importante destacar el hecho de que los nodos IoT, por regla general, disponen de capacidades limitadas. En este apartado se presentarán diferentes dispositivos IoT con sus características y se realizará una comparativa de las capacidades que ofrecen los distintos nodos en la Tabla 1.

2.1.1. Adafruit Feather HUZZAH con ESP8266

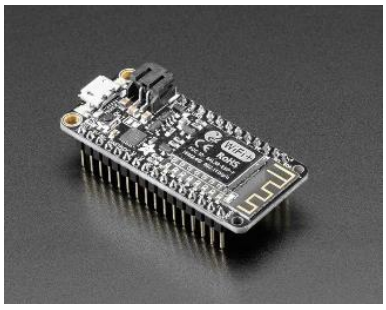

Figura 1. Adafruit Feather HUZZAH [2]

Procesador	Xtensa LX106
RAM	64 KB + 96 KB
Velocidad	80 MHz
Flash	4 MB
Arquitectura	Tensilica 32 bit
Almacenamiento externo	No
Tamaño	51 mm x 23 mm x 8 mm

2.1.2. Arduino Nano

Figura 2. Arduino Nano [3]

Procesador	ATmega328
RAM	2 KB
Velocidad	16 MHz
Flash	32 KB
Arquitectura	AVR 8 bit
Almacenamiento externo	No
Tamaño	18 mm x 45 mm

2.1.3. BeagleBone Black

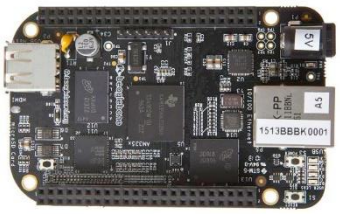

Figura 3. BeagleBone Black [4]

Procesador	AM3358 ARM Cortex-A8
RAM	512 MB
Velocidad	1 GHz
Flash	4 GB
Arquitectura	ARM 32 bit
Almacenamiento externo	MicroSD
Tamaño	86,36 mm x 53,34 mm

2.1.4. DE10-Nano

Figura 4. DE10-Nano [5]

Procesador	Dual-Core ARM Cortex-A9
RAM	1 GB
Velocidad	800 MHz
Flash	8 GB
Arquitectura	ARM 32 bit
Almacenamiento externo	MicroSD
Tamaño	68,6 mm x 107 mm

2.1.5. Intel Galileo

Figura 5. Intel Galileo [6]

Procesador	Intel Quark X1000
RAM	512 KB
Velocidad	400 MHz
Flash	8 MB
Arquitectura	Pentium 32 bit
Almacenamiento externo	MicroSD
Tamaño	106,68 mm x 71,12 mm

2.1.6. Libelium Waspmote

Figura 6. Libelium Waspmote [7]

Procesador	ATmega1281
RAM	8 KB
Velocidad	16 MHz
Flash	128 KB
Arquitectura	AVR 8 bit
Almacenamiento externo	SD
Tamaño	73,5 mm x 51 mm x 13 mm

2.1.7. NodeMCU v2 ESP8266 WiFi

Figura 7. NodeMCU v2 [8]

Procesador	Xtensa LX3
RAM	128 KB
Velocidad	80 MHz
Flash	4 MB
Arquitectura	Tensilica 32 bit
Almacenamiento externo	MicroSD
Tamaño	49 mm x 26 mm x 12 mm

2.1.8. Raspberry Pi Zero W

Figura 8. Raspberry Pi Zero W [9]

Procesador	ARM1176
RAM	512 MB
Velocidad	1 GHz
Flash	4 MB
Arquitectura	ARM 32 bit
Almacenamiento externo	MicroSD
Tamaño	65 mm x 30 mm x 5 mm

2.1.9. WiPy 3.0

Figura 9. WiPy 3.0 [10]

Procesador	Dual-Core Xtensa LX6
RAM	4 MB
Velocidad	240 MHz
Flash	8 MB
Arquitectura	Tensilica 32 bit
Almacenamiento externo	MicroSD
Tamaño	42 mm x 20 mm x 2,5 mm

A continuación, se muestra la Tabla 1 donde se realiza la comparación de las distintas plataformas IoT mostradas anteriormente.

Tabla 1. Comparación entre plataformas IoT

Plataforma	**RAM**	**Flash**	**Velocidad**
Adafruit Feather HUZZAH	130 KB	4 MB	80 MHz
Arduino Nano	2 KB	32 KB	16 MHZ
BeagleBone Black	512 MB	4 GB	1 GHz
DE10-Nano	1 GB	8 GB	800 MHz
Intel Galileo	512 KB	8 MB	400 MHz
Libelium Waspmote	8 KB	128 KB	16 MHz
NodeMCU v2	128 KB	4 MB	80 MHz
Raspberry Pi Zero W	512 MB	4 MB	1 GHz
WiPy 3.0	4 MB	8 MB	240 MHz

2.2. Ataques en redes IoT

Una amenaza es una violación potencial de la seguridad que podría causar daños como la destrucción o modificación de la información. Un ataque es un acto intencional por el cual una entidad intenta evadir los servicios de seguridad y violar la política de seguridad de un sistema. Hablamos de ciberataque cuando hacemos referencia al conjunto de acciones ofensivas contra sistemas de información, en este caso, redes IoT.

De acuerdo con el trabajo realizado por Liang Xiao et al. [11], existen diversos tipos de ciberataques. En este apartado se estudiarán los diferentes ataques que pueden ocurrir en el ámbito del Internet de las Cosas.

2.2.1. Control de entidades

El ciberataque basado en el control de entidades es conocido como ataque *Sybil* y se caracteriza por la manipulación de entidades falsas con el objetivo de comprometer el correcto funcionamiento del sistema [12]. Esta clase de ataque es frecuente encontrarla en las redes sociales donde, además de difundir spam y anuncios, las cuentas falsas intentan infectar con virus al resto de usuarios. En el caso de las redes IoT, el atacante trata de controlar el mayor número de nodos con la intención de acceder a un recurso concreto o, incluso, controlar la red.

2.2.2. Denegación de servicio

El ciberataque de denegación de servicio o DoS (*Denial of Service*) consiste en inundar el servidor objetivo con peticiones superfluas con el objeto de saturar este e interrumpir el servicio. Este tipo de ataque se complica cuando, en vez de ser realizado por una única entidad, es llevado a cabo por múltiples entidades con diferentes direcciones IP (*Internet Protocol*) y es conocido como DDoS (*Distributed Denial of Service*).

2.2.3. Espionaje

El espionaje, también conocido como *eavesdropping*, *sniffing* o *snooping*, es un tipo de ataque en el cual se busca obtener información de forma ilícita. Esta acción puede ser desempeñada de diferentes modos. Un ejemplo de ataque de espionaje sería el realizado por

un nodo que suplantase la dirección MAC (*Medium Access Control*) con el propósito de acceder ilegalmente a los datos transmitidos a dicha dirección.

2.2.4. Filtración

De acuerdo al estudio realizado por Tianbo Gu et al. [13], las redes IoT pueden sufrir un ciberataque conocido como filtración de la intimidad cuando las comunicaciones se realizan de forma inalámbrica. Debido a que en este tipo de comunicación los datos van cifrados, el propósito es detectar los eventos IoT que ocurren con los campos de los mensajes que se transmiten en claro comprometiendo así la privacidad de la información.

2.2.5. Modificación de la información

El ciberataque conocido como hombre en el medio o MITM (*Man In The Middle*) ocurre cuando un nodo obtiene la capacidad de leer, modificar e insertar información en una comunicación de forma no lícita. Cuando este ataque se realiza sobre una comunicación Bluetooth, como se ilustra en el trabajo de G. Rajendran et al [14], se denomina ataque *BlueBorne* y el nodo malicioso hace entender a los nodos víctima que han realizado el procedimiento de *paring* con éxito. Tras ello, el atacante puede leer, modificar e inyectar información a voluntad en la comunicación.

2.2.6. Interferencia

La acción ofensiva conocida como interferencia o *jamming* trata de interrumpir las transmisiones radio de los dispositivos IoT a través del envío de señales falsas. La consecuencia de esta clase de ciberataque es el agotamiento del ancho de banda, la energía y los recursos de memoria y computacionales de los dispositivos conectados a la red IoT debido a los intentos fallidos de comunicación.

2.3. Soluciones seguras en redes IoT

Una vez descubiertos los posibles ciberataques en redes IoT, es importante incorporar los servicios de seguridad en el sistema para minimizar los daños. El primer paso para proteger la red es desarrollar dichos servicios. De acuerdo con la recomendación X.800 [15], un servicio de seguridad consiste en un servicio, provisto por una capa de sistemas abiertos interconectados, que garantiza la seguridad adecuada de los sistemas o de las transferencias de datos. La recomendación X.800 del ITU-T define los servicios de seguridad que se muestran a continuación:

- Confidencialidad

  El servicio de confidencialidad proporciona a los datos la protección necesaria para evitar que sean revelados a una entidad no autorizada. Sólo la entidad que disponga de autorización podrá acceder a los datos. Este servicio de seguridad protege a la red de los ataques de espionaje.

- Integridad

  El servicio de integridad consiste en garantizar que los datos enviados por el emisor coinciden con los recibidos por el receptor, es decir, garantizar que no se ha producido

ninguna manipulación de los datos enviados. Este servicio de seguridad protege a la red de ataques de modificación de la información.

- Autenticación

  El servicio de autenticación trata de confirmar la identidad de la entidad comunicante, en otras palabras, confirmar que el emisor es quien dice ser. Este servicio de seguridad protege a la red de ataques de suplantación de identidad.

- No repudio

  El servicio de no repudio permite al emisor y receptor de una comunicación evitar la negación de la participación en el intercambio de información. Este servicio de seguridad puede aplicarse en tres situaciones: con prueba de origen, con prueba de envío o con prueba de entrega.

- Control de acceso

  El servicio de seguridad de control de acceso evita el uso ilícito de los recursos de la red por entidades carentes de autorización.

En este apartado, se estudiarán diferentes soluciones de seguridad que implementan servicios de seguridad basadas en aprendizaje automático. Antes de mostrar dichas soluciones, es importante entender las bases de los mecanismos de aprendizaje automático. El aprendizaje automático o ML (*machine learning*) es una rama de la inteligencia artificial dedicada a la generación de modelos basados en el análisis de datos para identificar patrones y tomar decisiones. El ML construye de forma automática estos modelos mediante iteraciones en las que optimiza el sistema y mejora el resultado, es decir, aprende.

Dentro del aprendizaje máquina existen tres paradigmas: aprendizaje supervisado, aprendizaje no supervisado y aprendizaje por refuerzo. A continuación, se muestra una breve explicación de los paradigmas de acuerdo al trabajo de A. Nawrocka et al. [16]:

- Aprendizaje supervisado

  Este paradigma se caracteriza por conocer los valores de salida esperados durante la fase de entrenamiento. En otras palabras, el modelo es entrenado con datos de entrada y salida para comparar durante esta fase los resultados y ajustar los pesos.

- Aprendizaje no supervisado

  En el caso del paradigma de aprendizaje no supervisado, el modelo no tiene en cuenta la salida que se espera obtener. El resultado del uso de este mecanismo consiste en el agrupamiento de los datos o *clustering*.

- Aprendizaje por refuerzo

  El paradigma del aprendizaje por refuerzo se caracteriza por la recopilación de la información del entorno y la obtención de una recompensa según las acciones realizadas. Este mecanismo busca optimizar dicha recompensa mediante las iteraciones.

Después de conocer los paradigmas del aprendizaje automático, se realizará un estudio de las soluciones de seguridad para redes IoT basadas en ML. Seguidamente, se muestra dicho estudio de acuerdo al trabajo de M. A. Al-Garadi et al. [17].

2.3.1. Árboles de decisión

El método de árboles de decisión o DTs (*Decision Trees*) es un ejemplo de aprendizaje supervisado y consiste en clasificar las muestras en función de sus características a través de una representación en árbol donde cada nodo se corresponde con una característica y cada enlace denota el valor que puede tomar dicha característica para ser clasificada.

Generalmente, este mecanismo está compuesto por dos etapas: la etapa de construcción y la etapa de clasificación. En la primera consiste en generar el árbol vacío, asignar en primer lugar la característica capaz de separar mejor las muestras del conjunto de datos como nodo origen y continuar asignando el resto de características en los nodos posteriores de acuerdo al mismo principio. La segunda etapa trata de obtener los valores óptimos de los enlaces para obtener el mejor resultado posible en la clasificación.

Este método puede ser aplicado en redes IoT tomando como entrada la información relativa al tráfico para detectar nodos sospechosos de realizar un ataque de DDoS como muestra el estudio realizado por S. Alharbi et al [18].

2.3.2. Máquinas de vectores de soporte

Este método de aprendizaje supervisado es conocido como SVMs (*Support Vector Machines*) y es uno de los mecanismos más utilizados para realizar tareas de clasificación. Las SVMs llevan a cabo la clasificación mediante un hiperplano separador de tantas dimensiones como tengan las muestras. Dicho hiperplano trata de mantener la mayor distancia posible entre las muestras de las distintas clases para optimizar la clasificación.

En su origen, las máquinas de vectores de soporte fueron creadas para aplicarse en situaciones donde las clases de datos eran linealmente separables. Actualmente, gracias al uso de funciones que aumentan las dimensiones del espacio, este mecanismo es capaz de transformar los datos de entrada para ser linealmente separables y, una vez generado el hiperplano, retornar al espacio original.

En el escenario de las redes IoT, este método de aprendizaje automático puede ser aplicado para solventar problemas de intrusión en tiempo real con la capacidad de mejorar los patrones de clasificación dinámicamente como se muestra en el trabajo de W. Hu et al. [19] o el realizado por C. Wagner et al. [20].

2.3.3. K-medias

El método de K-medias o *K-means clustering* está basado en el aprendizaje no supervisado y busca agrupar los datos en K conjuntos. Esta agrupación se realiza de forma iterativa teniendo en cuenta las características de las muestras. Por tanto, cada conjunto contendrá muestras cuyas características son semejantes.

El procedimiento llevado a cabo por este mecanismo consiste en la repetición de dos etapas. Primero, se estiman K centros y se asigna cada muestra al centro más cercano según la

distancia euclídea. Tras haber agrupado todas las muestras, se calcula la media de las muestras de cada conjunto y se vuelven a estimar los centros. El proceso termina cuando no haya muestras que hagan reconfigurar los conjuntos, es decir, cuando todas las muestras estén perfectamente agrupadas.

Pese a que los algoritmos de aprendizaje no supervisado todavía no han sido muy explorados en IoT, en trabajos como el realizado por H.-B. Wang et al. [21] se detectan intrusiones en redes de sensores inalámbricos o WSN (*Wireless Sensor Networks*).

2.3.4. *Q-learning*

El algoritmo *Q-learning* es un método de aprendizaje por refuerzo que consiste en la generación de una tabla de normas que indiquen al sistema que acciones realizar según el estado actual. Esta tabla se construye a través del sistema de recompensas característico de este paradigma del aprendizaje automático. Esta técnica permite al sistema aprender la estrategia óptima en entornos dinámicos.

Un ejemplo de aplicación del mecanismo de *Q-learning* en redes IoT se muestra en el documento de L. Xiao et al. [22], en el cual detectan un ataque de suplantación o *spoofing*. En este trabajo se aplica teoría de juegos para realizar el procedimiento de autenticación de acuerdo con el estado del canal físico, buscando el umbral óptimo para obtener la mayor precisión en la detección de intrusos mediante aprendizaje por refuerzo.

2.4. Simuladores de redes IoT

Para llevar a cabo la simulación de redes de dispositivos IoT es necesario hacer uso de programas conocidos como simuladores de eventos discretos. Gracias a estos programas es posible probar distintos escenarios de red e, incluso, añadir nuevas funcionalidades a las redes diseñadas.

A continuación, se muestran diferentes simuladores de redes IoT con una breve descripción.

2.4.1. NS-2

NS-2 [23] es un simulador de eventos discretos para la investigación sobre redes de ordenadores. Está escrito en el lenguaje de programación C++ y permite modificar el código de acuerdo a las necesidades.

Para poder realizar la visualización de las redes en NS-2 es imprescindible utilizar NAM [24]. NAM es una herramienta para visualizar la simulación del tráfico en redes de ordenadores.

2.4.2. GNS3

GNS3 [25] es un potente simulador gráfico de redes que permite diseñar topologías de red y simular el tráfico en ellas. Este programa de simulación es famoso por ser utilizado como herramienta para la preparación de distintas certificaciones en el ámbito de las redes de telecomunicación. Está escrito en el lenguaje de programación *Python*.

2.4.3. OMNeT++

OMNeT++ [26] es un simulador modular de eventos discretos orientado al estudio de las redes de ordenadores y sus protocolos. Este programa está diseñado de forma modular, es decir, el núcleo de las simulaciones, los modelos, la interfaz gráfica, etc. están separados. Esta característica permite integrar el simulador con aplicaciones personalizadas.

Los módulos están escritos en el lenguaje de programación C++ y se agrupan en objetos de alto nivel mediante un lenguaje de descripción denominado NED (*NEtwork Description*). Es posible diseñar aplicaciones personalizadas dentro del propio simulador, característica que favorece la integración con otras aplicaciones.

3. Diseño de la solución propuesta

En este capítulo se mostrará el planteamiento del sistema propuesto como solución de seguridad en redes IoT basado en aprendizaje automático.

Antes de comenzar la descripción del sistema es importante analizar las especificaciones y restricciones generales del mismo. En segundo lugar, se estudiará el presupuesto necesario para realizar el proyecto. Seguidamente, se describirá el sistema automatizado propuesto. Y, por último, se realizará el estudio detallado del mismo con los respectivos diagramas UML.

3.1. Especificaciones y restricciones generales del sistema

El escenario general en el que se desarrollará el proyecto consiste en una red de sensores centralizada e inalámbrica. Los sensores pertenecientes a la red envían paquetes con información a un nodo central denominado nodo pasarela y este responde a los sensores con un mensaje de acuse de recibo ACK (*Acknowledgement*). No está permitida la comunicación entre sensores, únicamente se permite la comunicación entre el sensor y el nodo central. Por este motivo, cualquier comunicación ilícita se considerará un comportamiento malicioso.

En la Figura 10 se puede observar un ejemplo de red de sensores centralizada en la cual aparecen cuatro nodos sensores y un nodo pasarela.

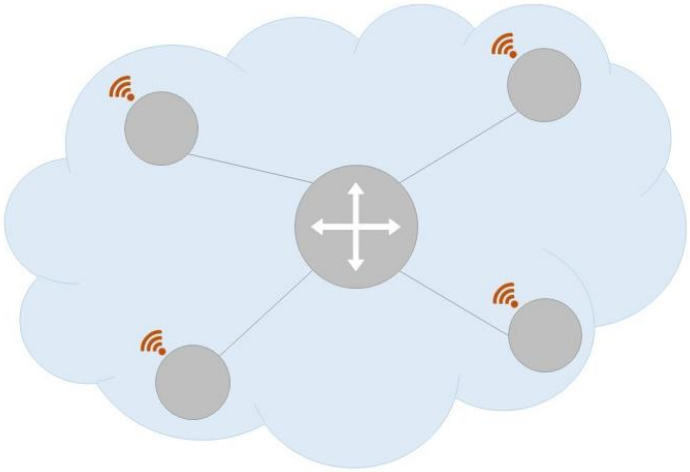

Figura 10. Ejemplo de red de sensores centralizada

3.1.1. Especificaciones

- Algoritmos avanzados de seguridad

  La solución propuesta estará basada en algoritmos avanzados de seguridad, concretamente, hará uso de mecanismos de aprendizaje automático.

- Dispositivos IoT simulados

  El sistema será implementado en una red de dispositivos IoT simulados. Por tanto, se hará uso de un simulador de redes para llevar a cabo el desarrollo de la solución propuesta.

- Comunicación inalámbrica y segura

  La comunicación entre los nodos de la red será inalámbrica y deberá ser segura. La seguridad de la comunicación tendrá su origen en el sistema desarrollado en este proyecto.

3.1.2. Restricciones

- Sistema eficiente

  El sistema diseñado deberá ser eficiente, los sensores disponen de capacidad de computación, memoria y batería limitadas.

- Software libre

  Se utilizarán, en la medida de lo posible, soluciones de software libre (*open source*). Esta restricción tiene relación directa con el presupuesto del proyecto.

3.2. Presupuesto

En este apartado se recogen los presupuestos requeridos para la elaboración del proyecto. Se realizará una división de los presupuestos de acuerdo a su naturaleza: presupuesto de mano de obra, presupuesto hardware y presupuesto software.

3.2.1. Recursos humanos

A continuación, en las Figuras 11 y 12, se muestra el diagrama de Gantt realizado para la planificación del proyecto.

Tareas	Inicio	Fin	Duración (horas)
Estado del arte	01/01/2021	31/01/2021	40
Diseño del sistema	01/02/2021	28/02/2021	40
Implementación y validación	01/03/2021	31/05/2021	140
Redacción del documento	25/01/2021	15/06/2021	100

Figura 11. Diagrama de Gantt - Desglose

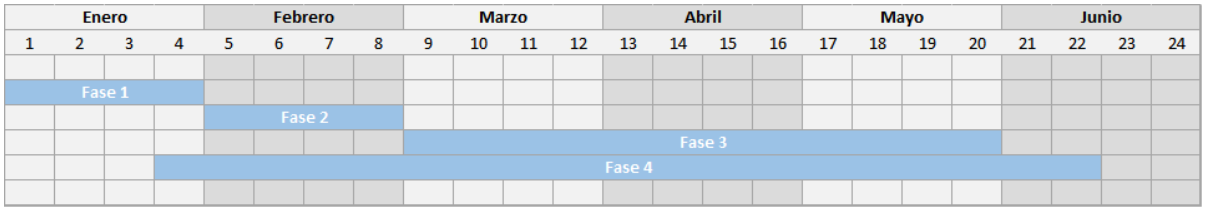

Figura 12. Diagrama de Gantt - Cronograma

En el desglose del diagrama de Gantt se pueden observar las tareas planificadas con sus respectivas fechas de inicio y fin además de las horas de trabajo estimadas para cada una de ellas. La duración total aproximada del proyecto consiste en cinco meses y dos semanas con

un total de trecientas veinte horas de trabajo. De acuerdo a esta estimación, se calcularán los costes asociados a los recursos humanos empleados.

Tabla 2. Costes estimados relativos a los recursos humanos necesarios

Concepto	Salario bruto anual	Meses de desarrollo	Coste estimado total
Ingeniero desarrollador	24.000 €	5,5 meses	11.000 €

3.2.2. Recursos hardware

Debido a que el proyecto se desarrolla sobre una red de sensores simulada, los requisitos hardware se reducen al equipo en el cual se ejecute el simulador y el sistema de detección de forma simultánea. Los costes estimados relacionados con los recursos hardware necesarios se muestran en la Tabla 3.

Tabla 3. Costes estimados relativos a los recursos hardware necesarios

Concepto	Coste estimado
Equipo con sistema operativo Windows instalado	700 €

3.2.3. Recursos software

De acuerdo a lo expuesto en el punto anterior, el proyecto se lleva a cabo en una red de dispositivos IoT simulados y, por tanto, es necesario un software específico para llevar a cabo tanto el desarrollo como la emulación.

Teniendo en cuenta las restricciones de diseño, se han escogido las opciones de software libre para elaborar el proyecto. Tanto el programa simulador como el entorno de desarrollo integrado utilizado son gratuitos tal y como se puede observar en la Tabla 4. El coste del sistema operativo se incluye en el precio estimado del equipo.

Tabla 4. Costes estimados relativos a los recursos software necesarios

Concepto	Coste estimado
Simulador OMNeT++	0 €
Entorno de desarrollo integrado PyCharm	0 €

3.2.4. Coste total

En la Tabla 5 se puede observar el coste total estimado de los recursos necesarios para realizar el proyecto. Se han incluido todos los recursos para tener una visión global del coste por la naturaleza del proyecto.

Tabla 5. Coste total estimado del proyecto

Concepto	Coste estimado
Recursos humanos	11.000 €
Recursos hardware	700 €
Recursos software	0 €
Total	11.700 €

3.3. Descripción general de la propuesta

Tal y como se puede observar en la Figura 13, siguiendo el ejemplo de escenario mostrado en el apartado 3.1, se añade a la red de sensores un sistema de detección de ataques basado en mecanismos de aprendizaje automático.

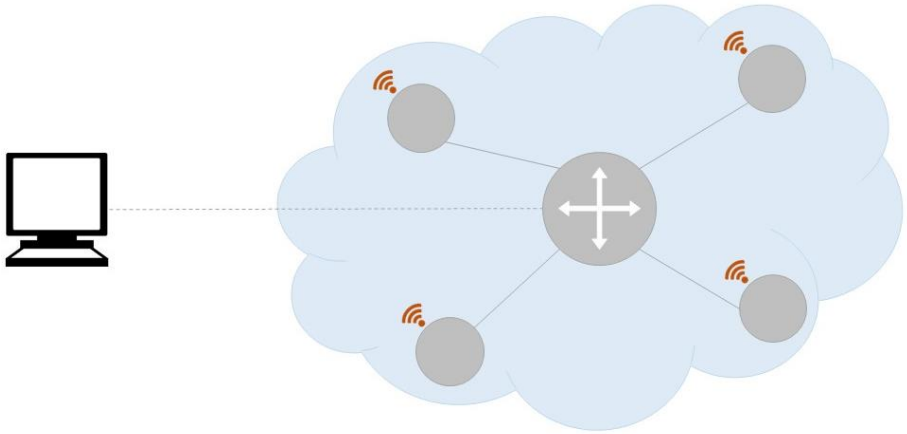

Figura 13. Ejemplo de red de sensores centralizada con sistema de detección

Durante la comunicación entre los nodos de la red, el nodo pasarela realiza capturas PCAP del tráfico. Esto es posible debido a que la comunicación es inalámbrica y, gracias a ello, es posible detectar posibles ataques a la red realizando el análisis en un único nodo.

El sistema de detección se encarga de analizar los paquetes intercambiados según son añadidos a la captura de tráfico. Si el sistema de detección determina que un paquete puede ser malicioso, se lanza un aviso de potencial ataque con la información de dicho paquete. El mecanismo que utiliza el analizador estudia el comportamiento normal de la red para

prepararse contra posibles ataques. De este modo, cuando el comportamiento de la red se ve alterado, detecta las posibles amenazas.

3.4. Diseño detallado. Diagramas UML

Para lograr un mayor entendimiento de la solución de seguridad propuesta, en este apartado se estudiarán los diagramas UML diseñados para el sistema y el mecanismo basado en aprendizaje automático utilizado para determinar si se está produciendo un posible ataque a la red de dispositivos IoT.

3.4.1. Diagrama de despliegue

En la Figura 14 se puede observar el diagrama de despliegue de la solución propuesta.

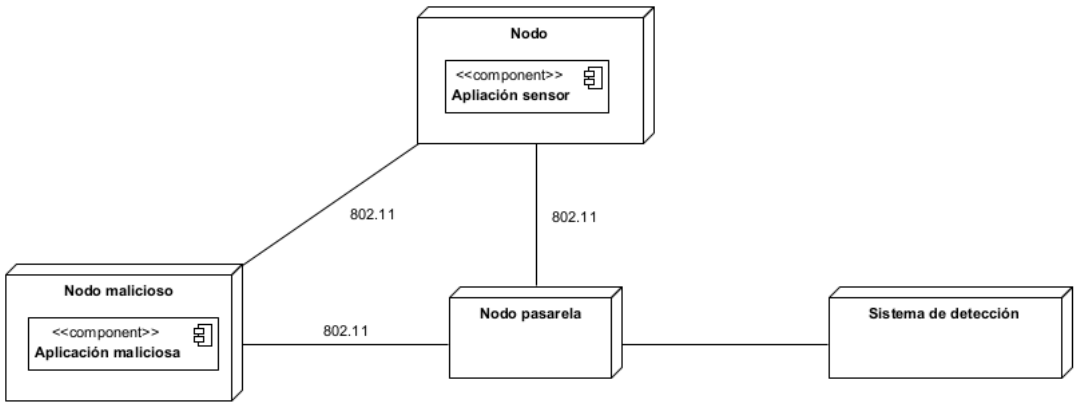

Figura 14. Diagrama de despliegue

Debido a la naturaleza inalámbrica de la conexión entre los nodos de la red, los paquetes intercambiados en ella son escuchados por todos los nodos. Por este motivo, en la Figura 14 se muestra un enlace entre nodos sensores. Una vez explicado ese detalle, el despliegue ilustrado consiste en un conjunto de nodos sensores que se comunican con un nodo central denominado nodo pasarela el cual transmite al sistema de detección la captura PCAP del tráfico para ser analizada y determinar un posible ataque. El nodo malicioso representado se trata de un nodo sensor que actúa de forma anómala, ya sea intentando comunicarse directamente con otro nodo sensor o, por ejemplo, transmitiendo información a tasas distintas a las usadas por el resto de nodos.

Los nodos sensores y el nodo pasarela se comunican entre sí de forma inalámbrica, concretamente mediante el estándar 802.11. Los nodos sensores a nivel de aplicación crean paquetes de datos con la información de sensado y los envían a los niveles inferiores para ser transmitidos al nodo pasarela. La aplicación sensor se encarga de dotar a los nodos de dicho comportamiento. El nodo malicioso actúa de forma similar al nodo sensor pero omite reglas como la centralización del tráfico o la tasa de transmisión. Por otra parte, el nodo pasarela simplemente actúa de sumidero y se comunica con el sistema de detección mediante la transferencia de capturas PCAP del tráfico.

El sistema de detección implementa un mecanismo de aprendizaje automático para analizar las capturas PCAP, determinar si los paquetes intercambiados son fraudulentos y determinar, por tanto, un potencial ataque a la red.

3.4.2. Diagrama de secuencia del comportamiento normal de la red

La comunicación entre los nodos sensores y el nodo pasarela se ilustra en la Figura 15.

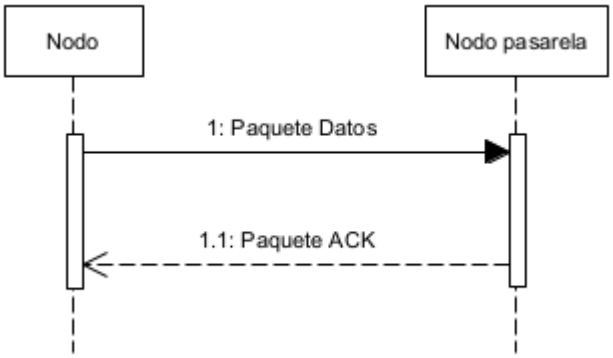

Figura 15. Diagrama de secuencia del comportamiento normal de la red

Como se puede observar en la Figura 15, el intercambio de información entre los nodos sensores y el nodo pasarela es simple. El nodo sensor envía un paquete de datos con la información recogida y el nodo central, para el cual va dirigido dicho paquete, confirma la recepción con un acuse de recibo o ACK.

Según se explica en el apartado anterior, debido a que la comunicación es inalámbrica el resto de nodos de la red recibe de igual forma el paquete con información enviado por el nodo sensor. Sin embargo, ninguno de ellos transmite el paquete de acuse de recibo debido a que no son el destinatario del mismo.

3.4.3. Diagrama de secuencia del comportamiento normal de la red junto con el sistema de detección

Si se añade el sistema de detección a la red, el diagrama de secuencia que se obtiene se muestra en la Figura 16.

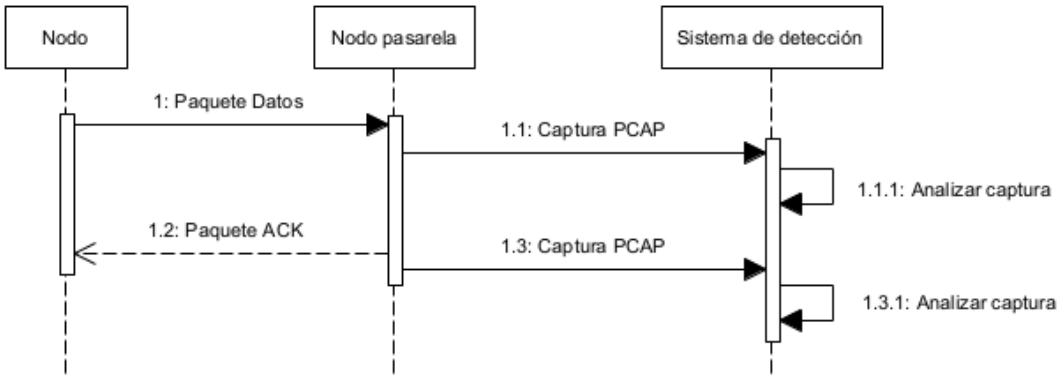

Figura 16. Diagrama de secuencia del comportamiento normal de la red junto con el sistema de detección

De acuerdo con lo expuesto en los apartados anteriores, el nodo pasarela envía al sistema de detección la captura PCAP del intercambio de paquetes producido en la red. Este procedimiento se repite cada vez que el nodo central escucha la transmisión de un paquete. De nuevo, por la naturaleza inalámbrica de las comunicaciones en la red, el nodo pasarela es capaz de escuchar el tráfico completo de la red y, por este motivo, sólo es necesario implementar el sistema de detección en únicamente un nodo.

El sistema de detección según recibe la captura PCAP del intercambio de mensajes, la analiza y determina si se está produciendo un comportamiento anómalo en la red, indicándolo en caso afirmativo.

3.4.4. Diagrama de secuencia de un posible comportamiento malicioso de la red

Un posible comportamiento anómalo de la red consistiría en la transmisión de paquetes de un nodo sensor a otro nodo sensor, lo cual podría ocurrir de forma intencionada y ser, por tanto, un comportamiento malicioso. Esta situación se ilustra en el diagrama de secuencia de la Figura 17.

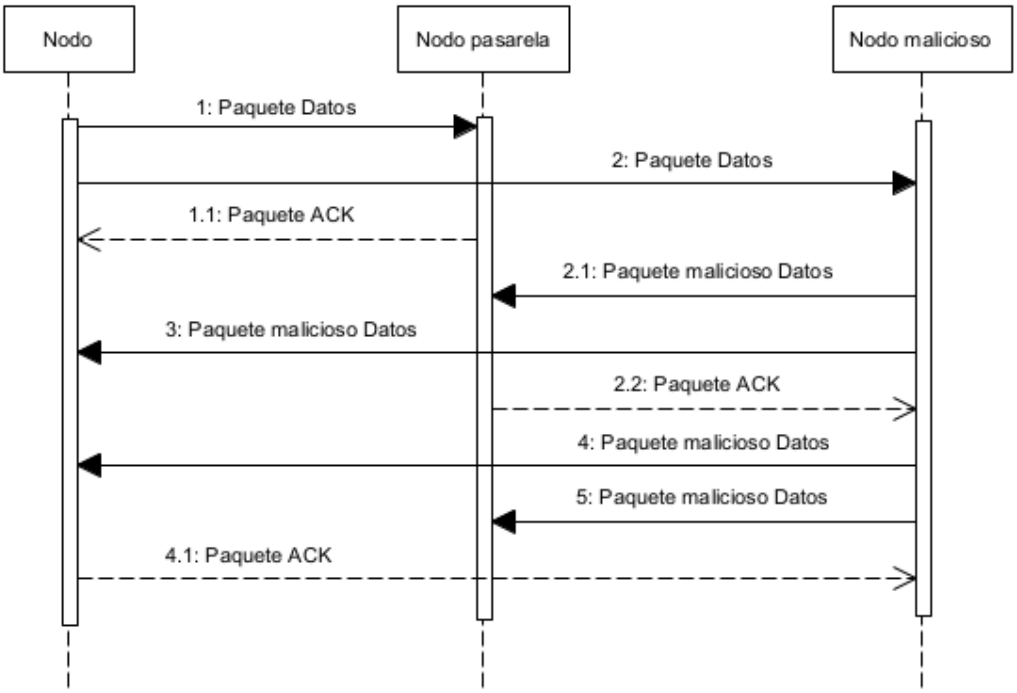

Figura 17. Diagrama de secuencia de un posible comportamiento malicioso de la red

Los nodos de la red transmiten un paquete de acuse de recibo o ACK cuando reciben un paquete de información destinado a ellos. Por tanto, como se puede observar en la Figura 17, sólo uno de los nodos que reciben un paquete de datos transmite un paquete ACK.

El diagrama de secuencia muestra un posible escenario en el cual un nodo sensor, de forma intencionada, intenta intercambiar información con otro nodo sensor. Como se ha expuesto en los apartados anteriores, la red es centralizada y todos los paquetes de información deben ser transmitidos al nodo pasarela. Por ende, el intento de un nodo sensor

de comunicarse con otro nodo sensor es interpretado como un posible comportamiento malicioso.

3.4.5. Diagrama de secuencia de un posible comportamiento malicioso de la red junto con el sistema de detección

Como se observa en la Figura 18, al incorporar el sistema de detección a la red, el nodo pasarela transmite la captura PCAP del intercambio de mensajes en cada ocasión que escucha la transmisión de un paquete.

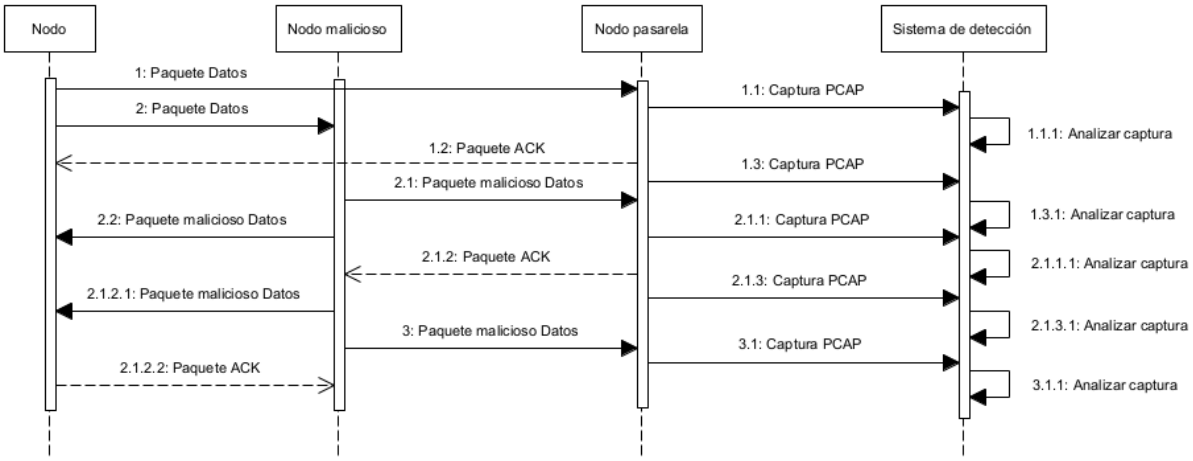

Figura 18. Diagrama de secuencia de un posible comportamiento malicioso de la red junto con el sistema de detección

En la Figura 18 se muestra la transmisión de la captura PCAP del tráfico de la red al sistema de detección con cada paquete transmitido en la red. En esta ocasión, el sistema de detección al analizar la captura PCAP de los mensajes transmitidos por el nodo malicioso debería determinar que está ocurriendo un posible ataque al estudiar su comportamiento anómalo.

Es importante destacar que el sistema de detección se basa en el funcionamiento normal de la red para determinar si se está produciendo un ataque y, por este motivo, si se da la situación de un nodo sensor actuando de forma incorrecta por un error de funcionamiento, esta situación sería detectada igualmente. Por tanto, el sistema de detección determina una posible amenaza tanto un funcionamiento erróneo de un nodo sensor como un acto intencionado de actuar de forma ilícita en la red.

3.4.6. Mecanismo de aprendizaje automático

El mecanismo de aprendizaje automático que utiliza el sistema de detección es un mecanismo distinto a los vistos en los antecedentes del marco tecnológico. El mecanismo escogido para aprender el comportamiento normal de la red y detectar posteriormente anomalías en él es conocido como codificador automático o *autoencoder*.

El *autoencoder* es un tipo de red neuronal artificial con una serie de particularidades. Antes de explicar el funcionamiento del codificador automático, es importante entender primero las redes neuronales artificiales.

Una red neuronal artificial es un sistema computacional formado por unos elementos conocidos como neuronas artificiales conectados entre sí. El modo de conexión de las neuronas artificiales consiste en transmitir el valor de salida de una de ellas a la entrada de la siguiente. Una neurona artificial tiene el aspecto mostrado en la Figura 19 y su funcionamiento se basa en recibir unos valores de entrada, aplicarles una función lineal y entregar a la salida el resultado de introducir el último valor obtenido en una función de activación encargada de eliminar la linealidad del modelo. Existen diferentes funciones de activación, las más conocidas son las funciones rectificadora o *ReLU*, sigmoide y tangente hiperbólica.

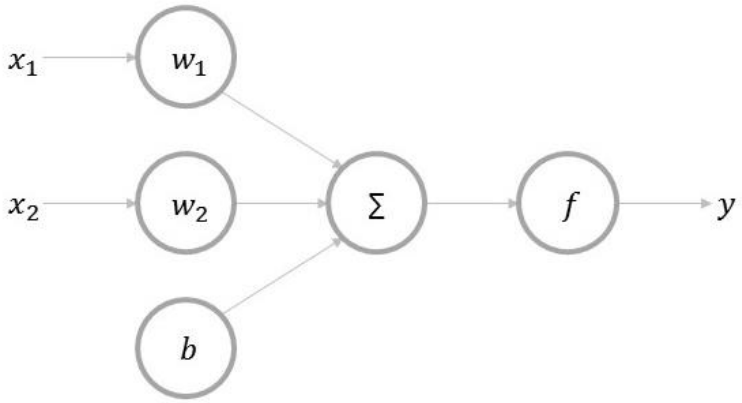

Figura 19. Neurona artificial

En la Figura 19 se pueden observar los valores de entrada x, el sesgo b de la función lineal, la función de activación f y el valor de salida y. El globo que contiene en su interior el sumatorio $\sum$ se encarga de sumar los valores ponderados que recibe y entregar el resultado a la función de activación. Es importante destacar que los pesos w y el sesgo b de la función lineal son valores propios de cada neurona artificial y son los valores que se ajustarán más adelante. Este modelo de neurona artificial se simplifica en la representación de la red neuronal artificial completa como se muestra en la Figura 15.

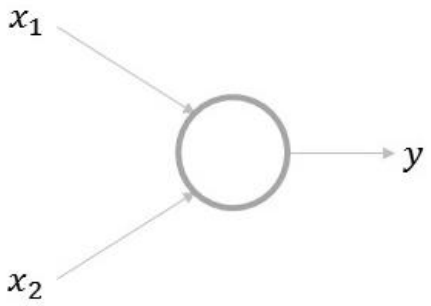

Figura 20. Neurona artificial simplificada

Una red neuronal artificial está formada por neuronas artificiales como las explicadas en los párrafos anteriores organizadas en capas y su aspecto se ilustra en la Figura 21. Este

modelo computacional se caracteriza por la capacidad de ajustarse mediante iteraciones para lograr el resultado deseado. Este proceso se conoce como entrenamiento de la red neuronal artificial y se basa en un método de cálculo denominado propagación hacia atrás.

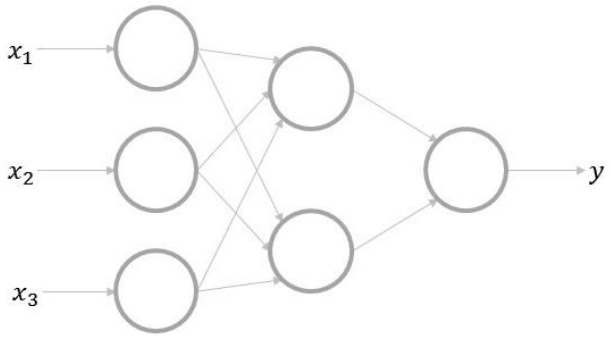

Figura 21. Red neuronal

El método de propagación hacia atrás trata de minimizar la función de pérdida de la red. Está función mide la diferencia entre la salida obtenida en la red neuronal artificial y la salida esperada. Existen multitud de funciones de pérdida y cada una de ellas tiene sus propias características. Una de las funciones de pérdida más utilizada es el error cuadrático medio. Para minimizar la función de pérdida se ajustan los pesos w y los sesgos b de las neuronas calculando su influencia en el resultado final mediante derivadas parciales. Este procedimiento se repite en cada iteración hasta obtener unos resultados satisfactorios.

El *autoencoder* es una red neuronal artificial compuesta por dos etapas. Una primera etapa de compresión en la cual se reduce la dimensionalidad de la entrada de la red y una segunda en la cual se aumenta de nuevo la dimensionalidad hasta igualar la del dato de entrada. En otras palabras, el codificador automático comprime la información de entrada a un espacio latente para después expandirlo hasta lograr obtener la entrada. Este mecanismo se entiende mejor con la Figura 22.

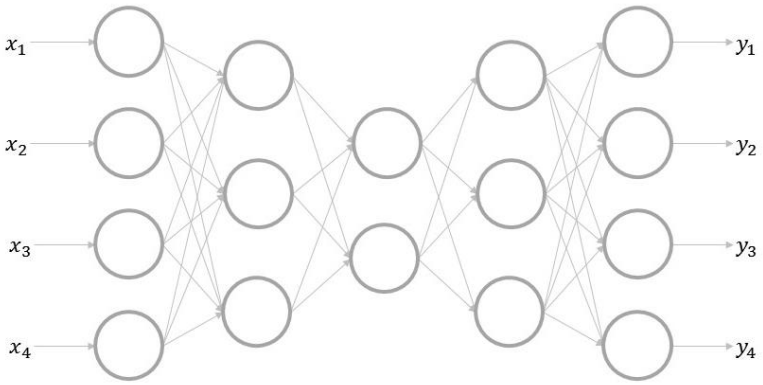

Figura 22. Autoencoder

El objetivo de la red neuronal artificial de la Figura 22 es obtener la salida y de igual valor que la entrada x. La salida de la capa de neuronas centrales se conoce como espacio latente y es el resultado de comprimir la información de entrada. En el ejemplo de la Figura

22, se reduce la información de entrada de la red a la mitad de su dimensionalidad y se vuelve a expandir.

Esta clase de redes neuronales artificiales es utilizada para detectar fraudes o eliminar el ruido de la información. El sistema de detección trata de aprender el correcto funcionamiento de la red para determinar un potencial ataque cuando analiza tráfico fraudulento en la red. Los valores que toma de entrada el codificador automático del sistema de detección son la información transportada en las cabeceras de los protocolos de los paquetes intercambiados en la red.

El entrenamiento del *autoencoder* utiliza únicamente el tráfico de la red sin comportamientos maliciosos para ajustar los parámetros. El entrenamiento finaliza tras lograr obtener un valor de error satisfactorio. Una vez la red neuronal artificial ha sido entrenada con éxito, se analiza el tráfico de la red completo y al procesar paquetes fraudulentos el error supera un umbral determinando así una posible amenaza para la seguridad de la red.

4. Implementación

En este capítulo se explicará la implementación de la solución de seguridad propuesta para redes de sensores. Para ello, se comenzará describiendo los escenarios utilizados en la implementación. Seguidamente, se estudiará el mecanismo de aprendizaje automático usado por el sistema de detección y se mostrará su implementación. Y, por último, se abordará la integración de los escenarios diseñados y el sistema de detección.

4.1. Descripción de los escenarios

Para llevar a cabo la implementación del sistema propuesto es necesario diseñar un escenario sobre el que aplicar la solución de seguridad. Se han creado tres escenarios sobre los que hacer uso del sistema de detección. Dos de ellos consisten en una topología de red de estrella y el tercero se trata de una topología de red de árbol. A continuación, se describirán en detalle los escenarios y se mostrarán los fragmentos de código más relevantes para su comprensión.

4.1.1. Características generales de los escenarios

Antes de mostrar los escenarios es importante explicar las características generales de los mismos.

Según se expuso en capítulos anteriores, el proyecto se desarrolla sobre una red de dispositivos IoT simulados en OMNeT++. Este simulador utiliza un lenguaje propio de descripción de redes NED para el diseño de la red y el lenguaje de programación C++ para dotar de comportamiento a los distintos elementos de la misma.

Utilizando el lenguaje NED se han creado los tres escenarios y todos ellos tienes características en común. Los tres utilizan los mismos dispositivos, los cuales tienen la misma pila de protocolos a excepción de la capa de aplicación:

- Nivel de enlace

  El protocolo de nivel de enlace de los dispositivos IoT simulados es el protocolo 802.11. En la cabecera de este protocolo se indica la dirección física del transmisor y la dirección física del receptor entre otros campos.

  Los paquetes de acuse de recibo se transmiten a nivel de enlace.

- Nivel de red

  Los sensores utilizan a nivel de red el protocolo IP, el cual transporta en su cabecera las direcciones IP origen y destino de los paquetes.

- Nivel de transporte

  Respecto a la capa de transporte, el protocolo utilizado por los dispositivos IoT simulados es el protocolo UDP (*User Datagram Protocol*). En la cabecera del protocolo de nivel de transporte se transmiten los puertos origen y destino de los paquetes.

La longitud de los paquetes de datos es de 3 Bytes. Los paquetes generados por el nivel de aplicación son de ese tamaño debido a la consideración de que las medidas que podrían ser tomadas por los sensores pueden ser representadas con 3 Bytes. Este valor es arbitrario como se verá más adelante en este mismo capítulo.

Los intervalos de transmisión de los paquetes por los nodos de la red siguen una distribución uniforme entre uno y dos segundos. Por tanto, los instantes de tiempo en los cuales los paquetes de datos son transmitidos se rigen por una variable aleatoria con función de distribución uniforme entre uno y dos segundos.

El código escrito en el lenguaje de programación C++ que dota de comportamiento a los nodos sensores a nivel de aplicación se muestra en el Fragmento de código 1.

Fragmento de código 1. Aplicación sensor

```
[…]
class INET_API AppSensor : public ApplicationBase, public UdpSocket::ICallback {
  […]
  protected:
    virtual void sendPacket();
  […]
}
[…]
void AppSensor::sendPacket() {
    auto data = makeShared<BytesChunk>();
    data->setBytes({1,2,3}); // Valores arbitrarios
    Packet *packet = new Packet("Datos", data);
    […]
    socket.sendTo(packet, chooseDestAddr(), destPort);
}
```

En el Fragmento de código 1 es posible apreciar el método más importante de la aplicación que dota de comportamiento a los nodos sensores. El método *sendPacket()* es el encargado de enviar la información recogida por los sensores. Esta información es la que se añade al campo datos del paquete y, en este caso, consiste en valores arbitrarios pues el algoritmo de detección no utiliza la información del campo de datos. El paquete creado es enviado a una de las direcciones IP pasadas como parámetro en el documento de configuración que se mostrará más adelante y al respectivo puerto también indicado en el documento.

4.1.2. Escenario 1

En la Figura 23 se observa el primer escenario. Como se ha indicado previamente, se trata de una red de sensores inalámbrica centralizada que sigue una topología de red en estrella.

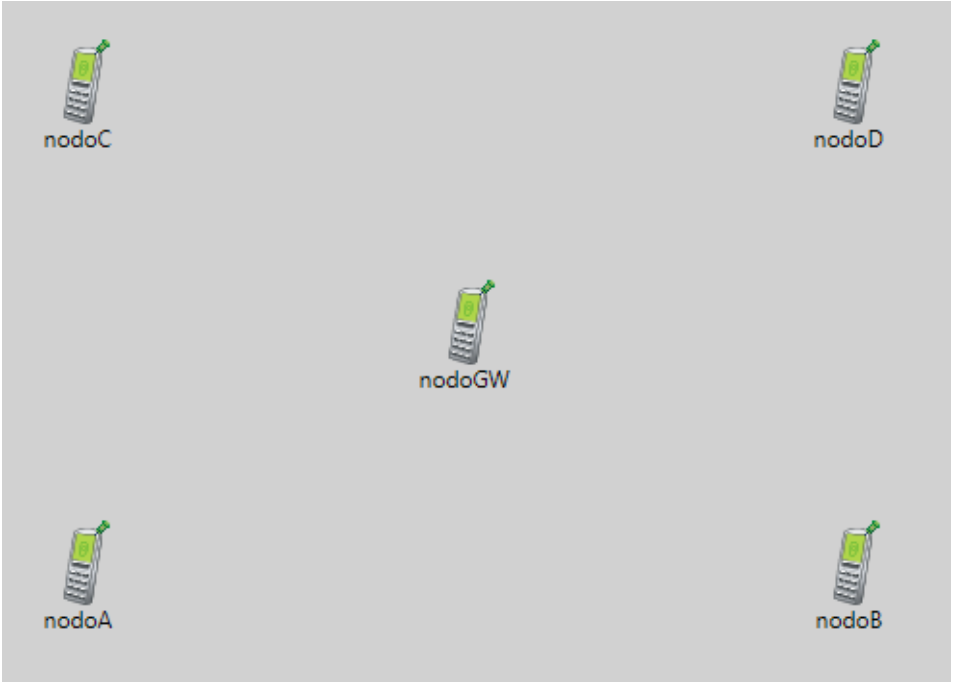

Figura 23. Escenario 1

El código NED utilizado para crear el primer escenario se muestra en el Fragmento de código 2.

Fragmento de código 2. Descripción de red del Escenario 1

```
[…]
network Escenario1
{
    submodules:
        […]
        nodoGW: AdhocHost {…}
        nodoA: AdhocHost {…}
        nodoB: AdhocHost {…}
        nodoC: AdhocHost {…}
        nodoD: AdhocHost {…}
}
```

En el Fragmento de código 2 se observan los cinco nodos que forman la red del Escenario 1: cuatro nodos sensores y un nodo pasarela, todos ellos dotados de capacidad de comunicación inalámbrica. *AdhocHost* es el objeto que representa un nodo inalámbrico en el simulador OMNeT++.

En este escenario cuatro nodos sensores intercambian información con el nodo pasarela. Todos los dispositivos IoT se encuentran en la misma subred, la subred 10.0.0.0/29. Los nodos sensores transmiten los paquetes de datos a la dirección IP 10.0.0.1 correspondiente al nodo central y al puerto 5000, en el cual escucha dicho nodo.

Cuando se inicia la simulación, los nodos intercambian información de control ARP para resolver las direcciones IP correspondientes. Tras ello, comienzan a transmitir los nodos sensores información hacia el nodo central. El contenido de los paquetes de datos es arbitrario,

pues en caso usar cifrado en las comunicaciones, el contenido no sería interpretable sin procesarse previamente.

El caso de estudio del Escenario 1 consiste, como se puede ver en la Figura 24, en utilizar el sistema de detección cuando un nodo malicioso entra en la red y comienza a transmitir paquetes de forma maliciosa al resto de nodos.

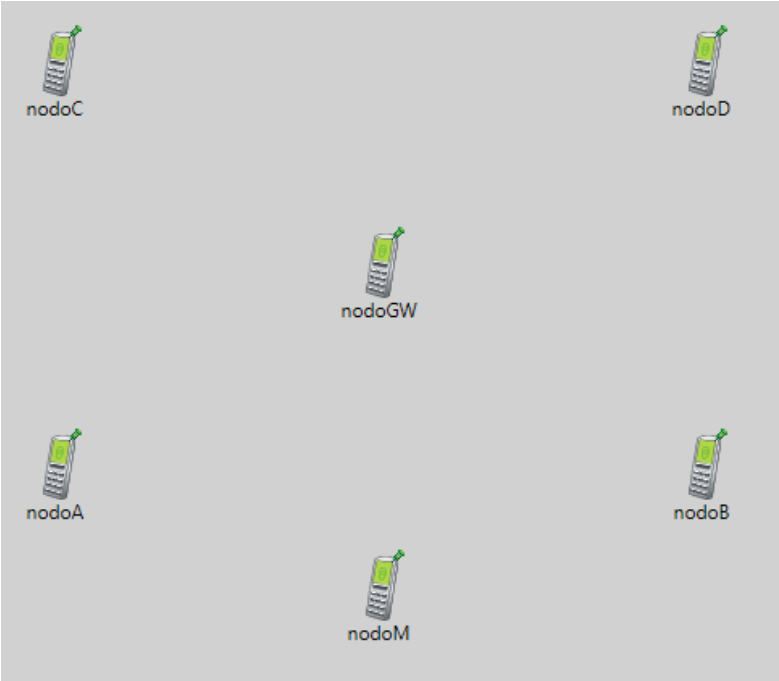

Figura 24. Escenario 1 malicioso

El código NED necesario para añadir el nodo malicioso es el mostrado en el Fragmento de código 3.

Fragmento de código 3. Descripción de red del Escenario 1 con nodo malicioso

```
[…]
network Escenario1M extends Escenario1
{
    submodules:
        nodoM: AdhocHost {…}
}
```

En el Fragmento de código 3 se extiende la descripción mostrada en el Fragmento de código 2 añadiendo un sexto nodo inalámbrico. Esto es posible debido a la capacidad de herencia que provee el lenguaje de descripción.

En este escenario, el nodo malicioso transmite paquetes a todos los nodos, omitiendo la restricción de la red centralizada, y lo hace siguiendo una variable aleatoria con función de distribución exponencial de 500 milisegundos. En este caso se estudia el sistema de detección en una situación en la cual un nodo intenta atacar a la red o actúa erróneamente de forma no intencionada por una avería.

En el Fragmento de código 4 se muestra el archivo de configuración del escenario normal y en el Fragmento de código 5 se muestra el correspondiente al escenario malicioso.

Fragmento de código 4. Archivo de configuración del Escenario 1

```
[Config escenario1]
# SIM
network = Escenario1
[…]
# NodoGW
*.nodoGW.numApps = 1
*.nodoGW.app[0].typename = "UdpSink"
*.nodoGW.app[0].localPort = 5000
# NodoA
*.nodoA.numApps = 1
*.nodoA.app[0].typename = "AppSensor"
*.nodoA.app[0].destAddresses = "nodoGW"
*.nodoA.app[0].destPort = 5000
*.nodoA.app[0].messageLength = 3B
*.nodoA.app[0].sendInterval = uniform(1s,2s)
*.nodoA.app[0].packetName = "PktNodoA"
[…]
```

Fragmento de código 5. Archivo de configuración del Escenario 1 con nodo malicioso

```
[Config escenario1M]
extends = escenario1
# SIM
network = Escenario1M
[…]
# L4 - NodoM
*.nodoM.numApps = 1
*.nodoM.app[0].typename = "AppSensorM"
*.nodoM.app[0].destAddresses = "nodoGW nodoA nodoB nodoC nodoD"
*.nodoM.app[0].destPort = 5000
*.nodoM.app[0].messageLength = 3B
*.nodoM.app[0].sendInterval = exponential(500ms)
*.nodoM.app[0].packetName = "PktNodoM"
[…]
```

En el Fragmento de código 4 se observa la asignación de las diferentes aplicaciones a los nodos de la red. Cada aplicación requiere de parámetros distintos: la aplicación *UdpSink* únicamente necesita el puerto local donde recibir los paquetes mientras que la aplicación *AppSensor* requiere de las direcciones destino, el puerto destino, la longitud del campo de datos del paquete y el intervalo de transmisión.

En el Fragmento de código 5 se extiende el fragmento anterior con la asignación de la aplicación maliciosa al nodo fraudulento. Esta aplicación funciona del mismo modo que la que implementan los nodos sensores normales. En este caso, el nodo malicioso recibe como direcciones destino todas las posibles en el escenario y posee un intervalo de transmisión diferente al resto de nodos sensores.

4.1.3. Escenario 2

El Escenario 2 se muestra en la Figura 25 y coincide con el Escenario 1.

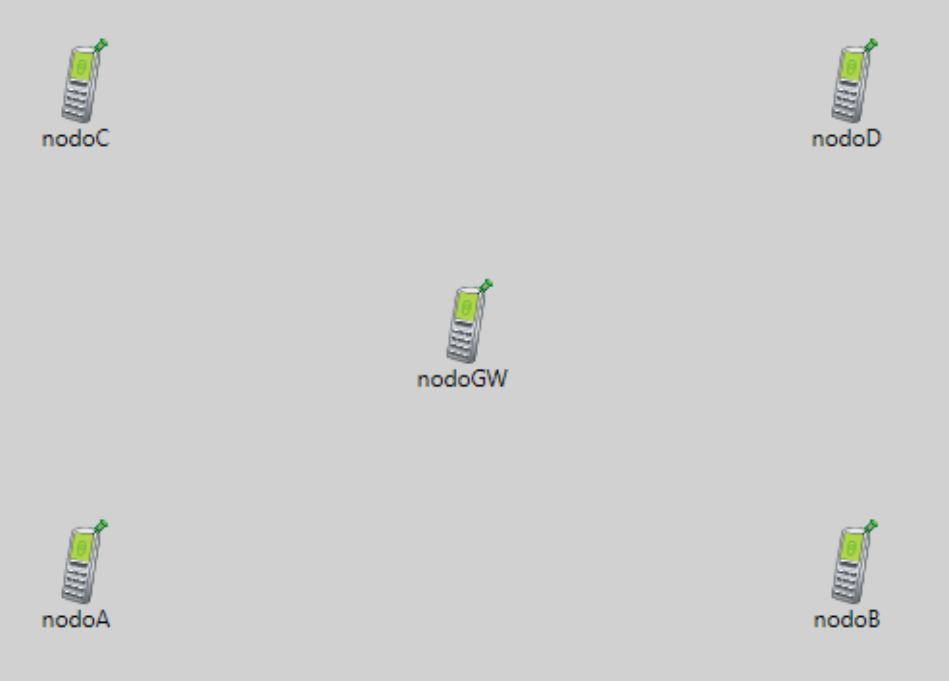

Figura 25. Escenario 2

En este segundo escenario, ilustrado en la Figura 26, se tiene como objetivo detectar un ataque de suplantación de identidad o *spoofing* añadiendo un nodo malicioso que suplante a un nodo sensor.

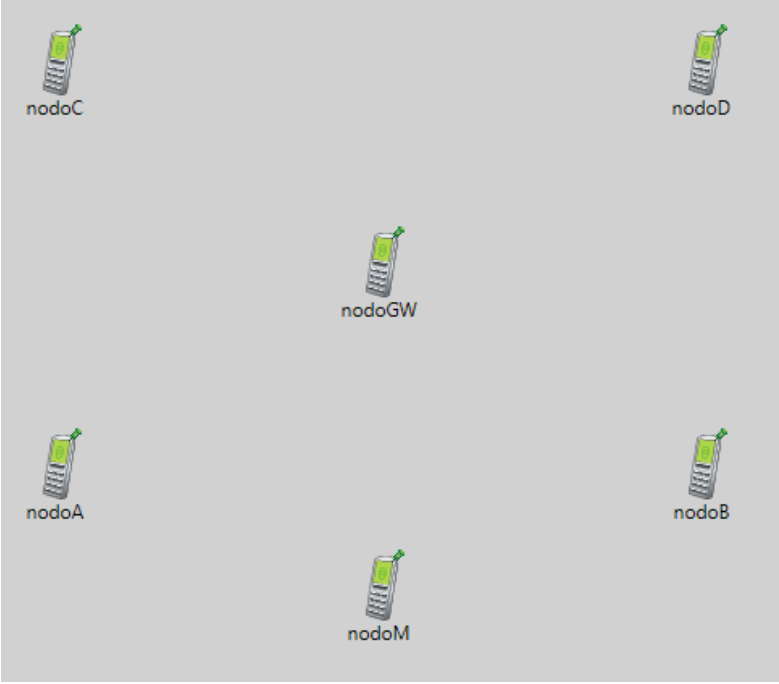

Figura 26. Escenario 2 malicioso

En este caso de estudio, el nodo atacante transmite paquetes a los nodos haciéndose pasar por otro nodo sensor. Concretamente, suplanta la dirección IP 10.0.0.2 del nodo sensor A. En este escenario se estudia el sistema de detección en una situación clara de ataque a la red por parte de un nodo malicioso.

En este segundo escenario, la descripción de la red en lenguaje NED y el comportamiento de los nodos en lenguaje C++ son los mismos que en el escenario anterior. En cambio, el archivo de configuración del escenario malicioso es distinto y se muestra en el Fragmento de código 6.

Fragmento de código 6. Archivo de configuración del Escenario 2 con nodo malicioso

```
[Config escenario2M]
extends = escenario1M
# SIM
network = Escenario1M
[…]
# Direcciones IP
*.configurator.assignUniqueAddresses = false
*.configurator.dumpAddresses = true
*.configurator.config = xml(
   "<config>
        <interface hosts='nodoGW' names='wlan0' address='10.0.0.1'/>
        <interface hosts='nodoA' names='wlan0' address='10.0.0.2'/>
        <interface hosts='nodoB' names='wlan0' address='10.0.0.3'/>
        <interface hosts='nodoM' names='wlan0' address='10.0.0.2'/>
        <interface hosts='**' address='10.0.0.x' netmask='255.255.255.248'/>
   </config>")
[…]
```

El segundo escenario malicioso mostrado en el Fragmento de código 6 extiende al primer escenario con nodo fraudulento configurando de forma manual la asignación de direcciones IP de los nodos de la red de sensores. En concreto, la misma dirección IP 10.0.0.2 es asignada tanto al primer nodo sensor como al nodo malicioso.

4.1.4. Escenario 3

El tercer escenario se muestra en la Figura 27. En este caso el escenario trata de nuevo de una red de sensores inalámbrica centralizada pero esta vez la tipología de la red es de tipo árbol.

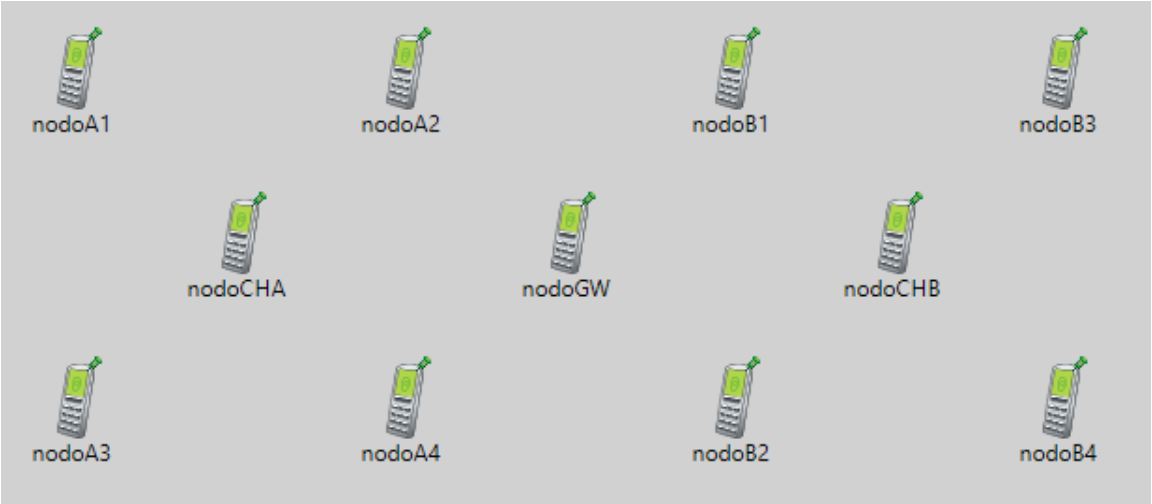

Figura 27. Escenario 3

La topología en árbol se construye usando dos nodos *Cluster Head* que encaminan los paquetes recibidos al nodo central. Los nodos sensores intercambian información únicamente con su correspondiente nodo *Cluster Head*. De esta forma, en este escenario se han creado diferentes subredes: la subred 10.1.1.0/29 para la agrupación A y la subred 10.2.1.0/29 para la agrupación B. Además, cada nodo *Cluster Head* se comunica con el nodo pasarela a través de un enlace: el enlace 10.1.0.0/30 para el nodo *Cluster Head* A y el nodo pasarela y el enlace 10.2.0.0/30 para el nodo *Cluster Head* B y el nodo central.

La descripción de la red del escenario se expone en el Fragmento de código 7 usando el lenguaje de descripción NED.

Fragmento de código 7. Descripción de red del Escenario 3

```
[…]
network Escenario3 {
    submodules:
        […]
        // Nodo Pasarela GW
        nodoGW: AdhocHost {…}
        // Nodos ClusterHead-A CHA
        nodoCHA: AdhocHost {…}
        nodoA1: AdhocHost {…}
        […]
        // Nodos ClusterHead-B CHB
        nodoCHB: AdhocHost {…}
        nodoB1: AdhocHost {…}
        […]
}
```

Del mismo modo que ocurría en el archivo de descripción de red de los primeros escenarios, en el Fragmento de código 7 se añaden a la red los correspondientes nodos inalámbricos. Esta vez, se añaden un nodo pasarela y dos nodos encaminadores que centralizan el tráfico de paquetes de cuatro nodos sensores cada uno.

Para que sea posible la jerarquización de las subredes y la organización del tráfico es necesario que los nodos *Cluster Head* dispongan de dos interfaces de red inalámbricas al igual

que el nodo pasarela. Adicionalmente, los nodos encaminadores de los paquetes disponen de una aplicación que transmite todos los mensajes que recibe de los nodos sensores hacia el nodo central. El comportamiento de dicha aplicación se muestra en el Fragmento de código 8.

Fragmento de código 8. Aplicación nodo encaminador

```
[…]
class INET_API AppClusterHead : public ApplicationBase, public
UdpSocket::ICallback {
  […]
  protected:
    virtual void socketDataArrive(UdpSocket *socket, Packet *packet) override;
  […]
}
[…]
void AppClusterHead::socketDataArrived(UdpSocket *socket, Packet *pk) {
    auto data = pk->peekData();
    delete pk;
    Packet *packet = new Packet("DatosCH", data);
    […]
    int destPort = par("destPort");
    L3Address destAddr = L3AddressResolver().resolve(par("destAddr"), 27);
    socket->sendTo(packet, destAddr, destPort);
}
```

En el Fragmento de código 8 se puede observar el método más importante de la aplicación que dota de comportamiento a los nodos encaminadores. El método *socketDataArrived()* tras recibir un paquete, extrae la información transportada en el campo de datos y crea un nuevo paquete con dicha información que transmite al nodo pasarela. La dirección y el puerto destino la recibe como parámetro en el archivo de configuración.

De acuerdo con lo expuesto en los párrafos anteriores, cualquier comunicación entre nodos sensores o entre nodo sensor y nodo pasarela no está permitida y se consideraría un comportamiento anómalo, es decir, un potencial ataque. En este caso de estudio, se busca utilizar el sistema de detección para determinar si se está produciendo un ataque en la red del tipo descrito.

El archivo de configuración del tercer escenario se muestra en el Fragmento de código 9 y seguidamente, en el Fragmento de código 10, el documento XML (*eXtensible Markup Language*) referente a la topología de la red.

Fragmento de código 9. Archivo de configuración del Escenario 3

```
[Config escenario3]
# SIM
network = Escenario3
[…]
# NodoGW
*.nodoGW.numWlanInterfaces = 2
*.nodoGW.numApps = 1
*.nodoGW.app[0].typename = "UdpSink"
*.nodoGW.app[0].localPort = 1000
[…]
# Nodo CHA
*.nodoCHA.numWlanInterfaces = 2
*.nodoCHA.numApps = 1
*.nodoCHA.app[0].typename = "AppClusterHead"
*.nodoCHA.app[0].destAddr = "10.1.0.1" # Nodo GW
*.nodoCHA.app[0].localPort = 1100
*.nodoCHA.app[0].destPort = 1000
*.nodoCHA.app[0].messageLength = 3B
*.nodoCHA.app[0].sendInterval = uniform(1s,2s)
*.nodoCHA.app[0].packetName = "PktNodoCHA"
[…]
# Nodos CHA
*.nodoA*.numApps = 1
*.nodoA*.app[0].typename = "AppSensor"
*.nodoA*.app[0].destAddresses = "10.1.1.1" # Nodo CHA
*.nodoA*.app[0].destPort = 1100
*.nodoA*.app[0].messageLength = 3B
*.nodoA*.app[0].sendInterval = uniform(1s,2s)
*.nodoA*.app[0].packetName = "PktNodoA"
[…]
# IP
*.configurator.dumpAddresses = true
*.configurator.config = xmldoc("topologia.xml")
[…]
```

Fragmento de código 10. Documento XML de configuración de la topología de red del Escenario 3

```
<config>
            <!-- Enlace GW-CHA -->
      <interface hosts='nodoGW' names='wlan0' address='10.1.0.1'/>
      <interface hosts='nodoCHA' names='wlan0' address='10.1.0.2'/>

            <!-- Enlace GW-CHA -->
      <interface hosts='nodoGW' names='wlan1' address='10.2.0.1'/>
      <interface hosts='nodoCHB' names='wlan0' address='10.2.0.2'/>

            <!-- Subred CHA -->
      <interface hosts='nodoCHA' names='wlan1' address='10.1.1.1'/>
      <interface hosts='nodoA1' names='wlan0' address='10.1.1.2'/>
      <interface hosts='nodoA2' names='wlan0' address='10.1.1.3'/>
      <interface hosts='nodoA3' names='wlan0' address='10.1.1.4'/>
      <interface hosts='nodoA4' names='wlan0' address='10.1.1.5'/>

            <!-- Subred CHB -->
      <interface hosts='nodoCHB' names='wlan1' address='10.2.1.1'/>
      <interface hosts='nodoB1' names='wlan0' address='10.2.1.2'/>
      <interface hosts='nodoB2' names='wlan0' address='10.2.1.3'/>
      <interface hosts='nodoB3' names='wlan0' address='10.2.1.4'/>
      <interface hosts='nodoB4' names='wlan0' address='10.2.1.5'/>
</config>
```

En el Fragmento de código 9 se realiza la asignación de aplicaciones a los nodos del mismo modo que ocurría en los escenarios anteriores. En esta asignación se añade la correspondiente al nodo encaminador que requiere los parámetros relativos a la recepción de paquetes como en la aplicación *UdpSink* y los referentes a la transmisión como en la aplicación *AppSensor*.

Además, se configura la topología de red usando el documento XML mostrado en el Fragmento de código 10, En él se asignan las direcciones IP correspondientes a los nodos para lograr así una topología de red en árbol.

Por último, en el Fragmento de código 11 se expone el archivo de configuración del tercer escenario malicioso, donde uno de los nodos sensores trata de comunicarse con el nodo pasarela directamente.

Fragmento de código 11.Archivo de configuración del Escenario 3 con nodo malicioso.

```
[Config escenario3M]
extends = escenario3
[…]
# Nodo M
*.nodoA2.numApps = 1
*.nodoA2.app[0].typename = "AppSensorM"
*.nodoA2.app[0].destAddresses = "10.1.0.1 10.2.0.1" # Nodo GW
*.nodoA2.app[0].destPort = 1000
*.nodoA2.app[0].messageLength = 3B
*.nodoA2.app[0].sendInterval = uniform(1s,2s)
*.nodoA2.app[0].packetName = "PktNodoM"
```

En el Fragmento de código 11 donde se configura el nodo malicioso del último escenario se asignan como direcciones destino las correspondientes a las interfaces de red del nodo pasarela. Esta condición omite la regla de centralización del tráfico por parte de los encaminadores al no comunicarse con el nodo *Cluster Head* que le correspondería. Respecto a los parámetros, actúan como en el resto de escenarios.

4.2. Autoencoder

Como se explicó en el capítulo anterior, el mecanismo de aprendizaje automático utilizado por el sistema de detección es el codificador automático. La red neuronal artificial recibe a la entrada la información transportada en la cabecera de los protocolos de los paquetes transmitidos en la red de sensores, la comprime a un espacio latente e intenta reconstruir la información de entrada a partir de este.

La información de entrada que procesa la red se muestra en la Tabla 6.

Tabla 6. Información de entrada procesada por el autoencoder

Dirección física del nodo receptor	Dirección física del nodo transmisor
Primer octeto de la dirección IP destino	Primer octeto de la dirección IP origen
Segundo octeto de la dirección IP destino	Segundo octeto de la dirección IP origen
Tercer octeto de la dirección IP destino	Tercer octeto de la dirección IP origen
Cuarto octeto de la dirección IP destino	Cuarto octeto de la dirección IP origen
Puerto destino	Puerto origen
Tamaño del paquete	Diferencia de tiempo con el paquete anterior

Por tanto, el codificador automático recibe catorce características de entrada de las cuales ninguna es el campo de datos del paquete procesado. Por este motivo, si la información fuese cifrada el algoritmo seguiría siendo válido o si el tamaño de los paquetes fuese distinto también seguiría siendo válido.

La topología de la red neuronal artificial implementada consiste en una capa de entrada de catorce neuronas artificiales y dos capas ocultas de doce y diez neuronas artificiales respectivamente en la etapa de codificación. El espacio latente resultante es de dimensión diez. La etapa de decodificación está formada por dos capas ocultas de diez y doce neuronas artificiales respectivamente y una capa de salida de catorce neuronas artificiales nuevamente. Esta topología se muestra en el resumen obtenido al ejecutar la red neuronal de la Figura 28.

```
Model: "Autoencoder"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(1, 14)]                 0
_________________________________________________________________
dense (Dense)                (1, 12)                   180
_________________________________________________________________
dense_1 (Dense)              (1, 10)                   130
_________________________________________________________________
dense_2 (Dense)              (1, 12)                   132
_________________________________________________________________
dense_3 (Dense)              (1, 14)                   182
=================================================================
Total params: 624
Trainable params: 624
Non-trainable params: 0
```

Figura 28. Topología del codificador automático

Como se puede observar, el *autoencoder* dispone de seiscientos veinticuatro parámetros ajustables los cuales se corresponden con los pesos *w* y los sesgos *b* de las neuronas artificiales que lo componen. Por ejemplo, los primeros ciento ochenta parámetros se corresponden con los catorce pesos de las doce neuronas junto con los doce sesgos correspondientes.

El codificador automático ha sido desarrollado en el entorno de desarrollo integrado *PyCharm* en el lenguaje de programación Python. El software desarrollado hace uso de las librerías *numpy* y *tensorflow* para implementar el *autoencoder* y de la librería *scapy* para la lectura de las capturas PCAP. En el cuadro siguiente se muestra el código referente a la red neuronal artificial implementada.

Fragmento de código 12. Implementación del codificador automático

```
[…]
# Entrada
entrada = tf.keras.Input(shape=14, batch_size=1)
# Codificador
(x) = Dense(units=12, use_bias=True, activation='tanh') (entrada)
# Espacio latente
(x) = Dense(units=10, use_bias=True, activation='relu') (x)
# Decodificador
(x) = Dense(units=12, use_bias=True, activation='tanh') (x)
# Salida
salida = Dense(units=14, use_bias=True, activation='relu') (x)
# Construcción
self.modelo = tf.keras.Model(name='Autoencoder', inputs=entrada, outputs=salida)
self.modelo.compile(
optimizer=tf.keras.optimizers.SGD(learning_rate=0.001, momentum=0.5), loss='mse')
[…]
```

Como se puede observar en el Fragmento de código 12, la red neuronal artificial implementada utiliza neuronas artificiales de la clase *Dense* de la librería *tensorflow*, las cuales se corresponden con las explicadas anteriormente. Todas las neuronas utilizan sesgo en su función lineal y las funciones rectificador y tangente hiperbólica como funciones de activación. El método de propagación hacia atrás se configura con un *learning rate* de 0.001 y un *momentum* de 0.5. Estos valores representan la velocidad a la que la red neuronal artificial aprende y cuánto influye el valor anterior en la actualización de los pesos y los sesgos. Además, el método utilizado para minimizar la función de pérdida es el método del descenso del gradiente que utiliza las derivadas parciales para ajustar los parámetros del codificador automático. La función de pérdida escogida es el error cuadrático medio.

4.3. Integración del sistema de detección y los escenarios

Para finalizar la implementación de la solución de seguridad es necesario integrar el sistema de detección con la red de sensores inalámbrica. En este apartado se estudiará dicha integración.

Como se ha visto anteriormente, el nodo pasarela se comunica con el sistema de detección mediante el intercambio de capturas de tráfico PCAP. El sistema de detección lee dichas capturas y procesa los paquetes.

El procesamiento de los paquetes intercambiados en la red no puede ser directamente realizado por el codificador automático. El paso previo al procesado es el proceso conocido como extracción de características.

La extracción de características consiste, a grandes rasgos, en adaptar la información con la que se trabaja para que la red neuronal artificial sea capaz de procesarla. En este caso, en primer lugar, hay que leer la captura PCAP pertinente. Para ello, se hace uso de la librería *scapy* ya que permite leer la captura PCAP paquete a paquete. En segundo lugar, gracias a la misma herramienta, se convierten los paquetes leídos en paquetes de la clase *Dot11* de la

librería *scapy*. Una vez formateados es posible extraer la información del paquete. En este momento, se crea un vector de catorce dimensiones y se le asignan los valores mencionados en el apartado 4.2 extraídos del paquete de clase *Dot11*. Este vector es la entrada del *autoencoder*.

Las redes neuronales artificiales requieren de dos etapas. Una primera etapa en la cual se ajustan los parámetros y una segunda en la cual se validan los resultados. El sistema de detección primero escucha y aprende del funcionamiento normal de la red hasta ajustar los parámetros de forma satisfactoria. Una vez terminado el entrenamiento, escucha el funcionamiento completo de la red de sensores y detecta los posibles ataques cuando ocurren comportamientos maliciosos. El método para validar los resultados durante el entrenamiento consiste en procesar un conjunto de datos de tráfico fraudulento y comprobar la capacidad del sistema para detectarlos.

5. Validación y resultados

Tras haber diseñado e implementado el sistema de detección, el siguiente paso es comprobar su rendimiento y, para ello, se han llevado a cabo una serie de pruebas que se expondrán en este capítulo.

5.1. Descripción de las pruebas realizadas

Antes de comenzar con la descripción de las pruebas realizadas es importante destacar el tipo de problema que se trata de resolver. El sistema de detección tiene como objetivo clasificar los paquetes intercambiados en la red de sensores inalámbrica. El sistema se encarga de determinar si un paquete es malicioso o no.

Debido a la naturaleza del problema, el mecanismo de evaluación consistirá en la construcción de la matriz de confusión del algoritmo de detección. Gracias a ella es posible determinar la precisión del clasificador, así como la exhaustividad y la exactitud. Para ello es necesario extraer las métricas que se explican a continuación.

- Verdaderos negativos (VN)

  Se considera verdadero negativo aquel resultado negativo interpretado como negativo. En el caso de estudio, el sistema de detección al analizar un paquete no malicioso no determina que es malicioso.

- Verdaderos positivos (VP)

  Cuando el sistema de detección analiza un paquete malicioso y determina que es un paquete fraudulento, el resultado es considerado como verdadero positivo. Es decir, cuando el resultado es positivo y se interpreta como tal, el resultado es verdadero positivo.

- Falsos negativos (FN)

  Un resultado falso negativo consiste en un resultado que siendo positivo ha sido interpretado como negativo. En otras palabras, un resultado falso negativo es, en el caso de estudio, un paquete malicioso que ha sido capaz de evadir el sistema de detección.

- Falsos positivos (FP)

  Los resultados falsos positivos son conocidos también como falsas alarmas. Consisten en resultados negativos que han sido clasificados como positivos. Hablando de paquetes intercambiados, un paquete no malicioso interpretado por el sistema de detección como un posible ataque.

Una vez obtenidas las métricas recién expuestas, es posible construir la matriz de confusión del algoritmo de detección y con ella calcular el rendimiento del sistema de detección. Para estudiar la capacidad de detección del sistema, los valores calculados son los siguientes.

- Precisión

  La precisión del algoritmo de detección consiste en la relación de resultados verdaderos positivos y el total de resultados determinados como positivos.

$$Precisión = \frac{VP}{VP + FP}$$

- Exhaustividad

  Un algoritmo de detección es exhaustivo cuando la relación resultados verdaderos positivos y total de resultados positivos se aproxima a la unidad.

$$Exhaustividad = \frac{VP}{VP + FN}$$

- Exactitud

  La exactitud del algoritmo de detección es utilizada para medir el grado de acierto del mismo. Consiste en la relación de resultados interpretados correctamente y el número total de resultados.

$$Exactitud = \frac{VP + VN}{VP + FP + VN + FN}$$

Respecto a las pruebas realizadas y de acuerdo a lo expuesto en los capítulos anteriores, el sistema de detección se compone de dos etapas. Una primera etapa de entrenamiento y una segunda de validación.

La etapa de entrenamiento del sistema de detección hace uso del tráfico intercambiado durante una hora en la red de sensores sin dispositivos IoT en ella maliciosos para aprender el funcionamiento normal de la misma. Según el escenario estudiado, la cantidad de información utilizada para el entrenamiento varía. En los dos primeros escenarios, cuatro nodos sensores transmiten información en intervalos gobernados por una variable aleatoria que sigue una función de distribución uniforme entre uno y dos segundos. Mientras que en el tercer escenario el número de nodos sensores asciende a ocho sumados a los dos nodos encaminadores gobernados por una variable aleatoria similar. Por tanto, la información utilizada en la etapa de entrenamiento del sistema de detección es menor en los dos primeros escenarios respecto a la utilizada en el tercer escenario.

La etapa de validación del algoritmo consiste en evaluar el rendimiento del sistema, extrayendo los valores expuestos anteriormente en este capítulo, utilizando muestras de cinco minutos del tráfico de paquetes intercambiados. Estas muestras son tomadas de la red de sensores inalámbrica con el nodo malicioso en ella. De igual modo que en la etapa anterior, la información utilizada para evaluar el rendimiento del sistema de detección es mayor en el tercer escenario que en el resto de escenarios.

Para llevar a cabo las distintas pruebas, se ha simulado en el programa OMNeT++ la red de dispositivos IoT de cada escenario dos veces. Una primera vez para obtener la captura PCAP del tráfico de paquetes intercambiados durante el funcionamiento normal de la red y una segunda vez para obtener la captura PCAP relativa al funcionamiento fraudulento para

la validación del algoritmo de detección. La duración de la primera simulación es de tres mil seiscientos segundos mientras que la duración de la segunda es de únicamente trescientos segundos.

De acuerdo con los capítulos anteriores, las capturas PCAP del tráfico de paquetes intercambiados son interpretadas previamente para su posterior análisis. Por tanto, de dichas capturas PCAP se extraen las características de cada paquete y se almacenan en un documento .csv el cual procesará el codificador automático.

5.2. Prueba del Escenario 1

El primer escenario consiste en una red de sensores inalámbrica centralizada con topología de estrella. En ella se introduce un nodo malicioso el cual transmite a una tasa distinta al resto de sensores y omite la regla de centralización del tráfico comunicándose con el resto de nodos sensores.

Para realizar la prueba de rendimiento del sistema de detección en este escenario se capturaron diecinueve mil quinientos cuarenta y nueve paquetes durante el funcionamiento normal de la red para la etapa de entrenamiento de la red. Respecto a la etapa de validación, se capturaron tres mil setecientos sesenta y un paquetes durante el funcionamiento fraudulento de la red de los cuales mil seiscientos sesenta y cuatro eran paquetes maliciosos.

La prueba del primer escenario duró una hora, siete minutos y veinticinco segundos y la matriz de confusión obtenida se muestra en la Tabla 7.

Tabla 7. Matriz de confusión del Escenario 1

		Interpretación	
		Negativos	**Positivos**
Realidad	**Negativos**	2072	25
	Positivos	589	1075

A través de las métricas expuestas en la matriz de confusión de la Tabla 7, es posible determinar el rendimiento del algoritmo. A continuación, se muestran los resultados de la primera prueba.

$$Precisión = 97{,}72\ \%$$

$$Exhaustividad = 64{,}60\ \%$$

$$Exactitud = 83{,}67\ \%$$

Prácticamente el noventa y ocho por ciento de los paquetes que no son fraudulentos son interpretados de forma correcta, lo cual se traduce en un índice bajo de falsas alarmas. Respecto a la capacidad del sistema de detectar paquetes maliciosos en la red, más del sesenta

por cierto de los paquetes fraudulentos son detectados por el algoritmo. Como resultado final, la exactitud del sistema de detección supera el ochenta por ciento, es decir, tres mil ciento cuarenta y siete paquetes han sido analizados de forma correcta de tres mil setecientos sesenta y un paquetes totales.

5.2.1. Resultado de la validación del Escenario 1

El resultado ideal de la validación de un clasificador consistiría en no obtener en las pruebas ni falsos positivos ni falsos negativos, es decir, obtener una exactitud del cien por ciento. Debido a que la situación ideal no es realista es importante estudiar tanto la precisión como la exhaustividad del sistema de detección.

La precisión, como se expone en los apartados anteriores, determina el número de falsas alarmas. En otras palabras, un valor de precisión próximo al cien por cien se traduce en un bajo número de falsas alarmas o falsos positivos. Maximizar este resultado es interesante de cara a obtener un buen rendimiento.

Sin embargo, el resultado más importante en un sistema de detección de ataques es la exhaustividad, pues el principal objetivo es detectar el mayor número posible de amenazas. Por tanto, el valor que interesa maximizar en este proyecto es la exhaustividad.

En la prueba del primer escenario se ha obtenido un sesenta y cuatro como seis por ciento de exhaustividad lo cual implica que más de la mitad de las amenazas son detectadas con éxito y, además, al estar el valor de la precisión muy próximo al cien por ciento, cuando se produce una alarma en el sistema de detección es altamente probable que la haya producido un paquete fraudulento. Resumiendo, los resultados obtenidos exponen que más del ochenta por ciento de los paquetes son interpretados de forma correcta y que menos del tres por ciento de las alarmas no son justificadas por potenciales ataques, es decir, solamente el tres por ciento de las alertas son falsas alarmas.

5.3. Prueba del Escenario 2

El Escenario 2 mantiene la misma topología que el primer escenario. Sin embargo, en el segundo se trata de detectar un ataque de suplantación de identidad en el cual el nodo malicioso se hace pasar por un nodo sensor.

En este caso de estudio, para la etapa de entrenamiento se han capturado diecinueve mil quinientos cuarenta y ocho paquetes durante la simulación de la red de sensores sin el nodo malicioso. Y, para la etapa de validación, tres mil doscientos sesenta y nueve paquetes entre los cuales mil doscientos noventa eran transmitidos o recibidos por el nodo fraudulento.

La prueba del Escenario 2 tuvo una duración de una hora, 40 minutos y 23 segundos. Los resultados obtenidos se exponen la matriz de confusión de la Tabla 8.

Tabla 8. Matriz de confusión del Escenario 2

		Interpretación	
		Negativos	**Positivos**
Realidad	**Negativos**	1953	25
	Positivos	463	827

Con los resultados obtenidos, los valores indicadores del rendimiento del sistema de detección se muestran seguidamente.

$$Precisión = 97{,}06\ \%$$

$$Exhaustividad = 64{,}10\ \%$$

$$Exactitud = 85{,}06\ \%$$

De forma similar al primer escenario, la precisión del algoritmo de detección supera el noventa y siete por ciento, lo cual implica que el número de falsas alarmas es prácticamente insignificante. Además, el sistema es capaz de detectar más del sesenta y cuatro por ciento de los paquetes maliciosos intercambiados en la red de sensores inalámbrica, dando como resultado un ochenta y cinco por ciento de interpretación correcta de los paquetes analizados.

Frente a un ataque de suplantación de identidad, el ochenta y cinco por ciento de los paquetes analizados han sido interpretados de forma correcta. Es decir, dos mil setecientos ochenta paquetes han sido interpretados sin error de tres mil doscientos sesenta y ocho.

5.3.1. Resultado de la validación del Escenario 2

Al tratarse el Escenario 2 de un ataque de suplantación de identidad, los resultados esperados no eran tan altos como en el primer escenario. Sin embargo, la validación del segundo escenario ha superado con creces las expectativas.

En la validación del Escenario 2 se ha obtenido de nuevo un valor de precisión superior al noventa y siete por ciento. El número de falsos positivos es mínimo al igual que ocurría en el escenario anterior, lo cual se traduce en una mayor confianza en las alarmas. Cuando se producen alertas, sólo veinticinco de las ochocientas cincuenta y dos son falsas alarmas.

Respecto a la exhaustividad del algoritmo de detección en el segundo escenario, prácticamente el sesenta y cinco por ciento de los paquetes fraudulentos son detectados. Por tanto, más de la mitad de los potenciales ataques son detectados y, junto al valor de precisión, otorga al sistema de detección una exactitud superior al ochenta y cinco por ciento.

El sistema de detección frente a un ataque de suplantación de identidad en una red de sensores inalámbrica es capaz de alertar de forma correcta más del sesenta por ciento de los

paquetes fraudulentos relacionados y hacerlo con la seguridad de que menos del tres por ciento de las alertas son falsas.

5.4. Prueba del Escenario 3

Por último, el tercer escenario tiene como objetivo comprobar el funcionamiento del sistema de detección en una topología de red distinta. En este caso, la topología de la red de dispositivos IoT es de tipo árbol y se caracteriza por la distribución jerárquica del tráfico: los nodos sensores se comunican con el nodo encaminador y este con el nodo pasarela, no estando permitido el intercambio de información entre los nodos sensores y el nodo central.

En este caso de estudio, el número de nodos es mayor que en el resto de escenario, por tanto, la información utilizada para las etapas de entrenamiento y validación es también mayor. Concretamente, para la etapa de entrenamiento del sistema de detección se han capturado ciento cincuenta y cuatro mil setecientos sesenta y ocho paquetes durante la simulación de la red de sensores con un comportamiento normal y doce mil ciento cuarenta paquetes para la etapa de validación del algoritmo. De los paquetes capturados para evaluar el sistema, únicamente cuatrocientos veinte paquetes son fraudulentos. Esto se debe a que esta vez el nodo malicioso no es añadido si no que se sustituye un nodo sensor por él.

Los resultados de la tercera prueba, de cuarenta y dos minutos y veinticuatro segundos de duración, se muestran en la Tabla 9.

Tabla 9. Matriz de confusión del Escenario 3

		Interpretación	
		Negativos	**Positivos**
Realidad	**Negativos**	8497	3223
	Positivos	12	408

En el Escenario 3 los valores obtenidos en la matriz de confusión de la Tabla 9 permiten calcular los siguientes valores de evaluación del rendimiento del sistema de detección.

$$Precisión = 11{,}23\ \%$$

$$Exhaustividad = 97{,}14\ \%$$

$$Exactitud = 73{,}35\ \%$$

Los resultados obtenidos en la tercera prueba no son similares a los obtenidos en las pruebas anteriores. En esta ocasión, la exhaustividad del algoritmo es el punto a destacar pues supera el noventa y siete por ciento mientras que el número de falsas alarmas es muy elevado respecto a los verdaderos positivos. De ahí que el valor de precisión sea poco mayor del diez

por ciento. Sin embargo, la capacidad del sistema de detección de interpretar de forma correcta los paquetes analizados es positiva. El algoritmo interpreta correctamente más del setenta y tres por ciento de los paquetes intercambiados en la red de sensores.

5.4.1. Resultado de la validación del Escenario 3

El objetivo de la prueba de validación del tercer escenario era comprobar el rendimiento del sistema de detección en una red con una topología diferente a las anteriores. Los resultados obtenidos han superado las expectativas pues observando los resultados de las dos primeras pruebas, el obtenido en este escenario difiere totalmente.

En la validación del Escenario 3 se ha obtenido un valor de exactitud inferior a los anteriores, setenta y tres por ciento frente a más de ochenta y tres por ciento. Prácticamente tres cuartos de los paquetes analizados por el sistema de detección son interpretados de forma correcta. Sin embargo, el valor que más interesa maximizar en una solución de seguridad, la exhaustividad, ha sido el mayor obtenido de los tres escenarios, lo cual se traduce en el mayor porcentaje de paquetes fraudulentos detectados con más del noventa y siete por ciento de ellos interpretados como posibles ataques.

En contraposición, el número de falsas alarmas es elevado. Concretamente, más del ochenta y ocho por ciento de las alertas son falsos positivos. Este valor negativo podría ser suplido con el factor humano o con una segunda etapa de análisis y obtener, por tanto, un potente sistema de detección de ataques.

El algoritmo de detección de ataques es capaz de analizar e interpretar de forma correcta más del setenta y tres por ciento de los paquetes intercambiados en una red de sensores inalámbrica con topología de árbol en la cual dos nodos encaminan el tráfico recogido por los nodos sensores al nodo central. Además, únicamente doce paquetes fraudulentos de los más de cuatrocientos intercambiados no son detectados como posibles ataques dando un valor de exhaustividad del sistema de detección superior al noventa y siete por ciento.

6. Conclusiones y trabajos futuros

En este capítulo se presentan los resultados obtenidos tras la elaboración de este Proyecto de Fin de Grado y se presentarán propuestas de trabajos futuros.

6.1. Conclusiones

El trabajo llevado a cabo en este proyecto ha consistido en diseñar e implementar un sistema de seguridad para redes de dispositivos IoT con mecanismos de seguridad avanzados. Este sistema trata de detectar si se está produciendo un ataque en la red de sensores inalámbrica mediante algoritmos de aprendizaje automático.

Para realizar el proyecto, en primer lugar, se elaboró un estudio del marco tecnológico y los antecedentes. El estudio consistió al comienzo en entender las limitaciones que las plataformas IoT poseen para, con una visión global, extraer las limitaciones y diseñar de acuerdo a ellas la solución. Seguidamente, se analizaron los ciberataques con el objetivo de conocer el tipo de amenazas a las que el sistema de detección se podría enfrentar. Tras conocer los distintos tipos de ataques que pueden darse en una red de dispositivos IoT, se procedió al estudio de soluciones de seguridad basadas en mecanismos de aprendizaje automático anteriores en este tipo de redes para analizar las diferentes estrategias utilizadas. Por último, se compararon las herramientas de simulación con las que trabajar y poder validar el sistema desarrollado.

Una vez terminado el estudio del estado del arte, se llevó a cabo el diseño de la solución de seguridad. Para ello, primero se analizaron las especificaciones y las restricciones junto con el presupuesto necesario para su desarrollo. Tras el análisis, se diseñaron los diagramas UML que explican en detalle el funcionamiento de la solución de seguridad y se terminó explicando la propuesta de mecanismo de aprendizaje automático para el sistema de detección.

A continuación del diseño de la solución de seguridad, se realizó la implementación de la misma. Para ello, fue necesario implementar en un entorno de simulación las redes de sensores inalámbricas que iban a ser objeto de pruebas. Tras crear las redes de sensores inalámbricas, se desarrolló el codificador automático, el núcleo del sistema de detección, y se obtuvieron los conjuntos de datos necesarios para su entrenamiento. Con la información suficiente recopilada se prosiguió con el entrenamiento del *autoencoder* para posteriormente validar su rendimiento.

La última fase del proyecto consistió en comprobar el funcionamiento del sistema de detección. Se utilizaron los conjuntos de datos de validación recogidos durante la validación para obtener los resultados de validación del rendimiento del clasificador y se analizaron para comprobar el cumplimiento de los objetivos específicos propuestos:

- Viabilidad

  Con la realización de las distintas pruebas se pudo comprobar que es posible integrar soluciones de seguridad basadas en mecanismos de aprendizaje automático en redes de dispositivos IoT.

- Simulación

  Se demostró la capacidad de integrar un sistema de seguridad en un entorno de simulación de redes en el cual se diseñaron distintos escenarios genéricos. El uso de un entorno de simulación permite realizar pruebas y detectar fallos en una fase previa al despliegue lo que redunda en un ahorro de tiempo y costes.

- Rendimiento

  Los resultados de las pruebas de validación expusieron resultados satisfactorios del sistema de detección con valores de precisión, exhaustividad y exactitud por encima de las expectativas demostrando el buen rendimiento del mecanismo de aprendizaje automático.

Tras el análisis de los resultados obtenidos en las pruebas, es reseñable que el rendimiento del sistema de detección ha superado las expectativas al obtener, en el peor de los casos, una exactitud superior al setenta por ciento llegando a superar el ochenta y cinco por ciento en el mejor caso. Por tanto, los resultados de la validación son satisfactorios y el sistema automatizado de detección de ataques funciona adecuadamente.

Como conclusión, se afirma el cumplimiento de los objetivos propuestos para lograr el principal objetivo del Proyecto de Fin de Grado satisfactoriamente. Por consiguiente, el sistema automatizado de detección de ataques es una solución de seguridad válida para redes de dispositivos IoT independientemente de sus topologías y de la información que en ellas se transmita.

6.2. Trabajos futuros

Durante el desarrollo de la solución propuesta se han descubierto posibles complementos o mejoras al sistema.

Como mejora, dado que el rendimiento no es ideal, se podría implementar un sistema de análisis secundario ya sea para reducir el número de falsas alarmas o para reducir el número de falsos negativos. Esta mejora implicaría volver a procesar los paquetes que se hayan interpretado como positivos en caso de querer reducir las falsas alarmas o reprocesar los paquetes interpretados como negativos para evitar perder paquetes maliciosos, lo que producirá también un aumento del tiempo de respuesta.

Otra opción como trabajo futuro sería migrar el mecanismo de detección diseñado a redes de comunicaciones por cable convencionales. El algoritmo de aprendizaje automático utiliza campos de las cabeceras de los protocolos presentes también en las redes cableadas y es altamente probable que pueda ser aplicable sin prácticamente modificaciones.

Finalmente, sería interesante evaluar la posibilidad de migrar las tareas de aprendizaje automático que más recursos consumen a despliegues en la nube, para utilizar la potencia de estos sistemas y mejorar el rendimiento.

7. Referencias

[1] B. Dorsemaine, J. Gaulier, J. Wary, N. Kheir and P. Urien, "Internet of Things: A Definition & Taxonomy," *2015 9th International Conference on Next Generation Mobile Applications, Services and Technologies*, Cambridge, UK, 2015, pp. 72-77, doi: 10.1109/NGMAST.2015.71.

[2] Adafruit, "Adafruit Feather HUZZAH ESP8266", 2021. [En línea]. Disponible en: https://learn.adafruit.com/adafruit-feather-huzzah-esp8266. [Accedido: 25 febrero 2021].

[3] Arduino, "Arduino Nano", 2021. [En línea]. Disponible en: https://store.arduino.cc/arduino-nano. [Accedido: 25 febrero 2021].

[4] BeagleBoard, "BeagleBone Black", 2021. [En línea]. Disponible en: https://beagleboard.org/black. [Accedido: 27 febrero 2021].

[5] Intel, "DE10-Nano", 2021. [En línea]. Disponible en: https://software.intel.com/content/www/xl/es/develop/topics/iot/hardware/fpga-de10-nano.html. [Accedido: 27 febrero 2021].

[6] Intel, "Intel Galileo", 2021. [En línea]. Disponible en: https://www.intel.la/content/www/xl/es/support/articles/000005735/boards-and-kits/intel-galileo-boards.html. [Accedido: 28 febrero 2021].

[7] Libelium, "limbelium Waspmote", 2021. [En línea]. Disponible en: https://www.libelium.com/iot-products/waspmote. [Accedido: 28 febrero 2021].

[8] Node MCU Team, "NodeMCU V2", 2021. [En línea]. Disponible en: https://naylampmechatronics.com/espressif-esp/153-nodemcu-v2-esp8266-wifi.html. [Accedido: 28 febrero 2021].

[9] Raspberry Pi, "Raspberry Pi Zero W", 2021. [En línea]. Disponible en: https://www.electronicsdatasheets.com/manufacturers/raspberry-pi/parts/raspberry-pi-zero-w. [Accedido: 28 febrero 2021].

[10] Pycom, "WiPy 3.0", 2021. [En línea]. Disponible en: https://pycom.io/product/wipy-3-0. [Accedido: 25 febrero 2021].

[11] L. Xiao, X. Wan, X. Lu, Y. Zhang and D. Wu, "IoT Security Techniques Based on Machine Learning: How Do IoT Devices Use AI to Enhance Security?" in *IEEE Signal Processing Magazine*, vol. 35, no. 5, pp. 41-49, Sept. 2018.

[12] K. Zhang, X. Liang, R. Lu and X. Shen, "Sybil Attacks and Their Defenses in the Internet of Things," in *IEEE Internet of Things Journal*, vol. 1, no. 5, pp. 372-383, Oct. 2014, doi: 10.1109/JIOT.2014.2344013.

[13] T. Gu, Z. Fang, A. Abhishek and P. Mohapatra, "IoTSpy: Uncovering Human Privacy Leakage in IoT Networks via Mining Wireless Context," *2020 IEEE 31st Annual International Symposium on Personal, Indoor and Mobile Radio Communications*, London, UK, 2020, pp. 1-7, doi: 10.1109/PIMRC48278.2020.9217236.

[14] G. Rajendran, R. S. Ragul Nivash, P. P. Parthy and S. Balamurugan, "Modern security threats in the Internet of Things (IoT): Attacks and Countermeasures," *2019 International Carnahan Conference on Security Technology (ICCST)*, Chennai, India, 2019, pp. 1-6, doi: 10.1109/CCST.2019.8888399.

[15] *Security architecture for open systems interconnection for CCITT applications*, ITU-T Recommendation X.800, 1991.

[16] A. Nawrocka, A. Kot and M. Nawrocki, "Application of machine learning in recommendation systems," *2018 19th International Carpathian Control Conference (ICCC)*, Szilvasvarad, Hungary, 2018, pp. 328-331, doi: 10.1109/CarpathianCC.2018.8399650.

[17] M. A. Al-Garadi, A. Mohamed, A. K. Al-Ali, X. Du, I. Ali and M. Guizani, "A Survey of Machine and Deep Learning Methods for Internet of Things (IoT) Security," in *IEEE Communications Surveys & Tutorials*, vol. 22, no. 3, pp. 1646-1685, thirdquarter 2020, doi: 10.1109/COMST.2020.2988293.

[18] S. Alharbi, P. Rodriguez, R. Maharaja, P. Iyer, N. Subaschandrabose, and Z. Ye, "Secure the Internet of Things with challenge response authentication in fog computing," in *Proc. IEEE 36th Int. Perform. Comput. Commun. Conf. (IPCCC)*, 2017, pp. 1–2.

[19] W. Hu, Y. Liao, and V. R. Vemuri, "Robust support vector machines for anomaly detection in computer security," in *Proc. Int. Conf. Mach. Learn. Appl. (ICMLA)*, 2003, pp. 168–174.

[20] C. Wagner, J. François, and T. Engel, "Machine learning approach for IP-flow record anomaly detection," in *Proc. Int. Conf. Res. Netw.*, 2011, pp. 28–39.

[21] H.-B. Wang, Z. Yuan, and C.-D. Wang, "Intrusion detection for wireless sensor networks based on multi-agent and refined clustering," in *Proc. IEEE WRI Int. Conf. Commun. Mobile Comput. (CMC)*, vol. 3, 2009, pp. 450–454.

[22] L. Xiao, Y. Li, G. Han, G. Liu and W. Zhuang, "PHY-Layer Spoofing Detection with Reinforcement Learning in Wireless Networks," in *IEEE Transactions on Vehicular Technology*, vol. 65, no. 12, pp. 10037-10047, Dec. 2016, doi: 10.1109/TVT.2016.2524258.

[23] "The Network Simulator - ns-2", *Isi.edu*. [En línea]. Disponible en: https://www.isi.edu/nsnam/ns/. [Accedido: 30 mayo 2021].

[24] "Nam: Network animator", *Isi.edu*. [En línea]. Disponible en: https://www.isi.edu/nsnam/nam/. [Accedido: 30 mayo 2021].

[25] "The software that empowers network professionals", *Gns3.com*. [En línea]. Disponible en: https://www.gns3.com/. [Accedido: 30 mayo 2021].

[26] "OMNeT++ Discrete Event Simulator", *Omnetpp.org*. [En línea]. Disponible en: https://omnetpp.org/. [Accedido: 30 mayo 2021].