Universidad Internacional de la Rioja (UNIR)

ESIT

Máster Universitario en Inteligencia Artificial

Sistema de detección de ataques *Botnet* a redes *IoT* basado en grafos de comunicación

Trabajo Fin de Máster

Presentado por: Concejal Muñoz, David

Dirigido por: Del Corte Valiente, Antonio

Ciudad: Madrid

Fecha: 21 de julio de 2022

Índice de Contenidos

Índice de Ilustraciones

Índice de Tablas

Resumen

La seguridad de las redes de comunicación tiene gran relevancia para la sociedad actual. Términos como ciberataque, ciberguerra, *phising*, *spam*, denegación de servicio (*DoS*) se han incluido en el vocabulario colectivo de la sociedad y es un claro indicador de la importancia de la seguridad de estas redes. Este trabajo presenta un estudio sobre el uso de técnicas de aprendizaje automático enfocadas a la mejora de la seguridad en redes en general y los pequeños dispositivos conectados a redes *IoT* en particular.

Se ha desarrollado un prototipo para la detección de ataques realizados desde redes de equipos informáticos infectados por programas maliciosos (*botnets*). Se ha empleado redes neuronales profundas, denominadas *stacked autoencoders* (*SAE*), para aprender el comportamiento normal de una red y posibilitar la detección de anomalías del mismo por la presencia de *botnets*. Este prototipo aporta la utilización de grafos de comunicación para describir el comportamiento en lugar del flujo de red. La combinación de *SAEs*, y grafos de comunicación, reduce los requisitos computacionales e incorpora facetas del comportamiento que pasan desapercibidas en el análisis clásico del flujo de red. Este aspecto de reducción computacional es especialmente relevante para su utilización en dispositivos *IoT* que cuentan con capacidades limitadas.

Los experimentos se han realizado con datos de tráfico benigno y tráfico generado por *botnets* en un entorno real de una red de dispositivos *IoT*. Los datos han sido elaborados por Garcia et al. (2020) y recogidos bajo la denominación *Aposemat IoT-23*. Los resultados corroboran la hipótesis inicial.

La propuesta es una alternativa viable al problema de detección de ataques *botnets* en redes *IoT*, obteniendo una exactitud similar a otros modelos propuestos para redes de comunicaciones clásicas.

Palabras Clave: *autoencoder*, *botnet*, ciberataque, *IoT*, grafo de comunicación

Abstract

The security of communication networks is highly relevant to today's society. A clear indicator is that terms such as cyber-attack, cyber-war, phishing, spam, and denial of service (DoS) have become common language. This work is a study on machine learning techniques used to improve network security, particularly, IoT networks. A prototype to detect botnet attacks has been developed (attacks from a set of computers infected by bots, pieces of malicious software that gets orders from a botmaster). Deep neural networks, called stacked autoencoders (SAE), have been used to learn the network's normal behavior and by spotting anomalies detect the botnet's presence. Instead of network flow, the behavior is described with communication graphs. The combination of SAEs and communication graphs reduces the computational costs and incorporates behaviors that go unnoticed in classical network flow analysis. IoT devices that have limited computational capabilities will profit from this method. The experiments have been carried out with benign traffic data and traffic generated by *botnets* in a real environment of a network of *IoT* devices. The data has been produced by Garcia et al. (2020) and collected under the name *Aposemat IoT-23*. The results corroborate the initial hypothesis. The proposal is a viable alternative to the problem of detecting botnet attacks in IoT networks, obtaining an accuracy like that of other models proposed for classic communication networks.

Keywords: autoencoder, botnet, communication graph, cyber-attack, IoT

Capítulo 1. Introducción

Este trabajo presenta un prototipo de sistema de detección de intrusiones basado en anomalías (*AIDS*) del comportamiento de una red de comunicaciones. El prototipo emplea un tipo de redes neuronales profundas no supervisadas denominadas *Stacked autoencoders*. El entrenamiento se realiza con las características elaboradas a partir de la información recopilada en forma de grafos de comunicación. La finalidad es discriminar el tráfico generado por ataques con fines maliciosos a redes de computadores, aun cuando las capacidades de computación sean reducidas. Este es el caso de las redes de dispositivos *IoT*.

A continuación se describe la relevancia de este enfoque, así como la estructuración de esta memoria. Se desarrollan los fundamentos y las características de la solución aportada a lo largo de los capítulos que la componen.

1.1. Motivación

La ciberseguridad actualmente se considera necesaria desde un punto de vista estratégico, tanto en ámbitos industriales y corporativos como en ámbitos gubernamentales. Los informes elaborados por compañías especializados en ciberseguridad reflejan el incremento anual de ciberataques. El informe anual realizado por *CheckPoint (2021)* refleja un incremento de ataques del 17 % en Estados Unidos con 443 ataques por semana, del 37 % en EMEA (Europa, Oriente Medio y África) con 777 ataques por semana y un 13 % en APAC (Asía y el Pacífico) con 1338 ataques por semana. Los ataques realizados con *botnets* representan el 27 % del total de ataques.

Los avances en las comunicaciones de internet en las últimas décadas y, la democratización del acceso a la red, es la causa del aumento de redes de dispositivos interconectados. El informe de Cisco (2020) muestra la evolución al alza del número de dispositivos conectados, siendo las conexiones máquina a máquina (*M2M*) de dispositivos *IoT* dónde incide el crecimiento.

Se atribuye a K. Ashtom la introducción en 1999 del término internet de las cosas (*IoT*). El paulatino interés e incesante desarrollo del *IoT* ha reforzado el aumento de los datos y dispositivos. El *IoT* es ampliamente usado en la producción industrial y el ámbito social. Ofrece importantes ventajas en términos de comodidad, eficiencia y accesibilidad, pero como indica W. Zhou et al. (2019), también ha sido la causa de graves amenazas de seguridad y privacidad.

Uno de los fines fundamentales de los dispositivos *IoT* es recopilar datos con un fin concreto. Su uso puede ser de tipo doméstico, corporativo o militar. Son utilizados para fines como el control de temperatura o el control de acceso a una vivienda, mediciones biométricas, control de la calidad y seguridad de procesos industriales y otros muchos objetivos. Estos datos son almacenados con frecuencia en servidores ubicados en redes locales o se almacenan en la nube. Los datos pueden tener un carácter sensible o producir daños, incluso personales, si son alterados.

Los dispositivos *IoT* de manera habitual presentan una seguridad débil debida a diversos factores. Se han integrado rápidamente en el ámbito doméstico, dónde sus usuarios son generalmente tolerantes a bajos niveles de seguridad que puedan exponer sus datos. Además, los consumos y capacidades restringidas de estos dispositivos determina que los fabricantes no incorporen mecanismos de seguridad completos para enfrentarse a métodos de ataque cada vez más sofisticados. Son frecuentemente una puerta para la intrusión en otros sistemas a los que se conecta o simplemente causan el bloqueo de los consumidores de los datos recopilados por alteración o falta de estos. También pueden ser integrados en redes *botnets* y participar activamente en ataques maliciosos.

Actualmente, la inclusión de estos dispositivos en el ámbito médico, industrial, militar y la automoción han conducido a la demanda de la inclusión de mecanismos de seguridad más sofisticados en ellos. La comunidad científica, la industria y otros ámbitos sociales se han aunado para la mejora de mecanismos aplicables. Se requiere una seguridad de tipo individual para neutralizar escenarios de robo de información o daños materiales y físicos.

La producción científica sobre sistemas de detección de intrusiones (*IDS*), que emplean técnicas de aprendizaje automático e inteligencia artificial, es una tendencia actual. Un reto para estas soluciones es su uso en dispositivos *IoT*. El consumo y requisitos computacionales de estas técnicas hacen complejo su inclusión en estos dispositivos.

1.2. Planteamiento del trabajo

Los *IDS* utilizan diversas técnicas con el objetivo de diferenciar tráfico de red con fines legítimos del tráfico malicioso. El objetivo es determinar la existencia de un ataque para iniciar las acciones necesarias para su desactivación y mitigar los daños asociados a estos ataques.

Estos sistemas se basan principalmente en el análisis del flujo de las comunicaciones en la red para detectar diversos tipos de ataques. Se propone un enfoque alternativo basado en analizar el comportamiento de las comunicaciones punto a punto entre servidores a partir de grafos de comunicación basados en el flujo de red. Estos análisis permiten observar facetas de comportamiento que pasan desapercibidas en el análisis directo del flujo de comunicación. Se pueden extraer patrones de comportamiento benigno y malicioso de estos grafos.

Las técnicas de *Machine Learning* para la detección de anomalías son aplicables para el análisis del comportamiento de redes. Se propone el uso de técnicas de aprendizaje profundo (*deep learning*) implementadas mediante redes neuronales profundas denominadas *Stacked Autoencoders* (SAE). Este tipo de redes son adecuadas para conjuntos de datos no balanceados, siendo ésta una circunstancia habitual en los problemas de detección de tráfico malicioso.

Por otro lado, el análisis del flujo de red o del comportamiento son costosas desde un punto de vista computacional. Esto no permite albergar este tipo de controles en dispositivos con capacidades de consumo y computación reducidas, como es el caso de dispositivos *IoT*. Se tiende a centralizar el análisis de amenazas en servidores o servicios en la nube. Sin embargo, existen estudios que sostienen la viabilidad de la inclusión de autoencoders en dispositivos *IoT*. Se abre así una vía para la incorporación de los sistemas de detección de intrusión en los propios dispositivos. Esto supone un incremento en su seguridad.

1.3. Estructura de la memoria

La memoria se ha estructurado en siete capítulos que se describen seguidamente.

El capítulo 2 presenta un estudio exploratorio del contexto y estado del arte en el entorno académico respecto del problema que se va a abordar. Se describe cuáles son las principales líneas de investigación que se están siguiendo actualmente.

El capítulo 3 describe el objetivo general, así como los objetivos específicos, de este trabajo de investigación. También expone la metodología que se ha seguido durante el mismo.

El capítulo 4 expone los requisitos identificados para el prototipo desarrollado.

El capítulo 5 describe en detalle el prototipo. Se detalla el diseño de la herramienta desde un punto de vista técnico y de funcionamiento. Facilita el uso de la herramienta para el lector cara a su utilización y evaluación de sus prestaciones.

El capítulo 6 muestra la evaluación del prototipo en el desempeño de las funciones para las que fue desarrollado. Se comprueba la capacidad de detección de diversos tipos de ataques *botnets*.

Finalmente, el capítulo 7 presenta las conclusiones extraídas del trabajo de investigación realizado e identifica las futuras líneas de investigación derivadas de los resultados del trabajo.

Capítulo 2. Contexto y Estado del Arte

Este capítulo pone en contexto al lector describiendo los diferentes conceptos relevantes para el estudio realizado. Se introduce los conceptos fundamentales de las redes de comunicaciones, así como los diferentes aspectos que debe abordarse para ofrecer niveles adecuados de seguridad. Se describen los principales tipos de amenazas a las que se enfrentan las redes y soluciones clásicas para mitigarlas. Se profundiza en los ataques maliciosos realizados mediante las redes denominadas *Botnets*.

Se exponen diversos métodos de análisis para evaluar el comportamiento normal de las redes de comunicación. La información extraída de estos análisis son actualmente una fuente importante en las técnicas empleadas para la detección de amenazas.

Se especifica técnicas de aprendizaje automático empleadas para la detección de ataques y se muestran resultados obtenidos en investigaciones previas en este campo.

Finalmente, se exponen las conclusiones obtenidas del estado del arte.

2.1. Las redes de comunicaciones

El concepto de redes de comunicaciones es amplio y puede ser ambiguo. La denominación de redes de computadores se emplea habitualmente para una mayor precisión. Tanenbaum y Wetherall (2012) emplea este término para describir a un conjunto de ordenadores con funcionamiento autónomo y con capacidades para intercambiar información entre ellos. El calificativo de ordenador se puede extender a pequeños dispositivos electrónicos con estas características. Este es el caso de los dispositivos *IoT*. El intercambio de la información requiere de un medio para la comunicación, como cables y conexiones inalámbricas, y un conjunto de protocolos para coordinar los diferentes aspectos de la comunicación. Por tanto, el concepto de redes de comunicaciones se circunscribe, en este estudio, a las infraestructuras que interconectan dispositivos electrónicos y los protocolos necesarios para el intercambio de información entre ellas.

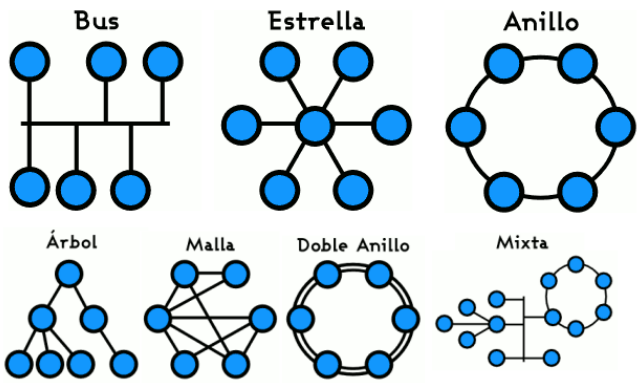
Las redes de comunicaciones se caracterizan por sus componentes y por su configu-

ración. La distribución de los componentes de la red determina su topología y la cobertura geográfica de la red. La configuración de la red describe el modo de interactuar de los diferentes dispositivos. Estas características son relevantes para los objetivos fundamentales de la seguridad de las redes.

Los tres objetivos que persigue los sistemas de seguridad en el entorno de redes son: la disponibilidad de la red, la integridad de la información y su confidencialidad(Z. Lu et al., 2010). Se debe mantener la accesibilidad y uso de los servicios ofrecidos por la red. La información no debe ser alterada de forma intencionada o accidental. Los servicios y la información solo deben ser accesibles a las personas autorizadas.

Las topologías comunes de una red son en línea, en anillo, en estrella, en malla, en árbol, híbridas e inalámbricas (Anjum & Apsar Pasha, 2015)(Z. Zhang, 2010)(Espina et al., 2007). La figura 2.1 representa las diferentes topologías descritas.

Figura 2.1: *Topologías de redes de comunicaciones.*



Fuente: https://sites.google.com/site/tecnocompu32/home/topologias-de-red

Las redes en línea (*bus*) tienen un canal de comunicaciones central, denominado *Backbone*, al cual se conectan todos los dispositivos. La topología en anillo conecta los dispositivos de manera circular y la información fluye en una única dirección. Los dispositivos tienen una distribución en estrella cuando uno de ellos se encarga del control de la comunicación con el resto y todos los mensajes pasan a través de él. Una malla es una disposición con conexiones redundantes entre dispositivos y se comunican entre ellos a través de estas conexiones directas o mediante múltiples saltos(*hops*) entre dispositivos. La topología en árbol o jerárquica es un caso de una malla dónde los dispositivos organizan sus conexiones por niveles de manera que se comunican con dispositivos de un nivel superior o inferior, pero no con los dispositivos al mismo nivel. Las topologías híbridas son combinaciones de las anteriores. Los dispositivos inalámbricos modifican la topología de la red de manera dinámica.

Los riesgos asociados a la disponibilidad de la red varían en función de la topología de la misma. Una red en bus puede provocar interrupciones o un deterioro del servicio cuando uno de los dispositivos falla. Las redes en estrella no estarán disponibles cuándo se daña el dispositivo central o cualquiera de los dispositivos en las redes en anillo. Las redes en malla son más tolerantes a los daños en un dispositivo puesto que incluyen redundancia en las conexiones. Solo partes de la red quedarían inoperativas.

Las redes también se clasifican en función de su cobertura geográfica. Los tipos de redes son: redes de área local(*LAN*), área metropolitana(*MAM*), área extensa(*WAN*), área personal (*PAN*) y privadas virtuales(*VPN*)(Forouza, 2014)(«IEEE Standard for Local and Metropolitan Area Networks–Port-Based Network Access Control», 2020). Una red *LAN* permite la comunicación rápida entre dos dispositivos sin necesidad de dispositivos intermedios. El área geográfica es reducida y los tiempos de respuesta de la red son del orden de milisegundos. Son usados tanto por organizaciones como particulares para la comunicación entre diferentes dispositivos en una misma ubicación. Estas redes se denominan *WLAN* cuándo la comunicación se realiza de modo inalámbrico. Las redes *PAN* también permiten la trasmisión en distancias cortas, pero se circunscriben a un grupo privado de dispositivos (por lo general dispositivos móviles, tablets, etc.) y se comunican de manera inalámbrica. El área geográfica de las redes *MAN* es mayor al de las redes de área local, pudiendo llegar a cubrir una ciudad. Son usados por organizaciones para las comunicaciones entre ubicaciones próximas. Las redes *WAN* permiten la comunicación entre dispositivos a grandes distancias como diferentes ciudades, países y continentes. Son operadas por proveedores de servicios de red (*ISP*) permitiendo las comunicaciones entre organizaciones y usuarios privados. Las redes de virtuales (*VPN*) son una simulación lógica de una red *LAN* sobre una red de área más amplia.

Los riesgos asociados a la confidencialidad de los datos y su integridad aumentan a medida que aumenta el área geográfica cubierta por las redes. El cifrado de la información trasmitida es uno de los métodos empleados reducir los riesgos.

Las redes pueden presentar una configuración punto a punto (*peer-to-peer*) o cliente/servidor. Todos los dispositivos poseen la misma condición en las redes punto a punto. Se comparten recursos entre los diferentes dispositivos, siendo un modo económico de compartir grandes cantidades de información y ancho de banda (Bawa et al., 2003). Sin embargo, existen riesgos de accesibilidad de los servicios y autenticidad de la información compartida por la dificultad en la identificación de los dispositivos y evaluación de los recursos disponibles. Las redes con una configuración cliente/servidor se organizan de

forma que unos dispositivos proveen servicios al resto de dispositivos que actúan con un rol de consumidores de los mismos (H. Zhang, 2013). Esta configuración de red también presenta riesgos asociados a la accesibilidad a los servicios ofrecidos por el servidor.

La comunicación de los dispositivos se estructura de manera modular, organizándose en capas jerarquizadas. Cada una de estas capas ofrece servicios que son utilizados por las capas de niveles superiores. Kurose y Ross (2013) indican que esta organización simplifica y facilita los cambios de implementación de los servicios ofrecidos por cada capa, pues el resto del sistema permanece sin cambios. Los protocolos de red establecen las reglas que dirigen las comunicaciones y se estructuran por capas. Un protocolo de red puede pertenecer a una o más capas de red, La implementación de los protocolos de red puede ser mediante *hardware*, *software* o una combinación de ambos.

El número de capas de las redes depende del modelo de implementación. Se estandarizó en los años 90 el modelo de interconexión de sistemas abiertos, conocido como modelo OSI, por la Organización Internacional de Estándares (*ISO*) y la Comisión Internacional de Electro tecnología (*ECI*) («ISO/IEC 7498-1 Information technology - Open System Inteconnexion - Basis Reference Model», 1994). El modelo contiene siete capas: física, enlace, red, transporte, sesión, presentación y aplicación.

La capa física es la responsable de la transmisión de los bits de información agrupados en una trama (*frame*). La capa de enlace divide en tramas los segmentos de información recibida (datagramas) y los transmite desde un dispositivo a otro enlazado punto a punto. La capa de red se hace cargo de distribuir los datagramas desde el dispositivo origen del mensaje al dispositivo de destino. La capa de transporte descompone en datagramas los mensajes de las aplicaciones al iniciar la comunicación en el dispositivo de origen y los recompone en el mensaje original en el dispositivo de destino. Es responsable de controlar el flujo de datagramas que se envían. La capa de sesión ofrece servicios para establecer puntos de control y recuperación sobre la información intercambiada entre aplicaciones. La capa de presentación proporciona servicios para la interpretación de los datos intercambiados. Se encarga del formato interno de almacenamiento de los datos de aplicación, así como de su compresión y cifrado. La capa de aplicación ofrece los servicios de red a los usuarios de las mismas.

Existen otros modelos de red con una organización en capas diferente. El modelo de Internet se dispone en cinco capas. Las primeras cuatro capas son similares al modelo *OSI*. Sin embargo, las funciones de las capas de sesión y presentación se delegan a la capa de aplicación. Por otro lado, no todos los dispositivos de una red tienen el mismo

número de capas. Hay dispositivos dedicados a encaminar los mensajes a los dispositivos de destino e implementan hasta la capa de red.

Existen amenazas de seguridad diferenciadas para cada una de las capas en las que se organizan las redes. Furdek et al. (2016) describen como las capas físicas y de enlace pueden sufrir ataques en forma de pulsos electromagnéticos y otras capas de nivel superior pueden sufrir ataques, provocando la denegación de servicios. Otro riesgo existente en las redes es la congestión por un flujo excesivo de mensajes.

A continuación, se describen dos paradigmas de seguridad empleados para la detección y mitigación de los efectos derivados de los ataques maliciosos: los sistemas de detección de intrusiones y las redes definidas por *software*(*SDN*).

2.1.1. IDS: Sistemas de detección de intrusiones

Liao et al. (2013) definen a los *IDS* como sistemas que identifican acciones maliciosas para poder mantener la seguridad. Los *IDS* pueden implementarse mediante elementos *software* y *hardware*.

Estos sistemas analizan el tráfico con un enfoque diferente a los tratamientos realizados por los *firewalls* clásicos. Se buscan características que permitan identificar al tráfico como malicioso.

Se pueden clasificar en dos grandes grupos (Khraisat et al., 2019): basados en firmas (*SIDS*) y basados en anomalías (*AIDS*).

Los sistemas basados en firmas usan técnicas de búsqueda de patrones en el tráfico de la red. Se disparan alarmas cuándo se encuentra tráfico que se identifique con la firma asociada a un ataque catalogado e incluido previamente en una base de datos. Estos sistemas son cada vez menos efectivos, según Khraisat et al. (2019). No son muy efectivos en la detección de ataques de tipo día cero. Además, los ataques son cada vez más sofisticados y emplean diferentes técnicas para pasar desapercibidos a la detección de firmas.

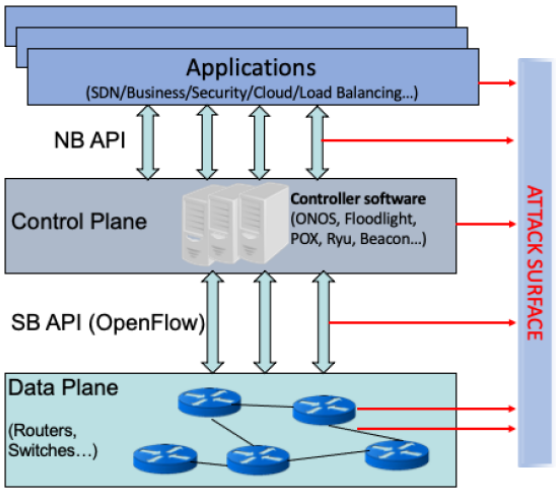
Los sistemas basados en anomalías aprenden el comportamiento normal de la red. Las desviaciones sobre este comportamiento se consideran una anomalía que puede deberse a un ataque. Se generan modelos de comportamiento mediante técnicas de aprendizaje automático o análisis estadísticos. El modelado requiere de una primera etapa de entrenamiento en la cual se aprenden las características del comportamiento normal. Una segunda etapa permite evaluar la precisión del sistema en la detección de anomalías. Este tipo de sistemas permiten detectar las amenazas de tipo día cero, según Alazab et al.

(2012). Estos sistemas además ofrecen una mayor protección. Los atacantes deben conocer el comportamiento normal de la red a fin de evitar la detección de sus actividades en la red.

2.1.2. SDN: Redes definidas por *software*

Benzekki et al. (2016) describen un *SDN* como un paradigma que permite un diseño, implementación y gestión de redes de comunicaciones, donde el control y el flujo de los datos se separan en dos planos diferentes. El plano de control es centralizado. Las políticas de la red se dictan desde un único controlador. Las decisiones de encaminamiento y la información de la topología lógica de la red se controlan en este plano. El plano de datos se encarga de la transmisión de los datos acorde a las políticas comunicadas desde el plano de control. Esta arquitectura simplifica aspectos de la gestión, monitorización y seguridad en las redes.

Figura 2.2: *Arquitectura SDN.*



Fuente: Sarica y Angin (2020)

Sarica y Angin (2020) explican que este tipo de redes se estructuran principalmente en tres capas: aplicación, control e infraestructura (figura 2.2).

Esta arquitectura contiene un controlador ubicado en la capa de control. Este controlador es el encargado de mantener la visión global de la red y las políticas de encaminamiento. Algunas aplicaciones, como los *IDS*, se ubican en la capa de aplicación y se comunican con el controlador. Los dispositivos de encaminamiento reciben del controlador las políticas en forma de reglas de flujo.

Shinan et al. (2021) afirman que un *SDN*, aparte de contar con herramientas para la

gestión de red, ofrece una mayor protección ante algunos tipos de ataques. Sin embargo, Ahmed et al. (2015) también describen vulnerabilidades derivadas de su uso. También debe considerarse que la arquitectura *SDN* centraliza la gestión de la red en el controlador. Esto convierte al controlador en un punto especialmente sensible a los fallos.

Estos sistemas presentan problemas de exactitud y rapidez en la detección de ataques porque hacen un uso intensivo de recursos para la recolección y monitorización del tráfico de red (Shinan et al., 2021).

2.2. Redes *IoT* y su seguridad

La aparición de pequeños dispositivos electrónicos, con funcionamiento autónomo y capacidades de conexión generalmente inalámbrica, ha supuesto nuevos esquemas de comunicación. Heer et al. (2011) describen a laa redes *IoT* como la interconexión de dispositivos muy heterogéneos y con diversos esquemas de comunicación entre humanos, máquinas (*things*) y entre ambos. Los protocolos de redes como Internet se diseñaron para la comunicación entre personas. Las redes *IoT* se implementaron inicialmente con estos protocolos. En la actualidad, ya existen protocolos específicos para la comunicación entre máquinas como *MQTT* (*Message Queue Telemetry Port*),*CoAP* (*Constrained Application Protocol*) y *OMA LWM2M* (*Open Mobile Alliance Light Weight M2M*).

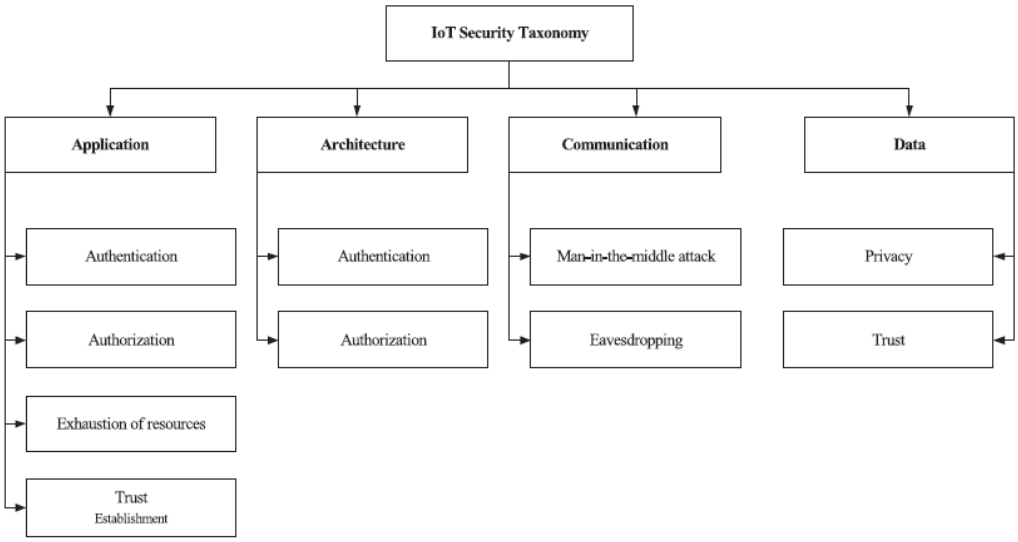
Las redes *IoT* incluyen componentes tales como sensores, actuadores, nodos de cómputo, receptores y comunicadores (Horrow, 2012). Los sensores captan información del mundo real y la transmiten a los nodos de cómputo dónde se toma una decisión que es ejecutada por los actuadores. Este modelo recibe la denominación de red de sensores inalámbrica (*WSN*). Este tipo de redes *ad hoc* se considera un elemento fundamental para la implementación de *IoT* (Alaba et al., 2017).

Se distinguen tres capas diferentes en las redes *IoT*: aplicación, percepción y protocolos de red (Zhao & Ge, 2013). La capa de aplicación ofrece múltiples servicios visibles y, según Yaqoob et al. (2017), se compone generalmente de un *middleware*, un protocolo de comunicación M2M, computación en la nube y una plataforma para el soporte de los servicios. La capa de percepción se encarga de la recopilación y el envío de la información. Está compuesta por dispositivos de percepción, como sensores, y de comunicación (Tsai et al., 2014). La capa de red se encarga de la transmisión y la seguridad. Además, aporta el medio para el acceso a la capa de percepción. En esta capa se incluye Internet, la computación en la red y dispositivos móviles (Pongle & Chavan, 2015).

Los requisitos de seguridad de las redes *IoT* son diferentes a los definidos para las

redes clásicas. Alaba et al. (2017) proponen una taxonomía para la seguridad específica para las redes *IoT*. La figura 2.3 presenta los cuatro dominios propuestos: aplicación, arquitectura, comunicación y datos.

Figura 2.3: *Taxonomía de seguridad en redes IoT.*



Fuente: Alaba et al. (2017)

En el dominio de las aplicaciones se debe disponer de mecanismos de autenticación, autorización, de control del consumo de recursos y de establecimiento de confianza entre dispositivos. La arquitectura de red debe aportar también mecanismos de autenticación y autorización. Se debe contar con métodos para evitar las escuchas inadvertidas (*Eavesdropping*) de las comunicaciones y los intermediarios (*man-in-the-middle*). La seguridad en el dominio de los datos debe ofrecer integridad y confianza en la información.

Mendes et al. (2019) exponen como el aumento de la participación de dispositivos *IoT* en ataques maliciosos, ha atraído la atención sobre aspectos relacionados con su baja seguridad y las vulnerabilidades explotadas. Estos dispositivos son incluidos en redes maliciosas (*botnets*) y participan activamente en ataques de tipo *DDoS*.

2.3. Tipos de amenazas de las redes de comunicaciones

Las redes de comunicaciones están expuestas a múltiples amenazas. La seguridad se despliega para mantener la accesibilidad a los servicios ofrecidos, conservando la integridad y confidencialidad de la información transmitida.

Pawar y J. (2015) clasifican las amenazas en dos grandes categorías y cataloga los tipos de ataques más frecuentes. Consideran una amenaza pasiva a la recopilación de

información que se transmite por la red. Una amenaza activa, sin embargo, provoca alteraciones del funcionamiento de la red.

Los ataques pasivos analizan el flujo de información de la red, suponiendo una amenaza a la confidencialidad de la información. El atacante no produce ninguna alteración de la información que transita por la red. Es habitual el uso de una técnica denominada *man-in-the-middle*. Así, el atacante se interpone en las comunicaciones de dos o más dispositivos, capturando el mensaje y dejándolo continuar a su destino. No suponen una amenaza sobre la integridad de la información ni sobre la accesibilidad de los servicios. Los tipos principales de ataques pasivos son:

a. Análisis del tráfico: Consiste en examinar el flujo de las comunicaciones entre los dispositivos de la red. El objetivo último es inferir información confidencial (Hafeez et al., 2019). Los mecanismos de encriptación no evitan este tipo de ataque, pues la información no se obtiene de los datos transmitidos sino de los patrones de conexión entre los diferentes dispositivos.

b. Escuchas en sigilo (*Eavesdropping*): Consiste en el uso de dispositivos o mecanismos para obtener información de los mensajes que transitan por la red. La información, que no está correctamente encriptada, es accesible para los atacantes. Se conoce también con otros nombres como *sniffing*. Es un tipo de ataque difícil de detectar y se emplea para recopilar información de utilidad para realizar otros tipos de ataques activos. Un ejemplo es la obtención de claves públicas o privadas para posteriores ataques de tipo *replay*.

Los ataques activos suponen una amenaza para la integridad de la información, cuándo provocan alteraciones en el contenido de la información transmitida, y a la accesibilidad a los servicios proporcionados por a red, cuándo producen una degradación o denegación de los mismos. Los tipos principales de ataques activos son:

a. Suplantación (*Spoofing*): Este ataque consiste en suplantar la identidad de un dispositivo legítimo en la red para enviar información en su nombre (Jindal & Sharma, 2014). Se distingue entre tres tipos distintos de suplantación:

    i. *Suplantación de MAC*: Los fabricantes de tarjetas de conexión de red asignan un identificador único a cada dispositivo denominado *MAC* (*Media Access Control*). El protocolo *ARP* (*Adress Resolution Protocol*) permite obtener la *MAC* de un dispositivo a partir de su dirección *IP*. El ataque consiste en alterar este

proceso de resolución de la dirección MAC (*ARP cache poisoning*) de manera que se asigne una dirección bajo el control del atacante. Esto permite recibir los mensajes enviados desde otros dispositivos hacia el dispositivo suplantado. En la práctica puede suponer denegación de servicios, obtención de acceso en procesos de autenticación o situaciones de tipo *man-in-the-middle*.

II. *Suplantación de IP*: Los dispositivos de una red tienen asignada una dirección *IP*. El ataque consiste en generar mensajes maliciosos usando la dirección *IP* de un dispositivo perteneciente a la red. El atacante puede provocar denegación de servicios, inundando la red con mensajes (*flooding*), o puede superar controles de identificación basados en listas de *IP* consideradas de confianza.

III. *Suplantación de DNS*: Las redes utilizan los servicios ofrecidos por el protocolo *DNS* (*Domain Name System*) para determinar la dirección *IP* a partir del nombre asignado al dispositivo. Este ataque consiste en alterar la resolución de la dirección *IP* de manera que se asigne una dirección controlada por el atacante. Las técnicas de envenenamiento de caché (*cache poisoning*) se emplean con este fin. El atacante puede conseguir la inyección de código malicioso o virus en los dispositivos de la red o conseguir información confidencial.

b. Modificación: Estos ataques tienen por objetivo provocar retrasos en el encaminamiento de mensajes. El atacante provoca cambios en la ruta de los mensajes, de manera que son enviados por caminos más largos. Gupta y Mittal (2017) describen este tipo de ataques en el ámbito de las redes móviles *ad-hoc*. Son redes compuestas por dispositivos móviles que cooperan para extender el rango de transmisión mediante el reenvío de mensajes.

c. Agujero de gusano (*Wormhole*): Un atacante puede registrar paquetes transmitidos por una red, o una parte de ellos, y enviarlos a otro lugar de manera encapsulada (*tunneling*) para replicarlo en la nueva localización (Hu et al., 2003a). Este tipo de ataques aplicado a redes móviles *ad-hoc* puede producir problemas de encaminamiento o de denegación de servicio. Los dispositivos encargados de determinar la ruta de los mensajes encuentran mensajes duplicados, no pudiendo distinguir los originales de los paquetes maliciosamente inyectados. También pueden usarse para la obtención de accesos no autorizados cuándo el paquete inyectado llega al destino antes que el paquete original.

d. Fabricación: Son ataques que afectan al encaminamiento de mensajes provocados

por la generación de mensajes de encaminamiento falsos. Li y Joshi (2008) exponen, como vulnerabilidades en los protocolos de red, pueden ser explotadas para alterar el encaminamiento de mensajes, sustituyendo los producidos por la red, por unos generados por los atacantes. Esto supone que los paquetes sean dirigidos a enlaces inexistentes y se inunde de mensajes a un dispositivo o se envíe información errónea sobre el estado de los enlaces.

e. Denegación de servicio (*Deneal of Service*): La denegación de servicios engloba una gran variedad de técnicas maliciosas cuyo objetivo final es reducir o interrumpir el acceso a los servicios ofrecidos por una red. Las técnicas empleadas se van adaptando para sortear las nuevas estrategias de seguridad orientadas a evitarlos (Loukas & Ulayoke, 2010). Estos ataques se han producido desde los inicios de las redes de computadoras, derivando en la actualidad en sofisticados ataques distribuidos (*DDoS*) donde múltiples dispositivos intervienen de manera coordinada.

f. Sumidero (*Sinkhole*): Este tipo de ataque se inicia comprometiendo un dispositivo interno a la red. Este nodo busca atraer todo el tráfico posible hacia él para desecharlo (Kibirige & Sanga, 2015). El dispositivo anuncia al resto de dispositivos unas métricas de encaminamiento fraudulentas. Las redes de sensores inalámbricas (*WSN*) son vulnerables a este tipo de ataques. Son redes compuestas por un conjunto de sensores autónomos que comunican a una base central sus mediciones. Este tipo de ataque persigue evitar que se envíe a la base la mayor cantidad de información posible.

g. *Sybil*: Douceur (2002) introduce esta denominación para ataques a redes con configuración *peer-to-peer* en los que un dispositivo malicioso presenta múltiples identidades. Este tipo de ataques supone una amenaza importante en redes de sensores inalámbricos (*WSN*) (Buford et al., 2009). Patil y Chen (2017) afirman que es sencilla su ejecución en este tipo de redes al tratarse de redes abiertas y con mensajes de difusión (*broadcast*). El dispositivo malicioso realiza el ataque, bien suplantando la identidad de otros dispositivos, bien generando nuevas identidades. Los mecanismos basados en votación, agregación y redundancia pueden ser desactivados por este tipo de ataque.

h. Agujero negro (*Black hole*): Este tipo de ataque tiene el objetivo de interrumpir o deteriorar la entrega de paquetes realizada por los servicios de encaminamiento de la red. Provocan denegación de servicio de al menos una parte de la red. Los servicios

de encaminamiento realizan la búsqueda de la ruta a seguir para la entrega de los mensajes de un dispositivo legítimo. El dispositivo malicioso se adelanta al resto de dispositivos de encaminamiento y propone una ruta a través de él. Los mensajes son entonces enviados hacia el dispositivo malicioso que los elimina (Deng et al., 2002). Se pueden encontrar ataques colaborativos de varios dispositivos maliciosos dónde se proponen mutuamente como parte de la ruta del mensaje (Motwani et al., 2015).

i. *Rushing*: Hu et al. (2003b) describen este ataque que afecta a las redes móviles *ad hoc*. Este ataque provoca la denegación de servicio en los protocolos de encaminamiento cuándo existe más de un salto en la ruta. Consiste en emitir solicitudes de reenvío de mensajes antes que los dispositivos legítimos. Esto aumenta la probabilidad de que los mensajes sean encaminados hacia rutas erróneas.

j. Re-inyección (*Replay*): Se engloban diferentes tipos de ataques bajo este término. De manera general, se puede considerar como un ataque donde un mensaje legítimo, usado en un contexto, se vuelve a inyectar en otro contexto diferente (Malladi et al., 2002). Son utilizados para la obtención de acceso a información o servicios restringidos suplantando una identidad autorizada. También es empleado para producir denegación de servicio inundando la red con la retransmisión de mensajes ya procesados.

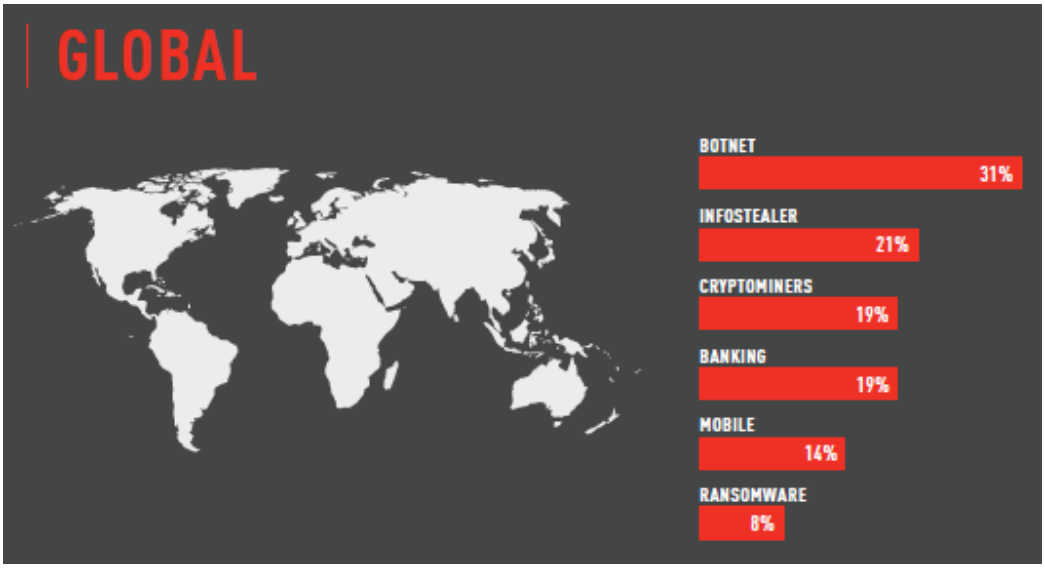
k. Bizantino (*Byzantine*): Un ataque bizantino consiste en un conjunto de dispositivos autenticados en la red que pueden de manera arbitraria producir detenciones de los servicios ofrecidos (Geetha & Sreenath, 2016). Estos dispositivos pueden producir bloqueos en el encaminamiento de los mensajes, eliminar paquetes o generar rutas que retrasen su entrega.

2.4. *Botnets*: Estructura y desafíos

Las *botnets* han supuesto la mayor amenaza a la seguridad durante el año 2021. Un tercio de los ataques registrados a corporaciones a nivel mundial se han realizado por *botnets* (CheckPoint, 2022), como se muestra en la figura 2.4.

Choi et al. (2007) describen las *botnets* como un conjunto de dispositivos comprometidos que son controlados por un *botmaster*. Son dispositivos heterogéneos (PCs, tablets, smartphones, sensores *IoT* y otros tipos) conectados mediante una red de comunicaciones. Una *botnet* no es maliciosa *per se*. La función que subyace es la ejecución automá-

Figura 2.4: *Porcentajes globales de ataques a redes corporativos por tipo malware.*



Fuente: CheckPoint (2022)

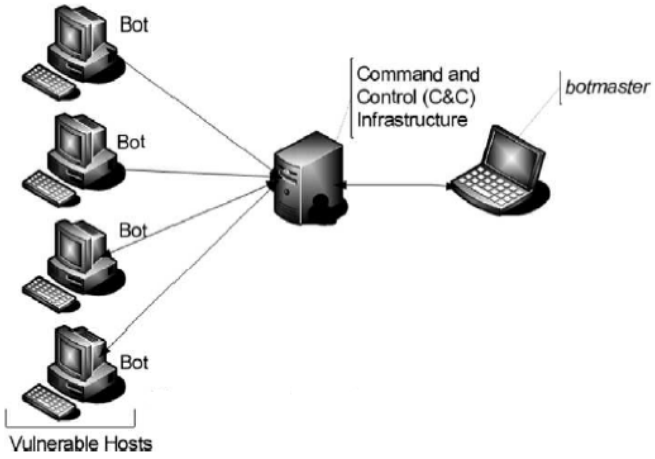
tica de comandos en múltiples dispositivos controlados desde un servidor remoto. Esta funcionalidad permite que un conjunto de dispositivos actúen de manera coordinada como una única entidad. Es una herramienta muy potente en diversas tareas legítimas, tales como despliegues de *software* o la computación distribuida. Estas facilidades han sido aprovechadas por la ciberdelincuencia con finalidades fraudulentas. La arquitectura ha ido evolucionando con los años, siendo cada vez más sofisticada a fin de evitar las medidas de seguridad. Se ha evolucionado tanto en la configuración de red como en los protocolos empleados. Se ha pasado de configuraciones *peer-to-peer* a configuraciones híbridas y de utilizar el protocolo *IRC* (*Internet Relay Chat*), para la comunicación de las órdenes de control, a usar protocolos HTTP o P2P.

2.4.1. Componentes

El análisis de las *botnets* permite identificar las partes que las componen. Silva et al. (2013) identifica los siguientes elementos: anfitriones (*host*) vulnerables, *bots*, *botmaster* y la infraestructura *C&C* (*Command-and-control*). La figura 2.5 representa estos elementos. Limarunothai y Munlin (2015) descomponen la arquitectura *C&C* en servidores y protocolos.

Los anfitriones vulnerables son dispositivos que han sido infectados con una pieza de código maliciosa denominada *bot*. La infección se realiza aprovechando vulnerabilidades existentes previamente en la seguridad del propio dispositivo. Se denominan dispositivos

Figura 2.5: *Componentes de las redes botnets*.



Fuente: Silva et al. (2013)

esclavos o *zombies*, puesto que son controlados a voluntad de un tercero denominado *botmaster*.

Los *bots* son, por tanto, programas potencialmente maliciosos que adquieren el control de los dispositivos anfitriones y que ejecutan las órdenes recibidas de un tercero. Esta característica diferencia a los *bots* de otros programas maliciosos que actúan de manera autónoma.

Los *botmasters* son quiénes controlan la *botnet* y son los responsables de emitir las órdenes a los *bots*. Se identifican con personas que persiguen obtener algún tipo de ventaja con fines ilícitos o fraudulentos.

La infraestructura *C&C* es el elemento que permite al *botmasters* comunicarse con los *bots* y, por lo tanto, es el punto crítico para desactivarlas. Un servidor se encarga de la intermediación entre el *botmasters* y los *bots*. Este servidor emplea protocolos de red para la comunicación. Los primeros empleaban el protocolo *(*IRC). Actualmente, se emplean protocolos *P2P*, *HTTP* o un modelo híbrido de ambos.

2.4.2. Arquitecturas

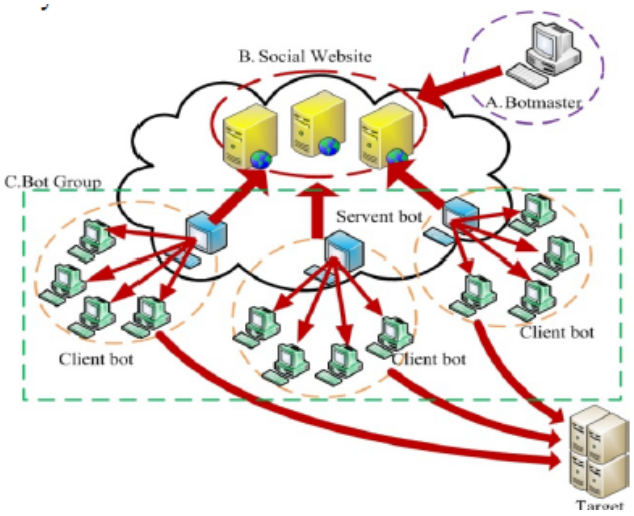
Zeidanloo y Manaf (2009) diferencian entre tres modelos diferentes: centralizado, distribuido e híbrido.

El modelo centralizado utiliza el protocolo *IRC* para la comunicación entre los *bots* y el *bootmaster*. Todas las comunicaciones pasan por un servidor *C&C*. Estas comunicaciones pueden ser utilizadas para la detección del servidor y deshabilitar la red completa, anulando este servidor. Es una arquitectura vulnerable a los mecanismos de seguridad.

El modelo distribuido emplea el protocolo *P2P* para las comunicaciones. Los dispositivos comprometidos tienen capacidad de actuar simultáneamente como servidor *C&C* y como *bot*. El *botmaster* puede enviar las órdenes a varios dispositivos comprometidos. Estos son los encargados de la distribución al resto siguiendo un esquema *P2P*. Este modelo evita la vulnerabilidad descrita en el modelo centralizado, pero se aumenta la complejidad del proceso. Una desventaja de este modelo radica en que el *botmaster* no dispondrá de información de retorno de los *bots*.

El modelo híbrido es más complejo cómo se muestra en la figura 2.6. T.-T. Lu et al. (2011) afirma que son estructuras avanzadas más difíciles de detectar que las *botnets* clásicas. Este tipo de redes usan las redes sociales como medio para la difusión de las órdenes y como punto desde dónde observar los detalles del proceso. Existe dos conjuntos de *bots* diferenciados entre los que actúan de servidores y los que actúan de clientes. Los servidores pueden unirse a las redes sociales para recoger las órdenes del *botmaster* y las retransmiten a los *bots* clientes.

Figura 2.6: *Modelo híbrido de arquitectura botnet.*



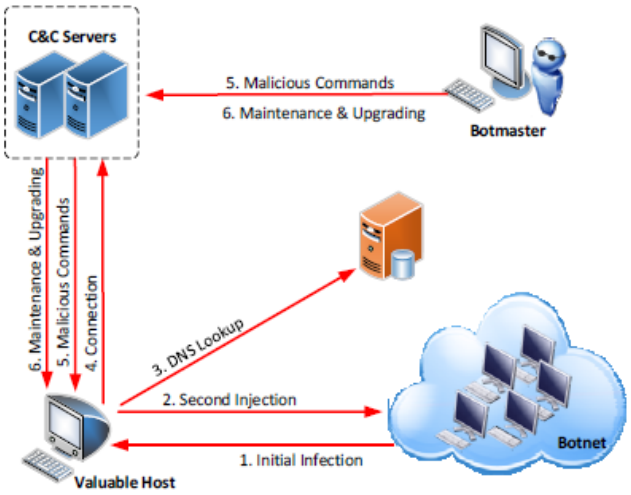
Fuente: T.-T. Lu et al. (2011)

2.4.3. Ciclo de vida

Limarunothai y Munlin (2015) describen el ciclo de vida en cinco etapas: infección inicial, inyección secundaria, búsqueda de *DNS* (*DNS lookup*), conexión, órdenes maliciosas y, por último, mantenimiento y actualización. La figura 2.7 muestra la representación de estas etapas.

El primer paso para la creación de una *botnet* es el reclutamiento de dispositivos para

Figura 2.7: *Ciclo de vida de una red botnet.*



Fuente: Limarunothai y Munlin (2015)

que ejecuten las órdenes del *botmaster*. Este proceso consiste en infectar dispositivos para que hospeden el código que los controlará y los transformará en *bots* de la red. La infección se realiza aprovechando vulnerabilidades en la seguridad del dispositivo o ingeniería social que convierte a un usuario legítimo en el vector de entrada.

Una vez infectado el dispositivo se produce una inyección secundaria que consiste en la descarga e instalación del código del *bot* que convierte al dispositivo en *zombie*. Se han empleado diferentes protocolos para descargar el código ejecutable del *bot*, tales como *FTP*, *HTTP* o, inclusive, *P2P*. El código binario descargado tiene una firma digital que es empleado por sistemas de seguridad (*SIDS*) para detectar e impedir el proceso de descarga.

Limarunothai y Munlin (2015) incluyen, respecto a otros trabajos, la fase de búsqueda *DNS*. El *bot* para completar su proceso de reclutamiento debe conectar con el servidor *C&C*. El *bot* debe conocer su dirección *IP*. Hay diferentes mecanismos para obtener la *IP*. Un método consiste en registrar el dominio de manera que los *bots* pueden obtener esta dirección mediante servicios *DNS*. Esto dificulta la localización del servidor *C&C*. El patrón de la comunicación de esta búsqueda es utilizado para la detección de *botnets*.

La fase de conexión, o *Rallying*, corresponde con el establecimiento de conexión del *bot* con el servidor *C&C*. Se considera que el proceso de reclutamiento se completa tras establecerse la conexión. En ese momento, el *bot* pasa a estar en espera de órdenes que ejecutar. El proceso de conexión se debe realizar cada vez que el dispositivo se enciende. Este comportamiento da lugar a patrones que son explotados por algunos sistemas de

seguridad para detectar las *botnets*.

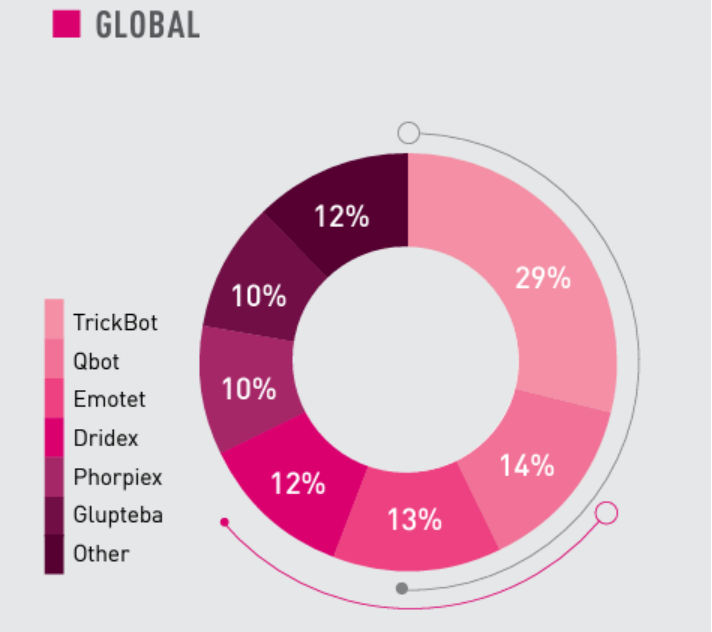
A continuación, se inicia la fase de recepción y ejecución de las órdenes *botmaster* por parte de los *bots*. Esta se alterna con la fase de mantenimiento y actualización, encargada de poner al día el código del *bot* con el objetivo de solucionar errores, incrementar la potencia o pasar desapercibido a los sistemas de seguridad.

2.4.4. Tipos de amenazas

Shinan et al. (2021) afirman que las *botnets* son más dañinas que otros tipos de códigos maliciosos. Cubren una amplia variedad de amenazas. Se emplean para fraudes bancarios, denegación de servicio distribuida (*DDoS*), minado de criptomonedas, robo de información, ciberguerra y otras muchas actividades ilícitas.

El tipo de ataque más frecuente de las redes *botnets* son *DDoS*, provocando la denegación de servicios de red. El informe de seguridad emitido por CheckPoint (2022), muestra como los programas *malware* más relevantes tienden a incorporar capacidades *botnet*.

Figura 2.8: *Botnets con mayor prevalencia en 2022.*



Fuente: CheckPoint (2022)

La figura 2.8 presenta la lista de *malware* con mayor prevalencia que incorporan funciones de las redes *botnet*.

Trickbot pertenece a la familia de troyanos bancarios. Es actualizado con frecuencia y se lo relaciona con ataques de tipo *ransomware*.

Emotet ha sido la red *botnets* con mayor prevalencia. Su objetivo principal era tomar el

control del mayor número posible de dispositivos. Posteriormente, grupos de los *bots* eran vendidos a grupos de ciberdelincuencia para su utilización en actividades específicas.

Dridex tiene capacidades para el robo de información y pertenece a la familia de los troyanos bancarios.

Qbot pertenece a la familia de *malware* destinado al robo de información relacionada con credenciales bancarias. También, se ha relacionado con campañas de correo no deseado y difusión de *ransomware*.

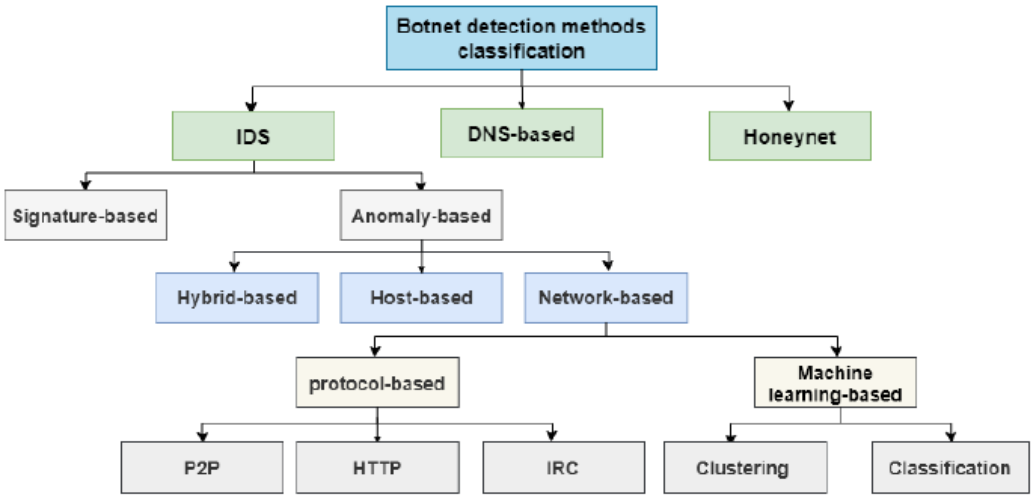
Phorpiex está relacionado con campañas masivas de correo no deseado. También se lo ha relacionado con la difusión de *ransomware* y otros tipos de extorsión.

Glupteba pertenece a la familia de *malware* destinado al robo de información completa de los navegadores. Su principal novedad es el uso de las listas públicas de *BitCoin* para difundir la dirección del *C&C*.

2.4.5. Técnicas de detección

El conocimiento del ciclo de vida de una *botnet*, según Silva et al. (2013), es importante para la definición de métodos de detección. La figura 2.9 resume las técnicas principales empleadas.

Figura 2.9: *Clasificación de técnicas de detección de redes botnet.*



Fuente: Shinan et al. (2021)

Los tres principales mecanismos son: basados en *DNS*, las redes *HoneyNet* y los sistemas de detección de intrusiones (*IDS*).

Los mecanismos basados en *DNS* consisten en detectar patrones de comportamiento de las consultas realizadas a los servicios de *DNS*. El comportamiento difiere cuándo

se realiza por *botnets*, al producirse accesos simultáneos al servidor *DNS*, en lugar de realizarse de manera continua. Permiten identificar a los dispositivos de la *botnet*, puesto que son los únicos que hacen consultas sobre ese dominio.

Los *HoneyNet* no permiten la detección de *botnets* por sí mismas según Karim et al. (2014). Son redes configuradas a propósito con vulnerabilidades con el objetivo de atraer ataques. La finalidad es identificar las técnicas de infección empleadas, el servidor *C&C* y la intensidad de ataque.

Los sistemas de identificación de intrusiones (*IDS*) monitorizan el tráfico de red para el reconocimiento de *botnets*. Estos sistemas bien buscan patrones en los datos transmitidos que se puedan identificar con firmas previamente inventariadas, o bien detectan anomalías en el comportamiento normal de la red. También es posible una combinación de ambos. Se han aplicado desde 2006 diferentes técnicas de aprendizaje automático para la detección de intrusiones basadas en anomalías de comportamiento (Shinan et al., 2021).

Autores como Khraisat et al. (2019) y Shinan et al. (2021) recopilan las propuestas de *AIDS* basados en aprendizaje automático con métodos supervisados, no supervisados y semi-supervisados. Los principales métodos de aprendizaje automático utilizados son árboles de decisión, Naïves-Bayes, redes neuronales artificiales, máquinas de vectores de soporte y KNN, entre los supervisados, y K-medias entre los algoritmos de agrupamiento (clustering). En menor medida se han empleado algoritmos genéticos (Murray et al., 2014). La tabla 2.1 muestra un resumen histórico de diferentes trabajos que han empleado algoritmos de aprendizaje automático para la detección de *botnets*.

También se han realizado desde el año 2015 (Shinan et al., 2021) múltiples estudios en los que se implementan *AIDS* sobre una arquitectura *SDN* ubicados en la capa de aplicación o de control.

Niyaz et al. (2017) proponen el uso de *autoencoders* para la detección de ataques *DDoS* en redes *SDN*. El modelo propuesto utiliza un aprendizaje no supervisado del comportamiento normal de la red basado en *SAE*. El modelo es capaz de clasificar hasta ocho tipos de ataques distintos con un ratio muy bajo de falsos positivos. Sus experimentos obtienen una exactitud (*accuracy*) del 99,82 % al diferenciar entre el tráfico benigno y el malicioso. Obtiene un 95,65 % de exactitud en la detección de ataques *DDoS*.

Luo y Nagarajan (2018) han demostrado la viabilidad del uso de *autoencoders* en sensores *IoT* para la detección de anomalías. Proponen un modelo donde una réplica de un *autoencoder*, definido mediante una red neuronal de tres capas, reside en cada sensor *IoT*. Esto permite una detección distribuida y sin comunicación de datos entre los sensores.

Tabla 2.1: *Resumen de uso de aprendizaje automático en AIDS para detección de redes botnet.*

Año	Método	Métrica/Resultado	Protocolos
2006	Redes Bayesianas	Falsos positivos(*FPR*): 15.04 %	*IRC*
2008	*Clustering*	Exactitud(*accuracy*: 99.06 %	*IRC*, *HTTP*, *P2P*
2011	KNN, SVM, Naive-Bayes, Redes neuronales	Exactitud: 97 %	*P2P*
2012	*Ramdom Forest*, J48 (árboles de decisión), SVM	Verdaderos positivos (*TPR*): 65 %	*IRC*, *HTTP*, *P2P*
2014	C4.5 (Árboles decisión)	Exactitud: 75 %	*IRC*, *HTTP*, *P2P*
2016	C4.5 (Árboles decisión), *Ramdom Forest*	Exactitud: 99.98 %	*P2P*
2017	*Ramdom Forest*	Exactitud: 93.6 %	*IRC*, *HTTP*, *P2P*
2018	Redes neuronales convolucionales, *Autoencoders*	Verdaderos positivos (*TPR*): 91 %	*IRC*, *P2P*
2018	J48 (árboles de decisión), SVM	Exactitud: 95 %(IRC), 80 %(HTTP)	*IRC*, *HTTP*
2018	Árboles de decisión, redes neuronales, *Gausian Naive-Bayes*	*F1 score*: 99 %	*IRC*, *HTTP*, *P2P*
2019	Árboles de decisión, KNN, redes neuronales, regresión logística	Exactitud: 98.7 %	*P2P*
2019	Redes neuronales convolucionales LSTM	Exactitud: 97.8 %	*IRC*, *HTTP*
2019	Árboles de decisión, Naive-Bayes, redes neuronales	Exactitud: 94.4 %	*P2P*
2020	Árboles de decisión, redes neuronales, KNN, Naive-Bayes	Exactitud: 98.08 %	*IRC*, *HTTP*
2020	J48 (árboles de decisión), Naive-Bayes, redes neuronales	Exactitud: 99 %	*IRC*, *HTTP*, *P2P*
2020	*Gausian Naive-Bayes*	Exactitud: 99.5 %	*P2P*
2020	J48 (árboles de decisión	Exactitud: 99.94 %	*P2P*
2020	Redes neuronales recurrentes LSTM, MCFP (problema de flujo de coste máximo)	Exactitud: 99.36 %	*IRC*, *HTTP*
2020	Naive-Bayes, Árboles decisión	Exactitud: 99.6 %	*IRC*, *HTTP*, *P2P*

Fuente: generada a partir de tabla elaborada por Shinan et al. (2021)

El aprendizaje se realiza periódicamente en un sistema *Cloud* a partir de la información recogida por los propios sensores. Los nuevos parámetros obtenidos por el proceso de aprendizaje son actualizados en los sensores. La arquitectura propuesta permite que la red se vaya adecuando a la evolución en el comportamiento de la red. Sin embargo, debe considerarse que la red es vulnerable a ataques de envenenamiento de redes neuronales (*poisoning ANN*) similares a los descritos por Yang et al. (2017).

2.5. Análisis del comportamiento de la red

Los estudios del comportamiento de una red se ha dirigido tradicionalmente al análisis de un conjunto de características del flujo de comunicaciones de los dispositivos de la red.

Líneas de investigación prometedoras, según autores como Shinan et al. (2021), Chowdhury et al. (2017) y Daya et al. (2019), caracterizan el comportamiento de la red mediante grafos de comunicación.

2.5.1. Análisis del comportamiento de la red mediante el flujo de comunicación

Se denomina flujo de red a las conexiones, o canal de comunicación establecido, entre los diferentes dispositivos que la componen. El flujo queda definido por una tupla compuesta habitualmente por cinco características: las direcciones *IP* origen y destino, el protocolo y los puertos origen y destino (para aquellos protocolos que los requieran).

Las redes *SDN* presentan una capa de control donde un controlador recopila información del flujo de las comunicaciones de la red. Se emplea el protocolo de red *NetFlow* (Cisco, 2017) para la recopilación de esta información. Este algoritmo analiza las comunicaciones agregando múltiples atributos, como el número de paquetes de entrada y salida, y los asocia a cada tupla que identifica a un flujo. Se pueden establecer patrones de identificación de los paquetes transmitidos a partir de estos atributos. J. Zhang et al. (2011) y otros autores consideran que estos atributos se pueden considerar huellas digitales que distingan unos paquetes de otros o segmenten aquellos que compartan características comunes. Esta asunción ha sido la base de estudios que buscan estos patrones que diferencian a los flujos benignos de los flujos producidos por las *botnets*.

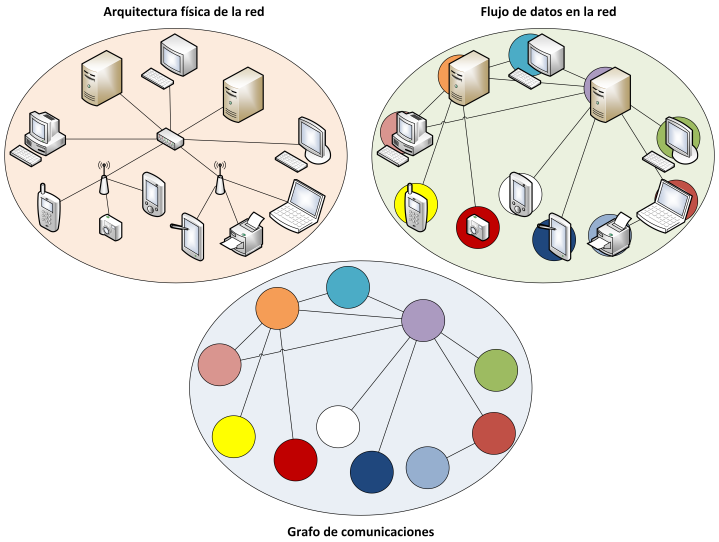
Los modelos propuestos capturan únicamente los efectos individuales de los ataques a un dispositivo, pues se basan en información estadística del flujo. Esto es una limitación según Chowdhury et al. (2017) al requerir la comparación de cada flujo con el resto de la red para determinar cuáles son anómalos. Además, permite que los atacantes eviten estas detecciones usando técnicas de encriptación o cambiado el volumen de información.

2.5.2. Análisis del comportamiento de la red mediante grafos de comunicación

Se analiza una representación de las comunicaciones en forma de un grafo (figura 2.10). Los nodos del grafo representan los dispositivos y los arcos corresponden a los flujos basándonos en cuatro características: las direcciones *IP* y puertos de origen y destino.

Se distinguen dos tipos de grafos para la detección de anomalías: estáticos y dinámicos. Los grafos estáticos pueden ser simples o contener atributos. En ambos casos hay autores que han propuestos modelos de detección anomalías basados en patrones de la

Figura 2.10: *Grafo de comunicaciones de una red.*



Fuente: Elaboración propia

estructura, o en patrones de las comunidades, o redes sociales. Los modelos propuestos en grafos dinámicos se basan en grafos que representan similitudes de propiedades relativas a la estructura (distribución, diámetro, etc.) descomponiéndolos en tensores o monitorizando la evolución en el tiempo de los grafos de comunidades.

Los grafos de comunicación evitan la comparación de flujos (Venkatesh et al., 2015), siendo un enfoque más eficiente. Chowdhury et al. (2017) proponen un enfoque rápido y que no requiere ninguna regla particular para la detección de *botnets*. Consiste en el cálculo de siete características:

a. entradas (*in degree*): número de flujos de entrada a un dispositivo. Es característico en las *botnets* un alto grado de flujos de entrada al servidor *C&C* debido a las conexiones iniciadas desde los *bots*.

b. salidas (*out degree*): número de flujos de salida desde un dispositivo. El servidor *C&C* y los *bots* tienen un grado de salidas debido al envío de órdenes y los intentos de reclutamiento de nuevos anfitriones vulnerables.

c. peso de las entradas (*in weight degree*): Número total de paquetes de entrada recibidos por un dispositivo de sus vecinos. Se asume que todos los *bots* de una red recibirán las mismas órdenes y, por tanto, recibirán el mismo número de paquetes desde el servidor *C&C*. Esto define un patrón que puede emplearse para segmentar a los componentes de una *botnet*.

d. peso de las salidas (*out weight degree*): Número total de paquetes de salida enviados por un dispositivo de sus vecinos. Se asume que los *bots* enviarán los mismos paquetes a los dispositivos que van a atacar.

e. coeficiente de segmentación (*clustering coefficient*): coeficiente que evalúa la cercanía existente entre los vecinos de un dispositivo. Se asume que las *botnets* de tipo *P2P* tienen un valor alto de este coeficiente.

f. intermediación entre dispositivos(*node betweenness*): número de veces que un dispositivo está en el conjunto compuesto por los caminos más cortos entre cada pareja de dispositivos. Se asume que las *botnets* de tipo *P2P* tienen un valor alto.

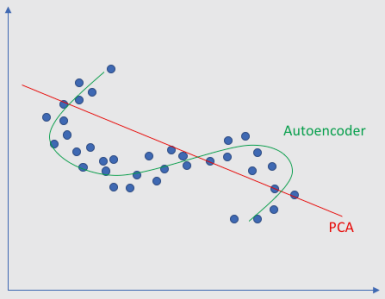
g. distancia entre dispositivos(*node closeness*): media de la distancia más corta desde todos los dispositivos que puedan alcanzar al que se mide. Esta medida es un factor relevante en la detección de *botnets* de tipo *P2P* (Sengupta et al., 2021).

h. Centralidad del vector propio (*eigenvector centrality*): es el peso del dispositivo en el grafo.

2.6. *Autoencoders* basados en redes neuronales

El concepto de *Autoencoders* fue introducido por primera vez por Rumelhart et al. (1986). Se trata de una red neuronal no supervisada que se entrena con el objetivo de extraer las principales características de la entrada de manera que pueda ser reconstruida. Las entradas son codificadas de forma que se comprime usando una función de pérdida de la información y, a continuación, se descodifica reconstruyendo la entrada con pérdida. La diferencia entre la entrada y salida se minimiza de manera que se realiza una reducción dimensional.

Figura 2.11: *Reducción no lineal respecto a la reducción lineal.*

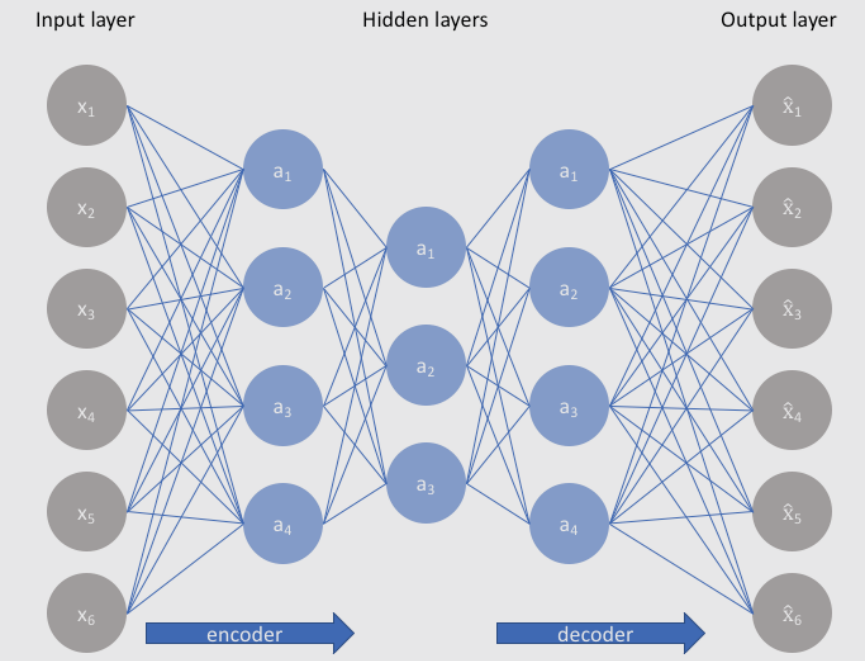


Fuente: Jordan (2016)

Bank et al. (2020) definen a los *autoencoders* como una generalización del análisis de las componentes principales (*PCA*) donde, en lugar de encontrar las relaciones lineales, aprende las no lineales (figura 2.11).

Los *autoencoders* basados en redes neuronales se componen de una capa de entrada, una o varias capas ocultas, denominadas *cuello de botella* o *bottleneck*, y una capa de salida con igual dimensión que la entrada. La figura 2.12 muestra una representación de la estructura de un *autoencoder* basado en redes neuronales.

Figura 2.12: *Estructura de un autoencoder.*



Fuente: Jordan (2016)

Una de las desventajas de los *autoencoders* basados en redes neuronales es que memoricen la entrada debido a la capacidad de aprendizaje de las mismas. Por tanto, se debe buscar un equilibrio de manera que el *autoencoder* sea capaz de alcanzar la exactitud buscada en la reconstrucción de la entrada sin producirse sobre aprendizaje. Se emplean técnicas de regularización de la red neuronal de manera que se reduce su capacidad de aprendizaje.

La regularización se puede realizar mediante una dimensión menor de la capa oculta, respecto a las capas de entrada y salida, o provocando una activación selectiva de las neuronas de las capas intermedias.

Los *autoencoders* dispersos (*sparse autoencoders*) activan las neuronas de las capas intermedias de manera selectiva. La elección de las neuronas activas se puede realizar bien aplicando técnicas similares a la regularización *L1* (Schmidt et al., 2009) o usar *KL-*

divergence (Ng et al., 2011).

Los *autoencoders* apilados (*stacked autoencoders*) son aquellos entrenados por capas de forma que cada capa de la etapa de codificación es la entrada a un *autoencoder* más interno hasta alcanzar el nivel más profundo de la red. De igual manera, las capas de la etapa de codificación se consideran la salida de un *autoencoder* más interno.

Los *autoencoders* tienen diferentes aplicaciones, como la reducción de la dimensionalidad, detección de anomalías, sistemas de recomendación, segmentación (*clustering*), clasificación y modelo generador (Bank et al., 2020).

Investigadores como Mirsky et al. (2018), Luo y Nagarajan (2018) y previamente C. Zhou y Paffenroth (2017), han propuesto modelos para la detección de anomalías en el dominio de la ciberseguridad.

2.7. Resumen de conclusiones

Los ataques de *botnets* continúan suponiendo la mayor amenaza a la seguridad, atendiendo a consideraciones como el volumen de ataques. El crecimiento de las redes *IoT*, conjuntamente con el menor nivel de seguridad, ha incrementado el riego. Los requisitos de bajo consumo y capacidad reducida de cómputo limitan en la práctica sus implementaciones de seguridad.

Los diferentes estudios descritos han obtenido buenos resultados en redes tradicionales, pero no son completamente aplicables en su conjunto a las redes *IoT* debido a sus especificidades. Este trabajo propone una implementación de un sistema de detección de intrusiones basado en anomalías que puede integrarse en redes definidas por *software* en el dominio de las redes *IoT*. La detección de anomalías se realizará mediante *stacked autoencoders* que caracterizarán el comportamiento normal de la red a partir de grafos de comunicaciones. Ambas técnicas son líneas prometedoras de investigación que combinadas pueden reducir los requisitos computacionales y de consumo necesarios para la detección de ataques *botnets*. El uso de algoritmos de aprendizaje automático no supervisado facilita el proceso de aprendizaje y ajuste periódico del modelo de detección de anomalías en el comportamiento de la red.

Capítulo 3. Objetivos

Este capítulo describe el objetivo principal del trabajo y una relación de objetivos más específicos. Un último apartado desarrolla la metodología seguida para verificar su consecución.

3.1. Objetivo general

El objetivo general del trabajo es desarrollar un prototipo de *AIDS* para la detección de ataques *botnets* en redes *IoT*, que analizará y tomará decisiones basándose en un grafo de comunicaciones de red, reduciendo en al menos un 30 % el volumen de datos a analizar y alcanzando una exactitud superior al 90 %.

Se partirá de diferentes propuestas realizadas por la comunidad académica y la industria de seguridad en el entorno de redes de comunicaciones, explorando algunas de las líneas prometedoras enunciadas por autores relevantes en la materia.

Para ello, el desarrollo *software* a realizar se encargará, en primera instancia, de elaborar un conjunto de características del comportamiento de la red a partir de los grafos de comunicaciones. Estos grafos serán generados a partir del flujo de comunicaciones similares a los recopilados por un *SDN*. Esta elaboración permitirá una reducción del volumen de información a analizar, siendo relevante en un entorno donde los recursos son limitados.

En segunda instancia, se determinará que tráfico es producido por una *botnet* mediante un *autoencoder* que analizará las características previamente elaboradas. Estos *autoencoders* podrán ser ubicados en dispositivos especializados en la detección de tráfico malicioso, pero como algunos autores han demostrado, podrían ubicarse incluso en los propios dispositivos *IoT*.

El prototipo será evaluado con tráfico de redes *IoT* realista dónde concurre tráfico normal y tráfico producido por *botnets*. Esto permitirá determinar su eficacia en laboratorio.

3.2. Objetivos específicos

- Determinar los componentes necesarios de las arquitecturas *SDN* con los sistemas *AIDS* que emplean algoritmos de aprendizaje automático.
- Identificar al menos seis características de los gráficos de comunicaciones que permiten establecer modelos de aprendizaje automático para la detección de ataques desde *botnets*.
- Evaluar la fiabilidad de *stacked autoencoders* en la detección de anomalías sobre el comportamiento de una red de comunicaciones. midiendo la exactitud y la precisión.
- Facilitar la reducción de recursos requeridos para la detección de ataques de *botnets* en un entorno de redes *IoT*, usando grafos de comunicaciones y comparando el volumen del flujo de red con el volumen de información a procesar por la solución propuesta para el aprendizaje.
- Cuantificar la eficacia de un sistema de detección de anomalías basado en grafos de comunicación con *autoencoders* sobre una red de dispositivos IoT realista.
- Verificar que la solución propuesta es una alternativa viable al problema de investigación estudiado, generando un modelo de predicción que alcance los umbrales establecidos.
- Explorar la línea de investigación prometedora basada en grafos de comunicación para el aumento de la seguridad en las redes IoT.

3.3. Metodología del trabajo

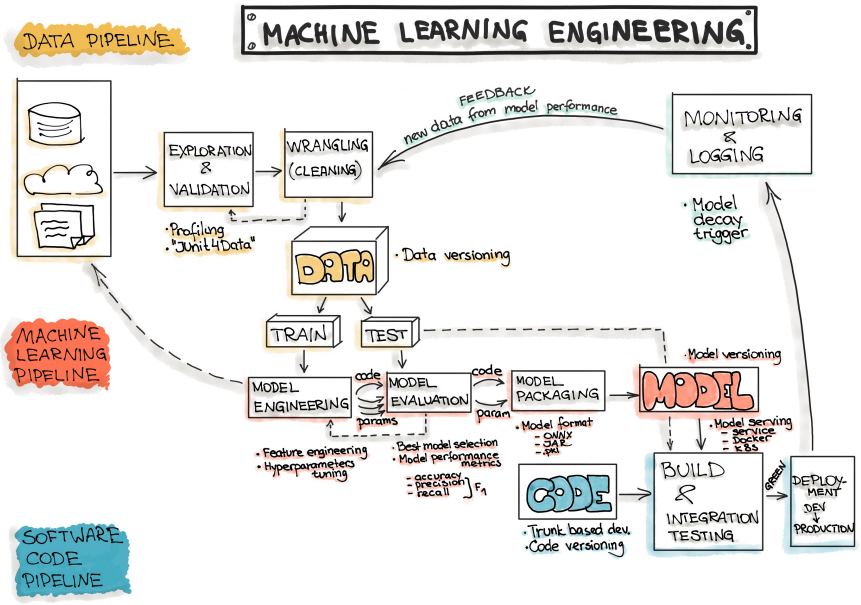
La hipótesis de este trabajo incide en el uso de técnicas de aprendizaje profundo funcionales en dispositivos *IoT*. Además, se pretende comprobar que los grafos de comunicaciones representan el problema con un menor volumen de información, obteniéndose modelos con una buena capacidad de predicción. Se van a realizar un conjunto de experimentos que permitan confirmar o refutar la hipótesis del trabajo.

Las metodologías clásicas para la gestión del ciclo de vida de aplicaciones no se adaptan completamente a productos focalizados en modelos de aprendizaje automático. Este prototipo seguirá un ciclo de vida *MLOps*, que extiende la metodología *DevOps*, poniendo

el foco en las tareas de ingeniería asociadas a la modelización con aprendizaje automático. Los principios básicos de esta metodología son: la automatización, la reproducibilidad, el despliegue y la monitorización.

Este proyecto no cubre todas las etapas *MLOps* debido al alcance establecido. Se centra en los procesos de ingeniería asociada al aprendizaje automático (*MLE*).

Figura 3.1: *Flujo de trabajo para la elaboración del prototipo.*



Fuente: MLOps.org (2022b)

La figura 3.1 muestra los tres *pipelines* que se abordan en el proyecto y el flujo de trabajo a seguir, exceptuando los trabajos de despliegue y monitorización.

a. El *pipeline* de datos se encarga de la adquisición, exploración, validación y preparación de los conjuntos de datos que van a usarse en la generación de los modelos.

   La fuente de datos de este proyecto es el dataset *Aposemat IoT-23*. Se trata de un conjunto de datos compuesto por flujos de red ya etiquetados. Este conjunto de datos debe procesarse para obtener los grafos de comunicaciones asociados y generar, a partir de ellos, los conjuntos de datos a emplear en el modelo.

   Los diferentes conjuntos de datos que se obtengan como resultado de estos trabajos de análisis, validación y preparación serán versionados con la herramienta *DVC* (*Open-source Version Control System for Machine Learning Projects*).

b. El *pipeline* de aprendizaje automático comprende la preparación, la evaluación y el empaquetado del modelo para su uso posterior.

El modelo basado en *autoencoders* no es supervisado. Esto facilita volver a entrenar el modelo cuándo decae su exactitud (*model decay*). Sin embargo, se parte de un conjunto de datos etiquetado que permite usar las etiquetas a modo de *ground truth*. Se establecerán métricas habituales en los aprendizajes supervisados para la evaluación de la capacidad de predicción del modelo.

Los conjuntos de datos disponibles serán empleados para la generación del modelo. Se van a dividir en 80 % para aprendizaje y 20 % para validación. Esta separación se va a realizar manteniendo el balance de tráfico benigno y malicioso del conjunto de datos original.

Se van a realizar cinco divisiones distintas para poder hacer una validación cruzada de los resultados. El *accuracy* y *F1-score* global del modelo tras los cinco *folds* se obtendrá mediante la media aritmética.

Se realizará una búsqueda aleatoria de hiperparámetros con la finalidad de encontrar la mejor configuración para el modelo.

Se obtendrán modelos de *stacked autoencoders* que representan el comportamiento de una red *IoT*. Por lo tanto, se establecerán mecanismos de trazabilidad y reproducibilidad de los diferentes modelos ensayados. La herramienta *mlflow* (*Open source platform for the machine learning lifecycle*) permitirá el versionado y trazabilidad de los modelos.

c. El *pipeline* de *software* se encarga de la codificación de los programas que integran el modelo en el producto final.

   Se desarrollarán algoritmos de elaboración de las características del comportamiento de una red a partir de grafos de comunicación. Estos desarrollos tomarán los flujos de comunicaciones capturados de una red y los transformarán en un grafo de comunicaciones en forma de *dataset* explotable por algoritmos de aprendizaje profundo. Se desarrollará un *API* con servicios que podrán integrarse con una red *SDN*.

   El desarrollo permitirá la automatización de la preparación del *dataset* que, posteriormente, se usará en el aprendizaje del modelo de detección de ataques de *botnets*. Se contará con métricas básicas que permitan evaluar la reducción del volumen esperado por el uso de grafos de comunicaciones.

   Existirán desarrollos específicos para la exportación de los modelos. Estas exportaciones serán utilizadas para la ejecución en los dispositivos que vayan a realizar los trabajos de detección de ataques.

Se desarrollarán servicios que determinen los flujos maliciosos a partir del tráfico de red recibido en entrada. Se deberá transformar el tráfico de red en entradas para el modelo aprendido. Esta transformación consiste en obtener el grafo de comunicaciones a partir del flujo de datos.

El código generado estará versionado en un repositorio *github*.

La metodología *MLOps* define un *canvas* genérico para los proyectos de modelado con aprendizaje automático basado en la propuesta de Agrawal et al. (2018). La tabla 3.1 muestra la caracterización para este proyecto.

Tabla 3.1: *Canvas de aprendizaje automático para el proyecto.*

Bloques	**Objetivos**	**Alcance en el proyecto**
Propuesta de valor	Definir el problema, su importancia y sus usuarios	*AIDS* con un menor coste computacional para redes *IoT*
Fuentes de datos	Identificar las fuentes esenciales	Los flujos de datos de la propia red
Tarea de predicción	Tipo de modelo a usar	*Stacked autoencoder*
Características (Ingeniería)	Cómo se van a representar los datos de entrada	Grafos de comunicaciones
Evaluación fuera de línea	Especificar y configurar los métodos y las métricas para evaluar el sistema	*Accuracy* y *F1 score* Función de pérdida *MSE*
Decisiones	Cómo se utilizan las predicciones para tomar decisiones	Generación de alertas al detectar flujos maliciosos
Haciendo predicciones	Cuándo y cómo se realizan las predicciones	Lotes de flujos de datos de red de forma periódica
Recolectando datos	Coste de recuperar y etiquetar nuevos datos	No requieren etiquetado
La construcción de modelos	Frecuencia y coste de volver a entrenar	Se vuelve a entrenar periódicamente
Evaluación y monitorización	Cómo se supervisa el rendimiento del sistema	Métricas sobre los ataques con supervisión humana

Fuente: generada a partir del *canvas* propuesto por MLOps.org (2022a)

Capítulo 4. Identificación de Requisitos

Los capítulos anteriores han descrito los problemas que suponen las redes *botnets* para la seguridad de las redes de comunicaciones en general y las redes de dispositivos *IoT* en particular. La tendencia actual en el ámbito de la seguridad es afrontar este desafío incorporando un *AIDS* en la capa de aplicaciones de redes *SDN*. Este sistema de detección de anomalías se comunica con el controlador ubicado en el plano de control de estas redes para recopilar los datos de entrada utilizadas en el análisis de amenazas.

Los *AIDS* son sistemas complejos compuestos de diversos componentes que se encargan de la monitorización de eventos en la red, el análisis de los eventos capturados y activar respuestas a las amenazas activadas. Además, los *AIDS* que incorporan modelos de aprendizaje automático cuentan con procesos para entrenar periódicamente el modelo, adaptándose a las condiciones locales y temporales de la red. Todos estos componentes se distribuyen en diferentes dispositivos de la red trabajando de forma coordinada.

El alcance temporal del proyecto no permite desarrollar todos los componentes de un *AIDS* completo integrado en un *SDN*.

Se desarrollarán los componentes de un *AIDS* que permitan verificar los beneficios asociados a la detección de ataques con *autoencoders*. Se analizan métricas de los grafos de comunicaciones generados a partir del flujo de datos de una red con una arquitectura *SDN* para redes *IoT*. Se ha dejado para futuros trabajos la integración y experimentación en un *SDN* real al no estar entre los objetivos de este trabajo.

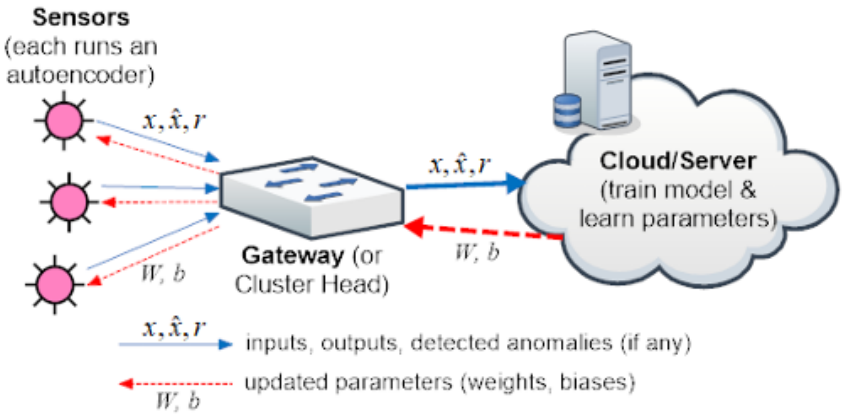
Los siguientes apartados describen los requisitos globales del desarrollo *software* que se realizará y otros requisitos más específicos. Estos requisitos se han establecido a partir del estudio del arte realizado, no habiendo sido posible consensuar con expertos en la materia la validez de los mismos o la necesidad de requisitos adicionales.

4.1. Requisitos globales

El prototipo de sistema *AIDS* deberá detectar ataques realizados desde redes *botnets*. El desarrollo deberá poder integrarse en un *SDN* para redes *IoT*. En concreto, será una aplicación adicional a las incluidas en la capa de aplicaciones de la arquitectura *SDN*.

El *AIDS* tendrá la arquitectura propuesta por Luo y Nagarajan (2018) (figura 4.1).

Figura 4.1: *Arquitectura de AIDS con autoencoders en redes IoT.*



Fuente: Luo y Nagarajan (2018)

Las tareas de sondeo de la actividad de la red en busca de ataques se realizará por todos los dispositivos que dispongan de la capa de aplicación, incluyendo los dispositivos *IoT*. El componente de detección empleará un modelo de *autoencoder*. Por tanto, todos los dispositivos que realicen las tareas de sondeo contarán con una copia del modelo. Este requisito es importante para distribuir la carga computacional del proceso de detección y para reducir los riesgos de indisponibilidad de este servicio de seguridad.

Los sondeos serán realizados periódicamente por los dispositivos. Se evaluarán las métricas calculadas en un grafo de comunicaciones generado a partir de los flujos de datos de la red. Un dispositivo necesitará únicamente las métricas de los nodos del grafo de comunicaciones asociadas al propio dispositivo. El grafo representará en un nodo cada conexión y puerto. Así, un dispositivo deberá evaluar las métricas asociadas a cada puerto que emplea para comunicarse.

La latencia de este sondeo deberá ser adecuado para evitar un aumento de consumo no asumible por los dispositivos *IoT*. Los grafos de comunicaciones facilitan aumentar la latencia. Se reduce el espacio de almacenamiento requerido al representar mediante una tupla el comportamiento de las comunicaciones de un dispositivo a través de uno de sus puertos.

Los resultados de las evaluaciones de cada dispositivo serán enviados a los servidores destinados a la toma de decisiones. Este envío se realizará de nuevo periódicamente para reducir el tráfico red y el consumo de los dispositivos. Una única transmisión tiene un menor consumo energético que múltiples envíos, aún con un mayor volumen de datos. La latencia de esta transmisión deberá ser por tanto superior a la definida para el sondeo. El dispositivo deberá almacenar la entrada y salida de cada evaluación realizada hasta que se realice la siguiente transmisión. Esto supondrá almacenar una matriz con tantos elementos como evaluaciones realizadas.

Los servidores de toma de decisiones se encargarán de evaluar los resultados obtenidos por los sensores y generarán las alarmas definidas. Un *AIDS* incorporado a un *SDN*, que disponga de lenguajes estructurados de dominio (*SDL*), podrá automatizar la generación de políticas dinámicas que se adapten a los cambios en el comportamiento de la red. Por ejemplo, las redes con controladores *Pyretic*, de la familia *Frenetic* (Foster et al., 2011), disponen del lenguaje *Kinetic* (Kim et al., 2015). Este lenguaje embebido en *Python* permite la automatización de políticas dinámicas mediante eventos controlados con una máquina de estados finitos.

Los servidores de toma de decisiones deberán actualizar el modelo periódicamente con la información recibida por los sensores. La adaptación del modelo a los cambios de la red se podrá automatizar. Es importante esta actualización del modelo para reducir los problemas de decaimiento de los resultados de predicción. La latencia deberá ser igual o mayor a la latencia de recepción de resultados de los sensores. El modelo obtenido en el nuevo entrenamiento será distribuido a todos los dispositivos.

4.2. Requisitos específicos

El desarrollo del *software* se realizará en el lenguaje de programación *Python* versión 3.10.4. Serán necesarias un conjunto de librerías para la implementación de las funcionalidades. La tabla 4.1 presenta las librerías y versiones requeridas.

Este desarrollo está centrado en un modelo de aprendizaje automático, donde el principal requisito es la existencia de datos suficientes y de calidad. También, la localidad (*locality*) es un factor adicional a considerar en los modelos de aprendizaje automático. Un modelo no siempre se puede transferir sin adaptación a otra localización. Por tanto, el modelo se deberá entrenar a partir de un tráfico de red realista de dispositivos *IoT*. Estas redes tienen topologías y una gestión de dispositivos diferentes a otros tipos de redes.

El modelo se generará con *stacked autoencoders*. Se entrenará con un conjunto de

Tabla 4.1: *Librerías requeridas para el desarrollo del proyecto.*

Librería	Versión	Función de uso
dvc	2.10.2	Versionado de conjuntos de datos usados en la generación del modelo
flask	2.1.2	*Framework* para aplicaciones *Web*
flask-Cors	3.0.10	Librerías para gestión del CORS de la consola *Web*
Flask-RESTful	0.3.9	Extensión de Flask para *API RESTful*
graphviz	0.20	Visualización de gráficos
matplotlib	3.5.1	Presentación de gráficos
mlflow	1.25.1	Trazabilidad y versionado de modelos de aprendizaje automático
networkx	2.8	Creación, manipulación y estudio de redes complejas
numpy	1.22.3	Cálculos matriciales
pandas	1.4.2	Análisis y manipulación de datos
pydot	3.0.8	Realiza *plots* al estilo de *MATLAB*
requests	2.27.1	Solicitudes de peticiones RESTFUL
sklearn	0.0	Utilidades de aprendizaje automático
tensorflow/keras	2.8.0	Generación del modelo de *autoencoders*

Fuente: Elaboración propia

datos compuesto por las métricas calculadas a partir del flujo de datos recibido en cada puerto y dispositivo de la red. El conjunto de datos contendrá las siguientes características: *in degree*, *out degree*, *in weight degree*, *out weight degree*, *clustering coefficient*, *node betweenness*, *node closeness* y *eigenvector centrality*.

Se deberá normalizar todas las características seleccionadas para el aprendizaje, tomando únicamente valores bajos pertenecientes al dominio de los números reales. Es una buena práctica en general en el aprendizaje automático y ayuda a mejorar el comportamiento de los modelos de redes neuronales.

Los sensores evaluarán el modelo cada 2 minutos y enviarán al servidor central los resultados obtenidos cada hora. La latencia de revisión del modelo se realizará cada semana, siendo actualizado en los sensores con esta misma latencia. Estos valores de latencia serán configurables.

La detección de anomalías se realizará con un esquema de clasificación binaria. No se hará distinción entre diferentes tipos de ataque. Se comprobará dos posibles configuraciones del modelo de *autoencoder*.

El primer escenario configurará un *autoencoder* con la etapa de codificación y la etapa de decodificación. La salida se clasificará como anómala cuando el error en la reconstrucción de la salida supere un umbral de certidumbre. El entrenamiento en este escenario se

realizará únicamente con ejemplos de tráfico benigno.

El segundo escenario configurará un modelo dónde se mantiene únicamente la etapa de codificación obtenida después del aprendizaje del *autoencoder*. El conjunto de datos obtenido por el codificador serán etiquetados con la clase original y se entrenará con regresión logística. Este caso se entrenará con tráfico benigno y malicioso.

Ambos escenarios se evaluarán mediante matrices de confusión, obteniendo el valor de *accuracy* y el valor de *F1-Score* de los resultados de ambas clases.

Los entrenamientos periódicos se realizarán con los datos de fechas anteriores, agregando los resultados de clasificación recopilados desde los sensores. Se establecerá una ventana temporal de manera que las observaciones que superen una semana no se incluirán en el nuevo entrenamiento.

El primer escenario no requiere ningún tipo de etiquetado para volver a entrenar periódicamente el modelo. Solo se incluirán en el proceso de aprendizaje aquellos comportamientos clasificados como benignos. El segundo escenario realizará el etiquetado con los resultados obtenidos por los sensores. Este etiquetado deberá supervisarse por humanos.

La monitorización del funcionamiento del modelo deberá cumplir las siguientes reglas:

a. El detalle de tráfico malicioso detectado se pondrá a disposición de un servicio encargado de automatizar la respuesta del sistema.

b. Se generarán alertas para la supervisión humana cuándo el porcentaje de detecciones supera un umbral establecido.

c. No se actualizará el modelo de manera automática si se obtiene una exactitud inferior al 90 %, salvo que mejore la exactitud previa.

Capítulo 5. Descripción de la herramienta software desarrollada

Este capítulo presenta el desarrollo de *software* que se ha realizado con sus diferentes etapas. Se ha seguido una metodología *MLops* como ciclo de vida del prototipo.

Los apartados siguientes muestran una descripción de la arquitectura, el funcionamiento del prototipo y una descripción de las tareas realizadas en los tres *pipelines* de la metodología de trabajo.

5.1. Arquitectura y funcionamiento del prototipo

El desarrollo de *software* consiste en un prototipo de los componentes principales de un sistema *AIDS*. La figura 5.1 presenta la arquitectura del prototipo y las principales líneas de su funcionamiento. Muestra la integración en una red *SDN* y, además, representa la relación de los diferentes actores en un ciclo de vida de aplicación con metodología *MLops*.

El ciclo de vida de estas aplicaciones es continuo y cíclico desde la concepción hasta su fin de vida. Se alternan los trabajos de análisis de datos y modelado con su operación en un entorno real.

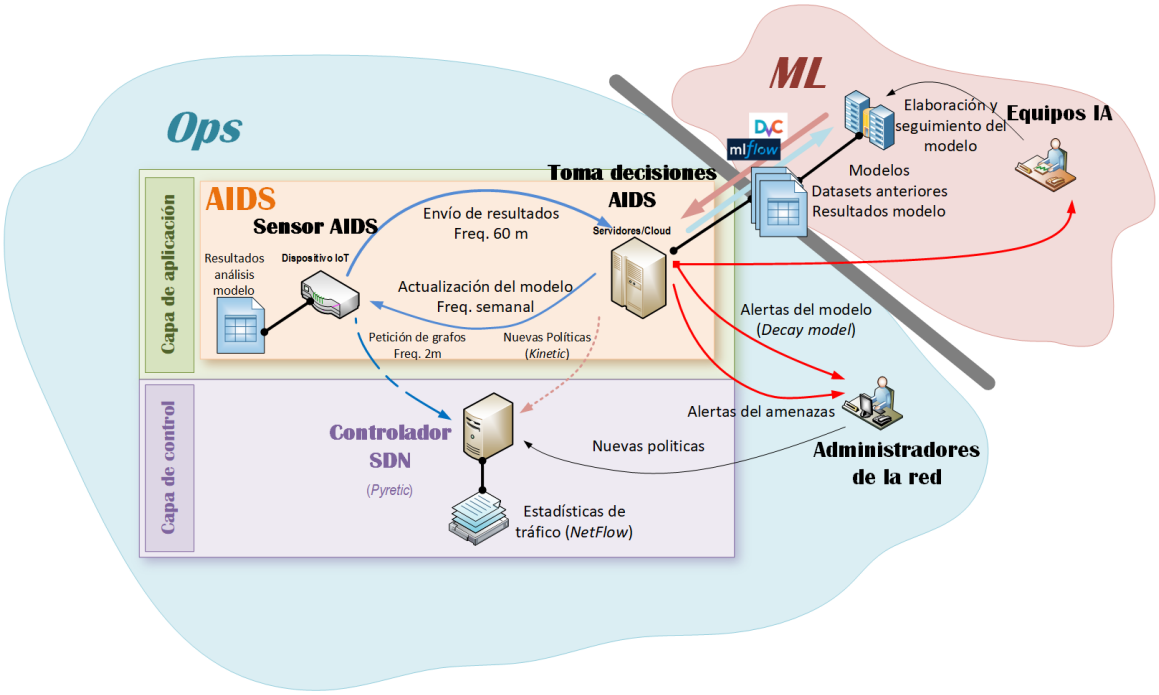
La primera etapa del ciclo de vida constará de tres *pipelines* de trabajo que se describen en detalle a lo largo de los siguientes apartados de este trabajo. Este trabajo solo cubre esta primera etapa del ciclo de vida.

El desarrollo es un prototipo con una finalidad investigadora. Sin embargo, se han incluido componentes, como parte del *pipeline* de codificación, que permiten la simulación del funcionamiento del sistema en una etapa de operaciones de una red *SDN* real.

La arquitectura planteada integra el prototipo de *AIDS* en la capa de aplicación de una red *SDN*. El sistema presenta diferentes componentes, siendo el sensor de monitorización y el servidor de toma de decisiones los más importantes.

El sensor de monitorización se ejecuta en todos los dispositivos con capa de aplicación,

Figura 5.1: *Arquitectura propuesta y funcionamiento del prototipo.*



Fuente: Elaboración propia

incluyendo los dispositivos *IoT*. Se encarga de evaluar el comportamiento del dispositivo que alberga al propio sensor. Engloba a un conjunto de procesos con ejecución periódica y tienen la finalidad de:

I. Analizar el comportamiento del dispositivo en la red. Comprueba si es anormal el patrón de las comunicaciones realizadas en sus puertos. Este sondeo se realiza a intervalos de 2 minutos. El análisis requiere disponer de las métricas asociadas al dispositivo en forma de grafo de comunicaciones de la red y una copia del modelo de *stacked autoencoder* entrenado.

   La información a analizar se solicita al controlador ubicado en el plano de control de la red *SDN*. Un controlador *Pyretic* sería adecuado para una mayor abstracción del sensor de la estructura de la red y facilitaría el desarrollo de la interfaz de consulta con lenguajes como *Kinetic*.

   El prototipo invocará a un servicio simulado de un controlador de red *SDN*. Este servicio entregará los datos de comportamiento que debe evaluar el sensor. La simulación consiste en recuperar datos preparados a partir de tráfico real de los *datasets IoT-23*. Esta información se tomará aleatoriamente del tráfico malicioso o benigno asociado al dispositivo. El tráfico usado no habrá participado en el proceso de preparación del modelo.

Los resultados de la evaluación, asociada a los datos de entrada, son almacenados por el sensor para su posterior envío al servidor de toma de decisiones.

II. Comunicar resultados al servidor de toma de decisiones. Se realiza cada hora enviando todos los resultados de las evaluaciones realizadas desde la comunicación anterior. La información comunicada se deja de persistir en el sensor.

III. Solicitar la versión más actualizada del modelo de *stacked autoencoder* al servidor de toma de decisiones. La solicitud se realiza semanalmente. La nueva versión es descargada en el sensor, actualizando el modelo. La latencia de solicitudes pasa a ser cada hora cuando no se haya obtenido una nueva versión en la primera llamada. Se retorna a una latencia semanal al completar la actualización.

El servidor de toma de decisiones es el componente encargado de recopilar los resultados del análisis de la red realizados por los sensores y determinar las actuaciones a realizar. Las funciones principales del servidor de toma de decisiones son:

I. Analizar los resultados comunicados desde un sensor y notificar los resultados. La relación de dispositivos y puertos, con comportamiento malicioso, es enviada a un servicio dedicado a automatizar la generación de políticas dinámicas de seguridad. Se generan notificaciones y se almacenan en la consola de monitorización de los administradores de la red. El componente de automatización es un servicio del *AIDS* que no será desarrollado de manera funcional en el marco de este proyecto.

   El analizador genera una alerta específica para los administradores de red cuándo el ratio de resultados maliciosos sobre los benignos supera un umbral. Esta situación requiere una atención especial por parte de los administradores de la red. Se puede deber a un mal funcionamiento del modelo ante un patrón de comportamiento legítimo (*model decay*) o a un ataque intenso a la red. Los administradores pueden determinar las contra medidas adicionales a las adoptadas por el servidor de toma de decisiones. Se emite una alerta a los equipos de trabajo responsables del modelo para que analicen si se requiere su ajuste. Se genera una alerta similar cuándo el sistema no detecta comportamiento anómalo durante un periodo prologando de tiempo. Esta situación puede deberse a una reducción de la sensibilidad del modelo a los ataques.

II. Unificar todos los resultados recibidos por los sensores segregándolos por el tipo: benigno o malicioso. Esta información será la entrada de los procesos semanales de

entrenamiento del modelo. Estos datos se ponen a disposición de servidores dedicados al entrenamiento del modelo. Estos entrenamientos se realizarán con datos de entrenamientos anteriores y los nuevos datos. Una parte de los datos de entrenamientos anteriores se descarta de acuerdo a una ventana temporal establecida sobre la fecha en la cual se obtuvieron. El objetivo es evitar que el sistema se sobre ajuste al comportamiento de una semana concreta. Los servidores dedicados al entrenamiento se encargarán de generar un nuevo modelo bajo la supervisión de los científicos de datos. Estos podrán realizar exploración y análisis de los nuevos datos recabados, iniciándose una nueva iteración del ciclo de vida *MLops*.

III. Poner a disposición de los sensores los nuevos modelos obtenidos periódicamente. El prototipo se ha simplificado exponiendo servicios que pueden consumir los sensores para actualizarse. La actualización en un *AIDS* real debería ser gobernada por el servidor de toma de decisiones con envíos de tipo *broadcast* a los dispositivos de la red.

Todos los procesos previos se describen con más detalle en apartados posteriores de este capítulo.

5.2. Preparación de datos para el modelo

Este *pipeline* está compuesto por tareas dedicadas a la preparación de los conjuntos de datos destinados al entrenamiento del modelo. Se debe determinar las fuentes de datos a utilizar y la frecuencia de actualización de las mismas. Además, se debe realizar un análisis exploratorio de los datos, determinando su validez y las adaptaciones necesarias para su posterior utilización. Se seleccionarán los datos más adecuados realizando una preparación antes de comenzar el entrenamiento del modelo.

La fuente de información elegida es el conjunto de datos *IoT-23* (Garcia et al., 2020) compuesta por 23 ficheros con 20 escenarios que incluyen flujos de datos maliciosos y 3 escenarios con únicamente flujos benignos. Las capturas se han realizado con dispositivos *IoT* reales durante 24 horas en varias sesiones. Los escenarios maliciosos incluyen tráfico de diferentes tipos de *malware*, actuando con varios protocolos de comunicación.

Los ficheros *IoT-23* se han elaborado a partir de las capturas recogidas en ficheros *pcap*. Los registros han sido etiquetados mediante un conjunto de reglas definidas por analistas humanos. Se pueden encontrar las siguientes etiquetas: *Attack* indica algún tipo de ataque desde el dispositivo, *Benign* indica la falta de sospecha de actividad maliciosa,

C&C indica que el dispositivo se ha conectado a un *C&C*, *DDoS* indica que el dispositivo está efectuando un ataque de denegación de servicios distribuido, *FileDownload* indica la descarga de *malware* al dispositivo, *HeartBeat* indica que se han enviado paquetes de traza desde el servidor *C&C*, *PartOfAHorizontalPortScan* indica una búsqueda horizontal de puertos para recopilar información, *Mirai*, *Okiru* y *Torii* indican conexiones características de esas *botnets*.

El resultado de la exploración previa del contenido de los ficheros muestra que la información, monitorizada en los flujos de la red, es suficiente para obtener grafos de comunicaciones que representen el comportamiento de la red. Existen valores nulos o vacíos que no afectan a los tratamientos de transformación. Por tanto, no se requiere un tratamiento de los valores nulos.

Los ficheros presentan un formato generado, a partir de los ficheros *pcap* originales, por el analizador de red *Zeek*.

Los trabajos de preparación de los *datasets* para el entrenamiento de modelos es costoso desde un punto de vista computacional. Se ha decidido no generar los ficheros definitivos en un solo proceso de transformación. Se va a realizar de manera escalonada para evitar reprocesamientos completos, si hay que hacer ajustes en la preparación de los datos. Se realizará en tres pasos: i) transformación a formato *csv*, ii) transformación de tráfico de datos a su representación mediante un grafo de red (dónde se genera un *csv* con las medidas de centralidad asociadas) y iii) la normalización de las medidas que se almacenan en el *csv* a utilizar en los modelos.

Estos ficheros se han transformado al formato *csv* por el *script Python* io23_2_csv.py desarrollada con este fin. Este *script*, además de transformar el formato, genera dos ficheros *csv* adicionales con el tráfico benigno y malicioso segregado (figura 5.2).

Figura 5.2: *Ejecución del script Python io23_2_csv.py.*

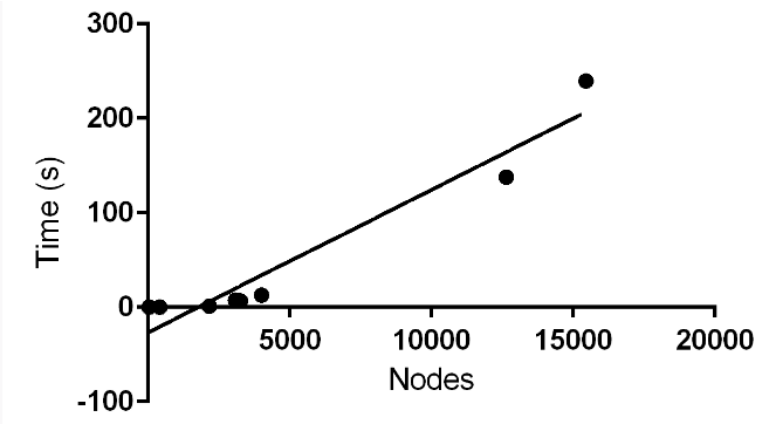
```
PS C:\ESTUDIOS\UNIR\MO_Inteligencia Artificial\C24_Trabajo Fin de Máster (12 ECTS)\workspace\saeDBot> & C:/Python310/python.exe "C:\ESTUDIOS\UNIR\
MO_Inteligencia Artificial\C24_Trabajo Fin de Máster (12 ECTS)\workspace\saeDBot\src\io23_2_csv.py" "X:\UNIR\iot23\bro" -S=1 -A=SC
Order action "SC" for files *.labeled (in path "X:\UNIR\iot23\bro") with limit size of 1000000 bytes.

Starting process of file 'X:\UNIR\iot23\bro\44-01_IoTMalware.labeled' .....
Total rows..... 237
Total rows in benign ..... 211
Total rows in malicious ... 26
Ended process of file 'X:\UNIR\iot23\bro\44-01_IoTMalware.labeled' .....

Starting process of file 'X:\UNIR\iot23\bro\04-01_Honeypot.labeled' .....
Total rows..... 452
Total rows in benign ..... 452
Total rows in malicious ... 0
Ended process of file 'X:\UNIR\iot23\bro\04-01_Honeypot.labeled' .....
```

Fuente: Elaboración propia

La versión *Python* del paquete *Networkx* se ha empleado para representar de los flujos de datos en forma de grafo de comunicaciones. Esta librería permite la representación del

grafo y, adicionalmente, dispone de funciones para la obtención de las métricas requeridas. La implementación de estos cálculos utiliza algoritmos eficientes. Aun así, la métrica *betweenness centrality* tiene unos requisitos computacionales muy elevados (Brandes, 2001). No ha sido posible obtener los recursos de computación necesarios para la generación de esta métrica para todos los nodos del grafo. El algoritmo presenta una complejidad logarítmica. Este patrón se ha podido observar en el seguimiento de las ejecuciones realizadas con conjuntos de datos con un bajo volumen de muestras (figura 5.3).

Figura 5.3: *Patrón de complejidad temporal del cálculo de betweenness centrality.*



Fuente: Elaboración propia

Se puede realizar una aproximación del cálculo *betweenness centrality* mediante la selección de una muestra aleatoria de los nodos destino a considerar en cada nodo del grafo. La métrica *local clustering coefficient* también presenta tiempos de cálculo elevados. Además, se ha constatado que el filtrado aplicado sobre los ficheros *pcap*, se traduce en grafos con topologías que no recogen esta propiedad del comportamiento de la red. El filtrado de las capturas está orientado a recoger los flujos de entrada y salida de los dispositivos *IoT*, omitiendo flujos entre otros dispositivos de la red.

Se ha decidido limitar el número de nodos para la obtención de *betweenness centrality* a 15000 destinos y omitir la métrica *local clustering coefficient* cuyo valor en estos conjuntos de datos es siempre 0. Estas circunstancias pueden estar introduciendo sesgos al modelo y distorsionar los resultados obtenidos.

La transformación de los *dataset* originales en formato *csv* en un grafo de comunicaciones, y la generación de un nuevo *csv* con las características de comportamiento, se ha realizado mediante el *script Python* io23csv_2_graph.py (figura 5.4). Este *script* registra los parámetros de invocación utilizados e información estadística por cada fichero procesado.

Los nuevos *datasets* obtenidos en general tienen un menor número de observaciones

Figura 5.4: *Ejecución del script Python io23csv_2_graph.py.*

```
PS C:\ESTUDIOS\UNIR\MO_Inteligencia Artificial\C24_Trabajo Fin de Máster (12 ECTS)\workspace\saeDBot> & C:/Python310/python.exe "C:\ESTUDIOS\UNIR\
MO_Inteligencia Artificial\C24_Trabajo Fin de Máster (12 ECTS)\workspace\saeDBot\src\io23_2_csv.py" "X:\UNIR\iot23\bro" -S=1 -A=SC
Order action "SC" for files *.labeled (in path "X:\UNIR\iot23\bro") with limit size of 1000000 bytes.

Starting process of file 'X:\UNIR\iot23\bro\44-01_IoTMalware.labeled' .....
Total rows..... 237
Total rows in benign ..... 211
Total rows in malicious ... 26
Ended process of file 'X:\UNIR\iot23\bro\44-01_IoTMalware.labeled' .....

Starting process of file 'X:\UNIR\iot23\bro\04-01_Honeypot.labeled' .....
Total rows..... 452
Total rows in benign ..... 452
Total rows in malicious ... 0
Ended process of file 'X:\UNIR\iot23\bro\04-01_Honeypot.labeled' .....
```

Fuente: Elaboración propia

al representarse el comportamiento de la red con grafos de comunicaciones (tabla 5.1). Sin embargo, se ha observado aumentos en algunos casos. Se trata de capturas con mucha dispersión en el flujo de datos y con tráfico puntual entre dos dispositivos. No se han podido procesar todos los ficheros por falta de recursos de memoria para la generación de los grafos asociados.

El siguiente paso consiste en el análisis y preparación final de los datos disponibles en los *datasets* generados. Son ficheros compuestos por catorce columnas:

1. *ip*: Dirección en la red del dispositivo. Esta característica permite identificar al dispositivo, pero no aporta información de utilidad para el entrenamiento.

2. *port*: Puerto de comunicaciones. Las comunicaciones de entrada y salida se han agrupado por puerto. Esta información no es relevante para el entrenamiento.

3. *in_degree* y *in_degree_norm*: Número de dispositivos desde los que se recibe información por un puerto y su valor normalizado Los valores normalizados pertenecen al dominio de los números reales dentro del intervalo [0, 1]. Esta característica si aporta información relevante para la detección de *botnets*.

4. *out_degree* y *out_degree_norm*: Número de dispositivos a los cuales se envía información por un puerto y su valor normalizado. Los valores normalizados pertenecen al dominio de los números reales dentro del intervalo [0, 1]. Esta característica también es relevante para la detección de *botnets*.

5. *in_weight*: Número de paquetes que se ha recibido por el puerto de comunicaciones. Se trata de una característica no categórica con valores en el dominio de los números naturales. Esta característica es de interés para la elaboración del modelo.

   Se normalizará dividiendo el número de paquetes recibidos por el dispositivo a través de un puerto entre la suma del número de paquetes recibidos y enviados por el

Tabla 5.1: *Datasets incluidos en IoT-23.*

Captura	Flujos	*Malware*	*Url* de descarga	Tot. reg y compresión
44-01_IoTMalware	benignos: 211 maliciosos: 26	Mirai	https://mcfp.felk.cvut.cz/publicDatasets/IoT-23-Dataset/IndividualScenarios/CTU-IoT-Malware-Capture-44-1	benignos: 9 (95.73 %) maliciosos: 16 (38.46 %)
04-01_Honeypot	benignos:452	Phillips HUE	https://mcfp.felk.cvut.cz/publicDatasets/IoT-23-Dataset/IndividualScenarios/CTU-Honeypot-Capture-4-1	benignos: 454 (-0.44 %)
05-01_Honeypot	benignos: 1374	Amazon Echo	https://mcfp.felk.cvut.cz/publicDatasets/IoT-23-Dataset/IndividualScenarios/CTU-Honeypot-Capture-5-1	benignos: 944 (31.3 %)
20-01_IoTMalware	benignos: 3193 maliciosos: 16	Torii	https://mcfp.felk.cvut.cz/publicDatasets/IoT-23-Dataset/IndividualScenarios/CTU-IoT-Malware-Capture-20-1	benignos: 628 (80.33 %) maliciosos: 8 (50.00 %)
21-01_IoTMalware	benignos: 3272 maliciosos: 14	Torii	https://mcfp.felk.cvut.cz/publicDatasets/IoT-23-Dataset/IndividualScenarios/CTU-IoT-Malware-Capture-21-1	benignos: 1924 (41.2 %) maliciosos: 8 (42.86 %)
42-01_IoTMalware	benignos: 4420 maliciosos: 6	Trojan	https://mcfp.felk.cvut.cz/publicDatasets/IoT-23-Dataset/IndividualScenarios/CTU-IoT-Malware-Capture-42-1	benignos: 3099 (29.89 %) maliciosos: 8 (-33.33 %)
08-01_IoTMalware	benignos: 2181 maliciosos: 8222	Hakai	https://mcfp.felk.cvut.cz/publicDatasets/IoT-23-Dataset/IndividualScenarios/CTU-IoT-Malware-Capture-8-1	benignos: 10 (99.54 %) maliciosos: 2058 (74.97 %)
34-01_IoTMalware	benignos: 1923 maliciosos: 21222	Mirai	https://mcfp.felk.cvut.cz/publicDatasets/IoT-23-Dataset/IndividualScenarios/CTU-IoT-Malware-Capture-34-1	benignos: 293 (84.76 %) maliciosos: 4202 (80.2 %)
03-01_IoTMalware	benignos: 4536 maliciosos: 151567	Muhstik	https://mcfp.felk.cvut.cz/publicDatasets/IoT-23-Dataset/IndividualScenarios/CTU-IoT-Malware-Capture-3-1	benignos: 5609 (-23.66 %) maliciosos: 88588 (41.55 %)
01-01_IoTMalware	benignos: 469275 maliciosos: 539473	Hide and Seek	https://mcfp.felk.cvut.cz/publicDatasets/IoT-23-Dataset/IndividualScenarios/CTU-IoT-Malware-Capture-1-1	benignos: 441334 (5.95 %) maliciosos: 210749 (60.93 %)
60-01_IoTMalware	benignos: 2476 maliciosos: 3578552	Gagfyt	https://mcfp.felk.cvut.cz/publicDatasets/IoT-23-Dataset/IndividualScenarios/CTU-IoT-Malware-Capture-60-1	benignos: 63 (97.46 %) maliciosos: 65516 (98.17 %)
48-01_IoTMalware	benignos: 3734 maliciosos: 3390604	Mirai	https://mcfp.felk.cvut.cz/publicDatasets/IoT-23-Dataset/IndividualScenarios/CTU-IoT-Malware-Capture-48-1	benignos: 2319 (37.9 %) maliciosos: 1699525 (49.88 %)
49-01_IoTMalware	benignos: 3665 maliciosos: 5406896	Mirai	https://mcfp.felk.cvut.cz/publicDatasets/IoT-23-Dataset/IndividualScenarios/CTU-IoT-Malware-Capture-49-1	benignos: 3116 (14.98 %) maliciosos: 4993856 (7.64 %)
09-01_IoTMalware	benignos: 22548 maliciosos: 6355745	Linux.Hajime	https://mcfp.felk.cvut.cz/publicDatasets/IoT-23-Dataset/IndividualScenarios/CTU-IoT-Malware-Capture-9-1	benignos: 18639 (17.34 %) maliciosos: 5276051 (16.99 %)
35-01_IoTMalware	benignos: 8262389 maliciosos: 2185398	Mirai	https://mcfp.felk.cvut.cz/publicDatasets/IoT-23-Dataset/IndividualScenarios/CTU-IoT-Malware-Capture-35-1	benignos: 4120109 (50.13 %) maliciosos: 65547 (97.00 %)
07-01_IoTMalware	benignos: 75955 maliciosos:11378759	Linux,Mirai	https://mcfp.felk.cvut.cz/publicDatasets/IoT-23-Dataset/IndividualScenarios/CTU-IoT-Malware-Capture-7-1	benignos: 113538 (-49.48 %) maliciosos: recursos insuficientes
36-01_IoTMalware	benignos: 2663 maliciosos: 13642435	Okiru	https://mcfp.felk.cvut.cz/publicDatasets/IoT-23-Dataset/IndividualScenarios/CTU-IoT-Malware-Capture-36-1	benignos: 1170 (56.06 %) maliciosos: recursos insuficientes
52-01_IoTMalware	benignos: 1794 maliciosos: 19779584	Mirai	https://mcfp.felk.cvut.cz/publicDatasets/IoT-23-Dataset/IndividualScenarios/CTU-IoT-Malware-Capture-52-1	benignos: 1369 (23.69) maliciosos: recursos insuficientes
33-01_IoTMalware	benignos: 1380791 maliciosos: 53073800	Kenjiro	https://mcfp.felk.cvut.cz/publicDatasets/IoT-23-Dataset/IndividualScenarios/CTU-IoT-Malware-Capture-33-1	benignos: 1362849 (1.3 %) maliciosos: recursos insuficientes
17-01_IoTMalware	benignos: 31438 maliciosos: 54628417	Kenjiro	https://mcfp.felk.cvut.cz/publicDatasets/IoT-23-Dataset/IndividualScenarios/CTU-IoT-Malware-Capture-17-1	benignos: 25206 (19.82 %) maliciosos: recursos insuficientes
43-01_IoTMalware	benignos: 20574934 maliciosos: 46746875	Mirai	https://mcfp.felk.cvut.cz/publicDatasets/IoT-23-Dataset/IndividualScenarios/CTU-IoT-Malware-Capture-43-1	benignos: recursos insuficientes maliciosos: recursos insuficientes
39-01_IoTMalware	benignos: 7337 maliciosos: 73561644	IRCBot	https://mcfp.felk.cvut.cz/publicDatasets/IoT-23-Dataset/IndividualScenarios/CTU-IoT-Malware-Capture-39-1	benignos: 8534 (-16.72 %) maliciosos: recursos insuficientes

Fuente: Elaboración propia

dispositivo en ese puerto.

6. *out_weight*: Número de paquetes que se han enviado por el puerto de comunicaciones. Se trata de una característica no categórica con valores en el dominio de los números naturales. Esta característica es de interés para la elaboración del modelo.

   Se normalizará dividiendo el número de paquetes enviados por el dispositivo a través de un puerto entre la suma del número de paquetes recibidos y enviados por el dispositivo en ese puerto.

7. *cc* (*closeness centrality*): Media de la distancia más corta a los dispositivos que pueden llegar desde un dispositivo. Es un valor continuo perteneciente al dominio de los números reales positivos. Esta medida se ha calculado ya normalizada.

8. *bc* (*betweenness centrality*): Número de veces que un dispositivo está en el conjunto compuesto por los caminos más cortos entre cada pareja de dispositivos. Es un valor

continuo perteneciente al dominio de los números reales positivos. Esta medida se ha calculado ya normalizada dividiendo el valor de centralidad del dispositivo entre (N-1)(N-2) donde N representa el número total de dispositivos.

9. *lcc* (*local clustering coefficient*): Coeficiente que evalúa la cercanía existente entre los vecinos de un dispositivo. Su valor equivale a la fracción de triángulos posibles a través del dispositivo. Esta información es relevante para la detección de patrones de comportamiento en *botnets* con comunicaciones *P2P*. Las capturas del flujo de datos del *dataset* de origen no permiten recoger esta característica y no se incluirá como entrada del modelo.

10. *in_eig* y *out_eig* (*eigenvector centrality*): Peso del dispositivo en la red de comunicaciones. Representa la centralidad de un dispositivo en función de la centralidad de sus vecinos. Existe un valor de entrada y otro de salida al tratarse de un grafo de tipo dirigido.

    El valor se normalizará dividiendo el valor de centralidad de cada dispositivo por la centralidad máxima que se obtiene sumando la centralidad de todos los dispositivos.

11. *label*: Etiqueta para identificar el tráfico como benigno o malicioso. Se obtiene a partir del etiquetado original. Tiene una codificación simbólica binaria.

El *script Python* graph_2_ds_normalized.py realiza la normalización de aquellas características que lo requieran. En concreto, se normalizan las columnas *in_weight*, *out_weight*, *in_eig* y *out_eig*. Se genera un fichero adicional con los parámetros usados en la ejecución y el resumen estadístico.

Estos ficheros son versionados en *DVC* para mantener una traza de los *datasets* que se han generado para la preparación del modelo.

5.3. Preparación del modelo

El modelo se ha generado a partir de los ficheros obtenidos en el *pipeline* de preparación de datos. Se ha trabajado sobre dos modelos distintos de *autoencoders*.

El primer modelo se encuadra dentro del aprendizaje no supervisado. Este modelo consiste en un *stacked autoencoder* completo. Estos modelos los componentes principales no lineales de las características de entrada, reduciendo la dimensionalidad del problema. Estas componentes principales son suficientes para la reconstrucción de las características de entrada.

El entrenamiento se realiza únicamente con información del comportamiento de la red considerado benigno. Es decir, el modelo aprende cuáles son las componentes principales del comportamiento esperado de la red.

El modelo aprendido es capaz de reconstruir aquellas entradas que correspondan a un comportamiento con el mismo patrón a partir de las componentes principales. Sin embargo, otros tipos de comportamientos no será posible reconstruirlos, constituyendo una anomalía que debe interpretarse como comportamiento malicioso.

Se cuantifica la diferencia existente entre las características del comportamiento recibido en entrada y aquellas obtenidas después de la reconstrucción. Esta diferencia es irrelevante cuándo el comportamiento es benigno, pero será significativa cuándo corresponda a un comportamiento malicioso o no habitual en la red.

Es necesario establecer una medida que cuantifique esta diferencia y un umbral que establezca cuando el error de reconstrucción es relevante y debe considerarse anómalo.

La función utilizada para obtener la diferencia en la reconstrucción es el error cuadrático medio (*MSE*), representada por la expresión matemática 5.1. Se trata de una medida basada en la distancia euclídea y se obtiene calculando la media de la suma del cuadrado de la diferencia entre el valor de salida y el valor de entrada de cada una de las componentes.

$$MSE\left(X, \hat{X}\right) = \frac{1}{N}\sum_{i=1}^{N}(\hat{x}_i - x_i)^2 \tag{5.1}$$

Se considera admisible el error de reconstrucción, y por tanto un comportamiento normal, cuando esté dentro del intervalo definido por valores extremos obtenidos a partir del rango inter cuartil (*IQR*). Este rango representa la diferencia entre el percentil 75 (Q_3) y el percentil 25 (Q_1) de los valores *MSE* obtenidos para el conjunto de datos de entrenamiento. Se denomina valor atípico extremo (*outlier*) a aquellos valores inferiores al Q_1 menos tres veces el valor *IQR* o superiores al Q_3 más tres veces el valor *IQR* (5.2).

$$IQR(mse) = Q(mse)_3 - Q(mse)_1$$

$$Outlier\,(x) = \begin{cases} \text{si } Q_1 - (3 * IQR) <= x <= Q_3 + (3 * IQR) \rightarrow 0 \\ \text{si } x < Q_1 - (3 * IQR) \rightarrow 1 \\ \text{si } x > Q_3 + (3 * IQR) \rightarrow 1 \end{cases} \tag{5.2}$$

El segundo modelo se encuadra dentro del aprendizaje supervisado. Este modelo consiste en un ensamble secuencial de un *stacked autoencoder* y un modelo de clasificación

por regresión logística.

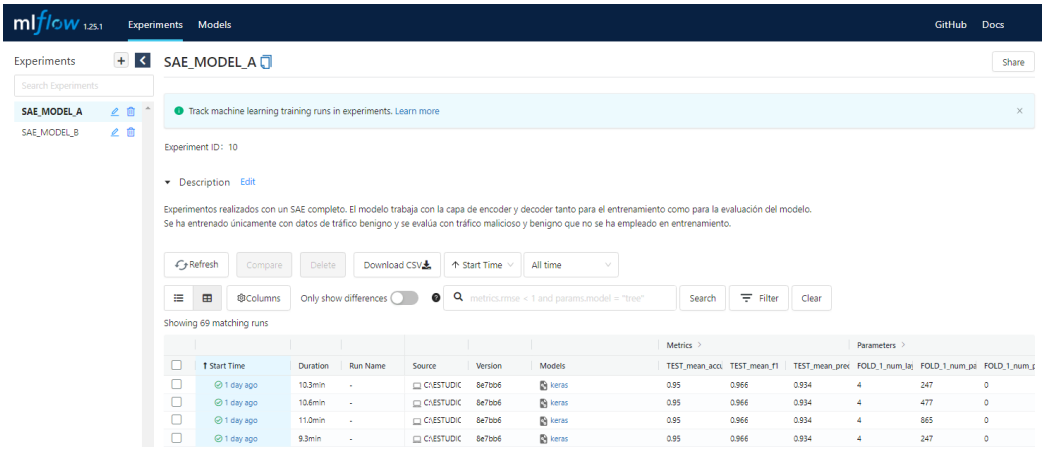
El entrenamiento se realiza con un conjunto de datos compuesto por comportamientos benignos y maliciosos. El *autoencoder* obtiene las características principales mediante la etapa de codificación y reconstruye la entrada mediante la etapa de decodificación. Una vez completado esta primera fase, se mantiene únicamente el codificador del *autoencoder*, siendo la salida de esta etapa, la entrada del modelo de aprendizaje automático de regresión logística. Este segundo modelo es el encargado de distinguir entre un comportamiento benigno o malicioso. Es posible el uso de otros modelos que permitan clasificaciones no binarias con el objetivo adicional de identificar el tipo de ataque (*DDoS*, *FileDownload*, *malware* o *HeartBeat*, etc.).

Se ha desarrollado un *script Python* independiente para la realización de los diferentes experimentos. El objetivo es evaluar y preparar el modelo más adecuado.

Se ha planificado un experimento para cada uno de los escenarios. Ambos experimentos se componen de un conjunto de casos de prueba que permiten evaluar el comportamiento de diferentes configuraciones de red y sus hiperparámetros.

Todos los experimentos se han integrado con *mlflow*. Este *framework* permite el seguimiento de todas las pruebas realizadas (figura 5.5).

Figura 5.5: *Seguimiento de la ejecución de experimentos en mlflow.*



Fuente: Elaboración propia

Se ha registrado todos los casos de prueba realizados en cada uno de los experimentos. En cada uno de ellos, se registra todas características y variables empleadas. También, se registran los modelos generados. La información almacenada permite el análisis de los resultados, reproducir los experimentos y el empaquetado del modelo para su distribución a un entorno de ejecución real.

Todos los casos de prueba se han realizado sobre el fichero de 01-01_IoTMalware.

Este fichero se ha elaborado a partir de 1.023.054 de flujos de datos, donde 469.275 se han etiquetado como benignos y 539.473 son maliciosos. Estos flujos se han representado con un grafo de comunicaciones con 605.076 dispositivos, donde 421.394 tienen un comportamiento benigno con 441.334 comunicaciones establecidas a través de sus puertos. Las comunicaciones maliciosas establecidas son 210.749.

El vector de características empleado para el aprendizaje se compone de los valores: *in_degree_norm*, *out_degree_norm*, *in_weight_norm*, *out_weight_norm*, *closeness centrality*, *betweenness centrality*, *in_eigenvector* y *out_eigenvector*. Todas las características están normalizadas.

Todos los casos de pruebas se han realizado con validación cruzada iterativa dividida en 5 *folds*. Este método divide el conjunto de datos de entrada en 5 grupos con un número similar de muestras seleccionados de manera aleatoria y sin repetición. Así, una muestra solo pertenece a uno de los grupos. En cada iteración, un grupo distinto será empleado en la validación y el resto de los grupos constituyen los datos de entrenamiento. El 80 % de los datos son utilizados en entrenamiento y el 20 % en validación.

Se ha utilizado las métricas de exactitud (*accuracy*), precisión, *recall* y f1-score para la evaluación del modelo.

La métrica *accuracy* determina la tasa de aciertos sobre el total de predicciones realizadas (5.3). La precisión mide la capacidad de predecir una clase sin equivocarse y representa el porcentaje de aciertos sobre las predicciones realizadas sobre esa clase (5.4). *Recall* mide la sensibilidad para detectar una clase y es el porcentaje de aciertos sobre el total de muestras de esa clase(5.5). *F1-score* evalúa el balance entre la precisión y *recall*. Se obtiene mediante la media armónica de ambos valores (5.6).

$$Accuracy = \frac{aciertos}{aciertos + fallos} \tag{5.3}$$

$$Precision = \frac{aciertos_{clase}}{aciertos_{clase} + fallos_{clase}} \tag{5.4}$$

$$Recall = \frac{aciertos_{clase}}{muestras_{clase}} \tag{5.5}$$

$$f1 - score = 2 * \frac{precision * recall}{precision + recall} \tag{5.6}$$

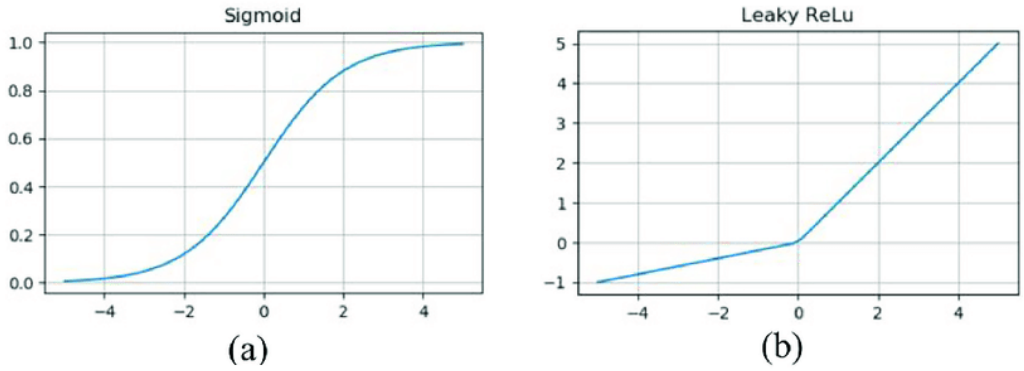
El modelo se evalúa teniendo en cuenta todas las iteraciones realizadas durante la validación cruzada. Se obtienen métricas agregadas a partir de la media aritmética de los resultados de cada iteración.

5.3.1. Modelo no supervisado

Este modelo no requiere de etiquetado del conjunto de datos de entrenamiento y validación. Sin embargo, se ha utilizado las etiquetas disponibles para evaluar la capacidad de predicción del modelo.

El *autoencoder* se ha implementado con las librerías de *Keras* y *TensorFlow* y una estructura de capas completamente interconectadas (*fully connected*). Todas las capas definidas son de tipo *Dense*. La función de activación es *LeakyReLu* (figura 5.6(b)) para todas las capas, excepto la capa de salida. En esta última capa se ha configurado con una función de activación *sigmoid* (figura 5.6(a)). Se ha incluido una normalización por capa, incluyendo una capa funcional de tipo *BatchNormalization*. La función de pérdida utilizada es *MSE*. Por último, se ha definido el optimizador *adam* para la adaptación dinámica del ratio de aprendizaje. Se ha mantenido la configuración por defecto de este optimizador.

Figura 5.6: *Funciones de activación del modelo.*



Fuente: Elaborada a partir de imagen en Kuo et al. (2020).

El experimento se ha automatizado con el *script Python* model_a_sae.py. Este *script* ejecuta un conjunto de casos de prueba aplicando la técnica *GridSearch* para combinar diferentes configuraciones de hiperparámetros y topologías para el *autoencoder*. Se ha previsto combinaciones distintas en los hiperparámetros:

a. Topología: Establece las capas que componen el *autoencoder*. Se puede definir el número de neuronas, la función de activación y el método de regularización para cada capa.

b. *Bottleneck*: Establece el número de neuronas de la capa central dónde se obtiene las componentes principales.

c. Factor *IQR*: Este hiperparámetro permite ajustar el intervalo de certidumbre del re-

sultado de la reconstrucción del comportamiento de la red. Un valor más elevado aumenta la tolerancia.

d. *Epochs*: Número de veces que se va a presentar el conjunto de entrada completo al modelo para el entrenamiento.

e. *Batch size*: Número de entradas que se presentan al modelo a la vez.

Tabla 5.2: *Rango de los hiperparámetros del experimento del modelo A.*

Topología	*Bottleneck*	Factor *IQR*	*Epochs*	*Batch size*
L1: 7 neuronas L2: 5 neuronas	1	1.5	10	32
L1: 8 neuronas L2: 7 neuronas L3: 6 neuronas L4: 5 neuronas	2	3	20	64
L1: 15 neuronas L2: 10 neuronas L3: 7 neuronas L4: 5 neuronas	3	5		128
	4	10		
		20		

Fuente: Elaboración propia

Se han realizado ensayos con diferente estructura en la red. Se han definido *autoencoders* con diferente número de capas y neuronas por capa. La tabla 5.2 presenta el rango de valores utilizado en *GridSearch* para cada uno de los hiperparámetros. Se han generado 360 casos de prueba a partir de la combinación de estos valores.

El caso de prueba con resultados más relevantes se presenta en la tabla 5.3. Se muestra el *accuracy* obtenido en el entrenamiento y el obtenido en validación. Las columnas *Precisión*, *Recall* y *F1-score* son los resultados obtenidos en la validación.

Tabla 5.3: *Resultados del experimento con el modelo A.*

Caso	Hiperparámetros	*Accuracy*	Precisión	*recall*	*f1-score*
19	L1: 7 neuronas L2: 5 neuronas	*Train*: 88.8 %			
	Bottleneck: 3	*Test Benign*:	100 %	89 %	94 %
	Factor *IQR*: 1.5	*Test Malicious*:	95.5 %	100 %	97.7 %
	Epochs: 20	*Test*: 96.7 %	98 %	94 %	96 %
	Batch size: 32				

Fuente: Elaboración propia

Los resultados obtenidos con las diferentes configuraciones del modelo son muy similares.

5.3.2. Modelo supervisado

La clasificación en este modelo se realiza mediante una regresión logística binaria. El modelo requiere etiquetado del conjunto de datos de entrenamiento y validación. Se ha utilizado las etiquetas disponibles para evaluar la capacidad de predicción.

El *autoencoder* incluido en el modelo tiene la misma implementación que en el modelo no supervisado. Las capas son de tipo *Dense* y se crean por defecto con la función de activación *LeakyReLu* y una capa de regularización de tipo *BatchNormalization*. La función de pérdida utilizada es *MSE* y el optimizador es *adam* con la configuración por defecto de *Keras*. El modelo de regresión logística se ha implementado con las librerías incluidas en *sklearn*.

El experimento se ha automatizado con el *script Python* model_b_sae.py. Este *script* ejecuta un conjunto de casos de prueba aplicando la técnica *GridSearch* para combinar diferentes configuraciones de hiperparámetros y topologías para el *autoencoder*. Se ha previsto combinaciones distintas en los hiperparámetros:

a. Topología: Establece las capas que componen el *autoencoder*. Se puede definir el número de neuronas, la función de activación y el método de regularización para cada capa.

b. *Bottleneck*: Establece el número de neuronas de la capa central dónde se obtiene las componentes principales.

c. *Epochs*: Número de veces que se va a presentar el conjunto de entrada completo al modelo para el entrenamiento.

d. *Batch size*: Número de entradas que se presentan al modelo a la vez.

Tabla 5.4: *Rango de los hiperparámetros del experimento del modelo B.*

Topología	*Bottleneck*	*Epochs*	*Batch size*
L1: 7 neuronas L2: 5 neuronas	1	10	32
L1: 8 neuronas L2: 7 neuronas L3: 6 neuronas L4: 5 neuronas	2	20	64
L1: 15 neuronas L2: 10 neuronas L3: 7 neuronas L4: 5 neuronas	3		128
	4		

Fuente: Elaboración propia

Se han realizado ensayos con diferente estructura en la red. Se han definido *autoencoders* con diferente número de capas y neuronas por capa. La tabla 5.4 presenta el rango de

valores utilizado en *GridSearch* para cada uno de los hiperparámetros. Se han generado 72 casos de prueba a partir de la combinación de estos valores.

El caso de prueba con resultados más relevantes se presenta en la tabla 5.5. Se muestra el *accuracy* obtenido en el entrenamiento y el obtenido en validación. Las columnas *Precisión*, *Recall* y *F1-score* son los resultados obtenidos en la validación.

Tabla 5.5: Resultados del experimento con el modelo B.

Caso	Hiperparámetros	*Accuracy*	Precisión	*recall*	*f1-score*
1	L1: 7 neuronas L2: 5 neuronas	*Train*: 98,96 %	98 %	99 %	99 %
	Bottleneck: 1	*Test Benign*:	100 %	98 %	99 %
	Epochs: 10	*Test Malicious*:	97 %	100 %	98 %
	Batch size: 32	*Test*: 98.94 %	98 %	99 %	99 %

Fuente: Elaboración propia

Los resultados obtenidos con las diferentes configuraciones del modelo son muy similares.

5.4. Preparación del código del prototipo

Este *pipeline* consiste en la elaboración del código fuente necesario para automatizar los dos *pipelines* anteriores. Además, se implementa el código fuente del prototipo del sistema *AIDS* que integra los modelos preparados.

El código fuente necesario para la automatización de la preparación de los datos y del modelo se ha mencionado en los apartados anteriores. La tabla 5.6 presenta la relación de *scripts* desarrollados.

Las *scripts* de preparación de datos tratan bloques de ficheros en entrada, generando los nuevos ficheros transformados en la ruta de destino seleccionada. Se han incluido parámetros para limitar el tamaño de los ficheros a procesar. Adicionalmente, cada *script* puede tener parámetros específicos para las funciones que realiza.

Las *scripts* de preparación de modelos realizan un lote de pruebas de un experimento en relación con un fichero de entrenamiento preparado con las *scripts* anteriores. Reciben un fichero de configuración de hiperparámetros que determinan las pruebas del experimento. Todos los casos de prueba son registrados en *mlflow*. Los modelos generados se conservan como parte del caso de prueba, pudiéndose empaquetarse para su distribución al sistema *AIDS*.

Tabla 5.6: *Scripts de preparación de los datos y de los modelos.*

Pipeline	Nombre	Función	Parámetros
Datos	io23_2_csv.py	Convierte a *csv* los ficheros *IoT-23* en formato del analizador *Zeek*	dir_files: Ficheros a convertir
			out_dir: Ruta destino
			-S: Tamaño máximo
			-A: Acción a realizar
	io23csv_2_graph.py	Obtiene un *csv* del grafo de comportamiento	pattern_files:Patrón de nombre ficheros
			out_dir: Ruta destino
			-S: Tamaño máximo
			-K: Límite nodos *betweenness centrality*
			-X: Columnas a excluir
	graph_2_ds_normalized.py	Genera un *csv* preparado para entrenamiento	dir_files: Ficheros a convertir
			out_dir: Ruta destino
Modelo	model_a_sae.py	Ejecuta los casos de prueba del modelo A	-D: Fichero de datos
			-T: Fichero de test
			-H: Fichero de hiperparámetros
			-O: Ruta base de salida de resultados
			-E: Nombre del experimento
			-R: Reiniciar desde un caso de pruebas
	model_b_sae.py	Ejecuta los casos de prueba del modelo B	-D: Fichero de datos
			-H: Fichero de hiperparámetros
			-O: Ruta base de salida de resultados
			-E: Nombre del experimento
			-R: Reiniciar desde un caso de pruebas

Fuente: Elaboración propia

Las otras tareas abordadas en este *pipeline* consisten en el desarrollo de los componentes del sistema *AIDS*. Se han desarrollado los componentes principales para el prototipo: el sensor, el servidor de toma de decisiones y la consola de monitorización. Se ha incluido un conjunto de desarrollos que simulan los servicios solicitados a componentes de una red *SDN* real. En concreto, se ha simulado el servicio de obtención del grafo de comunicaciones asociado a un dispositivo de la red y la aplicación de políticas de seguridad.

La tabla 5.7 presenta la relación de servicios por cada componente. Todos los servicios de intercambio de información entre componentes se han implementado con un *API RESTful*. El intercambio de mensajes se realiza en formato *JSON*. El *API* de estos servicios se ha generado en la plataforma *Postman* para automatizar las pruebas.

Las principales funcionalidades cubiertas por los desarrollos realizados se detallan a continuación:

I. Sondear el comportamiento del dispositivo. Se realiza en la capa de aplicación de los dispositivos de la red, incluyendo los dispositivos *IoT*. El figura 5.7 presenta el diagrama de secuencia compuesto conceptualmente por los componentes y servicios implicados.

   Este proceso se realiza periódicamente con una frecuencia definida por un parámetro

Tabla 5.7: *Servicios del sistema AIDS.*

Componente	Nombre Servicio	Función	Tipo
Sensor	behavior_scan	Verificación periódica del comportamiento del dispositivo	interno
	send_predictions	Envío periódico de los resultados de los sondeos	interno
	model_updater	Comprobación periódica de actualización del modelo	interno
Servidor toma decisiones	collect_predictions	Recopilación de las predicciones realizadas por los sensores	*POST RESTful*
	analyse_predictions	Análisis periódico de las predicciones	interno
	send_alerts	Registra en la consola las alertas	interno
	send_alerts_too_many_attacks	Registra en el cuadro de mando una alerta de demasiadas detecciones	interno
	send_warning_few_detections	Registra en el cuadro de mando un aviso de pocas detecciones	interno
	apply_security_policy	Comunica al controlador *SDN* nuevas políticas de seguridad	interno
	send_policy	Registra en la consola las políticas aplicadas	interno
	last_model_id_version	Obtiene el identificador de la versión más reciente del modelo	*GET RESTful*
	get_model_update	Recupera la versión mas reciente del modelo	*GET RESTful*
	generate_model	Genera un nuevo modelo periódicamente	interno
Controlador *SDN*	get_behavior	Recuperar el comportamiento de un dispositivo	*GET RESTful*
	update_security_policy	actualización de la política de seguridad	*PUT RESTful*

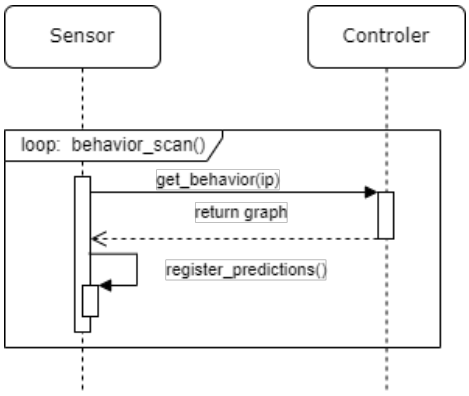
Fuente: Elaboración propia

del sistema. El sondeo se realiza por defecto cada 2 minutos. Se realiza mediante el servicio *behavoir_scan()*.

El sensor solicita, al controlador de la red *SDN*, las métricas recopiladas de sus comunicaciones con otros dispositivos. Esta petición se realiza mediante el servicio *GET RESTful* denominado *get_behavoir()*. Este servicio del controlador *SDN* se ha simulado para el prototipo. Se selecciona aleatoriamente entre las métricas existentes en los ficheros *IoT-23* asociadas al dispositivo.

El sensor predice el tipo de comportamiento del tráfico mediante el modelo entrenado y almacena los resultados invocando al servicio *register_predictions()*.

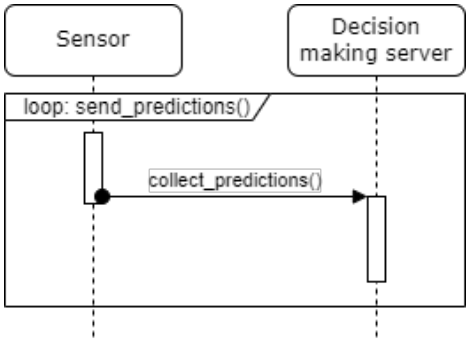
Figura 5.7: *Diagrama de secuencia del sondeo del comportamiento de un dispositivo.*



Fuente: Elaboración propia.

II. Enviar las predicciones. Se realiza en la capa de aplicación de los dispositivos de la red. La figura 5.8 presenta el diagrama de secuencia.

Figura 5.8: *Diagrama de secuencia del envío de predicciones.*



Fuente: Elaboración propia.

Este proceso se realiza periódicamente con una frecuencia definida por un parámetro del sistema. El envío se realiza por defecto cada hora. Se realiza mediante el servicio *send_predictions()*.

El sensor envía al servidor de toma de decisiones, todas las predicciones realizadas desde el último envío completado. La comunicación de las predicciones se realiza con el servicio *POST RESTful* denominado *collect_predictions()*. El servidor almacena todas estas predicciones para su análisis posterior.

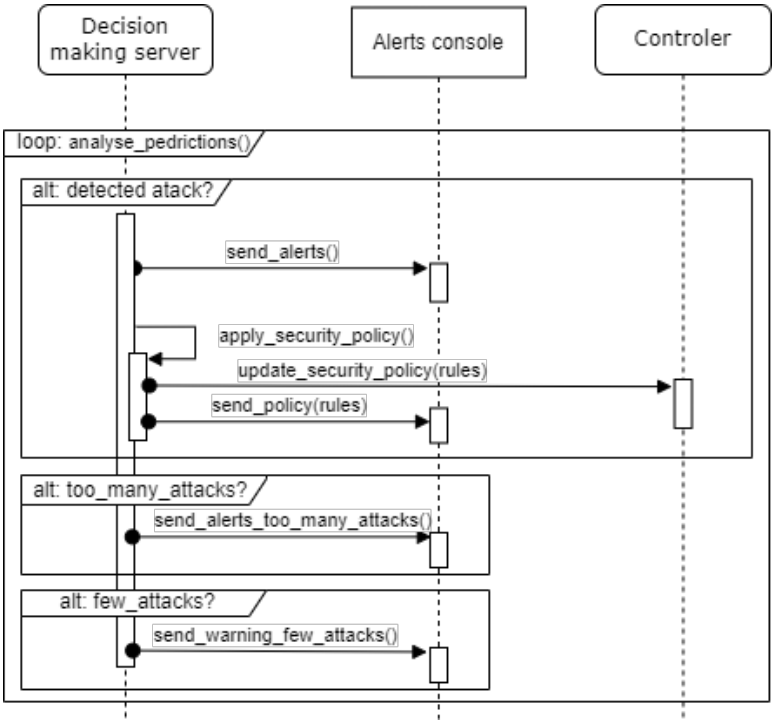
III. Analizar las predicciones. Se realiza en la capa de aplicación del servidor dedicado a la toma de decisiones. La figura 5.9 presenta el diagrama de secuencia.

Este proceso se realiza periódicamente con una frecuencia definida por un parámetro del sistema. El análisis se realiza por defecto cada hora. Se realiza mediante el servicio *analyse_predictions()*.

El servidor analiza todas las predicciones recopiladas de los sensores. Separa en dos conjuntos diferentes las predicciones maliciosas y benignas. Estas predicciones se preparan para su utilización posterior en la generación de nuevos modelos. Se calculan adicionalmente estadísticas relativas al número de predicciones tratadas y los ratios de predicciones benignas y maliciosas, sobre el total de predicciones analizadas.

El servidor actúa cuando se han encontrado predicciones maliciosas. Se registra una alerta en la consola de monitorización con el servicio *send_alerts()*. Estas alertas pueden ser consultadas por los administradores de la red en la consola de monitorización. El servidor además genera políticas de seguridad dinámicas para mitigar las consecuencias del ataque detectado. El servicio *apply_security_policy()* es responsable de generar estas políticas. Actualiza las políticas en el controlador de la

Figura 5.9: *Diagrama de secuencia del análisis de predicciones del sistema AIDS.*



Fuente: Elaboración propia.

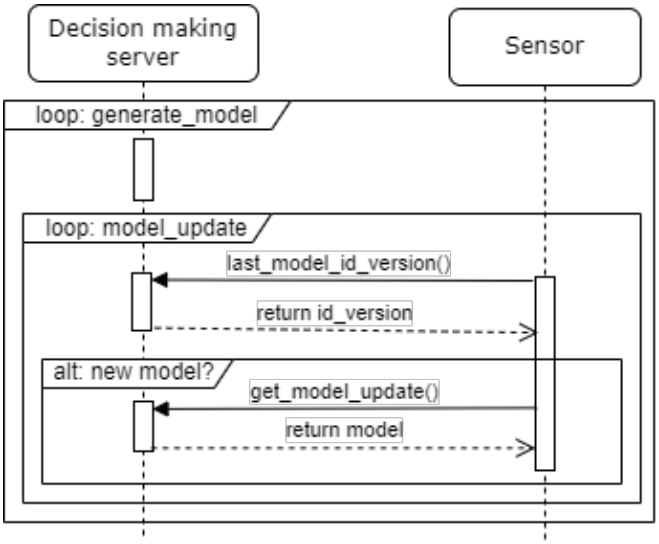
red *SDN* con el servicio *PUT RESTful* denominado *update_security_policy()* y se comunica a la consola de monitorización las reglas generadas con el servicio *send_policy()*. Esto permite a los administradores supervisar las medidas adoptadas por el sistema *AIDS* de manera automática y, en caso necesario, complementarlas.

El servidor comprueba los ratios de detecciones realizadas. Se registran alertas y avisos en el cuadro de mando de la consola de monitorización. Se registra una alerta con el servicio *send_alert_too_many_attacks()* cuando el ratio de predicciones maliciosas es superior a unos límites configurados por los administradores de la red. De igual forma, el servicio *send_warning_few_attacks()* permite registrar cuando las predicciones de comportamiento malicioso es inferior al límite establecido. Estos mensajes en el cuadro de mando avisan de la posible existencia de un decaimiento del modelo.

IV. Generar y actualizar el nuevo modelo de aprendizaje automático. Se realiza en la capa de aplicación del servidor dedicado a la toma de decisiones. La figura 5.10 presenta el diagrama de secuencia.

El servidor genera periódicamente un nuevo modelo. Se incluyen las predicciones realizadas por el sistema *AIDS* en el conjunto de entrenamiento. Se extraen del con-

Figura 5.10: *Diagrama de secuencia de la generación y actualización del modelo.*



Fuente: Elaboración propia.

junto de entrenamiento datos de capturas antiguas. Así, se establece una ventana deslizante de vigencia de los datos.

La supervisión del modelo es realizada por los científicos de datos, pero se considera que el nuevo modelo es candidato a ser utilizado si el *accuracy* de validación mejora al obtenido por el modelo actual. Además, el *accuracy* debe superar el 90 % excepto cuándo el modelo actual no lo supere. Este porcentaje se puede configurar.

Los sensores con periodicidad semanal comprueban la existencia de un nuevo modelo. Se solicita esta información al servidor de toma de decisiones con el servicio *GET RESTful* denominado *get_last_model_id_version()*. El sensor comprobará con una frecuencia mayor la existencia de un modelo cuándo no ha sido posible obtener uno nuevo tras una semana. Una vez disponible un nuevo modelo, el sensor solicita el envío del mismo con el servicio *GET RESTful* denominado *get_model_update()*.

Se ha incluido como parte del desarrollo una consola para facilitar la evaluación del prototipo. La consola desarrollada cubre únicamente aspectos básicos que, como se ha indicado, facilita la ejecución de los diferentes componentes del prototipo y la consulta básica de los resultados del funcionamiento del sistema. Elementos como cuadros de mando, perfiles de acceso a los servicios, entre otros, son necesarios en una consola administrativa de un *AIDS* completo.

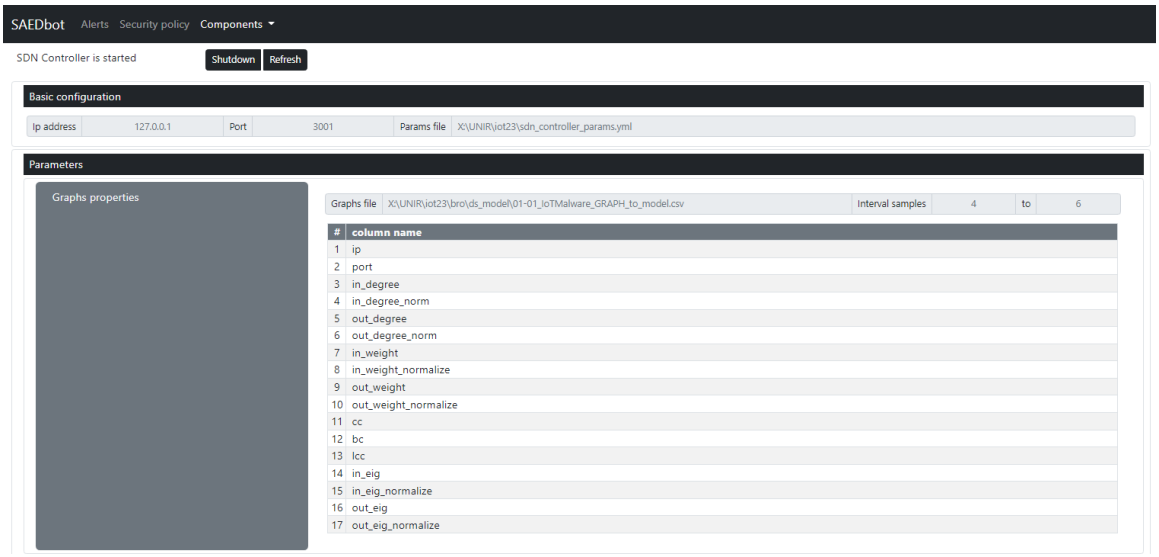
La consola desarrollada se inicia mediante el *script Python saedbot_console.py*. Una vez iniciada la consola, se puede acceder a ella mediante cualquier navegador *web*. La

página *html* está estructurada en tres secciones accesibles desde la barra de menú superior.

I. Componentes. Esta sección permite consultar la configuración de los tres componentes básicos para el funcionamiento del prototipo: el controlador *SDN* simulado, el servidor de toma de decisiones y los sensores.

   La figura 5.11 presenta la página para la gestión del controlador *SDN* simulado. Se puede activar o desactivar la simulación del controlador desde esta página. Se muestra la ubicación del fichero de configuración, así como la dirección *ip* y el puerto de acceso a los servicios simulados. También se muestra el contenido del fichero de parámetros. En concreto, la ubicación del fichero con la información de los grafos de comunicaciones, las columnas a tener en cuenta y el rango de muestras a extraer en cada petición de un sensor.

Figura 5.11: *Gestión del controlador SDN simulado desde la consola.*

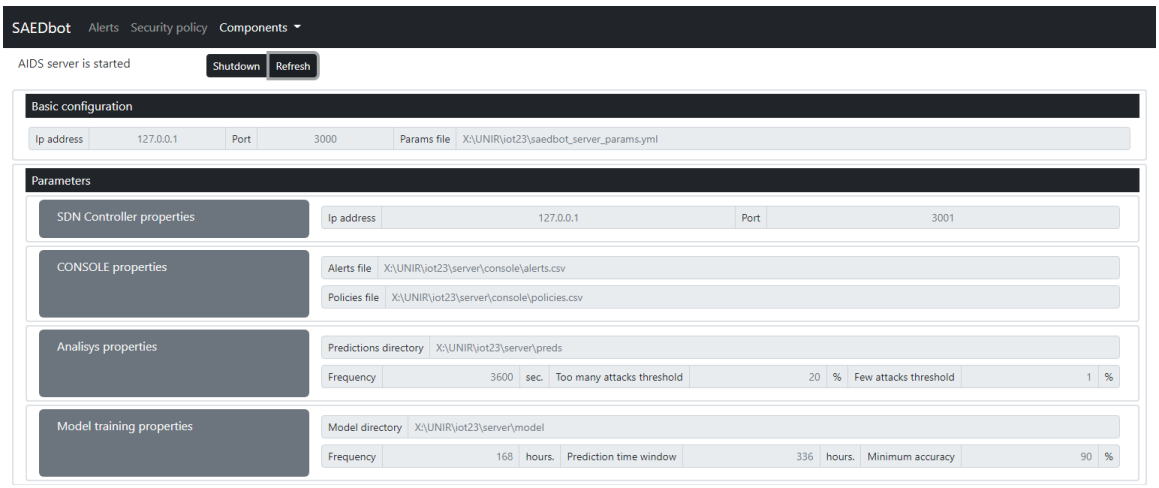


Fuente: Elaboración propia.

La figura 5.12 presenta la página para la gestión del servidor de toma de decisiones. Se puede activar o desactivar el servidor desde esta página. Se muestra la ubicación del fichero de configuración, así como la dirección *ip* y el puerto de acceso al servidor. También se muestra el contenido del fichero de parámetros agrupados por funcionalidad. Puede consultarse la configuración de la dirección *ip* y el puerto de acceso al controlador *SDN*. Otro bloque muestra la ubicación de los ficheros de la consola para almacenar las alertas y las políticas de seguridad creadas por el servidor de toma de decisiones. El siguiente bloque presenta las propiedades definidas

para el análisis de las predicciones realizadas por los sensores. Estas propiedades establecen la ubicación de los ficheros de predicciones, la frecuencia del análisis y los umbrales para determinar cuándo debe emitirse una alarma por existir un volumen alto o excesivamente bajo de ataques detectados. El último bloque contiene las propiedades para el re-entrenamiento del modelo. Las propiedades definen la ubicación del modelo, la frecuencia de entrenamiento, la ventana temporal de los datos a considerar del contenido de los ficheros de predicciones y el valor mínimo de exactitud admisible para el nuevo modelo.

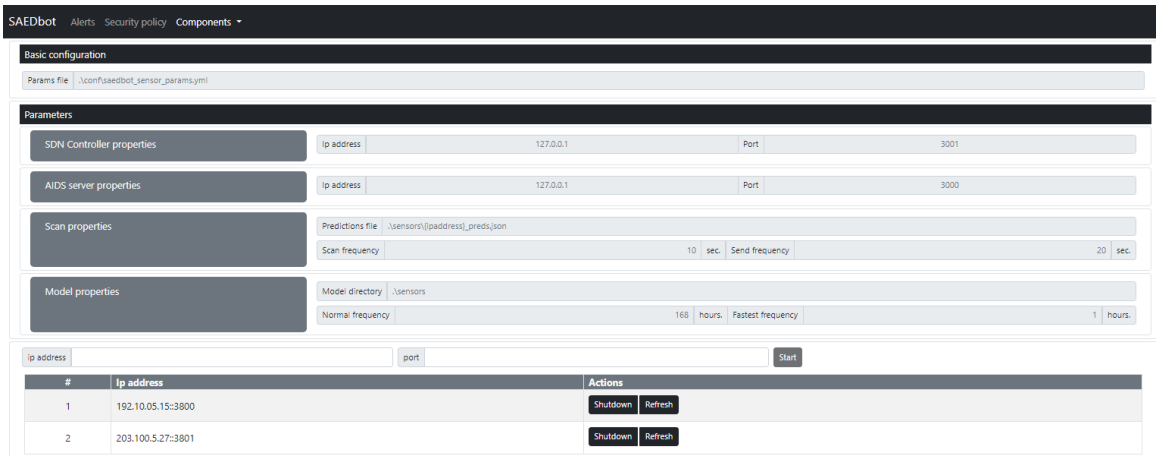
Figura 5.12: *Gestión del servidor de toma de decisiones desde la consola.*



Fuente: Elaboración propia.

La figura 5.13 presenta la página para la gestión de los sensores. Se puede activar o desactivar los diferentes sensores desde esta página.

Figura 5.13: *Gestión de los sensores desde la consola.*

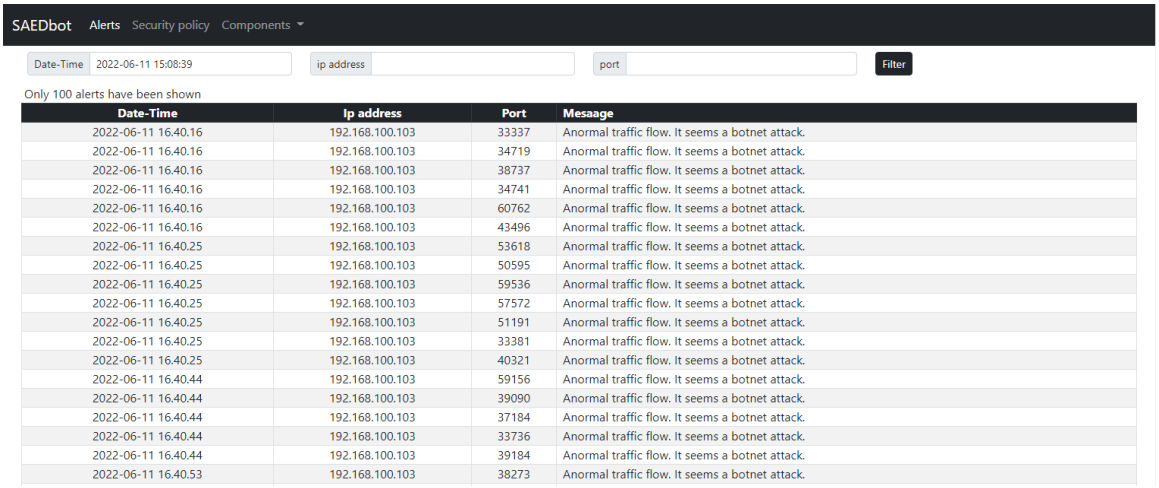


Fuente: Elaboración propia.

El prototipo simula la existencia de sensores *IoT* y permite indicar la dirección *ip* asignada al sensor, aun cuando realmente el proceso simulado realmente se está ejecutando de manera simulada en local.

Se muestra la ubicación del fichero de configuración compartido por todos los sensores simulados. También se muestra el contenido del fichero de parámetros agrupados por funcionalidad. Puede consultarse la configuración de la dirección *ip* y el puerto de acceso al controlador *SDN* y del servidor de toma de decisiones. El siguiente bloque presenta las propiedades definidas para el sondeo del comportamiento del dispositivo realizadas por los sensores. Estas propiedades establecen la ubicación de los ficheros de predicciones, la frecuencia de sondeo y envío al servidor de toma de decisiones. El último bloque contiene las propiedades para la actualización del modelo. Las propiedades definen la ubicación del modelo, la frecuencia normal de refresco y la frecuencia cuando no se ha podido refrescar en primera instancia.

Figura 5.14: *Consulta de alertas desde la consola.*



Date-Time	Ip address	Port	Mesaage
2022-06-11 16.40.16	192.168.100.103	33337	Anormal traffic flow. It seems a botnet attack.
2022-06-11 16.40.16	192.168.100.103	34719	Anormal traffic flow. It seems a botnet attack.
2022-06-11 16.40.16	192.168.100.103	38737	Anormal traffic flow. It seems a botnet attack.
2022-06-11 16.40.16	192.168.100.103	34741	Anormal traffic flow. It seems a botnet attack.
2022-06-11 16.40.16	192.168.100.103	60762	Anormal traffic flow. It seems a botnet attack.
2022-06-11 16.40.16	192.168.100.103	43496	Anormal traffic flow. It seems a botnet attack.
2022-06-11 16.40.25	192.168.100.103	53618	Anormal traffic flow. It seems a botnet attack.
2022-06-11 16.40.25	192.168.100.103	50595	Anormal traffic flow. It seems a botnet attack.
2022-06-11 16.40.25	192.168.100.103	59536	Anormal traffic flow. It seems a botnet attack.
2022-06-11 16.40.25	192.168.100.103	57572	Anormal traffic flow. It seems a botnet attack.
2022-06-11 16.40.25	192.168.100.103	51191	Anormal traffic flow. It seems a botnet attack.
2022-06-11 16.40.25	192.168.100.103	33381	Anormal traffic flow. It seems a botnet attack.
2022-06-11 16.40.25	192.168.100.103	40321	Anormal traffic flow. It seems a botnet attack.
2022-06-11 16.40.44	192.168.100.103	59156	Anormal traffic flow. It seems a botnet attack.
2022-06-11 16.40.44	192.168.100.103	39090	Anormal traffic flow. It seems a botnet attack.
2022-06-11 16.40.44	192.168.100.103	37184	Anormal traffic flow. It seems a botnet attack.
2022-06-11 16.40.44	192.168.100.103	33736	Anormal traffic flow. It seems a botnet attack.
2022-06-11 16.40.44	192.168.100.103	39184	Anormal traffic flow. It seems a botnet attack.
2022-06-11 16.40.53	192.168.100.103	38273	Anormal traffic flow. It seems a botnet attack.

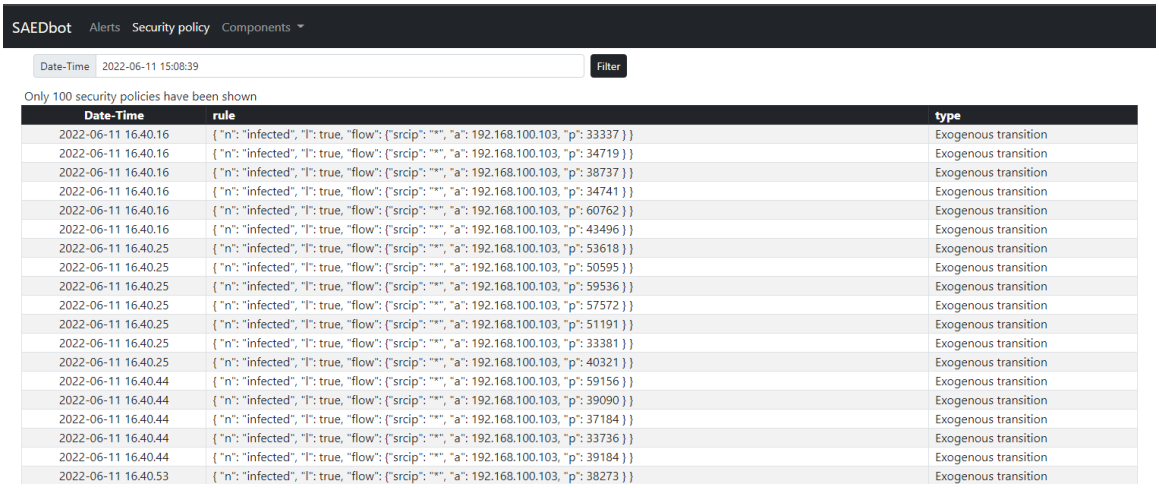
Fuente: Elaboración propia.

II. Alertas. Esta página permite consultar todos los tipos de alertas emitidas por el servidor de toma de decisiones. La figura 5.14 presenta el contenido de la página. Se muestra un máximo de 100 alertas. Se puede filtrar las alertas a recuperar estableciendo la fecha y hora a partir de donde consultar. Adicionalmente, se puede seleccionar una dirección *ip* concreta y/o puerto.

III. Políticas de seguridad. Esta página permite consultar todas las políticas de seguridad comunicadas desde el servidor de toma de decisiones al controlador *SDN*. La figura 5.15 presenta el contenido de la página. Se muestra un máximo de 100 políticas. Se

puede filtrar las alertas a recuperar estableciendo la fecha y hora a partir de donde consultar.

Figura 5.15: *Consulta desde la consola de las políticas de seguridad generadas.*



Date-Time	rule	type
2022-06-11 16.40.16	{ "n": "infected", "l": true, "flow": {"srcip": "*", "a": 192.168.100.103, "p": 33337 } }	Exogenous transition
2022-06-11 16.40.16	{ "n": "infected", "l": true, "flow": {"srcip": "*", "a": 192.168.100.103, "p": 34719 } }	Exogenous transition
2022-06-11 16.40.16	{ "n": "infected", "l": true, "flow": {"srcip": "*", "a": 192.168.100.103, "p": 38737 } }	Exogenous transition
2022-06-11 16.40.16	{ "n": "infected", "l": true, "flow": {"srcip": "*", "a": 192.168.100.103, "p": 34741 } }	Exogenous transition
2022-06-11 16.40.16	{ "n": "infected", "l": true, "flow": {"srcip": "*", "a": 192.168.100.103, "p": 60762 } }	Exogenous transition
2022-06-11 16.40.16	{ "n": "infected", "l": true, "flow": {"srcip": "*", "a": 192.168.100.103, "p": 43496 } }	Exogenous transition
2022-06-11 16.40.25	{ "n": "infected", "l": true, "flow": {"srcip": "*", "a": 192.168.100.103, "p": 53618 } }	Exogenous transition
2022-06-11 16.40.25	{ "n": "infected", "l": true, "flow": {"srcip": "*", "a": 192.168.100.103, "p": 50595 } }	Exogenous transition
2022-06-11 16.40.25	{ "n": "infected", "l": true, "flow": {"srcip": "*", "a": 192.168.100.103, "p": 59536 } }	Exogenous transition
2022-06-11 16.40.25	{ "n": "infected", "l": true, "flow": {"srcip": "*", "a": 192.168.100.103, "p": 57572 } }	Exogenous transition
2022-06-11 16.40.25	{ "n": "infected", "l": true, "flow": {"srcip": "*", "a": 192.168.100.103, "p": 51191 } }	Exogenous transition
2022-06-11 16.40.25	{ "n": "infected", "l": true, "flow": {"srcip": "*", "a": 192.168.100.103, "p": 33381 } }	Exogenous transition
2022-06-11 16.40.25	{ "n": "infected", "l": true, "flow": {"srcip": "*", "a": 192.168.100.103, "p": 40321 } }	Exogenous transition
2022-06-11 16.40.44	{ "n": "infected", "l": true, "flow": {"srcip": "*", "a": 192.168.100.103, "p": 59156 } }	Exogenous transition
2022-06-11 16.40.44	{ "n": "infected", "l": true, "flow": {"srcip": "*", "a": 192.168.100.103, "p": 39090 } }	Exogenous transition
2022-06-11 16.40.44	{ "n": "infected", "l": true, "flow": {"srcip": "*", "a": 192.168.100.103, "p": 37184 } }	Exogenous transition
2022-06-11 16.40.44	{ "n": "infected", "l": true, "flow": {"srcip": "*", "a": 192.168.100.103, "p": 33736 } }	Exogenous transition
2022-06-11 16.40.44	{ "n": "infected", "l": true, "flow": {"srcip": "*", "a": 192.168.100.103, "p": 39184 } }	Exogenous transition
2022-06-11 16.40.53	{ "n": "infected", "l": true, "flow": {"srcip": "*", "a": 192.168.100.103, "p": 38273 } }	Exogenous transition

Fuente: Elaboración propia.

La secuencia simplificada de acciones a seguir para poner en funcionamiento el prototipo comienza por iniciar la ejecución de la consola y acceder a esta desde el navegador *web*. A continuación, se activa el controlador *SDN* y el servidor de tomas de decisiones. Finalmente, se van iniciando los sensores a evaluar.

Una vez realizados estas acciones, el prototipo del sistema *AIDS* está en ejecución y los procesos definidos se van efectuando de manera repetitiva con las frecuencias establecidas. Se puede consultar en la consola las alertas y políticas de seguridad que se van generando por el servidor de toma de decisiones a partir de las predicciones obtenidas en los sensores.

Recapitulando, los sensores actualizan el modelo al iniciarse. Solicitan periódicamente al controlador *SDN* su grafo de comunicaciones y predicen el comportamiento mostrado. Las predicciones se almacenan para realizar envíos periódicos al servidor de toma de decisiones. Este servidor analiza todas las predicciones recibidas con la frecuencia definida. Emite una alerta y genera políticas de seguridad por cada predicción de tráfico malicioso recibida. Las políticas se envían al controlado *SDN* que es el responsable de aplicarlas. Además, el servidor de toma de decisiones emite alarmas si encuentra un número de alertas superior a un umbral o inferior a la media detectada durante la vigencia del modelo anterior. Finalmente. el servidor de toma de decisiones se encarga de obtener un nuevo modelo reentrenado de manera periódica. El modelo generado será distribuido a los sensores si mejora la exactitud del conjunto de validación respecto del modelo anterior o,

aunque empeore, la exactitud supera el 90 %.

El prototipo puede detenerse desde la consola, emitiendo las órdenes de detención de cada uno de los componentes.

Capítulo 6. Evaluación

Un sistema *AIDS* completo debe aportar más funcionalidades de las implementadas en el prototipo desarrollado. Se ha desarrollado los componentes fundamentales que deben constituir este tipo de sistemas y se ha descrito el ciclo de vida, siguiendo una metodología *MLops*.

El prototipo presentado en este trabajo se ha evaluado en dos aspectos: la usabilidad y la aplicabilidad para la resolución del problema abordado en el proyecto.

6.1. Usabilidad

La usabilidad de un sistema *AIDS* se puede evaluar atendiendo a diferentes criterios, considerando además que hay diferentes tipos de perfiles profesionales con funciones y herramientas diferentes. La evaluación de la usabilidad se ha circunscrito a los usuarios principales de estos sistemas; los administradores de seguridad.

Se han realizado encuestas que miden la escala de usabilidad de una aplicación según la definición de Brooke (1996). Esta encuesta permite valorar mediante 10 sencillas preguntas la percepción de los usuarios sobre la usabilidad de un sistema. Los resultados se bareman entre 0 y 100, siendo 100 la valoración más positiva. Se considera inaceptable una puntuación inferior a 50, neutral entre 50 y 68 y aceptable a partir de 68. Se han complementado con dos preguntas abiertas que solicita comentarios sobre que aspectos del prototipo resultan más útiles y, a criterio del encuestado, como se mejoraría el prototipo.

Se han elegido diferentes perfiles de profesionales dentro del sector IT, centrándose en el segmento de personas implicadas en el uso de este tipo de productos. Se ha seleccionado dos consultores seniors de arquitecturas IT, dos ingenieros de desarrollo, un administrador de sistemas y un estudiante de máster es IA. Todos ellos cuentan con una experiencia superior a 10 años en el sector IT. La tabla 6.1 presenta los resultados de las encuestas realizadas.

La media sobre los resultados de las encuestas obtiene un resultado de 80.63, indi-

cando la aceptación por parte de los encuestados. Las encuestas realizadas a perfiles de consultor han obtenido la mayor puntuación. Sin embargo, los ingenieros de desarrollo han otorgado la menor puntuación.

Tabla 6.1: *Resultados de las encuestas de escalas de usabilidad*

Preguntas de la encuesta	Encuestado 1	Encuestado 2	Encuestado 3	Encuestado 4	Encuestado 5	Encuestado 6
Creo que me gustaría utilizar frecuentemente este prototipo	De acuerdo	Totalmente de acuerdo	De acuerdo	Totalmente de acuerdo	De acuerdo	Neutral
Encontré el prototipo innecesariamente complejo	Muy en desacuerdo	Muy en desacuerdo	En desacuerdo	En desacuerdo	Muy en desacuerdo	Muy en desacuerdo
Pienso que el prototipo es fácil de usar	Totalmente de acuerdo	Totalmente de acuerdo	De acuerdo	Neutral	De acuerdo de acuerdo	Totalmente
Creo que necesitaré el apoyo de personal técnico para poder utilizar este prototipo	En desacuerdo	En desacuerdo	Neutral	En desacuerdo	Muy en desacuerdo	En desacuerdo
Encontré que varias de las funciones en el prototipo estaban bien integradas	Totalmente de acuerdo	De acuerdo	Totalmente de acuerdo	Totalmente de acuerdo	De acuerdo	De acuerdo
Pensé que había demasiada inconsistencia en este prototipo	Muy en desacuerdo	Muy en desacuerdo	En desacuerdo	Muy en desacuerdo	En desacuerdo	En desacuerdo
Me imagino que la mayoría de las personas podrían aprender a usar este prototipo muy rápido	De acuerdo	Totalmente de acuerdo	Neutral	Neutral	Neutral	De acuerdo
Encontré el prototipo muy difícil de usar	Muy en desacuerdo	Muy en desacuerdo	En desacuerdo	En desacuerdo	Muy en desacuerdo	Muy en desacuerdo
Me sentí muy confiado (seguro) al utilizar el prototipo	De acuerdo	De acuerdo	Totalmente de acuerdo	Totalmente de acuerdo	De acuerdo	Neutral
Necesité aprender muchas cosas antes de poder usar este prototipo	Muy en desacuerdo	Muy en desacuerdo	En desacuerdo	Neutral	Muy en desacuerdo	Muy en desacuerdo
Resultado	90,00	92.5	75.00	77.50	82.50	80,00

Fuente: Elaboración propia

Los encuestados consideran en general que la consola es el componente más útil del prototipo. Simplifica la puesta en marcha del prototipo sin necesidad de complejas configuraciones. Además, permite de una forma sencilla observar los acciones que realiza el *AIDS*, mostrando las alertas y las acciones asociadas en forma de políticas de seguridad. Adicionalmente, se ha considerado muy útil la actualización continúa y periódica del modelo.

Se han realizado un conjunto de recomendaciones por parte de los encuestados. En términos generales, se ha propuesto ampliar las funcionalidades aportadas por la consola. Se considera relevante incluir nuevas pantallas que permitan un seguimiento más global del sistema: la incorporación de un cuadro de mando con el resumen de la actividad del *AIDs* y pantallas específicas de seguimiento de la evolución del modelo, facilitando la detección del decaimiento del modelo y la desviación entre el entrenamiento y la inferencia (*training-serving skew*). El administrador de sistemas expresa que el *AIDS* debería contar con un modo de funcionamiento con detección de ataques, pero sin generación de políticas de seguridad automáticas. Además, indica que la detección de alertas cada 60 minutos no es eficaz ante ataques *DDoS* intensos. Apunta que sería una gran mejora que

la detección se realizará durante la propia sesión del flujo de comunicaciones, en lugar de analizar sesiones ya cerradas.

6.2. Aplicabilidad al problema

Se ha experimentado con dos modelos diferentes. Ambos han conseguido superar los objetivos establecidos, tanto en reducción de volumen de información en más de un 30 %, como en alcanzar una exactitud superior al 90 %.

Se ha conseguido una exactitud superior al 95 % a partir de tan solo 8 características de entrada. Esto permite definir *autoencoders* con una estructura muy simplificada y un pequeño número de parámetros a entrenar: 247 parámetros para el modelo A y 109 para el modelo B. Los requisitos computacionales tanto espaciales como temporales son reducidos, siendo este un factor fundamental para su ejecución en dispositivos *IoT*.

Ambos modelos presentan un 100 % de precisión a la hora de identificar el tráfico benigno, si bien considera el 11 % de los casos benignos como comportamiento anormal en el modelo A y el 2 % en el modelo B. Análogamente, ambos modelos detectan el 100 % de los comportamientos maliciosos si bien la precisión es del 95.5 % en el modelo A y el 97 % en el modelo B.

Estos resultados indican que todo el tráfico procedente de redes *botnets* es identificado, pero parte del tráfico legítimo de la red será tratado inadecuadamente. La inclusión de la regresión logística binaría mejora la capacidad de los autoencoders para clasificar correctamente los comportamientos benignos.

El principal inconveniente encontrado es el coste computacional asociado a la obtención de las características a partir de los grafos de comunicaciones. El cálculo de *closeness centrality*, *betweenness centrality* y *eigenvector* requieren de una importante capacidad de cómputo. Aun cuando no es realizada por los dispositivos *IoT*, estos requisitos pueden ser un impedimento para la integración en una red *SDN* en un entorno de ejecución real.

No se ha observado una desviación reseñable entre el entrenamiento y la inferencia. Sin embargo, se debe considerar que el prototipo no se ha puesto en servicio en un entorno de producción real. Se ha ejecutado en un entorno simulado y controlado y donde la distribución de los datos de entrada es similar a la usada en la etapa de entrenamiento del modelo.

El modelo basado en un ensamble de un *autoencoder* y un regresor logístico binario presenta mejores resultados. La separación de clases, fijando umbrales sobre el error de reconstrucción, tiene una menor exactitud y precisión.

Capítulo 7. Conclusiones y Trabajo Futuro

7.1. Conclusiones

Este trabajo tenía por objetivo principal verificar la viabilidad de aplicar *autoencoders* sobre grafos de comunicación para la detección de ataques maliciosos, realizados mediante redes *botnets*, a redes de dispositivos *IoT*. Este tipo de ataques son actualmente uno de los vectores de ataque más relevantes y los informes presentados por la industria del sector indican que el software malicioso con mayor prevalencia tiende a incorporar funcionalidades cada vez más sofisticadas de redes *botnets*.

Se ha descrito las diferentes técnicas propuestas por la comunidad científica y los problemas adicionales que plantea su detección en redes formadas por dispositivos *IoT*. La alta proliferación del uso de estos dispositivos, los niveles de seguridad reducidos y las limitaciones de consumo y cómputo ha conducido a la comunidad científica, y a la industria del sector, al estudio de técnicas que se adecuen a sus especificidades.

Nuestra propuesta describe el diseño de un *AIDS* para la identificación del comportamiento anómalo de dispositivos gestionados por un *SDN*, siendo una parte o el total de dispositivos *IoT*. El trabajo no solo muestra un prototipo con la arquitectura propuesta, sino que describe una metodología adecuada para el ciclo de vida completo de este tipo de desarrollos.

Se ha mostrado como un modelo de aprendizaje automático basado en *SAEs* es capaz de predecir el comportamiento normal de los dispositivos en la red a partir de grafos de comunicación con una exactitud superior al 95 %. Por tanto, los *SAEs* son una buena elección para el análisis del comportamiento de dispositivos *IoT*, considerando que los requisitos computacionales son asumibles por estos dispositivos.

Se ha experimentado con dos modelos distintos, cumpliéndose el objetivo marcado al superarse en ambos casos el 95 % de exactitud.

El primero se basa en aprendizaje no supervisado y utiliza únicamente un *SAE*. Se obtiene una exactitud total del 96.7 %. La precisión es del 100 %, con un *recall* del 89 % del

tráfico benigno, frente al 95.5 % de precisión, con un *recall* del 100 %, del tráfico malicioso

El segundo modelo es un ensamble de un SAE y una regresión logística con aprendizaje supervisado. Se obtiene una exactitud total del 98.94 %. La precisión es del 100 %, con un *recall* del 98 % del tráfico benigno frente al 97 % de precisión, con un *recall* del 100 %, del tráfico malicioso.

Los resultados de los experimentos realizados muestran como el uso de grafos de comunicación permite modelar el problema con un número reducido de dimensiones. Los modelos propuestos se componen con ocho dimensiones para describir el comportamiento de la red. Se ha mostrado como estos grafos de comunicaciones además suponen, en general, una reducción del volumen de información a procesar. Los grafos de red reducen de media un 39 % el número de registros originales y los ficheros pesan un 63,21 % menos, cumpliéndose el objetivo de reducción del 30 % del volumen de datos.

El principal inconveniente encontrado en el uso de grafos de comunicación es el alto coste computacional asociado a algunas de las métricas relevantes para modelar el comportamiento de la red.

Nuestra propuesta ha completado tanto los objetivo general como los objetivos específicos establecidos.

7.2. Líneas de trabajo futuro

El desarrollo realizado es un prototipo básico de un sistema *AIDS* con un entorno simulado a partir de datos reales del tráfico en redes *IoT*. Se debe continuar con nuevas evoluciones del prototipo con el objetivo de integrarlo en una red *SDN* real que gestione dispositivos *IoT*. La puesta en servicio del prototipo integrado en una red *SDN* permitirá ratificar los resultados obtenidos por las simulaciones realizadas en este trabajo.

Otra línea que debe ser explorada es la aplicación de políticas de seguridad automatizadas. Se pueden emplear técnicas de inteligencia artificial que generen políticas de seguridad, dotando al sistema con una mayor capacidad adaptativa en la respuesta de mitigación y anulación de los ataques detectados. La inclusión de regresión logística multinomial para identificar el tipo de ataque malicioso puede aportar información muy relevante para la generación de políticas más eficientes.

Los modelos presentados se han basado en ocho características de los grafos de comunicaciones, pero algunas de estas tienen un alto coste computacional tanto espacial como temporal. Se debe profundizar en el estudio para determinar las características que mejor modelan el comportamiento normal de los dispositivos en una red. En caso de ser

computacionalmente costosos, se debe buscar nuevos algoritmos o características más eficientes que representen igualmente el comportamiento de la red.

Finalmente, creemos que sería interesante la búsqueda de modelos basados en grafos de comunicaciones que sean capaces de analizar el comportamiento de la red en tiempo real, es decir, no esperar a la finalización de la sesión entre dispositivos. Este punto conlleva desafíos importantes en el contexto de detección de ataques a redes de comunicación en general, pero muy en especial a redes *IoT*.

Bibliografía

Agrawal, A., Gans, J. & Goldfarb, A. (2018). *Prediction Machines: The Simple Economics of Artificial Intelligence*. Harvard Business Review Press.

Ahmed, U., Raza, I., Hussain, S. A., Ali, A., Iqbal, M. & Wang, X. (2015). Modelling cyber security for software-defined networks those grow strong when exposed to threats. *Journal of Reliable Intelligent Environments*, *1*, 123-146. https://doi.org/10.1007/s40860-015-0008-0

Alaba, F. A., Othman, M., Hashem, I. A. T. & Alotaibi, F. (2017). Internet of Things security: A survey. *Journal of Network and Computer Applications*, *88*, 10-28. https://doi.org/10.1016/j.jnca.2017.04.002

Alazab, A., Hobbs, M., Abawajy, J. & Alazab, M. (2012). Using feature selection for intrusion detection system. *2012 International Symposium on Communications and Information Technologies (ISCIT)*, 296-301. https://doi.org/10.1109/ISCIT.2012.6380910

Anjum, A. & Apsar Pasha, S. (2015). A Brief View of Computer Network Topology for Data Communication and Networking. *International Journal of Engineering Trends and Technology (IJETT)*, *22*.

Bank, D., Koenigstein, N. & Giryes, R. (2020). Autoencoders. *ArXiv*, *abs/2003.05991*.

Bawa, M., Cooper, B., Crespo, A., Daswani, N., Ganesan, P., Garcia-Molina, H., Kamvar, S., Marti, S., Schlosser, M., Sun, Q., Vinograd, P. & Yang, B. (2003). Peer-to-peer research at Stanford. *SIGMOD Record*, *32*, 23-28. https://doi.org/10.1145/945721.945728

Benzekki, K., El Fergougui, A. & Elbelrhiti Elalaoui, A. (2016). Software-defined networking (SDN): A survey. *Security and Communication Networks*, *9*(18), 5803-5833. https://doi.org/10.1002/sec.173

Brandes, U. (2001). A Faster Algorithm for Betweenness Centrality. *Journal of Mathematical Sociology*, *25*, 163-177.

Brooke, J. (1996). *"SUS-A quick and dirty usability scale.Üsability evaluation in industry* [ISBN: 9780748404605]. CRC Press. https://www.crcpress.com/product/isbn/9780748404605

Buford, J. F., Yu, H. & Keong, L. E. (2009). Security. *P2P Networking and Applications*, 319-340. https://doi.org/10.1016/B978-0-12-374214-8.00014-3

CheckPoint. (2021). Cyber attack trends report mid year 2021. Accedido el 20 de marzo de 2022. Recuperado de: https://securitydelta.nl/media/com_hsd/report/443/document/cyber-attack-trends-report-mid-year-2021.pdf

CheckPoint. (2022). Cyber Security report year 2022. Accedido el 18 de marzo de 2022 . Recuperado de: https://pages.checkpoint.com/rs/750-DQH-528/images/2022_Security_Report_01-24-22.pdf

Choi, H., Lee, H., Lee, H. & Kim, H. (2007). Botnet Detection by Monitoring Group Activities in DNS Traffic, 715-720. https://doi.org/10.1109/CIT.2007.90

Chowdhury, S., Khanzadeh, M., Akula, R., Zhang, F., Zhang, S., Medal, H., Marufuzzaman, M. & Bian, L. (2017). Botnet detection using graph-based feature clustering. *Journal of Big Data*, *4*, 14. https://doi.org/10.1186/s40537-017-0074-7

Cisco. (2017). Cisco IOS NetFlow. Accedido el 18 de marzo de 2022. Recuperado de: https://www.cisco.com/c/en/us/products/ios-nx-os-software/ios-netflow/index.html

Cisco. (2020). Cisco Annual Internet Report (2018–2023) White Paper. Accedido el 22 de marzo de 2022. Recuperado de: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html

Daya, A. A., Salahuddin, M. A., Limam, N. & Boutaba, R. (2019). A Graph-Based Machine Learning Approach for Bot Detection. https://doi.org/10.48550/ARXIV.1902.08538

Deng, H., Li, W. & Agrawal, D. (2002). Routing security in wireless ad hoc networks. *Communications Magazine, IEEE*, *40*, 70-75. https://doi.org/10.1109/MCOM.2002.1039859

Douceur, J. R. (2002). The Sybil Attack (P. Druschel, F. Kaashoek & A. Rowstron, Eds.), 251-260.

Espina, J., Falck, T. & Mülhens, O. (2007). *Network Topologies, Communication Protocols and Standards*. https://doi.org/10.1007/1-84628-484-8_5

Forouza, B. A. (2014). *Data communications and networking* (fifth). Mc Graw Hill Education.

Foster, R., N. ans Harrison, Freedman, M., Monsanto, C., Rexford, J., Story, A. & Walker, D. (2011). Frenetic: A network programming language. *International Conference on Functional Programming (ICFP)*.

Furdek, M., Wosinska, L., Goścień, R., Manousakis, K., Aibin, M., Walkowiak, K., Ristov, S., Gushev, M. & Marzo, J. L. (2016). An overview of security challenges in communication networks. *Proceedings of 2016 8th International Workshop on Resilient Networks Design and Modeling, RNDM 2016*, 43-50. https://doi.org/10.1109/RNDM.2016.7608266

Garcia, S., Parmisano, A. & Erquiaga, M. J. (2020). IoT-23: A labeled dataset with malicious and benign IoT network traffic (Version 1.0.0) [Data set]. *Zenodo*. https://doi.org/10.5281/zenodo.4743746

Geetha, A. & Sreenath, N. (2016). Byzantine Attacks and its Security Measures in Mobile Adhoc Networks. *International Journal of Computing, Communication and Instrumentation Engineering*, *3*. https://doi.org/10.15242/ijccie.ae0116013

Gupta, K. & Mittal, P. K. (2017). An Overview of Security in MANET. *International Journal of Advanced Research in Computer Science and Software Engineering*, *7*, 151-156. https://doi.org/10.23956/ijarcsse/V7I6/0254

Hafeez, I., Antikainen, M. & Tarkoma, S. (2019). Protecting IoT-environments against Traffic Analysis Attacks with Traffic Morphing, 196-201. https://doi.org/10.1109/PERCOMW.2019.8730787

Heer, T., Morchony, 0., Hummen, R., Keoh, S., Kumar, S. & Wehlre, K. (2011). Security Challenges in the IP-based Internet of Things. *Wireless Personal Communications: An International Journal Archive*, *61*, 527-542. https://doi.org/10.1007/s11277-011-0385-5

Horrow, S., S.and Anjali. (2012). Identity management framework for cloud based Internet of Things. *Proceedings of the First International Conference on Security of Internet of Things, SecurIT '12*, 200-203.

Hu, Y.-C., Perrig, A. & Johnson, D. B. (2003a). Packet Leashes: A Defense against Wormhole Attacks in Wireless Networks. *in Proceedings of IEEE INFOCOM'03*.

Hu, Y.-C., Perrig, A. & Johnson, D. B. (2003b). Rushing Attacks and Defense in Wireless Ad Hoc Network Routing Protocols. *in Proceedings of ACM MobiCom Workshop - WiSe'03*.

IEEE Standard for Local and Metropolitan Area Networks–Port-Based Network Access Control. (2020). *IEEE Std 802.1X-2020 (Revision of IEEE Std 802.1X-2010 Incorporating IEEE Std 802.1Xbx-2014 and IEEE Std 802.1Xck-2018)*, 1-289. https://doi.org/10.1109/IEEESTD.2020.9018454

ISO/IEC 7498-1 Information technology - Open System Inteconnexion - Basis Reference Model. (1994). *International Standard)*.

Jindal, K. & Sharma, K. (2014). Analyzing Spoofing Attacks in Wireless Networks. *International Conference on Advanced Computing and Communication Technologies, ACCT*, 398-402. https://doi.org/10.1109/ACCT.2014.46

Jordan, J. (2016). Introduction to autoencoders. Accedido el 19 de abril de 2022. Recuperado de: https://www.jeremyjordan.me/autoencoders/

Karim, A., Salleh, R., Shiraz, M., Shah, S., AWAN, I. & Anuar, N. (2014). Botnet detection techniques: review, future trends and issues. *Zhejiang Univ. - Sci. C*, *15*. https://doi.org/10.1631/jzus.C1300242

Khraisat, A., Gondal, I., PVamplew, P. & Kamruzzaman, J. (2019). Survey of intrusion detection systems: techniques, datasets and challenges. *Cybersecurity*, *2*, 20. https://doi.org/10.1186/s42400-019-0038-7

Kibirige, G. & Sanga, C. (2015). A Survey on Detection of Sinkhole Attack in Wireless Sensor Network. *International Journal of Computer Science and Information Security*, *13*, 1-9.

Kim, H., Reich, J., Gupta, A., Shahbaz, M., Feamster, N. & Clark, R. (2015). Kinetic: Verifiable Dynamic Network Control. *Proceedings of the 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI '15)*.

Kuo, P.-H., Lin, S.-T. & Hu, J. (2020). DNAE-GAN: Noise-free acoustic signal generator by integrating autoencoder and generative adversarial network. *International Journal of Distributed Sensor Networks*, *16*, 155014772092352. https://doi.org/10.1177/1550147720923529

Kurose, J. F. & Ross, K. W. (2013). *Computer networking. A top-down approach* (sixth). Pearson Education.

Li, W. & Joshi, A. (2008). Security Issues in Mobile Ad Hoc Networks -A Survey. *White House Papers Graduate Research in Informatics at Sussex*, *17*.

Liao, H.-J., Lin, C.-H. R., Lin, Y. C. & Tung, K.-Y. (2013). Intrusion detection system: A comprehensive review. *J. Netw. Comput. Appl.*, *36*, 16-24.

Limarunothai, R. & Munlin, M. A. (2015). *Trends and Challenges of Botnet Architectures and Detection Techniques* (N.º 1). https://doi.org/10.14456/jist.2015.6

Loukas, G. & Ulayoke, G. o. (2010). Protection against Denial of Service Attacks: A Survey. https://doi.org/10.1093/comjnl/bxh000

Lu, T.-T., Liao, H.-Y. & Chen, M.-F. (2011). An Advanced Hybrid P2P Botnet 2.0.

Lu, Z., Lu, X., Wang, W. & Wang, C. (2010). Review and Evaluation of Security Threats on the Communication Networks in the Smart Grid. *The 2010 Military Communication Conference - Cyber Security and Network Management*.

Luo, T. & Nagarajan, S. G. (2018). Distributed Anomaly Detection Using Autoencoder Neural Networks in WSN for IoT. *2018 IEEE International Conference on Communications (ICC)*, 1-6. https://doi.org/10.1109/ICC.2018.8422402

Malladi, S., Alves-Foss, J. & Heckendorn, R. B. (2002). On Preventing Replay Attacks on Security Protocols. *Department of Computer Science University of Idaho*.

Mendes, L. D., Aloi, J. & Pimenta, T. C. (2019). Analysis of IoT Botnet Architectures and Recent Defense Proposals, 186-189. https://doi.org/10.1109/ICM48031.2019.9021715

Mirsky, Y., Doitshman, T., Elovici, Y. & Shabtai, A. (2018). Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection. https://doi.org/10.48550/ARXIV.1802.09089

MLOps.org. (2022a). MLOps Principles. Accedido el 11 de mayo de 2022. Recuperado de: https://ml-ops.org/content/phase-zero

MLOps.org. (2022b). An Overview of the End-to-End Machine Learning Workflow. Accedido el 11 de mayo de 2022. Recuperado de: https://ml-ops.org/content/end-to-end-ml-workflow

Motwani, A., Sondhi, J. & Dhote, V. (2015). A Review on Black Hole Attack in Mobile Adhoc Network Vimal Dhote. *International Journal of Computer Applications*, *116*, 975-8887.

Murray, S., Walsh, B., Kelliher, D. & O' Sullivan, D. (2014). Multi-Variable Optimization of Thermal Energy Efficiency Retrofitting of Buildings using Static Modelling and Genetic Algorithms - A Case Study. *Building and Environment*, *75*. https://doi.org/10.1016/j.buildenv.2014.01.011

Ng, A. et al. (2011). Sparse autoencoder. *CS294A Lecture notes*, *72*(2011), 1-19.

Niyaz, Q., Sun, W. & Javaid, A. Y. (2017). A Deep Learning Based DDoS Detection System in Software-Defined Networking (SDN). *ICST Transactions on Security and Safety*, *4*(12), 153515. https://doi.org/10.4108/eai.28-12-2017.153515

Patil, H. K. & Chen, T. M. (2017). Wireless Sensor Network Security: The Internet of Things. *Computer and Information Security Handbook*, 317-337. https://doi.org/10.1016/B978-0-12-803843-7.00018-1

Pawar, M. V. & J., A. (2015). Network Security and Types of Attacks in Network [International Conference on Computer, Communication and Convergence (ICCC 2015)]. *Procedia Computer Science*, *48*, 503-506. https://doi.org/https://doi.org/10.1016/j.procs.2015.04.126

Pongle, P. & Chavan, G. (2015). A survey: Attacks on RPL and 6LoWPAN in IoT, 1-6. https://doi.org/10.1109/PERVASIVE.2015.7087034

Rumelhart, D. E., Hinton, G. E. & Williams, R. J. (1986). Learning Internal Representations by Error Propagation. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations* (pp. 318-362). MIT Press.

Sarica, A. K. & Angin, P. (2020). Explainable security in SDN-based IoT networks. *Sensors*, *20*(24), 7326.

Schmidt, M., Fung, G. & Rosales, R. (2009). Optimization methods for l1-regularization. *University of British Columbia, Technical Report TR-2009-19*.

Sengupta, T., De, S. & Banerjee, I. (2021). A Closeness Centrality Based P2P Botnet Detection Approach Using Deep Learning. *2021 12th International Conference on Computing Communication and Networking Technologies (ICCCNT)*, 1-7. https://doi.org/10.1109/ICCCNT51525.2021.9579547

Shinan, K., Alsubhi, K., Alzahrani, A. & Ashraf, M. U. (2021). Machine Learning-Based Botnet Detection in Software-Defined Network: A Systematic Review. *Symmetry*, *13*(5). https://doi.org/10.3390/sym13050866

Silva, S. S. C., Silva, R. M. P., Pinto, R. C. G. & Salles, R. M. (2013). Botnets: A survey. *Computer Networks*, *57*(2), 378-403. https://doi.org/10.1016/J.COMNET.2012.07.021

Tanenbaum, A. & Wetherall, D. (2012). *Redes de computadoras* (fifth). PEARSON EDUCACIÓN.

Tsai, C., Lai, C. & Vasilakos, A. (2014). Future Internet of Things: open issues and challenges. *Wireless Networks*, *20*(8), 2201-2217. https://doi.org/10.1007/s11276-014-0731-0

Venkatesh, B., Hazra Choudhury, S., Nagaraja, S. & Balakrishnan, N. (2015). BotSpot: fast graph based identification of structured P2P bots. *J Comput Virol Hack Tech*, 1-15. https://doi.org/10.1007/s11416-015-0250-2

Yang, C., Wu, Q., Li, H. & Chen, Y. (2017). Generative poisoning attack method against neural networks. *arXiv preprint arXiv:1703.01340*.

Yaqoob, I., Hashem, I. A. T., Mehmood, Y., Gani, A., Mokhtar, S. & Guizani, S. (2017). Enabling Communication Technologies for Smart Cities. *IEEE Communications Magazine*, *55*(1), 112-120. https://doi.org/10.1109/MCOM.2017.1600232CM

Zeidanloo, H. R. & Manaf, A. B. A. (2009). Botnet Command and Control Mechanisms. *2009 Second International Conference on Computer and Electrical Engineering*, *1*, 564-568.

Zhang, H. (2013). Architecture of Network and Client-Server model. https://doi.org/10.48550/ARXIV.1307.6665

Zhang, J., Perdisci, R., Lee, W., Sarfraz, U. & Luo, X. (2011). Detecting stealthy P2P botnets using statistical traffic fingerprints. *2011 IEEE/IFIP 41st International Conference on Dependable Systems & Networks (DSN)*, 121-132.

Zhang, Z. (2010). Research on Wireless Sensor Networks Topology Models. *Journal of Software Engineering and Applications*, *03*, 1167-1171. https://doi.org/10.4236/jsea.2010.312137

Zhao, K. & Ge, L. (2013). A Survey on the Internet of Things Security. *2013 Ninth International Conference on Computational Intelligence and Security*, 663-667.

Zhou, C. & Paffenroth, R. C. (2017). Anomaly detection with robust deep autoencoders. *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, 665-674.

Zhou, W., Jia, Y., Peng, A., Zhang, Y. & Liu, P. (2019). The Effect of IoT New Features on Security and Privacy: New Threats, Existing Solutions and Challenges Yet to Be Solved. *IEEE Internet of Things Journal*, *6*(2), 1606-1616. https://doi.org/10.1109/JIOT.2018.2847733

Anexo A. Apéndices

A.1. Glosario de abreviaturas

Este anexo presenta la lista de las abreviaturas que se han empleado en el trabajo para facilitar la identificación del significado asociado a cada una de ellas.

AIDS:	Anomaly Intrusion Detection System
ANN:	Artificial Neronal Network
C&C:	Command and Control
CoAP:	Constrained Application Protocol
DDoS:	Distribuited Deneal of Service
DoS:	Deneal of Service
DVC:	Data Version Control
EUS:	Escala de Usabilidad del Sistema
IDS:	Intrusion Detection System
IoT:	Internet of Things
IRC:	Internet Relay Chat
ISP:	Internet Service Provider
LAN:	Local Area Network
M2M:	Machine to Machine
MAN:	Metropolitan Area Network
MQTT:	Message Queue Telemetry Port
MSE:	Mean Square error
OMA LWM2M:	Open Mobile Alliance Light Weight M2M
OSI:	Open System Interconnection

P2P Peer-to-peer
PAN: Personal Area Network
SAE: Stacked Auto Encoder
SDN: Software Defined Network
SIDS: Signature Intrusion Detection System
VPN: Vitual Private Network
WAN: Wide Area Network
WSN: Wireless Sensor Network

A.2. Código fuente del prototipo

El código fuente del prototipo es accesible a través del proyecto *GitHub* saeDBot. Este proyecto puede consultarse o descargarse en la *url* https://github.com/daviconUnir/saeDBot

A.3. Artículo de investigación

Sistema de detección de ataques Botnet a redes IoT basado en grafos de comunicación

David Concejal Muñoz

Universidad Internacional de la Rioja, Logroño (España)

21 de julio de 2022

RESUMEN

Este trabajo presenta un prototipo de *AIDs* para la detección de ataques *botnets* a redes de dispositivos *IoT* gestionados por un *SDN*. Se discrima el comportamiento normal de la red, y el producido por una *botnet*, mediante un modelo de *SAE* que analiza los grafos de comunicaciones de la red. La combinación de *SAEs* y grafos de comunicación reduce los requisitos computacionales, permitiendo sondear el comportamiento en los propios dispositivos *IoT*. Los resultados de los experimentos confirman que la arquitectura propuesta es una alternativa viable. Se ha obtenido una exactitud de 96.7 %, empleando un modelo no supervisado compuesto por un *SAE*, y una exactitud de 98.94 % con un ensamble supervisado de un *SAE* y una regresión logística. Estos valores de exactitud son similares a otros modelos propuestos para redes de comunicaciones clásicas.

PALABRAS CLAVE

Autoencoder, *Botnet*, ciberataque, *IoT*, grafo de comunicación

I. INTRODUCCIÓN

La ciberseguridad actualmente se considera necesaria desde un punto de vista estratégico, tanto en ámbitos industriales y corporativos como en ámbitos gubernamentales. Los informes elaborados por compañías especializados en ciberseguridad reflejan el incremento anual de ciberataques producidos. El informe anual realizado por *[5]* refleja un incremento de ataques del 29 % a nivel global. Los ataques realizados por *botnets* representan el 27 % del total de ataques.

El paulatino interés e incesante desarrollo del *IoT* ha reforzado el aumento de los datos y dispositivos. El *IoT* es ampliamente usado en la producción industrial y ámbito social. Ofrece importantes ventajas en términos de comodidad, eficiencia y accesibilidad, pero como indica [41], también ha sido la causa de graves amenazas de seguridad y privacidad.

Uno de los fines fundamentales de los dispositivos *IoT* es recopilar datos con un fin concreto. Son utilizados para fines como el control de temperatura o el control de acceso a una vivienda, mediciones biométricas, control de la calidad y seguridad de procesos industriales y otros muchos objetivos. Estos datos son almacenados con frecuencia en servidores ubicados en redes locales o se almacenan en la nube. Los datos pueden tener un carácter sensible o producir daños, incluso personales, si son alterados.

Los dispositivos *IoT* de manera habitual presentan una seguridad débil. Son frecuentemente una puerta para la intrusión en otros sistemas a los que se conecta o simplemente causan el bloqueo de los consumidores de los datos recopilados por alteración o falta de estos. También pueden ser integrados en redes *botnets* y participar activamente en ataques maliciosos.

Actualmente, la inclusión de estos dispositivos en el ámbito médico, industrial, militar y la automoción han conducido a la demanda de la inclusión de mecanismos de seguridad más sofisticados en ellos. La comunidad científica, la industria y otros ámbitos sociales se han aunado para la mejora de mecanismos aplicables. Se requiere una seguridad de tipo individual para neutralizar escenarios de robo de información o daños materiales y físicos.

Los *IDS* utilizan diversas técnicas con el objetivo de diferenciar tráfico de red con fines legítimos del tráfico malicioso. El objetivo es determinar la existencia de un ataque para iniciar las acciones necesarias para su desactivación y mitigar los daños asociados a estos ataques.

Estos sistemas se basan principalmente en el análisis del flujo de las comunicaciones en la red para detectar diversos tipos de ataques. Se propone un enfoque alternativo basado en analizar el comportamiento a partir de grafos de comunicación basados en el flujo de red. Estos análisis permiten observar facetas de comportamiento que pasan desapercibidas a otros análisis. Se propone el uso de técnicas de aprendizaje profundo implementadas mediante redes neuronales denominadas *Stacked Autoencoders* (SAE). Este tipo de redes son adecuadas para conjuntos de datos no balanceados, siendo esta una circunstancia habitual en los problemas de detección de tráfico malicioso. Por otro lado, el análisis del flujo de red es costoso computacionalmente. Esto no permite albergar estos procesos en dispositivos *IoT*, con capacidades de consumo y computación reducidas. Sin embargo, existen estudios que sostienen la viabilidad de la inclusión de *autoencoders* en dispositivos *IoT*.

El trabajo propone una arquitectura para un sistema de detección de anomalías por intrusión (*AIDS*), integrado en una red definida por *software* (SDN). Los dispositivos *IoT* sondean su comportamiento en la red mediante un modelo de aprendizaje profundo. La capa de aplicación del dispositivo identifica el tráfico malicioso a partir de grafos de comunicaciones provistos por el controlador SDN de forma regular. El sistema cuenta con un servidor dedicado a la toma de decisiones que se encarga de dictaminar las acciones a realizar a partir de las predicciones de los dispositivos. Además, el servidor se encarga de reentrenar el modelo periódicamente, adaptándose a la evolución de tráfico benigno de la red.

Se ha experimentado con dos modelos distintos: un modelo no supervisado compuesto por un *SAE* y un modelo supervisado que ensambla la etapa de codificación de un *SAE* con un regresor logístico binario. Los modelos han obtenido una exactitud del 96.7 % y el 98.94 %, respectivamente. Ambos modelos tienen a priori requisitos computacionales compatibles con dispositivos *IoT*. Además, estos modelos requieren un menor número de características para representar el comportamiento de la red con un menor número de instancias. Los grafos de red generados han reducido un 39 % el número de instancias del tráfico de red original y los ficheros pesan un 63,21 % menos. El prototipo propuesto para redes *IoT* obtiene unos resultados semejantes a otras arquitecturas actuales propuestas para redes clásicas.

II. ESTADO DEL ARTE

[36] define las redes de comunicaciones como un conjunto de ordenadores con funcionamiento autónomo y con capacidades para intercambiar información entre ellos. Esta definición se puede extender a los dispositivos *IoT*.

Los tres objetivos que persigue los sistemas de seguridad en el entorno de redes son: la disponibilidad de la red, la integridad de la información y su confidencialidad[22].

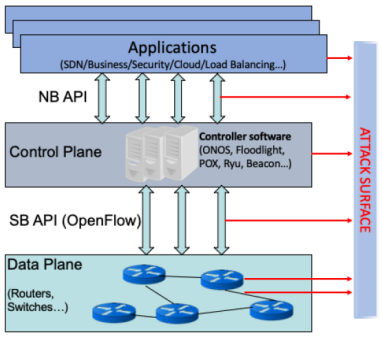
A. Paradigmas de seguridad

Hay diferentes paradigmas de seguridad empleados para la detección y mitigación derivados de los ataques maliciosos: los sistemas de detección de intrusiones y las redes definidas por *software*(*SDN*). Las tendencias actuales combinan ambos paradigmas.

[19] define a los *IDS* como sistemas que identifican acciones maliciosas para poder mantener la seguridad. Estos sistemas analizan el tráfico con un enfoque diferente a los tratamientos realizados por los *firewalls* clásicos. Se buscan características que permitan identificar al tráfico como malicioso. Se pueden clasificar en dos grandes grupos [17]: (i) *SIDS*, buscan de patrones en los datos que transitan la red y emiten alarmas al hallar tráfico coincidente con una firma catalogada. Estos sistemas son cada vez menos efectivos[17]. (ii) *AIDS*, aprenden el comportamiento normal de la red, considerando un ataque las desviaciones sobre este comportamiento. Se generan modelos de comportamiento mediante técnicas de aprendizaje automático. Los atacantes deben conocer el comportamiento normal de la red a fin de evitar la detección de sus actividades en la red.

Un *SDN* es un paradigma que permite un diseño, implementación y gestión de redes de comunicaciones donde el control y el flujo de los datos se separan en dos planos diferentes[4]. El plano de control es centralizado, dictándose las políticas de la red desde un único controlador. Este plano toma las decisiones de encaminamiento y gestiona la topología lógica de la red. El plano de datos se encarga de la transmisión de los datos acorde a las políticas del plano de control.

Figura 1: *Arquitectura SDN.*



Fuente: Sarica y Angin [32]

[32] describe su arquitectura en tres capas (figura 1): aplicación, control e infraestructura. La capa de control contiene un controlador encargado de mantener la visión global de la red y las políticas de encaminamiento. Aplicaciones tales como los *IDS* se ubican en la capa de aplicación y se comunican con el controlador. Los dispositivos de encaminamiento reciben del controlador las políticas en forma de reglas.

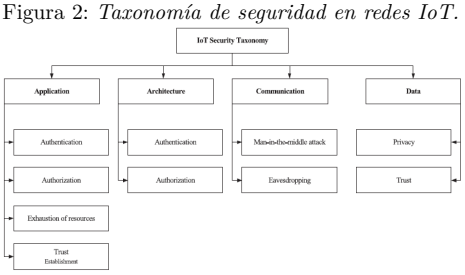
Un *SDN* tiene una mayor protección ante algunos tipos de ataques[34]. Sin embargo, también hay vulnerabilidades derivadas de su uso[1]. El controlador en un punto especialmente sensible a los fallos al centralizar la gestión de la red..

B. Seguridad en redes *IoT*

Las redes *IoT* incluyen componentes tales como sensores, actuadores, nodos de cómputo, receptores y comunicadores [13]. [39] distingue tres capas diferentes en las redes *IoT*: (i) aplicación, ofrece múltiples servicios visibles, (ii) percepción, encargada de la recopilación y el envío de la información, y (iii) protocolos de red, responsable de a transmisión y la seguridad.

Los requisitos de seguridad de las redes *IoT* son diferentes a los definidos para las redes clásicas. [2] propone una taxonomía específica para las redes *IoT* relativa a la seguridad, estableciendo cuatro dominios (figura 2): aplicación, arquitectura, comunicación y datos.

Figura 2: *Taxonomía de seguridad en redes IoT.*



Fuente: Alaba et al. [2]

El aumento del uso de dispositivos *IoT* en ataques maliciosos ha atraído la atención sobre aspectos como la baja seguridad y las vulnerabilidades que están siendo explotadas[25]. Estos dispositivos son incluidos en redes *botnets*, participando en ataques de tipo *DDoS*.

C. Amenazas en las redes de comunicación

Las redes de comunicaciones están expuestas a múltiples amenazas. [30] clasifica las amenazas en dos grandes categorías: pasivas y activas.

Los ataques pasivos analizan el flujo de información de la red, suponiendo una amenaza a la confidencialidad de la información. Los tipos principales de ataques pasivos son: (i) análisis del tráfico, se infiere información confidencial [12] y (ii) *Eavesdropping* o *sniffing*, se obtiene información de los mensajes que transitan por la red.

Los ataques activos suponen una amenaza para la integridad de la información y a la accesibilidad a los servicios de la red. Los tipos principales son: (i) *spoofing*, suplantación de la identidad de un dispositivo legítimo para enviar información en su nombre [15], (ii) modificación, provocan retrasos en el encaminamiento de mensajes, (iii) *wormhole*, se registran paquetes transmitidos por una red enviándolos mediante *tunneling* a una nueva localización[14], (iv) fabricación, generación de mensajes de encaminamiento falsos, (v) *deneal of service*, reduce o interrumpe el acceso a los servicios ofrecidos por una red, (vi) *sinkhole*, se compromete un dispositivo de la red que atrae el tráfico para desecharlo[18], (vii) *sybil*, un dispositivo malicioso presenta múltiples identidades en redes *P2P* [9], (viii) *black hole*, interrumpe o deteriora la entrega de paquetes realizada por los servicios de encaminamiento de la red, (ix) *rushing*, emite solicitudes de reenvío de mensajes antes que los dispositivos legítimos, (x) *replay*, un mensaje legítimo en un contexto se inyecta en otro contexto diferente [24] y (xi) *byzantine*, un conjunto de dispositivos autenticados pueden producir detenciones de los servicios ofrecidos de manera arbitraria [11].

D. Redes *botnets*

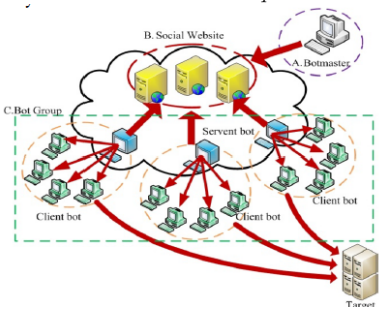
Las *botnets* son un conjunto de dispositivos comprometidos que son controlados por un *botmaster*[6]. Son dispositivos heterogéneos conectados mediante una red de comunicaciones. Una *bot-*

net no es maliciosa *per se*, permitiendo la ejecución coordinada de comandos en múltiples dispositivos. Estas facilidades han sido aprovechadas por la ciberdelincuencia con finalidades fraudulentas. La arquitectura ha evolucionado, pasando de configuraciones *peer-to-peer* a configuraciones híbridas y de protocolos *IRC* a usar protocolos *HTTP*, *P2P* o un modelo híbrido de ambos.

[35] describe los componentes de una *botnet*: (i) *host* vulnerables, dispositivos que han sido infectados con una pieza de código maliciosa, (ii) *bots*, programas potencialmente maliciosos que adquieren el control de los *host* y ejecutan las órdenes recibidas de un tercero, (iii) *botmaster*, controla la *botnet* y son los responsables de emitir las órdenes a los *bots*, y (iv) la infraestructura *C&C* (*Command-and-control*), permite al *botmasters* comunicarse con los *bots*. [20] descompone la arquitectura *C&C* en servidores y protocolos.

Las *botnets* pueden presentar diferentes arquitecturas. [38] diferencia entre tres modelos diferentes: (i) centralizado, todas las comunicaciones pasan por un servidor *C&C*, (ii) distribuido, los *host* pueden actuar simultáneamente como servidor *C&C* y *bot*, e (iii) híbrido, usan las redes sociales como medio para la difusión de las órdenes (figura 3).

Figura 3: *Modelo híbrido de arquitectura botnet.*



Fuente: Lu, Liao y Chen [21]

El ciclo de vida de una *botnet* consta de cinco etapas[20]: (i) infección inicial, consiste en infectar dispositivos para que hospeden el código que los controlará y los transformará en *bots*, (ii) inyección secundaria, consiste en la descarga e instalación del código del *bot*, (iii) *DNS lookup*, consiste en localizar la *IP* del servidor *C&C*, (iV) *Rallying*, comprende el establecimiento de conexión del *bot* con el servidor *C&C* y (v) órdenes maliciosas, mantenimiento y actualización.

El conocimiento del ciclo de vida de una *botnet* es importante para la definición de métodos de detección[35]. Los tres principales mecanismos son: (i) basados en *DNS*, consiste en detectar patrones de comportamiento de las consultas realizadas por los *bots* a los servicios de *DNS*, (ii) las redes *HoneyNet*, configuradas a propósito con vulnerabilidades con el objetivo de atraer ataques, aunque no permiten la detección de *botnets* por sí mismas [16] y (iii) los sistemas *IDS*.

E. Sistemas de detección de intrusiones

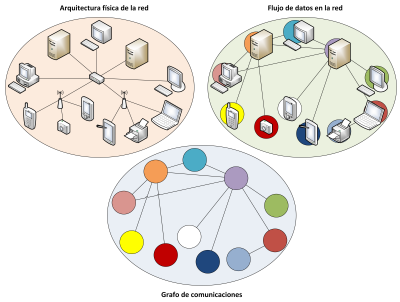
[17] y [34] recopilan las propuestas de *AIDS* basados en aprendizaje automático con métodos supervisados, no supervisados y semi-supervisados. Los principales métodos de aprendizaje automático utilizados son árboles de decisión, Naïves-Bayes, redes neuronales artificiales, máquinas de vectores de soporte y KNN, entre los supervisados, y K-medias entre los algoritmos de agrupamiento (clustering). En menor medida se han empleado algoritmos genéticos [28].

También se han realizado desde el año 2015 [34] múltiples estudios en los que se implementan *AIDS* sobre una arquitectura *SDN* ubicados en la capa de aplicación o de control.

Los modelos de aprendizaje automático se han realizado tradicionalmente sobre un conjunto de características del flujo de comunicaciones de los dispositivos de la red. Líneas de investigación prometedoras, según [34], [7] y [8], caracterizan el comportamiento de la red mediante grafos de comunicación.

Se analiza una representación de las comunicaciones en forma de un grafo (figura 4). Los nodos del grafo representan los dispositivos y los arcos corresponden a los flujos basándonos en cuatro características: las direcciones *IP* y puertos de origen y destino.

Figura 4: *Grafo de comunicaciones de una red.*



Fuente: Elaboración propia

Se han propuesto modelos de detección anomalías basados en patrones de la estructura, o en patrones de las comunidades, o redes sociales. Los grafos de comunicación evitan la comparación de flujos [37], siendo un enfoque más eficiente. [7] propone un enfoque rápido que consiste en el cálculo de ocho características: (i) *in degree*, número de flujos de entrada a un dispositivo, siendo alto en el servidor *C&C*, (ii) *out degree*, número de flujos de salida desde un dispositivo, siendo altos en el servidor *C&C* y los *bots*, (iii) *in weight degree*, número total de paquetes de entrada recibidos por un dispositivo de sus vecinos, donde se asume que todos los *bots* de una red recibirán el mismo número de paquetes desde el servidor *C&C*, (iv) *out weight degree*, número total de paquetes de salida enviados por un dispositivo de sus vecinos, donde se asume que los *bots* enviarán los mismos paquetes a los dispositivos que van a atacar, (v) *clustering coefficient*, evalúa la cercanía existente entre los vecinos de un dispositivo, donde se asume un valor alto un las *botnets* de tipo *P2P*, (vi) *node betweenness*, número de veces que un dispositivo está en el conjunto compuesto por los caminos más cortos entre cada pareja de dispositivos, donde se asume un valor alto en las *botnets* de tipo *P2P*, (vii) *node closeness*, media de la distancia más corta desde todos los dispositivos que puedan alcanzar al que se mide, siendo relevante en *botnets P2P* [33], y (viii) *eigenvector centrality*, peso del dispositivo en el grafo.

F. *Autoencoders* para detección de anomalías

El concepto de *Autoencoders* fue introducido por [31]. Se trata de una red neuronal no supervisada que se entrena con el objetivo de extraer las principales características de la entrada de manera que pueda ser reconstruida. Las entradas son codificadas y a continuación descodificadas, produciéndose pérdida de información. La diferencia entre la entrada y salida se minimiza obteniéndose una reducción dimensional. Los *Autoencoders* son una generalización del análisis *PCA* donde, en lugar de encontrar las relaciones lineales, aprende las no lineales [3].

Los *SAE* son *autoencoders* entrenados por capas de forma que cada capa de la etapa de codificación es la entrada a un *autoencoder* más interno hasta alcanzar el nivel más profundo de la red. De igual manera, las capas de la etapa de decodificación se consideran la salida de un *autoencoder* más interno.

[26], [23] y previamente [40], han propuesto modelos para la detección de anomalías en el dominio de la ciberseguridad. [29] propone el uso de *autoencoders* para la detección de ataques *DDoS* en redes *SDN*, analizado el comportamiento normal de la red con un *SAE*. El modelo es capaz de clasificar hasta ocho tipos de

ataques distintos con un ratio muy bajo de falsos positivos. Sus experimentos obtienen una exactitud (*accuracy*) del 99,82 % al diferenciar entre el tráfico benigno y el malicioso.

Se ha demostrado la viabilidad del uso de *autoencoders* en sensores *IoT* para la detección de anomalías [23]. Es un modelo donde reside una réplica del *autoencoder* en cada sensor *IoT*. Esto permite una detección distribuida y sin comunicación de datos entre los sensores. El modelo se reentrena periódicamente a partir de la información recogida por los sensores. La arquitectura propuesta permite que la red se vaya adecuando a la evolución en el comportamiento de la red.

G. Resumen de conclusiones

Los trabajos más recientes tienden a integrar los *AIDS* con redes *SDN*. Existen trabajos previos que proponen esta arquitectura para redes *IoT*, dedicando servidores a la detección. Otros trabajos proponen *AIDS* en redes *WSN* donde la detección se realiza en los *IoT* mediante *autoencoders*. Todos los modelos propuestos analizan el tráfico de la red. Hasta donde alcanza nuestro conocimiento, aún no se han propuesto sistemas *AIDS* para redes *IoT* que analicen grafos de comunicaciones.

Este trabajo propone un sistema *AIDS* que puede integrarse con un *SDN* en el dominio de las redes *IoT*. La detección de anomalías se realizará con *SAEs* que caracterizarán el comportamiento normal de la red a partir de grafos de comunicaciones. Se combina líneas prometedoras de investigación buscando reducir los requisitos computacionales y de consumo necesarios para los dispositivos *IoT*.

III. OBJETIVOS Y METODOLOGÍA

El objetivo general del trabajo es desarrollar un prototipo de un *AIDS* para la detección de ataques *botnets* en redes *IoT*, que analizará y tomará decisiones basándose en un grafo de comunicaciones de red, reduciendo en al menos un 30 % el volumen de datos a analizar y alcanzando una exactitud superior al 90 %.

Este objetivo general se ha desglosado en objetivos mas específicos: (i) determinar los componentes necesarios de las arquitecturas *SDN* con los sistemas *AIDS*, (ii) identificar al menos seis características de los gráficos de comunicaciones para la detección de ataques desde *botnets* en redes *IoT*, (iii) evaluar la exactitud y la precisión de los *SAEs* en la detección de anomalías del comportamiento de redes *IoT*, (iv) reducir de recursos para detectar ataques *botnets* a redes *IoT*, (v) verificar que la solución propuesta es una alternativa viable al problema estudiado y (vi) explorar el uso de grafos de comunicación para el aumento de la seguridad en las redes IoT.

Este prototipo seguirá un ciclo de vida *MLOps* poniendo el foco en las tareas de ingeniería asociadas a la modelización con aprendizaje automático. Este proyecto no cubre todas las etapas *MLOps* debido al alcance establecido. Se centra en los procesos de ingeniería asociada al aprendizaje automático (*MLE*). No se han abordado los trabajos de despliegue y monitorización.

Acorde a la metodología *MLOps* se han distribuido los trabajos en tres *pipelines*: preparación de datos, preparación del modelo y preparación del *software*.

El *pipeline* de datos se encarga de la adquisición, exploración, validación y preparación de los conjuntos de datos que van a usarse en la generación de los modelos. Se ha empleado, como fuente de datos, los *datasets* etiquetados *Aposemat IoT-23* [10] compuesto por flujos de redes *IoT* reales. Estos *datasets* se van a procesar para obtener loa grafos de comunicaciones asociados. Los datos preparados se han versionado en *DVC*.

El *pipeline* de aprendizaje automático comprende la preparación, la evaluación y el empaquetado del modelo para su uso posterior. Aun cuando los *autoencoders* no son supervisados, se emplea el etiquetado original como *ground truth* y permite evaluar la capacidad de predicción del modelo con métricas de aprendizaje supervisado. El proceso de entrenamiento se realiza con validación cruzada con cinco *folds*, obteniéndose las métricas globales de *accuracy* y *F1-score* mediante la media aritmética. Se realizará una búsqueda exhaustiva de hiperparámetros para determinar la mejor configuración el modelo. La herramienta *mlflow* se utiliza para mantener la trazabilidad y reproducibilidad de los diferentes modelos ensayados.

El *pipeline* de *software* se encarga de la codificación de los programas que integran el modelo en el prototipo. Hay desarrollos para la transformación de los datos originales y el entrenamiento del modelo. El desarrollo principal es el prototipo de *AIDS* que explota los modelos entrenados.

La tabla 1 muestra la caracterización para este proyecto acorde al canvas propuesto por *MLops*.

Tabla 1: *Canvas del proyecto.*

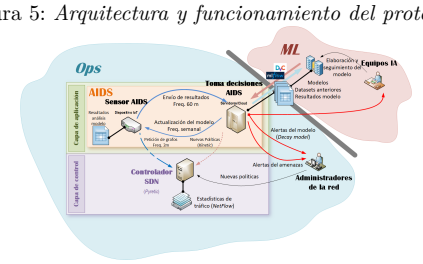
Bloques	Objetivos	Alcance en el proyecto
Propuesta de valor	Definir el problema y su importancia	*AIDS* con un menor coste computacional
Fuentes de datos	Identificar fuentes esenciales	flujos de datos de la red
Tarea de predicción	Tipo de modelo a usar	*SAEr*
Características	Cómo representar la entrada	Grafos de comunicaciones
Evaluación *offline*	Especificar métodos y métricas	*Accuracy* y *F1 score* *MSE*
Decisiones	Cómo utilizar las predicciones	Generar alertas
Haciendo predicciones	Cuándo y cómo	Por lotes periódicamente
Recolectando datos	Coste de obtener nuevos datos	No requieren etiquetado
construir modelos	Frecuencia y coste	Se vuelve a entrenar periódicamente
Evaluación y monitorización	Cómo se supervisa	Métricas con supervisión humana

Fuente: Adaptación del *canvas* propuesto por [27]

IV. CONTRIBUCIÓN

Se ha desarrollado un prototipo de los componentes principales de un sistema *AIDS*. La figura 5 presenta la arquitectura del prototipo y las principales líneas de su funcionamiento. Muestra la integración en una red *SDN* y, además, representa la relación de los diferentes actores en un ciclo de vida de aplicación con metodología *MLops*.

Figura 5: *Arquitectura y funcionamiento del prototipo.*



Fuente: Elaboración propia

El ciclo de vida de estas aplicaciones es continuo y cíclico desde la concepción hasta su fin de vida. Se alternan los trabajos de análisis de datos y modelado con su operación en un entorno real.

El desarrollo es un prototipo con una finalidad investigadora. Sin embargo, se han incluido componentes, como parte del *pipeline* de codificación, que permiten la simulación del funcionamiento del sistema en una etapa de operaciones de una red *SDN* real.

Se han realizado tres *pipelines* diferentes de acuerdo a la metodología *MLops*.

El primer *pipeline* abordado es la preparación de los datos. Las redes cuentan con herramientas de monitorización que capturan el flujo de comunicaciones. Estos flujos capturados se deben transformar en grafos de comunicaciones y, a partir de estos, se extrae las características que serán la entrada del entrenamiento de los modelos evaluados por el prototipo. Se ha automatizado estas transformaciones, realizándose en tres etapas: (i) transformación de las capturas en *csv*, se transforma desde el formato

original de las capturas a un formato estandarizado. (ii) obtención de los grafos de red y sus características, se genera el grafo de red a partir del *csv* y se calculan todas aquellas características que potencialmente sean útiles para el modelo, y (iii) adecuación de los datos, se analizan los datos y se realizan las adecuaciones necesarias, en este caso, ha sido suficiente con normalizar algunas de las características, puesto que no existían valores nulos, ni duplicados. Los ficheros generados se han versionado en *DVC*.

El siguiente *pipeline* abordado es la preparación del modelo. Todos los modelos evaluados tienen una estructura de capas completamente interconectadas. Todas las neuronas tienen una función de activación *LeakyReLu*, excepto la capa de salida que cuenta con una función de activación *sigmoid*. Se ha incluido una normalización por capa. Se ha probado con diferentes estructuras e hiperparámetros mediante una búsqueda *gridSearh*. Se han evaluado dos modelos diferentes basados en *SAE*: un modelo no supervisado y un modelo supervisado sobre un ensamble de la etapa de codificación de un *SAE* y una regresión logística. La generación y evaluación de los modelos se ha automatizado, registrando en *mlflow* las ejecuciones y los resultados obtenidos.

El último *pipeline* abordado es el desarrollo del prototipo. La arquitectura planteada integra el prototipo de *AIDS* en la capa de aplicación de una red *SDN*. El sistema presenta diferentes componentes, siendo el sensor de monitorización y el servidor de toma de decisiones los más importantes.

El sensor de monitorización se ejecuta en todos los dispositivos con capa de aplicación, incluyendo los dispositivos *IoT*. Se encarga de evaluar el comportamiento del dispositivo que alberga al propio sensor. Engloba a un conjunto de procesos con ejecución periódica y tienen la finalidad de:

I. Analizar el comportamiento del dispositivo en la red. Comprueba si es anormal el patrón de las comunicaciones realizadas en sus puertos. Este sondeo se realiza a intervalos de 2 minutos. El análisis requiere disponer de las caracteristicas asociadas al dispositivo en forma de grafo de comunicaciones de la red y una copia del modelo de *SAE* entrenado.

   La información a analizar se solicita al controlador ubicado en el plano de control de la red *SDN*. Un controlador *Pyretic* sería adecuado para una mayor abstracción del sensor de la estructura de la red y facilitaría el desarrollo de la interfaz de consulta con lenguajes como *Kinetic*.

   El prototipo invoca a un servicio simulado de un controlador de red *SDN*. Este servicio entrega los datos de comportamiento que debe evaluar el sensor. La simulación consiste en recuperar datos preparados a partir de tráfico real de los *datasets IoT-23*. Esta información se toma aleatoriamente del tráfico malicioso o benigno asociado al dispositivo. El tráfico usado no ha participado en el proceso de preparación del modelo.

   Los resultados de la evaluación, asociada a los datos de entrada, son almacenados por el sensor para su posterior envío al servidor de toma de decisiones.

II. Comunicar resultados al servidor de toma de decisiones. Se realiza cada hora enviando todos los resultados de las evaluaciones realizadas desde la comunicación anterior. La información comunicada se deja de persistir en el sensor.

III. Solicitar la versión más actualizada del modelo de *SAE* al servidor de toma de decisiones. La solicitud se realiza semanalmente. La nueva versión es descargada en el sensor, actualizando el modelo. La latencia de solicitudes pasa a ser cada hora cuando no se haya obtenido una nueva versión en la primera llamada. Se retorna a una latencia semanal al completar la actualización.

El servidor de toma de decisiones es el componente encargado de recopilar los resultados del análisis de la red realizados por los sensores y determinar las actuaciones a realizar. Las funciones principales del servidor de toma de decisiones son:

I. Analizar los resultados comunicados desde un sensor y notificar los resultados. La relación de dispositivos y puertos, con comportamiento malicioso, es enviada a un servicio dedicado a automatizar la generación de políticas dinámicas de seguridad. Se generan notificaciones y se almacenan en la consola de monitorización de los administradores de la red. El componente de automatización es un servicio del *AIDS* que no será desarrollado de manera funcional en el marco de este proyecto.

   El analizador genera una alerta específica para los administradores de red cuándo el ratio de resultados maliciosos sobre los benignos supera un umbral. Esta situación requiere una atención especial por parte de los administradores de la red. Se puede deber a un mal funcionamiento del modelo ante un patrón de comportamiento legítimo (*model decay*) o a un ataque intenso a la red. Los administradores pueden determinar las contra medidas adicionales a las adoptadas por el servidor de toma de decisiones. Se emite una alerta a los equipos de trabajo responsables del modelo para que analicen si se requiere su ajuste. Se genera una alerta similar cuándo el sistema no detecta comportamiento anómalo durante un periodo prologando de tiempo. Esta situación puede deberse a una reducción de la sensibilidad del modelo a los ataques.

II. Unificar todos los resultados recibidos por los sensores segregándolos por el tipo: benigno o malicioso. Esta información será la entrada de los procesos semanales de entrenamiento del modelo. Estos datos se ponen a disposición de servidores dedicados al entrenamiento del modelo. Estos entrenamientos se realizarán con datos de entrenamientos anteriores y los nuevos datos. Una parte de los datos de entrenamientos anteriores se descarta de acuerdo a una ventana temporal establecida sobre la fecha en la cual se obtuvieron. El objetivo es evitar que el sistema se sobre ajuste al comportamiento de una semana concreta. Los servidores dedicados al entrenamiento se encargarán de generar un nuevo modelo bajo la supervisión de los científicos de datos. Estos podrán realizar exploración y análisis de los nuevos datos recabados, iniciándose una nueva iteración del ciclo de vida *MLops*.

III. Poner a disposición de los sensores los nuevos modelos obtenidos periódicamente. El prototipo se ha simplificado exponiendo servicios que pueden consumir los sensores para actualizarse. La actualización en un *AIDS* real debería ser gobernada por el servidor de toma de decisiones con envíos de tipos *broadcast* a los dispositivos de la red.

El prototipo cuenta con una consola administrativa que permite iniciar o detener los diferentes componentes y seguir la evolución de las alertas y acciones tomadas por el prototipo.

V. RESULTADOS O EVALUACIÓN

Se han evaluado dos modelos distintos a partir del *dataset* obtenido a partir del fichero de 01-01_IoTMalware. El fichero original contaba con 1.023.054 de flujos de datos, donde 469.275 se han etiquetado como benignos y 539.473 son maliciosos. Se han transformado en un grafo de comunicaciones, donde 441.334 comunicaciones tienen un comportamiento benigno y 210.749 maliciosas. Se ha obtenido las siguientes características: *in_degree_norm*, *out_degree_norm*, *in_weight_norm*, *out_weight_norm*, *closeness centrality*, *betweenness centrality*, *in_eigenvector* y *out_eigenvector*. Todas las características están normalizadas. El cálculo de *betweenness centrality* presenta complejidad logarítmica, limitándose a un máximo de 15.000 destinos.

El primer modelo se encuadra dentro del aprendizaje no supervisado. Este modelo consiste en un *SAE* completo. El entrenamiento se ha realizado únicamente con información del comportamiento de la red considerado legítimo. El modelo evalúa la diferencia entre la entrada y los valores reconstruidos, siendo mayor cuanto más se aleje del comportamiento normal. La diferencia se obtiene mediante el *MSE* (1).

$$MSE\left(X, \hat{X}\right) = \frac{1}{N}\sum_{i=1}^{N}\left(\hat{x}_i - x_i\right)^2 \quad (1)$$

Se considera un comportamiento normal si la diferencia está dentro de un intervalo obtenido a partir del rango intercuartil (*IQR*). Son atípicas extremas las diferencias 3 veces superiores al *IQR* respecto a los extremos de este rango (2).

$$IQR(mse) = Q(mse)_3 - Q(mse)_1$$

$$Outlier\,(x) = \begin{cases} \text{si } Q_1 - (3 * IQR) <= x <= Q_3 + (3 * IQR) \rightarrow 0 \\ \text{si } x < Q_1 - (3 * IQR) \rightarrow 1 \\ \text{si } x > Q_3 + (3 * IQR) \rightarrow 1 \end{cases} \quad (2)$$

El segundo modelo es un ensamble secuencial de un *SAE* y un modelo de clasificación por regresión logística binaria. El entrenamiento se ha realizado con un conjunto de datos compuesto por comportamientos benignos y maliciosos. El *autoencoder* codifica la entrada (sin reconstruirla), siendo la entrada de la regresión logística.

A. Evaluación modelo A no supervisado

Se han generado 360 casos de prueba a partir de la combinación de estructuras de red y configuraciones de hiperparámetros. La tabla 2 muestra las combinaciones realizadas:

Tabla 2: *Hiperparámetros del experimento del modelo A.*

Topología	*Bottleneck*	Factor *IQR*	*Epochs*	*Batch size*
L1: 7 neuronas L2: 5 neuronas	1	1.5	10	32
L1: 8 neuronas L2: 7 neuronas L3: 6 neuronas L4: 5 neuronas	2	3	20	64
L1: 15 neuronas L2: 10 neuronas L3: 7 neuronas L4: 5 neuronas	3	5		128
	4	10		
		20		

Fuente: Elaboración propia

El mejor resultado obtenido se presenta en la tabla 3. Se muestra el *accuracy* obtenido en el entrenamiento y el obtenido en validación. Las columnas *Precisión*, *Recall* y *F1-score* son los resultados obtenidos en la validación.

Tabla 3: *Resultados del experimento con el modelo A.*

Caso	Hiperparámetros	*Accuracy*	Precisión	*recall*	*f1-score*
19	L1: 7 neuronas L2: 5 neuronas *Bottleneck*: 3 Factor *IQR*: 1.5 *Epochs*: 20 *Batch size*: 32	*Train*: 88.8 % *Test Benign*: *Test Malicious*: *Test*: 96.7 %	100 % 95.5 % 98 %	89 % 100 % 94 %	94 % 97.7 % 96 %

Fuente: Elaboración propia

Los resultados obtenidos con las diferentes configuraciones del modelo son muy similares.

B. Evaluación modelo B supervisado

Este modelo requiere de etiquetado del conjunto de datos de entrenamiento y validación. Se ha utilizado las etiquetas disponibles para evaluar la capacidad de predicción del modelo. Se han generado 72 casos de prueba a partir de la combinación de estructuras de red y configuraciones de hiperparámetros. La tabla 4 muestra las combinaciones realizadas:

El caso de prueba con resultados más relevantes se presenta en la tabla 5. Se muestra el *accuracy* obtenido en el entrenamiento y el obtenido en validación. Las columnas *Precisión*, *Recall* y *F1-score* son los resultados obtenidos en la validación.

Tabla 4: *Rango de los hiperparámetros del experimento del modelo B.*

Topología	*Bottleneck*	*Epochs*	*Batch size*
L1: 7 neuronas L2: 5 neuronas	1	10	32
L1: 8 neuronas L2: 7 neuronas L3: 6 neuronas L4: 5 neuronas	2	20	64
L1: 15 neuronas L2: 10 neuronas L3: 7 neuronas L4: 5 neuronas	3		128
	4		

Fuente: Elaboración propia

Los resultados obtenidos con las diferentes configuraciones del modelo son muy similares.

Tabla 5: Resultados del experimento con el modelo B.

Caso	Hiperparámetros	*Accuracy*	Precisión	*recall*	*f1-score*
1	L1: 7 neuronas L2: 5 neuronas *Bottleneck*: 1 *Epochs*: 10 *Batch size*: 32	*Train*: 98,96 % *Test Benign*: *Test Malicious*: *Test*: 98.94 %	98 % 100 % 97 % 98 %	99 % 98 % 100 % 99 %	99 % 99 % 98 % 99 %

Fuente: Elaboración propia

VI. DISCUSIÓN O ANÁLISIS DE RESULTADOS

Los dos modelos sobre los que se ha experimentado han conseguido superar los objetivos establecidos, tanto en reducción de volumen de información en más de un 30 %, como en alcanzar una exactitud superior al 90 %.

Se ha conseguido una exactitud superior al 95 % a partir de tan solo 8 características de entrada. Esto permite definir *autoencoders* con una estructura muy simplificada y un pequeño número de parámetros a entrenar: 247 parámetros para el modelo A y 109 para el modelo B. Los requisitos computacionales tanto espaciales como temporales son reducidos, siendo este un factor fundamental para su ejecución en dispositivos *IoT*.

Ambos modelos presentan un 100 % de precisión a la hora de identificar el tráfico benigno, si bien considera el 11 % de los casos benignos como comportamiento anormal en el modelo A y el 2 % en el modelo B. Análogamente, ambos modelos detectan el 100 % de los comportamientos maliciosos si bien la precisión es del 95.5 % en el modelo A y el 97 % en el modelo B.

Estos resultados indican que todo el tráfico procedente de redes *botnets* es identificado, pero parte del tráfico legítimo de la red será tratado inadecuadamente. La inclusión de la regresión logística binaría mejora la capacidad de los autoencoders para clasificar correctamente los comportamientos benignos.

El principal inconveniente encontrado es el coste computacional asociado a la obtención de las características a partir de los grafos de comunicaciones. El cálculo de *closeness centrality*, *betweenness centrality* y *eigenvector* requieren de una importante capacidad de cómputo. Aun cuando no es realizada por los dispositivos *IoT*, estos requisitos pueden ser un impedimento para la integración en una red *SDN* en un entorno de ejecución real.

VII. CONCLUSIONES

Este trabajo tenía por objetivo principal verificar la viabilidad de aplicar *autoencoders* sobre grafos de comunicación para la detección de ataques maliciosos a redes de dispositivos *IoT*, realizados mediante redes *botnets*. Nuestra propuesta describe la arquitectura y el ciclo de vida de un *AIDS* para la identificación del comportamiento anómalo de dispositivos gestionados por un *SDN*, siendo una parte o el total de los dispositivos *IoT*.

Se ha experimentado con dos modelos distintos, superándose en ambos casos la reducción del volumen de información en más

de un 30 % y el objetivo de exactitud fijado en el 90 %. Se ha analizado la reducción del volumen de información derivada del uso de grafos de comunicación. Los modelos propuestos requieren solo ocho dimensiones para describir el comportamiento de la red. Además, se ha reducido en general el volumen de información a procesar: un 39 % de media el número de registros originales y un 63,21 % en promedio el peso de los ficheros. El principal inconveniente encontrado es el alto coste computacional asociado a algunas de las métricas relevantes para modelar el comportamiento de la red. El primer experimento obtiene una exactitud total del 96.7 %, basándose en aprendizaje no supervisado. La precisión del tráfico benigno es del 100 %, con un *recall* del 89 %, frente al 95.5 % de precisión, con un *recall* del 100 %, del tráfico malicioso. El segundo modelo obtiene una exactitud total del 98.94 %, empleado un ensamble de un SAE y una regresión logística. Se obtiene. La precisión del tráfico benigno es del 100 %, con un *recall* del 98 % frente al 97 % de precisión, con un *recall* del 100 %, del tráfico malicioso.

Los experimentos se han realizado en un entorno simulado a partir de datos del tráfico real en redes *IoT*. Se va a trabajar en un futuro en integrar el prototipo en una red *SDN* real. Se debe profundizar en el módulo de políticas de seguridad con el objetivo de aplicar técnicas de inteligencia artificial para su generación, dotando al sistema de una mayor capacidad adaptativa en la respuesta de mitigación y anulación del ataque detectado.

Algunas de las características del modelo tienen un alto coste computacional. Se debe profundizar en el estudio para determinar las características que mejor modelen el comportamiento normal de los dispositivos en una red, estudiando nuevos algoritmos o características más eficientes que representen el comportamiento benigno de la red.

Se estudiará la inclusión de regresión logística multinomial para identificar, no solo ataques, sino el tipo de ataque malicioso. Esta identificación permite establecer políticas de seguridad más adaptadas a cada tipo de ataque.

Finalmente, creemos que debe explorarse el análisis del comportamiento de la red en tiempo real, no esperando a la finalización de la sesión entre dispositivos. Este punto conlleva desafíos importantes en el contexto de detección de ataques a redes de comunicación en general, pero muy en especial a redes *IoT*.

REFERENCIAS

[1] Usama Ahmed et al. «Modelling cyber security for software-defined networks those grow strong when exposed to threats». En: *Journal of Reliable Intelligent Environments* (2015).

[2] Fadele Ayotunde Alaba et al. «Internet of Things security: A survey». En: *Journal of Network and Computer Applications* (2017).

[3] Dor Bank, Noam Koenigstein y Raja Giryes. «Autoencoders». En: *ArXiv* (2020).

[4] Kamal Benzekki, Abdeslam El Fergougui y Abdelbaki Elbelrhiti Elalaoui. «Software-defined networking (SDN): A survey». En: *Security and Communication Networks* (2016).

[5] CheckPoint. *Cyber attack trends report mid year 2021. Accedido el 20 de marzo de 2022.* URL: `https://securitydelta.nl/media/com_hsd/report/443/document/cyber-attack-trends-report-mid-year-2021.pdf`.

[6] Hyunsang Choi et al. «Botnet Detection by Monitoring Group Activities in DNS Traffic». En: *7th IEEE International Conference on Computer and Information Technology (CIT 2007)* (2007).

[7] Sudipta Chowdhury et al. «Botnet detection using graph-based feature clustering». En: *Journal of Big Data* (2017).

[8] Abbas Abou Daya et al. «A Graph-Based Machine Learning Approach for Bot Detection». En: *arXiv* (2019).

[9] John R. Douceur. «The Sybil Attack». En: *Springer Berlin Heidelberg* (2002).

[10] Sebastian Garcia, Agustin Parmisano y Maria Jose Erquiaga. «IoT-23: A labeled dataset with malicious and benign IoT network traffic (Version 1.0.0) [Data set]». En: *Zenodo* (2020).

[11] A Geetha y N Sreenath. «Byzantine Attacks and its Security Measures in Mobile Adhoc Networks». En: *International Journal of Computing, Communication and Instrumentation Engineering* (2016).

[12] Ibbad Hafeez, Markku Antikainen y Sasu Tarkoma. «Protecting IoT-environments against Traffic Analysis Attacks with Traffic Morphing». En: (2019).

[13] S. Horrow S.and Anjali. «Identity management framework for cloud based Internet of Things». En: *Proceedings of the First International Conference on Security of Internet of Things, SecurIT '12* (2012).

[14] Yih-Chun Hu, Adrian Perrig y David B Johnson. «Packet Leashes: A Defense against Wormhole Attacks in Wireless Networks». En: *in Proceedings of IEEE INFOCOM'03* (2003).

[15] Keshav Jindal y Kamal Sharma. «Analyzing Spoofing Attacks in Wireless Networks». En: *International Conference on Advanced Computing and Communication Technologies, ACCT* (2014).

[16] Ahmad Karim et al. «Botnet detection techniques: review, future trends and issues». En: *Zhejiang Univ. - Sci. C* (2014).

[17] Ansam. Khraisat et al. «Survey of intrusion detection systems: techniques, datasets and challenges». En: *Cybersecurity* (2019).

[18] George Kibirige y Camilius Sanga. «A Survey on Detection of Sinkhole Attack in Wireless Sensor Network». En: *International Journal of Computer Science and Information Security* 13 ().

[19] Hung-Jen Liao et al. «Intrusion detection system: A comprehensive review». En: *J. Netw. Comput. Appl.* (2013).

[20] Ritthichai Limarunothai y Mohd Amin Munlin. *Trends and Challenges of Botnet Architectures and Detection Techniques.* 2015.

[21] Ta-Te Lu, Hung-Yi Liao y Ming-Feng Chen. *An Advanced Hybrid P2P Botnet 2.0.* 2011.

[22] Zhuo Lu et al. «Review and Evaluation of Security Threats on the Communication Networks in the Smart Grid». En: *The 2010 Military Communication Conference - Cyber Security and Network Management* (2010).

[23] Tie Luo y Sai G. Nagarajan. «Distributed Anomaly Detection Using Autoencoder Neural Networks in WSN for IoT». En: *2018 IEEE International Conference on Communications (ICC).* 2018.

[24] Sreekanth Malladi, Jim Alves-Foss y Robert B Heckendorn. «On Preventing Replay Attacks on Security Protocols». En: *Department of Computer Science University of Idaho* (2002).

[25] Lucas D.P. Mendes, James Aloi y Tales C. Pimenta. «Analysis of IoT Botnet Architectures and Recent Defense Proposals». En: (2019).

[26] Yisroel Mirsky et al. «Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection». En: *arXiv* (2018).

[27] MLOps.org. *MLOps Principles. Accedido el 11 de mayo de 2022.* URL: `https://ml-ops.org/content/phase-zero`.

[28] Sean Murray et al. «Multi-Variable Optimization of Thermal Energy Efficiency Retrofitting of Buildings using Static Modelling and Genetic Algorithms - A Case Study». En: *Building and Environment* (2014).

[29] Quamar Niyaz, Weiqing Sun y Ahmad Y. Javaid. «A Deep Learning Based DDoS Detection System in Software-Defined Networking (SDN)». En: *ICST Transactions on Security and Safety* (dic. de 2017).

[30] Mohan V. Pawar y Anuradha. J. «Network Security and Types of Attacks in Network». En: *Procedia Computer Science* (2015).

[31] D. E. Rumelhart, G. E. Hinton y R. J. Williams. «Learning Internal Representations by Error Propagation». En: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations* (1986).

[32] Alper Kaan Sarica y Pelin Angin. «Explainable security in SDN-based IoT networks». En: *Sensors* (2020).

[33] Tirthankar Sengupta, Sanghamitra De e Indrajit Banerjee. «A Closeness Centrality Based P2P Botnet Detection Approach Using Deep Learning». En: 2021.

[34] Khlood Shinan et al. «Machine Learning-Based Botnet Detection in Software-Defined Network: A Systematic Review». En: *Symmetry* (2021).

[35] S. S. C. Silva et al. «Botnets: A survey». En: *Computer Networks* (2013).

[36] Andrew Tanenbaum y David Wetherall. *Redes de computadoras*. PEARSON EDUCACIÓN, 2012.

[37] Bharath Venkatesh et al. «BotSpot: fast graph based identification of structured P2P bots». En: *J Comput Virol Hack Tech* (2015).

[38] Hossein Rouhani Zeidanloo y Azizah Bt Abdul Manaf. «Botnet Command and Control Mechanisms». En: *2009 Second International Conference on Computer and Electrical Engineering* (2009).

[39] Kai Zhao y Lina Ge. «A Survey on the Internet of Things Security». En: *2013 Ninth International Conference on Computational Intelligence and Security* (2013).

[40] Chong Zhou y Randy C Paffenroth. «Anomaly detection with robust deep autoencoders». En: 2017.

[41] W Zhou et al. «The Effect of IoT New Features on Security and Privacy: New Threats, Existing Solutions and Challenges Yet to Be Solved». En: *IEEE Internet of Things Journal* (2019).