Chapter 2

Marco Teórico

Es fundamental establecer un marco sólido que garantice la comprensión del contexto técnico del problema afrontado, además de resultar la base teórica de las decisiones tomadas en la realización del presente trabajo.

Entre los conceptos abordados en esta sección se encuentran los dos elementos principales de esta tarea, el Internet de las Cosas y las Botnets. Se expondrá la creciente expansión del IoT en las aplicaciones tanto industriales como domésticas y como este, por su propia idiosincrasia, presenta vulnerabilidades y desafíos de gran interés en el ámbito de la informática y, en particular, de la ciberseguridad. Se procederá a dar una descripción de algunos tipos de ataques de Botnets para entender en profundidad cómo actúan estas y cómo podrían enfocarse las contramedidas ante estas amenazas.

A continuación se darán detalles técnicos relativos a la computación en la niebla (*Fog Computing*) y la motivación detrás de la decisión de esta como la mejor opción para el despliegue del servicio. Ahondando en la aqruitectura interna de este sistema se sentarán las bases necesarias para comprender una arquitectura en microservicios y las ventajas detrás de sistemas de containerización como Docker.

Por último, se expondrán los diferentes modelos de Machine Learning empleados en las pruebas del servicio de análisis de tráfico. Estas especificaciones permiten entender las ventajas y limitaciones de cada uno de ellos y cómo contribuyen a la clasificación del tráfico como malicioso o benigno.

Estos fundamentos permitirán, por tanto, entender completamente la solución propuesta en los siguientes capítulos.

2.1 El Internet de las Cosas

Se ha vivido recientemente un aumento en la adopción del Internet de las Cosas por diferentes organizaciones y particulares. El IoT trae consigo enormes ventajas y ofrece un mundo de posibilidades para sus usuarios, sin embargo, conviene prestar especial atención a sus vulnerabilidades e implementar soluciones de forma acorde.

El Internet de las Cosas es la posibilidad de conectar a Internet elementos comunes inicialmente no concebidos para ello (como pueden ser puertas, medios de transporte o electrodomésticos). Esto permite generar entornos totalmente conectados que automatizan múltiples procesos frecuentes, facilitando tareas cotidianas para sus usuarios.

La arquitectura del IoT posee 4 capas. La primera de ellas la conforman los sensores y actuadores encargados de recopilar información del entorno. Suelen ser dispositivos de

tamaño y recursos reducidos, haciendo de ellos un objetivo prioritario para los atacantes en estos entornos pues deben equilibrar la implementación de seguridad con su funcionalidad principal dentro de la limitación de recursos que sufren.

La siguiente capa es la capa de red, la cual provee transporte de los datos entre los elementos que conforman la red.

La capa de procesamiento de datos emplea técnicas como el aprendizaje automático para limpiar, ordenar y formatear los datos para su análisis posterior o para facilitar la toma de decisiones en base a los mismos.

Por último, se encuentra la capa de aplicación. Esta capa supone la capa de interacción con el usuario, permitiendo, mediante el uso de interfaces sencillas y explicativas, que la persona usuaria pueda visualizar los datos extraídos y controlar los dispositivos de la capa de recopilación de datos.

Desde un punto de vista de seguridad la cantidad limitada de recursos previamente mencionada no es el único desafío que enfrentan las redes IoT. El entorno del Internet de las Cosas es extremadamente heterogéneo, tanto a nivel de software (protocolos, sistemas operativos...) como a nivel de hardware. Esta diversidad dificulta la adopción de técnicas comunes de seguridad y facilita que un atacante pueda descubrir una vulnerabilidad en alguno de los elementos que conforman la red. Además, como se verá en el apartado siguiente, las contraseñas por defecto con las que se comercializan muchos dispositivos IoT no son modificadas por el usuario, permitiendo su corrupción mediante ataques de Fuerza Bruta.

Estas vulnerabilidades representan una amenaza importante pues pueden dar acceso a la red a personas ajenas a la misma. Esta infiltración tendría consecuencias de enorme importancia si se realizara contra infraestructuras proveedoras de servicios en, por ejemplo, un sistema de riego automático de una ciudad o una red de sensores en Smart Farming. Sin embargo, no conviene infravalorar el impacto que esta pueda tener también para un ciudadano común, pues un ataque contra un sistema de Smart Home podría, por ejemplo, otorgar acceso a un tercero a imágenes privadas del hogar o permitir que este controle elementos del edificio como sistemas de aire acondicionado, pestillos o luces.

Como se discute en Amoo et al. (2024) resulta fundamental comprender los retos en ciberseguridad a los que se enfrenta el IoT y cómo afrontarlos.

2.2 Botnets: Una amenaza en auge

Considérese una red de dispositivos IoT. Se denomina *botnet* a un subconjunto de dichos dispositivos que se encuentran controlados remotamente por un usuario externo denominado *Botmaster*.

Estos aparatos, al pertenecer a la red IoT original, son empleados por su nuevo controlador para realizar ataques o acciones maliciosas en perjuicio de la red y/o en beneficio del *Botmaster*.

Dentro de la estructura de una *botnet* centralizada destaca también el Servidor de Comando y Control (C&C). Este es un dispositivo externo a la propia red (es decir, no es un dispositivo infectado si no uno en posesión del atacante) que se encarga de organizar la red de bots y de enviar las órdenes de ataque para su realización. Este tipo de servidores dan lugar a topologías como la Botnet en estrella (donde un Servidor C&C envía comandos a todos los bots) o la Botnet Multi-Server (en la que un conjunto de servidores C&C intercomunicados se encargan del envío de instrucciones).

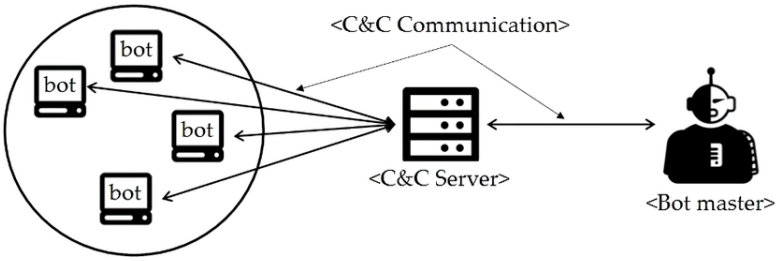

Figure 2.1: Topología de Botnet en estrella

Sin embargo, existe una clara desventaja ante este modelo de redes de bots que ha derivado en un uso creciente de botnets descentralizadas en los últimos años (Botnets Peer-to-Peer) y es el *punto de fallo*. Es decir, al existir un (o un conjunto reducido de) servidores de control la trazabilidad del ataque puede remontarse a dicho servidor y neutralizar la amenaza de una forma que, pese a ser compleja, es más sencilla que en una topología descentralizada.

Es entonces cuando surgen las Botnets Peer-to-Peer o Botnets descentralizadas. Esta concepción distribuida de la red permite al Botmaster enviar la instrucción a un único dispositivo infectado y son los propios bots quienes, mediante comunicación entre ellos, distribuyen los comandos a lo largo de toda la red. La inexistencia de un servidor de control y la capacidad para insertar las órdenes de ataque entre comunicación legítima de los dispositivos hacen de estas redes mucho más difíciles de detectar y, por tanto, mucho más resistentes a su desmantelamiento que aquellas con un Servidor C&C.

Se presentarán a continuación algunos de los ataques más habituales de las Botnets en el Iot, especificando la fase de vida de la Botnet en la que se producen.

- **Reconocimiento:** En esta etapa la Botnet se encuentra en proceso de escaneo de la red con el objetivo de detectar dispositivos vulnerables para su posterior infección. Como parte de este proceso destaca el **Ataque de Escaneo de Puertos/Port Scanning Attack**; en un escaneo de este tipo el atacante identifica puertos abiertos como posible indicio de vulnerabilidades explotables y son, por tanto, candidatos a facilitar el acceso y, en última instancia, el control del dispositivo. Esta técnica puede tomar distintas formas, entre ellas destacando el envío de flags de tipo SYN y, al recibir el SYN-ACK se descubre que el puerto está abierto, o el envío de pings (una técnica mucho más sencilla).

- **Infección:** Una vez detectadas las vulnerabilidades se procede a la infección del dispositivo. Es habitual el empleo de **técnicas de fuerza bruta** para la obtención de credenciales (siendo la no modificación de las credenciales predeterminadas de fábrica una vulnerabilidad muy habitual en el IoT) o el **File Downloading Attack** en el que se busca lograr que el dispositivo descargue un fichero malicioso.

- **Explotación:** Una vez la red de bots ya se encuentra establecida surgen otros ataques. El **ataque de denegación de servicio (DOS)** busca imposibilitar a la víctima su uso del servicio (por ejemplo, su conexión a Internet). Para ello se emplean diferentes técnicas (como se verá más adelante con el ejemplo de la Red Mirai) siendo una de las más habituales consiste en sobrecargar el servidor de forma que no pueda gestionar la cantidad tan elevada de peticiones que recibe y no pueda dar servicio a las peticiones legítimas. En el contexto del Internet de las Cosas también se observa el **Cryptojacking o Minado de Criptomonedas**; este tipo de ataque, de gran interés desde el auge de Bitcoin en la última década, consiste en emplear los

dispositivos de la víctima para minar cryptomonedas sin su conocimiento, resultando en un aumento no autorizado de los costes computacionales y de recursos, viéndose perjudicado el rendimiento del dispositivo y generando, además, posibles perjuicios económicos.

Estos ataques pueden comprometer gravemente la seguridad de una red IoT resultando en exposición de información sensible o inutilización de servicios. Un ejemplo sería un sistema domótico de seguridad en un hogar víctima de una Botnet; si un atacante encontrara una vulnerabilidad en una red de cámaras de una vivienda tendría acceso sin conocimiento de la víctima a imágenes en directo del hogar, además de otros posibles dispositivos de la red como un sensor que permitiera bloquear o desbloquear puertas en la casa.

Para una mayor comprensión del problema que suponen las botnets en el Internet de las Cosas se presenta a continuación el ejemplo más emblemático de este tipo de ataques y su impacto. La botnet **Mirai**, creada en sus inicios por un grupo de 3 jóvenes estadounidense, fue utilizada por diversos hackers después de que su código fuente fuera publicado en varios foros de Internet. Esta Botnet buscaba realizar **Ataques de Denegación de Servicio** y, para ello, se valía de **Ataques de Fuerza Bruta** para adivinar claves de dispositivos IoT que, muchas veces, eran aquellas que el fabricante implementó por defecto con poco grado de complejidad. Una vez se había logrado apoderar del dispositivo usaba su nueva red de bots para saturar los servidores mediante la generación masiva de tráfico. A finales de 2016 Mirai fue la causante de diversos ataques que inutilizaron algunas grandes compañías tecnológicas como la francesa OVH o la estadounidense DynDNS. El ataque de Mirai puso de manifiesto la vulnerabilidad de las redes IoT y los peligros que este nuevo paradigma traía consigo si se descuidaba la seguridad.

2.3 Fog, Edge y Cloud Computing

Si el lector ahonda en artículos relativos a la detección de botnets en Redes IoT, como Snehi et al. (2024), podrá observar que el concepto de Fog Computing cobra cada vez más fuerza en el ámbito del IoT y, en especial, en tareas de computación en redes IoT. Por ello, conviene abordar este nuevo paradigma y entender las ventajas y las desventajas que este ofrece como posible entorno de despliegue de la solución de análisis de tráfico.

La Computación en la Niebla (*Fog Computing*) busca ofrecer capacidad de cómputo y almacenamiento en un nivel intermedio entre los dispositivos frontera y la computación en la nube tradicional. Como se discute en Bonomi et al. (2012) esta nueva capa ofrece diversas ventajas entre las que destacan la disminución en la latencia, la descentralización y la cercanía al origen del dato. Esta concepción local requiere el despliegue de nodos como routers o microservidores a poca distancia de los elementos de la capa de recopilación, siendo muy claro como esto contribuye a reducir la latencia respecto a la computación en la nube. Además, desde un punto de vista de seguridad y robustez, este paradigma computacional descentralizado destaca en dos aspectos. Por un lado, la no necesidad de enviar datos a través de la nube y la capacidad para procesar datos sin salir del entorno local, pudiendo mantener en el mismo ciertos datos sensibles, evita posibles vulnerabilidades o transporte de datos que requerirían especial esfuerzo de seguridad para evitar su filtración ante ataques externos. Por otro lado, se garantiza la robustez del sistema al ser resiliente ante caídas de red y conexión con el servicio en la nube. Si la red pierde su capacidad de interactuar con el servicio centralizado todo el sistema estaría comprometido, mientras que la existencia de varios microservidores locales con capacidad de almacenamiento y cómputo permitiría que este siguiera en funcionamiento mientras se reestablece la conexión.

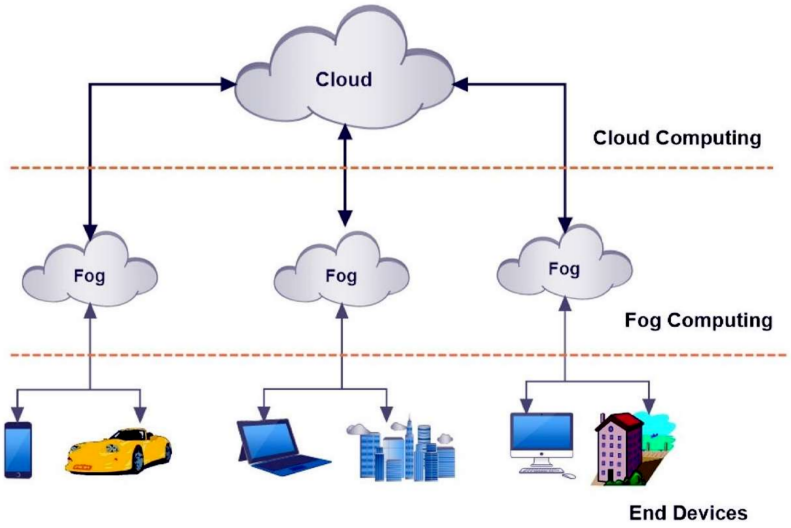

Figure 2.2: Paradigma de Computación en la Niebla

Conviene no confundir la Computación en la Niebla con otro modelo de también reciente popularidad, la Computación en la Frontera. En este caso, la computación se realiza en algunos de los propios dispositivos IoT, por ejemplo en ciertos sensores de la red. Este modelo lleva los cómputos lo más cerca posible de la capa de más bajo nivel de la red reduciendo la latencia lo máximo posible, como se presenta en Sarkar et al. (2025). No obstante, y aunque esta eficiencia en tiempo es fundamental en ciertos entornos e infraestructuras críticas del IoT como ciudades inteligentes o Health IoT, el análisis de tráfico malicioso en la red no requiere de dicha inmediatez. Además, realizar los propios cálculos en los dispositivos demanda una inversión económica y de recursos en hacer de estos nodos, muchas veces muy simples y con poca capacidad de procesamiento, capaces de soportar las computaciones requeridas. En el caso de modelos posiblemente complejos como una red neuronal seguramente no merece la pena la inversión por el beneficio.

Paradigma	Latencia	Consumo de Ancho de Banda	Escalabilidad	Control
Cloud Computing	Alta	Alta	Alta	Centralizado
Fog Computing	Baja	Baja	Media	Local
Edge Computing	Muy baja	Muy baja	Baja	Local

Table 2.1: Comparativa entre arquitecturas de computación

En el problema abordado en el presente trabajo la latencia y el consumo de ancho de banda no son elementos críticos a tener en cuenta, por tanto, y como se observa en el resumen presentado en 2.1, la Computación en la Niebla ofrece un equilibrio entre tiempo, escalabilidad y control que hace de ella una solución de especial interés para el despliegue del servicio de análisis de datos.

2.4 Arquitectura en Microservicios y Docker

Debido a la decisión de adoptar un modelo de arquitectura en microservicios, en lugar de una arquitectura monolítica, resulta importante comprender las características de este modelo de despliegue.

El principio fundamental sobre el que opera este tipo de arquitectura es el desacople.

Un enfoque en microservicios parte de la premisa de que una aplicación o servicio puede dividirse en componentes funcionales más pequeños, cada uno encargado de una responsabilidad específica y delimitada sin que sus funciones interfieran entre sí. En este esquema la aplicación es la unión de los microservicios, que trabajan de forma independiente pero comunicada para lograr un objetivo común.

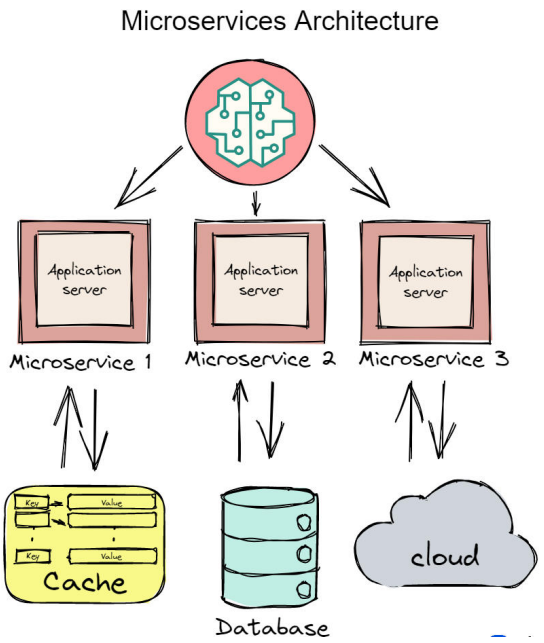

Figure 2.3: Diagrama de Arquitectura en Microservicios

A diferencia de la clásica arquitectura monolítica, donde la aplicación se concibe como una única unidad o servicio indivisible, los microservicios proporcionan modularidad, escalabilidad y facilidades en el despliegue y el mantenimiento como se discute en Richards and Ford (2020).

Cada microservicio tiene una funcionalidad clara, esto permite diferenciar claramente en qué parte del proceso se encuentran los cuellos de botella o las dificultades o deficiencias. Esta independencia permite, además, realizar el redespliegue únicamente de la tarea problemática y no de todo el servicio en su conjunto. Además, pueden escalarse los recursos de los microservicios que lo requieran.

Estas ventajas y su adecuación al sistema implementado en el presente trabajo, como se verá más adelante, justifican su adopción en la solución final.

Como herramienta para el despliegue de la arquitectura en microservicios se empleará Docker. Docker es un software de código abierto que permite desplegar aplicaciones de forma aislada en *contenedores*. Cada uno de estos contenedores agrupan todos los elementos que comprenden el dominio de dicha aplicación.

Para lograr esto se definen imágenes, plantillas que contienen el código y sus dependencias, y posteriormente se instancian creando los contenedores. Estas plantillas hacen de la aplicación un servicio portable y reproducible, pues cualquier sistema que posea la plantilla podrá instanciar exactamente el mismo contenedor. Docker permite también el empleo de volúmenes, almacenamientos persistentes de datos compartibles entre contenedores, que facilitan enormemente la integración de los microservicios tanto entre sí como con el host. Además, docker permite el despliegue ágil de múltiples contenedores en un solo sistema

anfitrión, lo cual unido a sus características de manejo de recursos y escalabilidad lo hacen una opción sólida y eficiente como herramienta de despliegue.

Se empleará la herramienta de orquestación Docker Compose para gestionar el despliegue y el ciclo de vida de múltiples microservicios. Este software permite agrupar dicho despliegue en un solo fichero yaml y gestionar de forma sencilla tanto los volúmenes compartidos como la red.

Estas herramientas y conceptos permiten entender el diseño del sistema implementado y el por qué se ha decidido adoptar una topología distribuida en detrimento de la arquitectura monolítica.

2.5 Aprendizaje Automático: Detectando lo Indetectable

De la mano del desarrollo tecnológico han aparecido máquinas cada vez más inteligentes. Los avances en los últimos años del *Machine Learning* permiten a los dispositivos aprender de los datos para tomar decisiones sobre datos que nunca antes han visto. Estas técnicas permiten automatizar tareas como la que se aborda en el presente trabajo, la detección de tráfico de bots en una red de IoT.

En esta sección se explicarán brevemente algunas de las técnicas más populares para lograr este objetivo, las cuales se mencionarán en secciones posteriores cuando se aborde el estado del arte, centrándose en la conexión de estas con el problema particular de la clasificación de tráfico.

La primera parte del proceso consistirá en determinar las características (*features*) más apropiadas para entrenar a los modelos. La elección de características es un paso fundamental pues se reciben múltiples métricas en la captura de tráfico y una sobrealimentación de estas a los modelos de aprendizaje automático puede hacer crecer drásticamente el tiempo de entrenamiento y predicción de los mismos sin aportar información nueva relevante. Se emplearán técnicas de selección de características como ANOVA o las importancias de modelos como el Random Forest. Además, un procedimiento recomendado es normalizar las variables numéricas y convertir las categóricas en identificadores numéricos, esto es necesario para el empleo de algunos de los modelos.

Se emplearán modelos de Aprendizaje Automático, siendo algunos de ellos modelos de Aprendizaje Profundo. En el Aprendizaje automático se diseñan sistemas capaces de aprender patrones y relaciones y tomar decisiones en base a los mismos sin haberles sido explícitamente definido cómo hacerlo. En el caso particular del Deep Learning el modelo trata de emular el razonamiento de un cerebro humano, implementando capas de neuronas que le permiten identificar, entender y emplear relaciones y procesos de mayor complejidad cuyo razonamiento no sabríamos o nos resultaría muy difícil programar como se argumenta en Buduma and Locascio (2017). Los modelos de Machine Learning, en definitiva, ofrecen una solución efectiva para la detección de tráfico Botnet en una red IoT al ser capaces de detectar los patrones subyacentes del tráfico malicioso.

Random Forest y Extra Trees

Entre los modelos cuyos resultados se analizarán el primero es **Random Forest**. Este modelo, un popular modelo de ensemble, consiste en la combinación de muchos *Árboles de Decisión* y selección por mayoría. En este tipo de árboles cada nodo planeta una posible división del dataset en base a una característica (por ejemplo, en el caso aquí abordado, flujos conversacionales con más de x bytes por paquete o con menos) mientras que los nodos hoja del árbol determinan la clasificación final de dicho elemento del conjunto. El

bosque de árboles proporciona estabilidad y robustez al modelo, pues basa su clasificación en la clasificación que hayan realizado la mayoría de sus miembros.

Además del **Random Forest** se hará uso de otro modelo de ensemble similar, **Extra Trees**. La diferencia entre ellos, siendo ambos combinaciones de árboles de decisión, radica en el sistema empleado para el entrenamiento de dichos árboles. **Random Forest** emplea *bagging* para seleccionar subconjuntos aleatorios para entrenar los árboles que lo componen, asegurando diversidad entre los mismos. Sin embargo, **Extra Trees** entrena los árboles de decisión con todo el dataset de entrenamiento que se le provee. Por tanto, para mantener la diversidad en las predicciones de sus miembros, realiza de forma aleatoria las divisiones de sus nodos. El empleo de la totalidad del dataset aumenta su sesgo pero reduce su varianza, es decir, disminuye el sobreajuste del entrenamiento, mientras que la aleatoriedad agiliza el proceso de entrenamiento. En el problema abordado en el presente trabajo, como se verá en secciones subsecuentes, existe un alto problema debido al desbalanceo de las clases en el dataset, haciendo del sobreajuste un riesgo a tener en cuenta.

Gradient Boosting, XGBoost y LightGBM

Además de los algoritmos de ensemble recién mencionados se realizarán pruebas con modelos que emplean potenciación del gradiente (**Gradient Boosting**). El procedimiento base sobre el que se fundamentan este tipo de algoritmos es el *boosting*, es decir, el entrenamiento secuencial de modelos débiles, como árboles de decisión, tratando de mejorar en cada uno el rendimiento de su predecesor. En este caso esto se realiza mediante la disminución de la función de pérdida empleando el descenso del gradiente, es decir, tomar la dirección opuesta al gradiente como método para avanzar de un paso del boosting al siguiente.

Como caso particular de este tipo de modelos se implementarán **XGBoost** y **LightGBM**. Por tanto, se procederá a aclarar las particularidades de estos y qué ventajas ofrecen para el problema de detección de tráfico malicioso. **Extreme Gradient Boosting** incorpora ciertas ventajas computacionales y estadísticas entre las que destacan la regularización L1 y L2 para evitar overfitting, post-pruning (eliminar ramas que no aportaban información relevante) y paralelización permitiéndole procesar de forma mucho más sencilla datasets de gran tamaño. Además, este algoritmo maneja de forma nativa valores faltantes obteniendo similares resultados a los casos en los que se imputan los valores faltantes empleando diferentes técnicas, como se discute en Ergul Aydin and Kamisli Ozturk (2021), lo cual es habitual en datos relativos a tráfico IoT. **LightGBM** es un método desarrollado por Microsoft que ofrece soporte a variables categóricas sin codificar (como podría ser el protocolo empleado), usa leaf-wise growth en lugar de level-wise (es decir, divide el nodo con mayor ganancia independientemente de su profundidad en lugar de dividir todos los nodos del mismo nivel antes de proceder con el siguiente, esto aumenta el riesgo de sobreajuste) y optimiza el tiempo de entrenamiento mediante el empleo de histogramas. Por tanto, ambas opciones presentan ventajas muy relevantes en el desafío que se aborda, algunas a nivel de optimización de recursos y tiempo y otras enfocadas en el sobreajuste.

Dando por concluido el conjunto de modelos de aprendizaje automático no profundo relevantes se procede a presentar de forma sencilla y breve las redes neuronales empleadas.

Perceptrón Multicapa

En primer lugar, se aborda el **perceptrón multicapa (MLP)**. El MLP es una red neuronal cuyas capas de neuronas están completamente conectadas y poseen una función de activación no lineal, haciendo de ella un modelo especialmente fuerte en casos donde los

datos no son linealmente separables. Este modelo supone un punto de partida óptimo antes del empleo de redes más complejas, su capacidad para procesar datos tabulares (como los csv) hace que se adapte muy bien a la situación contemplada.

Autoencoders

Se ha hecho uso también de **Autoencoders**. Estos modelos, de aprendizaje no-supervisado, se emplearán para tratar de reconstruir tráfico normal y detectar anomalías. Es decir, se entrena empleando tráfico únicamente no malicioso y luego, al recibir el tráfico real completo, la red neuronal es capaz de detectar desvíos en su intento de reconstrucción, siendo estos desvíos síntoma de posibles botnets.

GRU y Transformadores

Además, debido a que los datos de tráfico son datos ordenados temporalmente, se ha considerado relevante el empleo de redes neuronales secuenciales, en particular redes de tipo **GRU**. Este tipo de modelo es capaz de capturar dependencias temporales relevantes, descartando aquellas que no lo son, lo cual resulta muy efectivo si se conciben los envíos de paquetes como flujos temporales de conversaciones entre dispositivos en lugar de elementos por separado.

Por último, se han empleado **transformadores**. Estos modelos detectan relaciones globales en secuencias de flujos conversacionales empleando atención (es decir, asignación de importancias diferentes a características diferentes de los datos). Su coste computacional es, sin embargo, elevado, lo cual resulta un factor determinante en la adopción de dicho modelo como el más óptimo.

Los modelos introducidos en esta sección son la base de las pruebas realizadas y sus resultados motivan la decisión de implementar únicamente uno de ellos en la solución final.