UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería de Telecomunicación

Diseño e implementación de un sistema inteligente para la detección de ataques Man-in-the-Middle basados en ataques ARP poisoning

Trabajo Fin de Grado

Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación

AUTOR/A: Martínez Cadenas, José Alberto

Tutor/a: Palau Salvador, Carlos Enrique

CURSO ACADÉMICO: 2023/2024

Resumen

El TFG presentado irá de la mano con el proyecto GUARDIAN. GUARDIAN es un proyecto europeo en colaboración con la universidad de Wisconsin de Estados Unidos en donde se desarrollan herramientas de detección y recuperación de ataques *Ransomware* sobre datos en movimiento. La contribución del TFG se basa en el diseño y la implementación de un sistema de inteligencia artificial (IA) para la detección de ataques *Man-in-the-middle* (MITM) basados en ataques *ARP poisoning*. Para el desarrollo del sistema de inteligencia artificial usaremos el *dataset* IoTID20, el cual contiene información sobre el tráfico de una red de dispositivos *Internet of Things (IoT)*. Dicho *dataset*, consta de una gran variedad de ataques y 86 características de estos. Para hacer útil el *dataset*, debemos modificarlo y adaptarlo según nuestras necesidades y objetivos, para así poder detectar y obtener las características más relevantes e implementarlas como entradas para el entrenamiento de nuestro sistema de inteligencia artificial. Una vez desarrollado el modelo de IA se pasará a evaluar sobre un banco de pruebas, este entorno cuenta con una infraestructura de red y mecanismos de seguridad basados en entornos virtuales, donde un servidor aloja tres máquinas virtuales: Cliente, MITM y Servidor, diseñado para replicar la complejidad del ataque y de esta forma poder comprobar la correcta implementación del modelo de detección y su validación frente al problema planteado.

Palabras clave: Ciberseguridad; ransomware; inteligencia artificial; man- in the middle

Abstract

The TFG presented will go hand in hand with the GUARDIAN project. GUARDIAN is a European project in collaboration with the University of Wisconsin in the United States where tools for the detection and recovery of Ransomware attacks on data in motion are developed. The TFG contribution is based on the design and implementation of an artificial intelligence (AI) system for the detection of Man-in-the-middle (MITM) attacks based on ARP poisoning attacks. For the development of the AI system, we will use the IoTID20 dataset, which contains information about the traffic of a network of IoT devices. This dataset consists of a wide variety of attacks and their characteristics. To make the dataset useful, we must modify and adapt it according to our needs and objectives, to detect and obtain the most relevant features and implement them as inputs for the training of our artificial intelligence system. Once the AI model is developed, it will be evaluated on a testbed; this environment has a network infrastructure and security mechanisms based on virtual environments, where a server hosts three virtual machines: Client, MITM, and Server, designed to replicate the complexity of the attack and thus be able to verify the correct implementation of the detection model and its validation against the problem posed.

Keywords: cybersecurity; ransomware; artificial intelligence; man-in-the-middle; man-in-the-middle.

Resum

El TFG presentat anirà de bracet amb el projecte GUARDIAN. GUARDIAN és un projecte europeu en col·laboració amb la universitat de Wisconsin dels Estats Units on es desenvolupen ferramentes de detecció i recuperació d'atacs Ransomware sobre dades en moviment. La contribució del TFG es basa en el disseny i la implementació d'un sistema d'intel·ligència artificial (IA) per a la detecció d'atacs Man-in-the-middle (MITM) basats en atacs ARP poisoning. Per al desenvolupament del sistema d'intel·ligència artificial usarem el dataset IoTID20, el qual conté informació sobre el trànsit d'una xarxa de dispositius IoT. Este dataset, consta d'una gran varietat d'atacs i 86 característiques d'estos. Per a fer útil el dataset, hem de modificar-ho i adaptar-ho segons les nostres necessitats i objectius, per a així poder detectar i obtindre les característiques més rellevants i implementar-les com a entrades per a l'entrenament del nostre sistema d'intel·ligència artificial. Una vegada desenvolupat el model de IA es passarà a avaluar sobre un banc de proves, este entorn compta amb una infraestructura de xarxa i mecanismes de seguretat basats en entorns virtuals, on un servidor allotja tres màquines virtuals: Client, MITM i Servidor, dissenyat per a replicar la complexitat de l'atac i d'esta manera poder comprovar la correcta implementació del model de detecció i la seua validació enfront del problema plantejat.

Paraules clau: Ciberseguretat; ransomware; intel·ligència artificial; man-in-the-middle

RESUMEN EJECUTIVO

La memoria del TFG del Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación debe desarrollar en el texto los siguientes conceptos, debidamente justificados y discutidos, centrados en el ámbito de la ingeniería de telecomunicación

CONCEPT (ABET)	**CONCEPTO (traducción)**	**¿Cumple? (S/N)**	**¿Dónde? (páginas)**
1. IDENTIFY:	1. IDENTIFICAR:		
1.1. Problem statement and opportunity	1.1. Planteamiento del problema y oportunidad	S	5
1.2. Constraints (standards, codes, needs, requirements & specifications)	1.2. Toma en consideración de los condicionantes (normas técnicas y regulación, necesidades, requisitos y especificaciones)	S	17-19
1.3. Setting of goals	1.3. Establecimiento de objetivos	S	6, 12-16
2. FORMULATE:	2. FORMULAR:		
2.1. Creative solution generation (analysis)	2.1. Generación de soluciones creativas (análisis)	S	7-11, 20-23
2.2. Evaluation of multiple solutions and decision-making (synthesis)	2.2. Evaluación de múltiples soluciones y toma de decisiones (síntesis)	S	24-40
3. SOLVE:	3. RESOLVER:		
3.1. Fulfilment of goals	3.1. Evaluación del cumplimiento de objetivos	S	41
3.2. Overall impact and significance (contributions and practical recommendations)	3.2. Evaluación del impacto global y alcance (contribuciones y recomendaciones prácticas)	S	42

Índice General

Índice de Figuras

Índice de tablas

CAPÍTULO 1. INTRODUCCIÓN

Hoy en día, la ciberseguridad es una de las mayores amenazas y problemas los cuales enfrenta la sociedad; según datos de (WatchGuard, 2024) en 2023 se registró que cada 39 segundos tuvo lugar un ciberataque, generando así un total de 2.200 ataques diarios, en donde el *Ransomware* figura como uno de los ataques con mayor cantidad de víctimas, resultando como los sectores más afectados a: en 1er lugar el sector gubernamental y militar, en 2do lugar el sector financiero y bancario y en 3er lugar el sector de la educación e investigación. Además del mal trago que supone la pérdida de información, se estima que los gastos globales en tanto a la ciberseguridad superarán los 1.75 billones de dólares de forma acumulativa entre los años comprendidos del 2021 al 2025 (Braue, 2021).

Como bien sabemos, la ciberseguridad es un sector en auge y continuo crecimiento, en donde las empresas lideres del sector tecnológico (como Google, Amazon, Microsoft, etc.) han estado invirtiendo fuertemente para el desarrollo y estudio de los innumerables retos que presenta la ciberseguridad y que a día de hoy son un gran dolor de cabeza.

Debido a las increíbles prestaciones y campos de mejora que presenta el sector de ciberseguridad, decidí realizar este TFG, en donde a través de la Inteligencia Artificial, se llevará a cabo el desarrollo e implementación de un sistema inteligente para la detección de un ataque *Man-In-The-Middle* (MITM) basado en *ARP Spoofing* (modificación de las tablas ARP).

De esta forma, a lo largo de la memoria, se explicará la metodología, planteamiento y desarrollo seguido para poder brindar así, una propuesta de solución y poder contribuir, en cierta forma, con el desarrollo e investigación del sector de la ciberseguridad.

1.1 MOTIVACIÓN

Basándonos en los aterradores datos comentados anteriormente y siendo consciente de la problemática actual que presenta la ciberseguridad, decidí realizar una exploración sobre este campo, formular hipótesis y validarlas con experimentos prácticos y así, poder ayudar en el sector de ciberseguridad de alguna manera.

Durante mi periodo de prácticas, en el grupo de investigación de Sistemas de Tiempo Real y Distribuido (STARD) de la Universidad Politécnica de Valencia, se presentó la oportunidad de trabajar en el proyecto GUARDIAN, el cual es un proyecto europeo liderado por el grupo STARD de la Universidad Politécnica de Valencia en colaboración de la Universidad de Wisconsin, Madisson, USA. El proyecto GUARDIAN consiste en un estudio empírico y análisis práctico de los ataques *ransomware* sobre datos en movimiento; debido a mi interés por el sector de la ciberseguridad y los conocimientos adquiridos durante el grado de telecomunicaciones, el grupo me incorporó como parte de los integrantes del proyecto, en donde mi función principal sería aportar un sistema de IA capaz de detectar un posible ataque MITM.

GUARDIAN es un proyecto innovador, el cual busca atacar un sector del área de ciberseguridad del cual no se tiene prácticamente ningún estudio ni investigaciones pertinentes, de esta forma seriamos los primeros en aportar un estudio completo de ataques *Ransomware* sobre datos en movimiento. Juntando así este hecho, y que mi tarea asignada involucraría en gran parte la ciberseguridad y la implementación de modelos de inteligencia artificial, dicho proyecto supondría para mí una gran oportunidad, en la cual podría formar parte de un gran proyecto europeo y trabajar en sectores innovadores y áreas de gran alcance tecnológico.

De esta forma, es cómo surgió mi TFG, el cual encaramos con la gran motivación e ilusión que nos supondría alcanzar los objetivos planteado y las áreas de trabajo en las que estaría involucrado.

1.2 OBJETIVOS

El objetivo de este TFG es aportar el diseño e implementación de un modelo inteligente para la detección de ataques *Man-In-The-Middle* basados en ataques *ARP poisoning*. Para llevar a cabo dicho objetivo, el proyecto fue segmentado en varios subobjetivos:

- Estudio y análisis de ataques MITM y *Ransomware*
- Planteamiento del problema e identificación de puntos claves
- Estructuración de las herramientas necesarias para llevar a cabo el proyecto
- Implementación software para el diseño de un modelo de Inteligencia Artificial
- Validación de los modelos desarrollados sobre entornos prácticos que simulan escenarios de la vida real
- Aportación del modelo final desarrollado
- Aportación de las conclusiones obtenidas

1.3 ESTRUCTURA DE LA MEMORIA

La presentación de la memoria estará formada por 6 capítulos, en los cuales se abordará todos los temas correspondientes en tanto al trabajo realizado.

Capítulo 1. Introducción: se pondrá en contexto el TFG presentado, aportando los objetivos a desarrollar y el motivo de la elección de dicho TFG.

Capítulo 2. Estado del Arte: se presentarán todas las herramientas utilizadas durante la elaboración del proyecto, introduciendo así al lector al conocimiento de las mismas.

Capítulo 3. Metodología y desarrollo: en esta sección se plantera la organización y estructura presentada para llevar a cabo los objetivos mencionados.

Capítulo 4. Diseño de la solución: se presentará el modelo a desarrollar, estableciendo así la propuesta innovadora frente al problema en cuestión, y la metodología a seguir.

Capítulo 5. Implementación y validación de los experimentos: sección donde se detalla los experimentos a desarrollados, mostrando su validación y conclusiones obtenidas.

Capítulo 6. Conclusión y trabajos futuros: como ultima sección, se desarrollará la conclusión del proyecto, así como posibles mejoras y áreas de aplicación.

CAPÍTULO 2. ESTADO DEL ARTE

En esta sección repasamos el panorama actual de las tecnologías y metodologías exploradas en el proyecto. Mediante el estudio de los últimos avances, los trabajos publicados y las herramientas disponibles, pretendemos ofrecer una visión completa del estado del arte que presenta el proyecto. La exploración abarca los puntos clave en los que se basará el proyecto, como lo son el área de Inteligencia Artificial y los conjuntos de datos (*datasets*) utilizados en trabajos similares. De esta forma, se presentarán componentes específicos para el área de IA, como por ejemplo, los modelos y algoritmos más comunes, también se expondrán librerías de software destacadas, como Scikit-learn, y herramientas esenciales, como Wireshark y PyShark.

2.1 Inteligencia Artificial (AI)

El aprendizaje automático (Machine Learning, ML) es la capacidad de los sistemas inteligentes para aprender y mejorar a través de la experiencia adquirida por datos históricos, sin necesidad de programación ni de ninguna otra intervención humana (Michell, 1997). Existen varios tipos de ML, como el Supervisado, el No Supervisado, el Semisupervisado y el Aprendizaje por Refuerzo.

Los tipos más comunes de ML son el Supervisado y el No Supervisado.

- los modelos Supervisados, son aquellos que trabajan con una fuente de datos ya clasificados para el entrenamiento, poseen así una etiqueta que los identifica, de esta forma, el modelo debe ser capaz de clasificar correctamente dichos datos en donde se le debe asignar la misma etiqueta que tenían en un principio.
- y los modelos No Supervisados, no cuentan con una fuente de datos que previamente están etiquetados para su entrenamiento, por lo que el modelo etiqueta interiormente los datos, basándose en tendencias, parámetros y conductas similares.

Para llevar a cabo la predicción de los modelos, existen diferentes algoritmos: de clasificación, de regresión, de agrupamiento (*claustering*) y de reducción de dimensionalidad; dependiendo el problema que afrontemos, determinado algoritmo prestará mejor comportamiento que otro. Entre los algoritmos más comunes tenemos:

- **Árbol de decisión (*Decision Tree*): es** un algoritmo de aprendizaje supervisado, el cual basa la división de los datos según sus características, logrando mantener entre sus divisiones, conjuntos de datos que presentan valores y comportamientos similares
- **Bosque aleatorio (*Random Forest*):** es un algoritmo de aprendizaje supervisado, que basa sus predicciones en múltiples arboles de decisiones, buscando mejorar la precisión y reducir el sobreajuste.
- **Máquinas de vectores soporte (SVM):** Las SVM son potentes modelos de aprendizaje supervisado que se utilizan para tareas de clasificación y regresión. Funcionan encontrando el hiperplano que mejor separa las clases en un espacio de alta dimensión, maximizando el margen entre clases.

- **Esemble:** los modelos basados en un algoritmo esemble, se basan en la combinación de varios algoritmos para intentar conseguir así una respuesta final mejorada, normalmente el algoritmo de aprendizaje esemble se basa en un Decision Tree y un Random Forest.
- **K-Primeros-Vecinos (k-NN):** k-NN es un algoritmo de aprendizaje basado en instancias, el cual resulta sencillo pero eficaz, que se utiliza para tareas de clasificación y regresión. Clasifica los nuevos puntos de datos en función de la clase mayoritaria de sus k vecinos más cercanos en el espacio de características. k-NN no es paramétrico y no requiere un entrenamiento explícito del modelo, por lo que resulta adecuado tanto para conjuntos de datos pequeños como grandes.
- **Análisis Discriminante Lineal (LDA):** es un algoritmo de clasificación supervisado, el cual basa sus predicciones en encontrar las combinaciones lineales que mejor divida las clases en base de los datos proporcionados.

2.2 Scikit-learn

Scikit-learn (F.Pedregosa, 2011) es una librería de aprendizaje automático de software libre para el lenguaje de programación Python. Incluye varios algoritmos de clasificación, regresión y análisis de conglomerados, como máquinas de vectores soporte, bosques aleatorios, *Gradient boosting, K-means y DBSCAN*. Está diseñado para interoperar con las bibliotecas numéricas y científicas NumPy y SciPy.

La API de Scikit-Learn se ha desarrollado siguiendo ciertos principios para garantizar que pueda aplicarse fácilmente en una amplia gama de dominios. Sigue cinco principios que permiten un uso práctico y sencillo de la biblioteca, incluso para personas sin una sólida formación matemática:

- **Coherencia:** Todos los objetos comparten una interfaz común derivada de un conjunto limitado de métodos, con una documentación coherente.

- **Inspección:** Todos los valores de los parámetros especificados se exponen como atributos públicos.

- **Integración con bibliotecas científicas de Python:** La perfecta integración con otras bibliotecas científicas populares de Python como NumPy, Pandas y Matplotlib facilita la manipulación, el análisis y la visualización de datos dentro de un entorno cohesivo.

- **Jerarquía de objetos limitada:** Los algoritmos están representados únicamente por clases Python; los conjuntos de datos están representados en formatos estándar (arrays NumPy, DataFrames Pandas, matrices dispersas SciPy), y los nombres de los parámetros utilizan cadenas Python estándar.

- **Composición:** Muchas tareas de aprendizaje automático se pueden expresar como secuencias de algoritmos más fundamentales, y Scikit-Learn hace uso de esto siempre que sea posible.

- **Valores predeterminados razonables:** Cuando los modelos requieren parámetros especificados por el usuario, la biblioteca define valores predeterminados adecuados. Estos principios se describen directamente en este documento que describe los principios de la API de Scikit-Learn.

2.3 Wireshark

Wireshark (U. Banerjee, 210) es un analizador de protocolos multiplataforma que se utiliza para realizar análisis y solucionar problemas en redes de comunicaciones, análisis de datos y protocolos, y como herramienta educativa.

Wireshark ofrece una interfaz gráfica de usuario que permite explorar de forma interactiva los paquetes de red capturados. Los usuarios pueden capturar tráfico de red en directo desde varias interfaces o abrir archivos de captura pregrabados para analizarlos sin conexión. Es compatible con la capa física, la capa de enlace, los protocolos de red, la capa de transporte y los protocolos de la capa de aplicación. Wireshark es compatible con una amplia gama de protocolos de red, como Ethernet, Wi-Fi, TCP/IP, UDP, HTTP, DNS, DHCP y SSL/TLS, entre otros.

Wireshark ofrece numerosas ventajas:

- **Versatilidad:** Es un programa increíblemente versátil que soporta más de 480 protocolos diferentes, permitiéndonos trabajar con datos capturados de una sesión de red mediante paquetes. Estos pueden ser almacenados para su posterior análisis.

- **tcpdump:** Soporta un formato de fichero estándar llamado tcpdump, permitiendo la reconstrucción de sesiones TCP. Todo ello soportado por una interfaz gráfica bastante accesible, que se hace relativamente fácil de aprender con algo de práctica.

- **Abundantes prestaciones:** Sus características son muy beneficiosas para los administradores, que pueden utilizarlo para solucionar problemas de red como la congestión del tráfico o el consumo excesivo. También puede ayudar a identificar y prevenir vulnerabilidades de seguridad.

- **Propósitos de desarrollo:** Puede utilizarse para depurar implementaciones de protocolos de red en niveles de desarrollo.

- **Fines educativos:** Es una herramienta valiosa para aprender sobre las operaciones de red y diseccionar la información contenida en los paquetes de los dispositivos conectados a la red analizada. Podemos observar información como nombres de usuario, contraseñas, mensajes e incluso determinar la marca y el modelo de los dispositivos de la red.

- **Multiplataforma:** Puede funcionar en los principales sistemas operativos como Windows, Linux, OS X, entre otros.

- **Actualizaciones periódicas:** Recibe actualizaciones frecuentes, no sólo introduciendo nuevas funcionalidades sino también incorporando numerosos parches de seguridad.

2.4 Pyshark

PyShark es una librería Python que proporciona una interfaz programática para trabajar con tráfico de red capturado o en vivo. Va más allá de una simple envoltura para la popular herramienta de línea de comandos Tshark, ofreciendo una interfaz de alto nivel para interactuar con paquetes capturados. Esta sección profundiza en las funcionalidades y ventajas de PyShark, convirtiéndolo en un valioso activo en la caja de herramientas del investigador moderno (KimiNewt, 2023).

Las principales características de PyShark incluyen:

- **Soporte de archivos de captura**: PyShark permite a los usuarios leer y procesar archivos de captura de red en varios formatos, incluyendo PCAP (Packet Capture), PCAPNG (Next Generation Packet Capture) y otros. Esta capacidad permite el análisis offline del tráfico de red previamente capturado.

- **Captura en vivo:** PyShark soporta la captura de paquetes de red en vivo, permitiendo a los desarrolladores capturar el tráfico de red en tiempo real directamente desde las interfaces de red. Esta característica es inestimable para monitorizar la actividad de la red a medida que ocurre y realizar análisis en tiempo real.

- **API de alto nivel:** PyShark proporciona una interfaz Pythónica de alto nivel para interactuar con paquetes de red, facilitando a los desarrolladores trabajar con datos de paquetes sin necesidad de entender las complejidades de bajo nivel de las estructuras de paquetes. Esta abstracción simplifica el proceso de análisis y manipulación de la red.

- **Análisis e inspección de paquetes:** Con PyShark, los usuarios pueden analizar paquetes de red individuales e inspeccionar su contenido, incluyendo cabeceras de protocolo, datos de carga útil y otra información relevante. Esta capacidad permite un análisis detallado de los protocolos de red y los patrones de comunicación.

- **Soporte de Protocolos:** PyShark soporta una amplia gama de protocolos de red fuera de la caja, incluyendo protocolos comunes como TCP, UDP, HTTP, DNS, SSL/TLS, y muchos otros. Este amplio soporte de protocolos lo hace adecuado para analizar diversos entornos de red y aplicaciones.

- **Procesamiento basado en eventos:** PyShark permite a los desarrolladores definir controladores de eventos personalizados que se activan en función de condiciones específicas o eventos encontrados durante el procesamiento de paquetes. Esta arquitectura basada en eventos facilita el análisis avanzado de paquetes y permite a los desarrolladores implementar una lógica personalizada adaptada a sus requisitos específicos.

- **Integración con Wireshark:** Como PyShark está construido sobre la librería Wireshark, aprovecha las amplias capacidades de decodificación y disección de protocolos de Wireshark. Esta integración asegura un soporte robusto de protocolos y un análisis preciso de paquetes.

- **Compatibilidad multiplataforma:** PyShark está diseñado para funcionar sin problemas en diferentes sistemas operativos, incluyendo Windows, macOS y Linux, proporcionando flexibilidad y portabilidad para tareas de análisis de red.

2.5 Dataset IoTID20

El conjunto de datos IoTID20 (Ullah, 2020) es una colección de datos de tráfico de red diseñada específicamente para entrenar y probar sistemas de detección de intrusiones (IDS) para escenarios de *Internet of Things* (IoT). El *dataset* IoTID20, fue recopilado sobre el tráfico de una red de dispositivos de IoT sobre un ambiente de *smart home*, en este ambiente, se utilizaron laptops, *tablets*, *smartphones*, un punto de acceso SKT NGU y una cámara WI-FI, que estaban conectados a un router WI-FI. Los dispositivos SKT NGU y la cámara WI-FI fueron utilizados para recopilar los datos, y actuaban como victimas para el entorno anómalo de ataques que se quería diseñar sobre el ambiente de IoT, por otra parte, el resto de los dispositivos eran el medio por el cual se implementaban las acciones anómalas sobre la red, es decir, se comportaban como atacantes.

De esta manera, tenemos que el *dataset* cuenta con flujos de red Normales (es decir, la comunicación típica entre dos máquinas A y B, con su flujo de datos de petición y respuesta típicos, sin alteración ninguna en el proceso) y con 8 tipos de ataques informáticos basados en ataques Miraia, DoS, MITM y Scan. La estructura del *dataset* cuenta con una gran cantidad de datos, alrededor de 625.000 filas y 86 etiquetas (columnas) que corresponden con características muy específicas sobre cada flujo de datos recopilados en la red, estas características nos ayudan a poder realizar un estudio más profundo sobre cada flujo y por consiguiente sobre cada ataque, en donde podemos implementar prácticas de análisis, observación y discriminación para transformar dicha información de la manera más favorable según nuestros objetivos.

De esta forma, como podemos ver en la Figura 1, tenemos que el conjunto de datos IoTID20 clasifica el tráfico en cinco categorías:

- **Normal:** Tráfico de red legítimo.

- **Mirai:** Ataques asociados a la red de *bots* Mirai, que infecta dispositivos y lanza ataques de denegación de servicio.

- **DoS (denegación de servicio):** Ataques cuyo objetivo es interrumpir o inutilizar un servicio saturándolo de tráfico.

- **Scan:** Actividades en las que los atacantes sondean una red para identificar vulnerabilidades.

- **MITM (Man-in-the-Middle):** Ataques en los que los atacantes se sitúan entre dos dispositivos para espiar la comunicación o manipular datos.

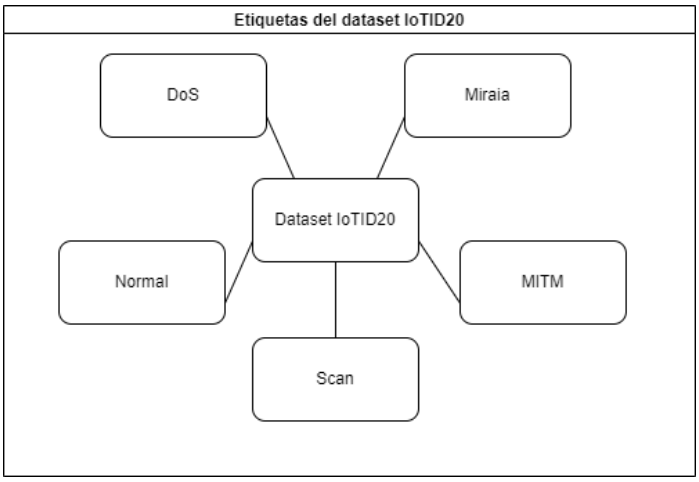


Figura 1. Estructura del dataset IoTID20 según la etiqueta de sus datos

CAPÍTULO 3. METODOLOGÍA

Figura 2. Machine Learning Process Flow

La metodología seguida en tanto a la realización del proyecto corresponde en gran parte a las tareas correspondientes de un *Data Scient* (Analistas de Datos), por lo que los pasos seguidos son los correspondientes al *machine learning process flow* (flujo de procedimiento de aprendizaje automático) correspondiente a la figura 2.

De esta forma, también definimos una estructura para llevar a cabo la organización e implementación del proyecto, la cual también será utilizada para el desarrollo de los experimentos, siguiendo así una base organizada para tratar con detalle el diseño, entrenamiento y validación de nuestros modelos; así, la estructura planteada es la siguiente:

1. **Adquisición de datos y preprocesado**

   En esta sección se explicará la manera de cómo se obtiene los datos que usaremos más adelante para el entrenamiento del modelo de IA, el preprocesado que realizamos para ajustar los datos, con el objetivo de que el modelo los pueda interpretar de la mejor manera y, por último, como el modelo es capaz de recoger estos datos.

2. **Modelo de IA**

   En esta sección se explicará al detalle todo el proceso que realizamos para el entrenamiento del modelo de IA, aportando los algoritmos y funciones utilizadas.

3. **Validación**

   Por último, en esta sección se recogen todos los resultados obtenidos en tanto a los estudios realizados, aportando las métricas y datos concretos para demostrar con base sólida el proceso de evaluación.

1. Adquisición de datos y preprocesado

El primer paso que debemos determinar es la fuente de información de la cual nos basaremos para realizar nuestro estudio, es decir, cabe determinar los datos que tomaremos como base para el entrenamiento e interpretación del modelo, estos datos son los que le pasaremos a nuestra IA como modo de información para alimentarla de conocimiento sobre nuestra estructura de datos y entorno, tras los cuales tendrá que caracterizar/diferenciar si se está produciendo un posible ataque MITM o no.

De esta forma, es de gran importancia tener una base sólida para entrenar nuestro modelo, por lo que surge la necesidad de encontrar un *dataset* que estuviera bien clasificado y etiquetado, que contuviera flujo anómalo y flujo normal, y que además contara con una gran cantidad de datos e información para poder realizar estudios y experimentos que se ajustaran a los resultados y previsiones esperadas. De esta forma, optamos por usar el *dataset* IoTID20, el cual recoge todos los requisitos planteados.

Definido nuestro *dataset* base, pasaremos a trabajarlo y ajustarlo según nuestro objetivo, lo que se conoce en *machine learning* como preprocesado de datos.

Para modelar un sistema de IA, los datos proporcionados deben ser lo menos complejos posibles, para de cierta forma evitar confusiones y tareas de clasificación complejas al modelo, lo que mejorará sus prestaciones en un futuro; De esta forma, cuando queramos trabajar con dicho *dataset*, primero debemos realizar un preprocesado; así, como primer paso, debemos analizar la estructura de datos que presenta el *dataset* y simplificarlo si es posible. Para ello, leemos el *dataset* que se encuentra en un formato CSV, lo pasamos a un *dataframe* de pandas y seguidamente aplicamos la instrucción df.info () para leer la naturaleza de los datos; en dicha resolución, encontramos varios datos de naturaleza tipo *object*, lo que no es nada conveniente para un modelo de IA, por lo que pasaremos dichos datos a tipo *Integer*, lo cual es una estructura de datos fácilmente manejables e interpretables en los modelos de IA.

Con este simple paso, ya podemos acceder directamente a los datos y trabajar con ellos para el desarrollo y entrenamiento de nuestro modelo de inteligencia artificial para detecta un posible ataque MITM.

De todas formas, recordemos que este *dataset* es simplemente nuestro primer punto de partida; con él, realizaremos los primeros estudios en los cuales esperaremos encontrar conclusiones de gran valor.

De igual manera, otra de nuestras aportaciones, será la realización de un nuevo *dataset*, el cual llamaremos DatasetUPV. El objetivo de este *dataset* es proporcionar a la comunidad científica y de investigación, otra basa de estudio para desarrollar, implementar y mejorar los estudios sobre los posibles ataques de MITM en una red de flujo de datos.

Para implementar nuestro DatasetUPV, estaremos generando y capturando tráfico de red a través de la herramienta Wireshark, durante 3 días. La información recogida durante este periodo de tiempo se encontrará en uno o varios pcap (extensión de los archivos provenientes de Wreshark), en donde haciendo uso de la librería PyShark de Python, seremos capaces de extraer la información recogida de nuestra red y traducirla para generar finalmente nuestro DatasetUPV. Obviamente, antes de utilizar el *dataset*, debemos de realizar el mismo preprocesado de datos que realizamos para el *dataset* IoTID20, en donde modificamos la naturaleza de los datos para que el modelo de IA pueda interpretar fácilmente los datos proporcionados.

2. IA

Como bien explicamos en la introducción y estado del arte, nuestro modelo de IA será nuestra arma de detección frente a un posible ataque de MITM.

Para el diseño e implementación de nuestro modelo de IA usaremos la librería Sciket-Learn, que es una herramienta en Python especializada para el entorno de inteligencia artificial. De esta forma, Scikit-Learn nos proporciona una amplia selección de sublibrerias y funciones, las cuales serán clave pen este proceso de creación del modelo, como por ejemplo, característica_importance, train_test_split, StandardScaler, etc.

3. Validación

En los experimentos que iremos desarrollando, realizaremos varios entrenamientos para los modelos de IA, cada uno con un tipo de datos de entrada distinto y ajustado sus características en función de los experimentos planteados; pero para llevar a cabo la validación de cada uno de ellos, utilizaremos la herramienta Wireshark para capturar todo el tráfico de una red en tiempo real.

El objetivo de estas capturas es obtener datos que los modelos nunca hayan visto, es decir, una cantidad de datos totalmente distinta a los datos con el que el modelo fue entrenado; de esta manera podremos validar las prestaciones y el comportamiento de nuestro modelo frente a nuevos datos y comprobar si es capaz de realizar de forma correcta la clasificación de MITM o no.

Ahora bien, no todo es tan fácil como parece; de las capturas Wireshark, lo que realmente queremos es extraer las características con las que entrenamos el respectivo modelo que vamos a evaluar, pero a la hora de extraer dicha información, nos dimos cuenta de que las características, no se encuentran etiquetadas de la misma manera dentro de los datos que proporciona Wireshark, es decir, en una captura Wireshark no vamos a encontrar un dato que sea, por ejemplo, “Flow_IAT_Mean” o “Flow_Byts/s”. Por lo que para extraer esas características, tendremos que jugar con la información que nos proporciona Wireshark.

Para solucionar esta tarea, nos vimos con la necesidad de programar un scrip en el cual pudiéramos leer toda la captura Wireshark y a partir de los datos que vayamos leyendo, ser capaces de extraer las características que nos interesan. Para ello, utilizaremos la herramienta Pyshark de Python, la cual nos permite leer y extraer datos directamente de una captura Wireshark.

De esta manera, seremos capaces de obtener al detalle las características que nos interesan, en donde las pasaremos como entrada a nuestro modelo y seguidamente, con la ayuda de las métricas de validación, podremos llevar a cabo nuestra validación.

Ahora bien, como nuestro objetivo es identificar si se está produciendo un posible ataque de MITM o no, nuestro modelo de IA se basa en un problema de clasificación; para este tipo de problemas, el objetivo es asignar una clase o categoría a cada instancia de datos basándose en sus características.

Y a la hora de evaluar el modelo, las métricas que cabrían estudiar serian la precisión, el recall, la accuracy y el F1 Score, los cuales obtendremos a partir de la matriz de confusión. Estas métricas son importantes ya que darán información de como nuestro modelo está trabajando las diferentes clases, donde, lo más importante para nosotros es minimizar la cantidad de falsos positivos y falsos negativos, es decir, detectar un ataque MITM que no lo es o dejar pasar un tráfico que es MITM, pero fue clasificado como normal.

Matriz de confusión

La matriz de confusión es una herramienta que nos permite evaluar el rendimiento de un modelo de forma visual, en donde se representa la cantidad de clasificaciones correctas frente a las incorrectas.

TN	FP
FN	TP

TN: El número de veces que el modelo predijo correctamente la clase negativa.

FN: El número de veces que el modelo predijo incorrectamente la clase negativa, es decir, clasifico cierta entrada como negativa cuando en realidad era positiva

TP: El número de veces que el modelo predijo correctamente la clase negativa.

FP: El número de veces que el modelo predijo incorrectamente la clase positiva, es decir, clasifico cierta entrada como positiva cuando en realidad era negativa.

Accuracy

Medida de rendimiento que define la proporción de predicciones correctas realizadas por un modelo en comparación con el número total de predicciones. Su definición viene dada por:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Recall

Métrica que evalúa la capacidad de un clasificador para identificar todas las instancias positivas relevantes. Su definición viene dada por:

$$Recall = \frac{TP}{TP + FN}$$

Precisión

Métrica de rendimiento que evalúa la calidad de las predicciones realizadas por un clasificador. Su definición viene dada por:

$$\text{Precisión} = \frac{TP}{TP + FP}$$

F1 Score

Métrica que combina tanto la precisión como el *recall* en un solo valor, está definida como el promedio harmónico de precisión y *recall*

$$F1\ Score = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

La interpretación de los márgenes en tanto a la evaluación de un modelo es la siguiente: para un valor correspondiente entre 0%-50% se trata de un modelo malo, entre 50%-60% se considera un modelo regular, ya que es un poco mejor que tirar una moneda al aire hablando sobre temas de probabilidad, pero de igual manera sigue siendo ineficiente, entre 60%-80% ya se considera un modelo bueno y con una fiabilidad aceptable, y entre 80%-90% se considera un modelo bastante bueno, en el cual se realizó un entrenamiento bastante correcto y de esa forma es capaz de clasificar sin problema.

Sin embargo, hay que tener mucho cuidado con los modelos que arrojan resultados mayores al 90% , esto puede ser una indicación de que algo inusual está ocurriendo, puede ser debido a un sobre-entrenamiento del modelo, una divergencia super clara entre clases , un posible desbalance en la cantidad de datos que pertenece a cada clase o posiblemente el error más común , que suele darse al proporcionar como entrada para el modelo, datos que estén afectando al cálculo de las métricas de evaluación , ya sea porque tenemos datos inútiles , alta correlación entre ellos o simplemente que no sean necesarios para nuestro caso.

CAPÍTULO 4. DISEÑO DE LA SOLUCIÓN

4.1 REQUISITOS

En esta sección se plantearán los requisitos funcionales y no funcionales que tienen lugar en el TFG desarrollado; entendemos por requisitos funcionales a las acciones que el sistema debe ser capaz de realizar y por otro lado, los requisitos no funcionales describen el comportamiento del sistema frente a acciones que no son las principales (es decir, ciertos comportamientos que pueden alterar la experiencia de uso del sistema).De esta forma a través de la elaboración de plantillas para cada caso, seremos capaces de establecer los requisitos iniciales que el proyecto debe cumplir y las posibles limitaciones que pueden presentar.

Requisitos Funcionales

RF-1	Leer ficheros Pcap
Prioridad	Alta
Descripción	El modelo debe ser capaz de leer los archivos pcap correspondiente al tráfico de red
Criterio de aceptación	Con el uso de la librería Pyshark en Python podemos visualizar la correcta interpretación de los datos provenientes del archivo pcap
Posibles Limitaciones	El software de Wireshark no cuenta con herramientas suficientes para extraer los datos deseados

Tabla 1: RF-1 Leer ficheros pcap

RF-2	Extracción de las características importantes
Prioridad	Alta
Descripción	El sistema debe ser capaz de extraer las características de mayor relevancia para el modelo de IA
Criterio de aceptación	Haciendo uso de librerías específicas de Scikit-Learn podremos comprobar las características más importantes que devuelve el modelo
Posibles Limitaciones	Incongruencia en las características importantes y dificultad en su uso

Tabla 2: RF-2 Extracción de las características importantes

RF-3	Clasificación del flujo de datos
Prioridad	Alta
Descripción	Tras diseñar y entrenar nuestro modelo de IA, el modelo debe ser capaz de llevar a cabo una correcta clasificación del tráfico analizado, y diferenciar entre flujo Normal y MITM
Criterio de aceptación	Tras pasar al modelo por una pruebas y métricas de evaluación, podemos verificar la correcta interpretación del modelo y clasificación de los datos
Posibles Limitaciones	Diseño de software ineficaz Enfoque y estructura de diseño mal planteada Características importantes mal seleccionadas Requerimiento de altas capacidades de software (CPU, Memoria, …)

Tabla 3: RF-3 Clasificación del flujo de datos

Requisitos No Funcionales

RNF-1	Manejar gran cantidad de datos
Prioridad	Media
Descripción	El sistema diseñado debe ser capaz de trabajar con una gran cantidad de datos de entrada, logrando así cumplir con la extracción de características, entrenamiento y validación del sistema sin ningún problema
Criterio de aceptación	Para poder validar dicho requisito, utilizaremos un *dataset* que maneje gran flujo de datos y características, de esta forma comprobaremos el correcto funcionamiento del modelo en tanto a al entrenamiento y validación con grandes cantidades de datos
Posibles Limitaciones	Tiempos de análisis demasiado elevados La propia adquisición de grandes cantidades de datos Requerimiento de altas capacidades de software (CPU, Memoria, …)

Tabla 4: RNF-1 Manejar gran cantidad de datos

RNF-2	Flexibilidad y adaptación para usos futuros
Prioridad	Media
Descripción	Una increíble mejora al sistema diseñado sería su capacidad de adaptación y flexibilidad para casos distintos, pudiendo así tener una base sólida para aplicar en otros ámbitos de estudio e investigación.
Criterio de aceptación	Realizaremos un estudio en base a dos enfoques de planteamiento (paquete a paquete y por conjunto de paquetes), de esta forma veremos la flexibilidad del sistema
Posibles Limitaciones	Cada caso de implementación debe trabajar de forma muy específica según los datos y el enfoque planteado, lo que provocaría replantear el diseño, implementación, entrenamiento y validación para cada nuevo caso de estudio.

Tabla 5: RFN-2 Flexibilidad y adaptación para usos futuros

En base a las plantillas anteriores, basaremos la estructura e implementación de nuestra arquitectura, buscando así satisfacer el cumplimiento de cada uno de los requisitos mencionados.

4.2 ARQUITECTURA GENERAL

La arquitectura del TFG desarrollado consiste en simular un ambiente típico, en el cual se establece una comunicación entre Cliente y Servidor, pero tenemos un tercer factor que es el atacante MITM, el cual, en un primer instante, es invisible para el Cliente, es decir, el Cliente no sabe de su existencia.

Así, en la figura 3, modelamos este escenario, donde representamos el ambiente en donde se generará nuestra solución que estará diseñada/instalada en la parte del Cliente.

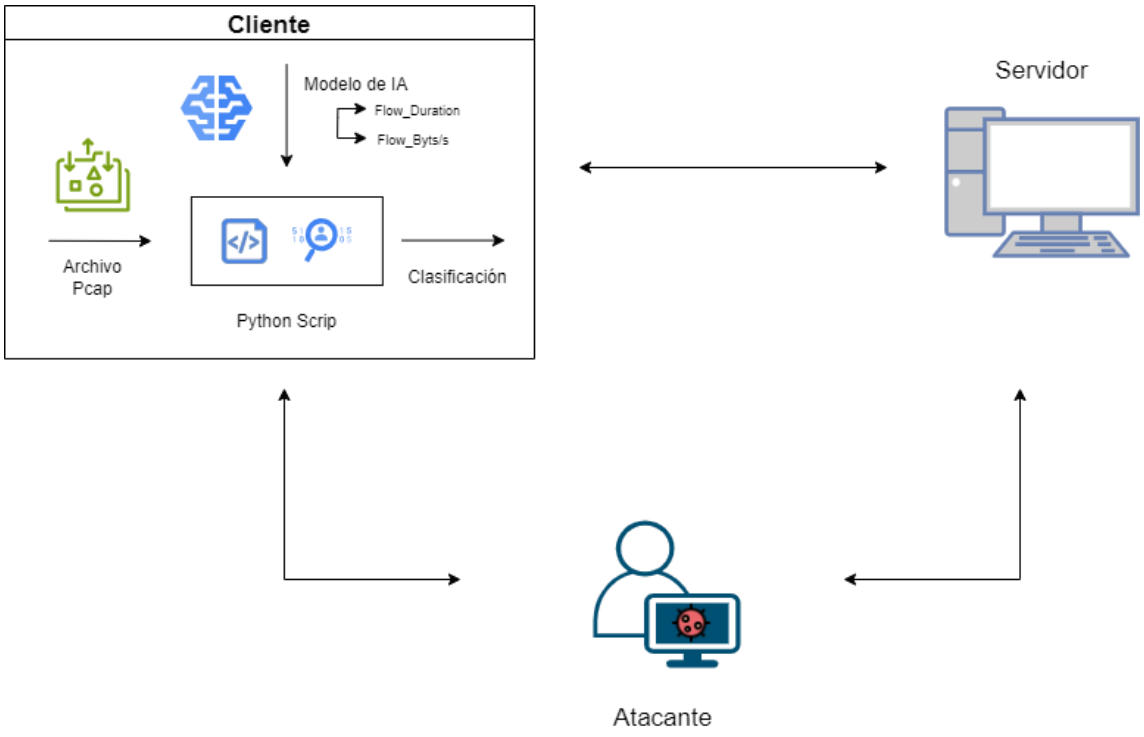


Figura 3. Arquitectura general

4.3 BANCO DE PRUEBAS

Para simular el entorno de trabajo y validar los experimentos planteados y el comportamiento de los modelos de IA desarrollados, utilizaremos el entorno de pruebas desarrollado en el proyecto GUARDIAN, el cual se representa en la figura 4. Este entorno cuenta con una infraestructura de red y mecanismos de seguridad basados en entornos virtuales, donde un servidor aloja tres máquinas virtuales: Cliente, MITM y Servidor. El escenario plantea una comunicación aparentemente normal entre Cliente y Servidor, la cual es interceptada por un atacante MITM. Este modifica la tabla del Protocolo de Resolución de Direcciones (ARP) para suplantar la dirección MAC del Servidor, logrando que todo el tráfico de red pase también por el atacante sin que el Cliente se percate de ello, Figura 5.

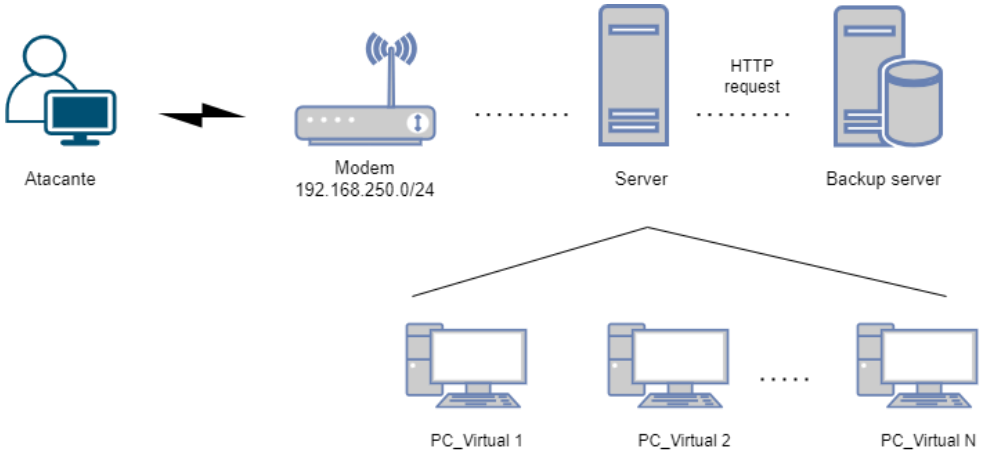


Figura 4. Entorno de pruebas GUARDIAN

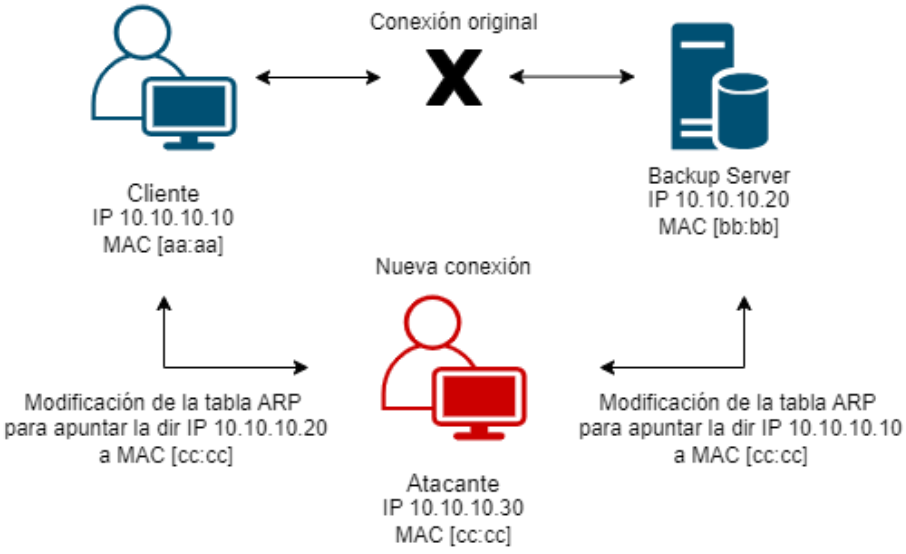


Figura 5. Ejemplo de ARP Spoofing

En este caso de estudio, el MITM intercepta la comunicación y encripta parcialmente los datos enviados del Cliente al Servidor durante las transferencias de imágenes (mediante peticiones POST y GET, realizadas sobre el protocolo de red TCP/HTTP). A pesar de que el Servidor responde con un código de estado 200 OK, la imagen que llega está cifrada, como se muestra en la Figura 6. Esta situación refleja que el Cliente no es consciente de la presencia del MITM, ya que la comunicación no muestra signos aparentes de alteración.

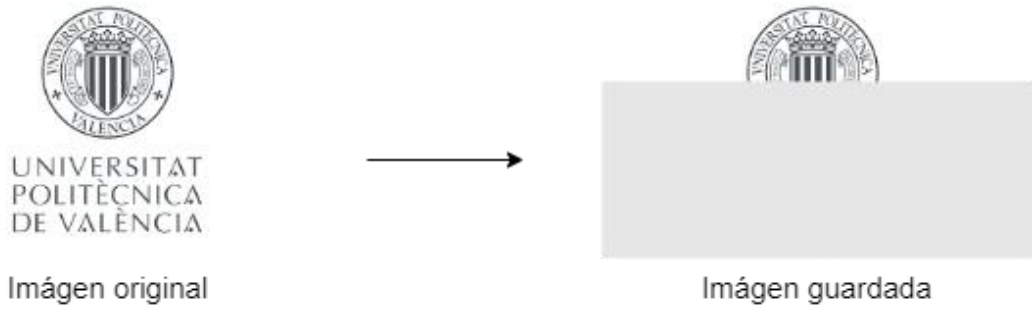


Figura 6. Ejemplo de la encriptación de los datos en movimiento

El objetivo de este experimento es detectar la presencia del MITM. A diferencia de un ataque típico de *Ransomware* que encripta todo el archivo, en este escenario se realiza una encriptación parcial del contenido de los datos. Esta simulación permite evaluar la efectividad de los modelos de IA desarrollados para detectar ataques MITM en tiempo real, asegurando la integridad y seguridad de las comunicaciones en redes informáticas.

4.4 EXPERIMENTOS

En esta sección, presentaremos los experimentos planteados y llevados a cabo para lograr la detección de un posible ataque MITM.

De esta forma, seguiremos nuestra estructura presentada en la fase general del proyecto, donde recordemos que el esquema a seguir para cada experimento se divide en tres secciones: la adquisición de datos, la parte de IA, en donde daremos a conocer el entrenamiento y diseño implementado, y por último la validación del modelo, aportando lo resultados obtenidos en cada experimento.

Así, plantemos 3 experimentos, los cuales se representan en la siguiente tabla (Tabla 6):

Experimentos	Dataset	Descripción
Exp-1	IoTID20	Enfoque de análisis paquete a paquete
Exp-2	DatasetUPV y DatasetUPV2	Enfoque de análisis paquete a paquete
Exp-3	DatasetUPV3	Enfoque de análisis por transacción

Tabla 6: Exposición de los experimentos para la detección del MITM

CAPÍTULO 5. IMPLEMENTACIÓN Y VALIDACIÓN DE LOS EXPERIMENTOS

5.1 EXPERIMENTO 1

En este primer experimento realizaremos el entrenamiento de nuestro modelo de inteligencia artificial sobre nuestro ambiente de pruebas (*testbet*), tomando como datos de entrada el *dataset* IoTID20.

El planteamiento a seguir para este experimento es analizar todo el flujo de datos de una red en tanto a paquetes, es decir, evaluaremos todo el flujo proveniente de una comunicación Cliente-Servidor y analizaremos paquete a paquete si se está produciendo o no un posible ataque MITM.

Antes de trabajar con el enfoque comentado, debemos determinar el algoritmo de clasificación que usaremos para entrenar nuestro modelo, de la misma forma, únicamente en esta ocasión, me tomaré la libertad de adelantarme un poco a la parte del preprocesado de los datos (que sería nuestro siguiente punto 5.1.1), en donde aprovecharemos los experimentos a desarrollar en tanto a la elección del algoritmo de clasificación, para determinar cuáles son las características más importantes que presenta nuestro *dataset* IoTID20.

Elección del algoritmo de clasificación y características más importantes

En primer lugar, debemos de tener en cuenta que, para entrenar un modelo de Inteligencia artificial, se debe establecer y determinar el algoritmo de clasificación o regresión que usaremos, lo cual es un punto fundamental para alcanzar los objetivos deseados, por ejemplo, si tratamos con problemas de regresión, usamos algoritmos como regresión lineal o arboles de regresión, y para problemas de clasificación, podemos usar algoritmos como *DecisionTree,o RandomForest*. En nuestro caso, lo que queremos es detectar un posible ataque de MITM, lo que supone un problema de clasificación. Por tanto , como primer paso de análisis , tomamos como base, los estudios realizados por un grupo de investigadores, en donde el paper redactado (Ullah, 2020) explica detalladamente el uso de los diferentes algoritmos que puede tomar un modelo de IA para la clasificación de distintos tipos de ataques informáticos que puede sufrir una maquina (como por ejemplo: DoS, Mirai, Scan, MITM, etc) .La conclusión obtenida por la investigación de este grupo era que para detectar un posible ataque MITM, los algoritmo que presentaba mejores prestaciones y comportamiento eran el *RandomForest, DecisionTree y Esemble.*

De igual manera, nos vimos con la necesidad de contrastar y verificar los resultados obtenidos en dicha investigación; de este modo, pasamos a implementar los tres algoritmos que, según el estudio realizado, presentaban mejores resultados, además, consideramos que sería destacable, probar con algún otro algoritmo, por ejemplo, LDA, para dejar en evidencia que, sin lugar a duda, los algoritmos que mejores se adaptaban era los mencionados anteriormente.

Al mismo tiempo, a la hora de diseñar y entrenar un modelo, debemos ser muy cuidadosos y selectivos a la hora de proporcionarle los datos y características con las que queremos trabajar, debido a que un exceso de datos podría estar afectando de forma no deseada a nuestro modelo,

generando así una posible alteración en las métricas de validación, como por ejemplo la accuracy o la precisión; Siguiendo esta idea , es fundamental realizar un estudio a profundidad de los propios datos (detectando su valía para el modelo y la información que aportan) para que de esta forma, podamos seleccionar las características verdaderamente importantes y depurando la información que no sea relevante.

De esta manera, hemos optado por plantear 3 experimentos, los cuales se encuentran representados en la Tabla 7, con el objetivo de abordar los puntos mencionados anteriormente (determinar el algoritmo de clasificación y los datos de entrada para el modelo).

Antes de pasar a los experimentos y teniendo conocimiento previo sobre modelos de IA y las características que presenta el *dataset* IoTID20, podemos predecir como se comportará el modelo si lo evaluamos directamente con todas las características. Sabemos que existen 2 características que nos realizaran de forma directa la clasificación, que son 'Cat' y 'Sub_Cat' ya que expresamente nos dicen si es o no MITM, por lo que hay que eliminarlas. Luego, también sabemos que 84 características son muchas, y que además, el sentido de algunas de ellas son irrelevantes para nuestro modelo, por lo que seguramente los valores de mis validaciones serán ineficientes e irreales.

De todas formas, este apartado va enfocado para explicar detalladamente todo el proceso desarrollado en tanto a la parte de IA, de manera que se pueda comprender y seguir fácilmente, así, siguiendo un orden, tomaremos este caso como el primer experimento que realizaremos.

Experimentos	**Algoritmos**	**Selección de características**
Exp-1	Decision Tree, Random Forest, Esemble y LDA.	Todas las caracteristicas (86)
Exp-2	Decision Tree, Random Forest, Esemble	Utilizamos la herramienta Recursive Feature Elimination (Nos quedamos con 10 características de 86)
Exp-3	Random Forest	Características finales (Nos quedamos con 6 características de 86)

Tabla 7: Exposición de los experimentos para determinar el algoritmo de clasificación y características importantes

Ahora, vamos a adentrarnos a explicar el enfoque que seguimos para cada experimento, aportando a la vez los resultados obtenidos y las conclusiones a la que llegamos tras lo estudios realizados.

Exp 1

El objetivo de este primer experimento es comprobar el comportamiento de los 4 algoritmos de clasificación seleccionados (*RandomForest, DecisionTree, Esemble y LDA*), utilizando como datos de entrada todas las características del *dataset* IoTID20 y analizando la respuesta de cada uno de ellos junto con las métricas de evaluación. metodología

Tras evaluar los diferentes algoritmos, obtuvimos un valor del 100% para todas las métricas que estamos aplicando para validar las prestaciones del modelo, lo cual rectifica nuestra idea/hipótesis inicial que habíamos planteado en tanto a los posibles resultados que obtendríamos para este primer experimento. Ahora bien, como explicamos anteriormente en tanto a la interpretación de los valores arrojados por los modelos, sabemos que esos valores del 100% son un inconveniente.

Teniendo conocimiento así del posible causante del error (datos innecesarios para el modelo), pasaremos a estudiar las características del *dataset*. En el *paper* donde encontramos este *dataset*, nos comentan que, para detectar este tipo de ataques, existen 12 Características que están muy relacionadas entre ellas, es decir, que presentan alta correlación, lo que perjudica al modelo. Por consiguiente, pasaremos a eliminar dichas características y nos quedamos con un total de 72.

Exp 2

En este segundo experimento seguiremos con el estudio de los algoritmos de clasificación y la depuración de las características.

En el experimento 1, pudimos comprobar que el utilizar las 86 características del *dataset* nos resulta en un problema, por lo que en este experimento cabría detectar cuales son las más importantes y las que presentan un mayor peso. De esta forma, haremos uso de la herramienta RFE (en inglés, *recursive feature elimination*) de la librería *sklearn.feature_selection*, la cual nos ayudará a seleccionar dichas características importantes a través de una eliminación recursiva, donde seguidamente con *features_importances* podremos obtener el valor exacto de contribución de dicha característica para el modelo.

Al utilizar RFE especificamos que nos devolviera las 10 características más importantes, en donde su elección fue la siguiente: *'Src_IP', 'Src_Port', 'Dst_IP', 'Dst_Port', 'Flow_Duration', 'Flow_Byts/s', 'Flow_Pkts/s', 'Flow_IAT_Mean', 'Bwd_Header_Len', 'Init_Bwd_Win_Byts'*

Así, evaluaremos nuevamente nuestros modelos de clasificación para seleccionar el que mejor se adapte a nuestro caso, los resultados obtenidos se reflejan en la Tabla 8.

Algoritmos	Accuracy	Recall	Precision	F1 Score
RandomForest	0.9964	0.9964	0.9964	0.9964
DecisionTree	0.9972	0.9972	0.9972	0.9972
Esemble	0.9972	0.9972	0.9972	0.9972
LDA	0.9200	0.9200	0.9200	0.9200

Tabla 8: Evaluación de los algoritmos según 10 características

Como podemos ver, nuestras métricas de evaluación nos siguen arrojando valores muy cercanos al 100%, aunque podemos ver que el algoritmo LDA está más cerca del 90% que del 100%.

Por consiguiente, podemos ver que, *RandomForest, DecisionTree y Esemble*, presentan prácticamente el mismo comportamiento para alcanzar nuestro objetivo, y de la misma forma, el algoritmo LDA se aleja un poco (por lo que queda descartado).

De esta forma, como los tres algoritmos presentan resultados prácticamente iguales, hemos optado por la elección del algoritmo *RandomForest*, Figura 7, que a diferencia del *DecisionTree* que utiliza un solo árbol de decisión, *RandomForest* crea un "bosque" de árboles (es decir, múltiples arboles de decisión), cada uno construido en un subconjunto aleatorio de las características disponibles, consiguiendo así mejorar la precisión y reducir el sobreajuste; en tanto a *Esemble* , este algoritmo se basa en *RandomForest y DecisionTree* , por lo que en base a nuestro criterio , preferimos usar un algoritmo simple y no compuesto para tratar en este caso, el problema planteado.

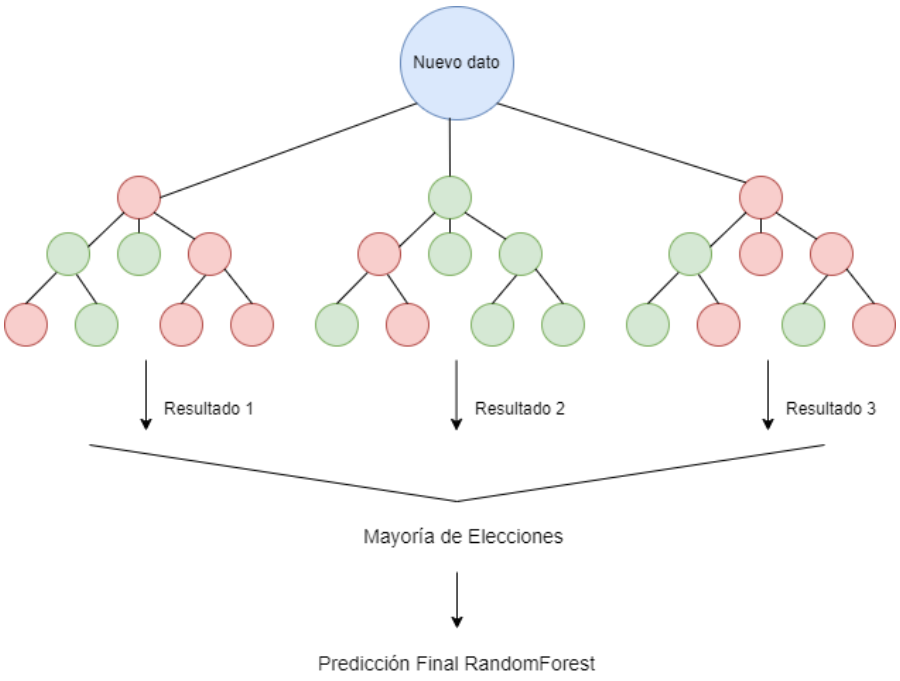


Figura 7. Estructura interna del algoritmo RandomForest

Exp 3

Al comprobar el peso, es decir, la importancia de las características obtenidas en el experimento 2, obteníamos que a partir de la quinta característica la importancia era sobre un valor del 0,002, es decir, una contribución prácticamente nula.

Lo que nos lleva a desarrollar este tercer experimento, en donde decidimos reducir aún más las características y quedarnos solo con 5, por lo que ajustamos el RFE, y nos devolvía las siguientes características: *'Src_Port', 'Dst_IP', 'Dst_Port', 'Flow_Duration', 'Flow_Pkts/s'*.

Tras analizar un poco las características en general y poniendo en práctica los conocimientos adquiridos durante la carrera en tanto a Telemática y ciberseguridad, nos dimos cuenta de que había características muy interesantes en tanto al comportamiento de un posible ataque MITM que cabrían de ser analizadas y que no se estaban tomando en cuenta, como lo son: *'Flow_IAT_Mean', 'Down/Up_Ratio' y 'Flow_Byts/s'*.

Por ejemplo, *Flow_IAT_Mean* nos indica el tiempo promedio entre los paquetes enviados en un flujo completo; en una comunicación normal, dicho tiempo es regular, lo que refleja la naturaleza de una comunicación de datos, sin embargo, un atacante MITM puede manipular estos intervalos de tiempos para maniobrar y tratar de pasar desapercibido; por lo que, si este intervalo de tiempo es muy bajo o muy alto, puede ser un gran indicar que existe un posible MITM en la comunicación. De la misma manera, ocurre con *Flow_Byts/s*, la cual corresponde con la tasa de bytes por segundos, lo que corresponde un valor regular para una comunicación normal, pero si dicha tasa presenta valores muy bajos o altos, podría estar ocurriendo un ataque de MITM en la comunicación.

Finalmente, decidimos elegir estas 5 características para entrenar el modelo: *'Flow_Duration', 'Flow_IAT_Mean', 'Down/Up_Ratio', 'Flow_Pkts/s' y 'Flow_Byts/s'*. En donde al incluir estas características que en un principio parecían no ser muy importantes, el modelo presenta la misma capacidad de clasificación que con 10 características. De esta forma, podemos apreciar en las Tablas 9 y 10, el resultado obtenido tras la evaluación del modelo y las características más relevantes seleccionadas del *Dataset* IoTID20, respectivamente.

Algoritmo	Accuracy	Recall	Precision	F1 Score
RandomForest	0.9575	0.9575	0.9575	0.9575

Tabla 9 Evaluación del modelo con el algoritmo y características finales

Caracteristicas	Tipo	Descripción
Flow_Duration	Integer	Duración total de un flujo de comunicación
Flow_Byts/s	Integer	Tasa de bytes por segundos
Flow_IAT_Mean	Integer	Media del intervalo de tiempos entre paquetes para el flujo completo
Down/Up_Ratio	Integer	Relación entre los bytes enviados desde el origen al destino y los bytes enviados desde el destino al origen.
Flow_Pkts/s	Integer	Tasa de paquetes por segundo

Tabla 10 Características más relevantes del Dataset IoTID20

Finalmente, tras este tercer experimento, podemos asegurar que el algoritmo de clasificación seleccionado será el *RandomForestClassifier* y que contamos con las características correctas para realizar el diseño e implementación del modelo de IA.

De esta forma, ahora si pasamos a evaluar nuestro modelo de IA siguiendo el enfoque planteado según un análisis de paquete a paquete.

5.1.1 Adquisición de datos y preprocesado

Nuestra base para el entrenamiento será el *dataset* IoTID20, que como bien comentamos en apartados anteriores, de 86 características nos quedamos con las 5 más importantes (*Flow_Duration", "Flow_IAT_Mean", "Flow_Byts/s, 'Down/Up_Ratio' y Flow_Pkts/s*), en donde además, ajustamos la naturaleza de los datos para tener datos de tipo *Integer* y que sean fácilmente interpretados por el modelo.

El siguiente paso es extraer de las capturas Wireshark las características que habíamos marcado como importantes; para ello, como bien comentamos en las especificaciones generales del *paper*, creamos unos *scrip* en Python que a partir de los datos de Wireshark podría extraer dichas características.

Así, obtenemos las características más importantes de la siguiente manera:

Flow_Duration lo obtenemos como el tiempo que dura un flujo de datos, es decir, el tiempo que dura un paquete o un flujo de comunicación; para extraer dicha característica, basta con restar el tiempo de inicio y fin de un paquete o el flujo completo de la comunicación.

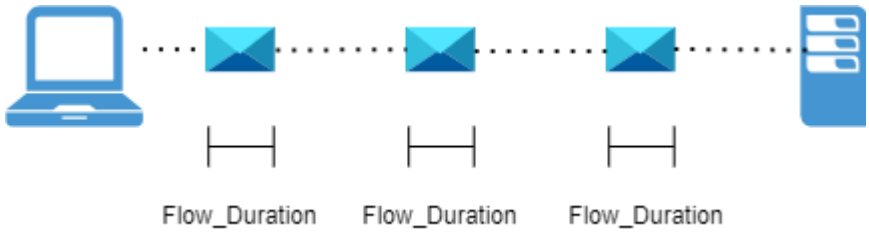


Figura 8. Ejemplo visual de la característica Flow_Duration

Flow_Bytes/s lo obtenemos sacando el Flow_Duration de dicho paquete, y luego dividimos la longitud del paquete (en bytes) por su Flow_Duration.

Flow_IAT_Mean lo obtenemos como el tiempo intermedio entre la finalización de un paquete y el comienzo del siguiente paquete.

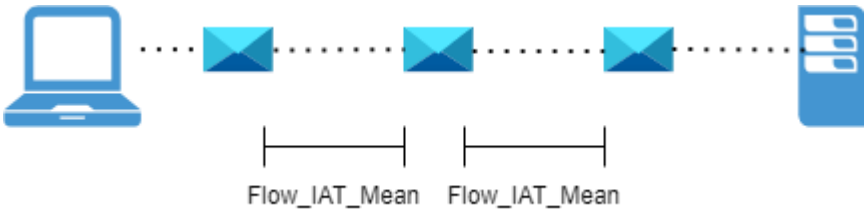


Figura 9. Ejemplo visual de la característica Flow_IAT_Mean

Flow_Pkts/s lo obtenemos contando el número de paquetes en un flujo específico y luego lo dividimos por su *Flow_Duration.*

Pero, a la hora de extraer estas características, nos dimos cuenta de que la mayoría de estas solo tienen un enfoque para ser evaluadas en tanto al flujo completo de la comunicación entre Cliente-Servidor, por lo que dichas características solo tienen valor/peso para este contexto.

Como el enfoque de este experimento es evaluar el flujo en tanto de paquete a paquete, el uso de todas las características es inútil. Sin embargo, de las cinco características, dos de ellas sí que podrían llegar a tener sentido en tanto a paquetes habláramos, que son: *Flow_Duration y Flow_Bytes/s*.

De esta forma, para el experimento 1, solo trabajaremos con estas 2 características.

5.1.2 IA

Así tenemos que nuestro modelo se basara en *Flow_Duration y Flow_Bytes/s*, en donde además ya habíamos acordado usar el algoritmo de clasificación de *RandomForestClassifier* implementando la librería de Scikit-lern

Para entrenar nuestro modelo, usaremos la técnica de Validación Cruzada(o en inglés, *Cross-Validation*), esta técnica en vez de realizar un Split de datos (una división de los datos), genera múltiples divisiones pero en tiempos diferidos, de esta forma, lograremos trabajar con todo el conjunto de datos, tanto para la validación como para el entrenamiento, y también nos proporciona cierta ventaja de cara a la validación, en donde nos permite obtener resultados más fiables debido a que los valores obtenidos de las métricas serán promediados según el número de divisiones realizadas.

Para implementar el *Cross-Validation*, debemos utilizar la *herramienta KFold, kfold = KFold (n_splits=k, shuffle=True, random_state=42)*, en el cual tenemos que especificar el número de divisiones que queremos realizar al conjunto de datos (k=4), si queremos que los datos tomados para dichas divisiones sean aleatorios (shuffle=True), y por último debemos indicar la semilla del estado inicial (*random_state*=42).

La división de los datos se realiza en torno a un 80% para el entrenamiento y un 20% para el test, en donde se le asigna una mayor cantidad de datos a la parte de entrenamiento para que el modelo se familiarice con el entorno y tenga cubierto una mayor cantidad de datos.

Otro paso importante es estandarizar/normalizar los datos, para que todos los datos de nuestro modelo presenten la misma escala y no haya errores a la hora de trabajar con ellos; Por lo tanto, debemos escoger el escalado o normalización que queremos usar, en nuestro caso, utilizamos el *StandarScale* el cual ajusta los valores de las características para que tengan una media de cero y una deviación estándar de uno.

Una vez estandarizados los datos, pasamos a crear nuestro modelo con el algoritmo *RandomForest, model = RandomForestClassifier (n_estimators=4, random_state=42*), en su implementación, debemos indicar el número de árboles de decisión queremos que tome el modelo *(n_estimators=4)*, debemos saber que mientras más redes y arboles hayan, nuestro modelo será más complejo, pero para la situación que estamos evaluando y los objetivos que queremos alcanzar, nos es suficiente con poner solo 4 divisiones de árboles de decisión, y en tano a la semilla de partida *(random_state=42)*, pondremos 42 como venimos haciendo normalmente.

Así, ya seremos capaces de entrenar el modelo con los datos de entrenamiento seleccionados, *model.fit(X_train, y_train)*, y predecir el conjunto de prueba para luego evaluar el modelo, *y_pred = model.predict(X_test)*, en donde obtuvimos un *accuracy* del 91% durante el entrenamiento, lo que validaremos más adelante.

Por último, guardamos el modelo y su *scaler* (importante, ya que, si luego abrimos el modelo, debemos establecer el mismo *scaler* que se utilizó para su entrenamiento, ya que cada implementación del *scaler* ajusta los datos de forma distinta, independientemente si se utilizó la misma semilla) y así, podremos trasladar el modelo en un futuro si así lo precisamos necesario.

5.1.3 Validación

Para evaluar el modelo de IA, modelamos 2 ambientes dentro de nuestro *testbet*, uno en donde se simula una comunicación Normal entre Cliente y Servidor, y en el otro se simula una comunicación entre Cliente y Servidor, pero con la existencia de un MITM, el cual intercepta todos los mensajes intercambiados en la comunicación. Para ambos casos, estamos capturando a través de Wireshark todo el flujo de la comunicación, en donde el Cliente realiza un envió de 50 Imágenes hacia el Servidor. El objetivo de estas capturas es verificar el comportamiento de nuestro modelo.

De esta forma, pasamos a comprobar, por ejemplo, el comportamiento del modelo para el caso de flujo Normal, en donde el resultado obtenido fue de un *accuracy* del 100%, lo que clasifica a la perfección cuando se trata de un paquete perteneciente a un flujo Normal. Pero, cuando pasamos a comprobar el caso de MITM, obteníamos un *accuracy* del 0%, es decir, todo flujo lo clasificaba como Normal, lo que es muy distinto a como el entrenamiento del modelo nos decía que se comportaría el modelo, aportando un *accuracy* del 91%.

Este resultado, nos lleva a pensar que el enfoque paquete a paquete no es una buena idea para la detección del MITM. Pero analizando los valores del *dataset* y los que obteníamos en nuestro entorno del *testbet*, los valores eran totalmente distintos; como se puede apreciar en las gráficas, representadas por las figuras 10 y 11, los valores en el *dataset* IoTID20 y los valores obtenidos en nuestro entorno son totalmente distintos, de esta forma, al haber una diferencia tan grande entre ambos datos, el modelo de IA no era capaz de identificar con claridad y clasificaba todo como flujo Normal.

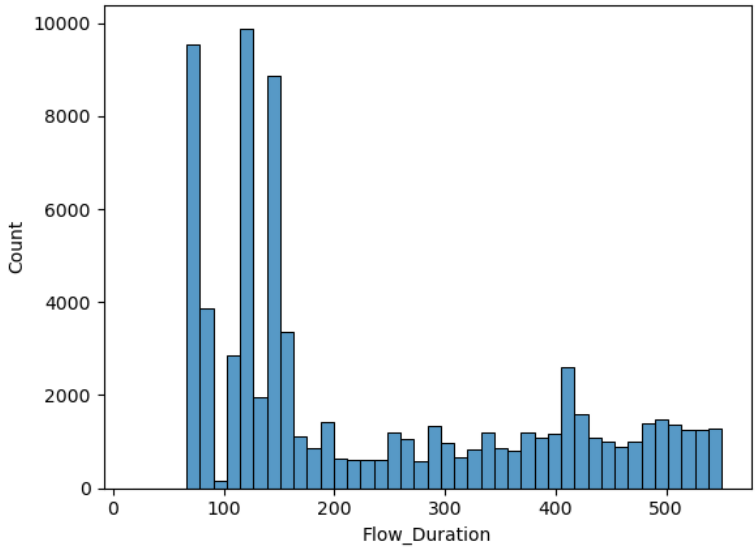


Figura 10. Distribución del Flow_Duration en el dataset IoTID20

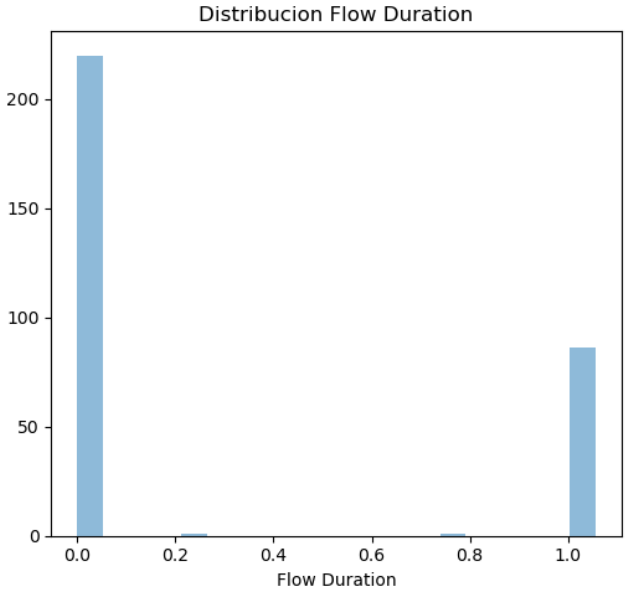


Figura 11. Distribución del Flow_Duration en nuestro entorno

Todo esto nos lleva a pensar que el *dataset* IoTID20 esta implementado para un caso en especial, el cual fue entrenado con una red y condiciones muy específicas, de esta manera, si validamos el comportamiento con un entorno distinto al que fue entrenado, el modelo no sabe clasificar y devuelve muy malos resultados.

De hecho, podemos llegar a pensar tan solo viendo los valores de *Flow_Duration* de que el *dataset* fue forzado para obtener ciertos resultados y demostrar simplemente un posible tipo de detección frente a estos tipos de ataques.

De esta manera, considero que la mejor implementación que puedo desarrollar para poder detectar un posible ataque MITM es crear un *dataset* propio y a partir de él, poder comprobar según nuestro entorno, que es controlado y del cual tenemos conocimiento, los valores que nos arroja el modelo tras la validación, lo que nos lleva a la implementación del segundo experimento.

5.2 EXPERIMENTO 2

Este segundo experimento surge de la idea de implementar nuestro propio *dataset* para poder diseñas, validar y entrenar nuestro modelo de IA; en este apartado se explicará el proceso de creación del nuevo *dataset* (DatasetUPV) y los resultados que obtuvimos. Todo esto, siguiendo el mismo enfoque de trabajo, que consiste en realizar un estudio y evaluación de paquete a paquete, ya que en el experimento 1, no pudimos sacar conclusiones claras sobre este enfoque y determinar si era el indicado o no para lograr nuestro objetivo.

5.2.1 Adquisición de datos y preprocesado

Anteriormente, habíamos utilizado como base de datos los valores provenientes del *dataset* IoTID20, pero dicho *dataset* ya queda descartado. De esta forma, pasaremos a la implementación de nuestro propio *dataset*, el cual incluirá valores provenientes de nuestra propia red y entorno.

Para crear el *dataset*, deberíamos contar con flujo anómalo y flujo normal, es decir, tener datos provenientes de una comunicación infectada/atacada por un MITM y datos de una comunicación normal sin ningún tipo de alteración. Para logara esto, debemos generar dichos ambientes/simulaciones, las cuales se llevarán a cabo en el *testbet*, donde el Cliente enviará, por un lado, 50 Imágenes al servidor de forma alterada por un MITM, y, por otro lado, enviará las mismas imágenes, pero de forma Normal, en donde nosotros estaremos capturando todo el tráfico para ambos casos a treves de Wireshark.

De esta forma, tenemos 2 archivos pcap, los cuales debemos leer y extraer las características con las que estamos trabajando que son *Flow_Duration y Flow_Byts/s*; para esto, ejecutaremos nuestro script de Python, el cual, a partir de un pcap, me extrae las características seleccionadas y las guardamos en un archivo CSV; estos pasos, los ejecutamos para ambas capturas, es decir, debemos meter el pcap de flujo Normal y el de MITM para poder generar en cada caso su archivo CSV.

Una vez tengamos ambos archivos CSV, los convertimos e *dataframes* y los juntamos para así crear nuestro DatasetUPV, a la hora de juntar ambos CSV es importante mezclar los datos del *dataset*, ya que de lo contrario me quedare con un *dataset* “dividido ”, el cual tenga en la parte superior todos los valore correspondientes al flujo Normal, y en la

parte inferior todos los valores que corresponden con los MITM, lo que resultaría en un gran problema a la hora de realizar los Split de divisiones de datos que efectúa el KFold en la parte del entrenamiento. Otro paso previo que tuvimos que realizar para ajustar el *dataset*, es la forma en que se estructuraba nuestro dataset, al juntar ambos archivos CSV, el *dataset* se estructuró con una distribución de 3 filas y X número de columnas (correspondientes a la cantidad de datos que tenía). Este diseño/estructura no es la que queríamos para trabajar los datos, por lo que lo reorganizamos y conseguimos un *datset* estructurado con 3 columnas (correspondientes a las dos características a evaluar y el *Label*) y X filas (correspondientes a la cantidad de datos), ya por último, tuvimos que renombrar las columnas con los nombres que correspondían (debido al cambio de estructura que realizamos) y finalmente, ya contábamos con nuestro DatsetUPV

5.2.2 IA

En tanto a la parte de IA y la forma en que desarrollamos el entrenamiento del modelo es igual a la realizada en el experimento 1, donde trabajamos con solo dos características (*Flow_Duration y Flow_Byts/s*), creamos el modelo con la herramienta de *Cross-Validation* para poder jugar con todos los datos correspondientes al DatasetUPV, teniendo en cada Split un 80% de los datos para el entrenamiento y un 20% para test, luego entrenamos el modelo y guardamos tanto el modelo como el *Scaler*.

5.2.3 Validación

Ahora, pasaremos a evaluar el modelo; tras el entrenamiento, obtuvimos un *accuracy* del 94%, lo que resulta ser en un principio bastante bueno, pero, de todas formas, la prueba definitiva para validar este *accuracy* era probar con unos datos que el modelo nunca haya visto. Así, realizamos otras 2 capturas diferentes a las que ya teníamos, en donde el Cliente realizo en esta ocasión una transferencia de 1994 imágenes de animales hacia el Servidor, el envío se realizó tanto para flujo Normal como para MITM.

Estas nuevas capturas serán las utilizadas para validar nuestro modelo; comenzamos por verificar el comportamiento presentado en tanto al Flujo MITM, en donde se obtuvo un *accuracy* del 61,4%, lo que llama la atención que ocurre una bajada en tanto al 30% del *accuracy* obtenido en el entrenamiento.

Esta caída del *accuracy* es un tanto preocupante, aunque de todas formas para ser una primera prueba y comparándolo con los resultados en el experimento 1, no está del todo mal. Ahora, lo que cabría es averiguar/investigar porque bajamos tanto este valor de *accuracy*.

Entre las hipótesis planteadas, pensamos que se debe a que el datasetUPV que generamos es muy pequeño (tiene tan solo 353 filas de datos, a comparación del *dataset* IoTID20 que tiene cerca de 70mil filas de datos)por lo que realmente estamos viendo/analizando muy pocos datos , e incluso estos datos se parecen mucho entre ellos, lo que nos resulta en un bajo conocimiento del entorno e interpretación del mismo, por lo que a la hora de meter otro tipo de datos, el modelo no sabe cómo tratarlos, y como podemos ver, el 40% de ellos, los clasifica de manera errónea.

Por tanto, dentro de este mismo experimento, pasaremos a generar un Dataset mucho más grande, para poder tratar una mayor cantidad de datos y ver si este puede ser el motivo de la caída del *accuracy*.

DatasetUPV2

5.2.1.2 Adquisición de datos

Para generar este nuevo *dataset*, DatasetUPV2, tendremos que generar un mayor tráfico de tramas entre Cliente y Servidor, anteriormente, ya habíamos generado unas capturas pcap en donde se transferían 1994 imágenes, pero estas capturas fueron utilizadas únicamente para validar el DatasetUPV; Ahora en esta ocasión, utilizaremos dichas capturas para crear nuestro DatasetUPV2. Los pcaps utilizados cuentan con aproximadamente 23mil tramas para MITM y 9900 para Normal, como este nuevo *dataset* es un pequeño experimento dentro del experimento 2, solo tomaremos 3000 tramas para MITM y 3000 para flujo Normal, manteniendo así un equilibrio de muestras por clase y pasamos de un *dataset* de 353 tramas a otro de 6000 tramas, lo que nos sería suficiente para comprobar si la hipótesis planteada del tamaño del *dataset* es el causante del problema.

Así, pasamos a crear el *datset*, haciendo los mismos pasos que hicimos para crear el primer *dataset* (DatasetUPV). De esta forma, generamos los dos archivos CSV, los juntamos, mezclamos las filas, reorganizamos el *dataset* y renombramos las columnas. En donde finalmente, ya tenemos listo nuestro nuevo DatasetUPV2, el cual cuenta con 6000 filas de datos para trabajar.

5.2.2.2 IA

El entrenamiento del modelo de IA tampoco cambia, ya que el enfoque y las características a trabajar son las misma, por tanto, es realizar nuevamente los pasos que seguimos para el primer entrenamiento, solo que esta vez, tomando como fuente de datos el DatasetUPV2. De esta forma, creamos el modelo con la herramienta *de Cross-Validation* para poder jugar con todos los datos correspondientes al DatasetUPV2, teniendo en cada Split un 80% de los datos para el entrenamiento y un 20% para test, luego entrenamos el modelo y guardamos tanto el modelo como el Scaler.

Con este *dataset*, obtuvimos en el entrenamiento un *accuracy* del 81%, lo cual nos deja muy buenas sensaciones.

5.2.3.2 Validación

Para comprobar que el 81% obtenido en la parte del entrenamiento no es algo ficticio, debemos generar nuevamente una serie de datos que el modelo no haya visto para validar así su comportamiento; como venimos haciendo hasta ahora. Es esta ocasión, se realizó la misma transferencia de 1994 imágenes de Cliente a Servidor, pero como las capturas se realizaron en momentos distintos, los datos que obtendremos no serán los mismos, es decir, tendremos *Flow_Duration* distintos y *Flow_byts/s* distintos, debido a que dichos valoren tienen mucho que ver con el flujo de la red.

Así, pasaremos a evaluar cómo se comporta el modelo con los datos del flujo MITM; para este escenario obtuvimos que nuestro modelo aportaba una *accuracy* del 83,6%, en donde podemos observar que incluso mejoró un 2,6% respecto al *accuracy* mostrado en su fase de entrenamiento, lo que nos lleva a confirmar que posiblemente hayamos encontrado un modelo que caracteriza bastante bien si un paquete fue modificado por un MITM o no, lo cual era el objetivo en que se enfocaba el proyecto.

De todas formas, nos pusimos manos a la obra para mirar si existía la posibilidad de mejorar aún más nuestro modelo, por lo que decidimos graficar los datos para ver el comportamiento que estaban tomando en el modelo e intentar sacar algún tipo de información que nos fuera útil. Por lo tanto, pasamos a representar la distribución de cada característica para evaluar el comportamiento que presentaba cada una de ellas para los casos de MITM y los flujos Normales.

Tras representar dichas gráficas, nos dimos cuenta de que la característica de *Flow_Duration* era prácticamente igual para los casos de MITM y Normal. Como podemos ver en la Figura 12, la fina línea amarilla de la gráfica, nos indica donde caen la mayoría de los valores para ambos casos, así, podemos ver que casi todos los valores de *Flow_duration* para los casos del MITM son prácticamente 0, y para el caso Normal, también están muy cercanos al 0, teniendo la mayoría de sus valores en torno al 0.001, lo que resultaría que *Flow_Duration* en vez de ayudar podría estar confundiendo al modelo.

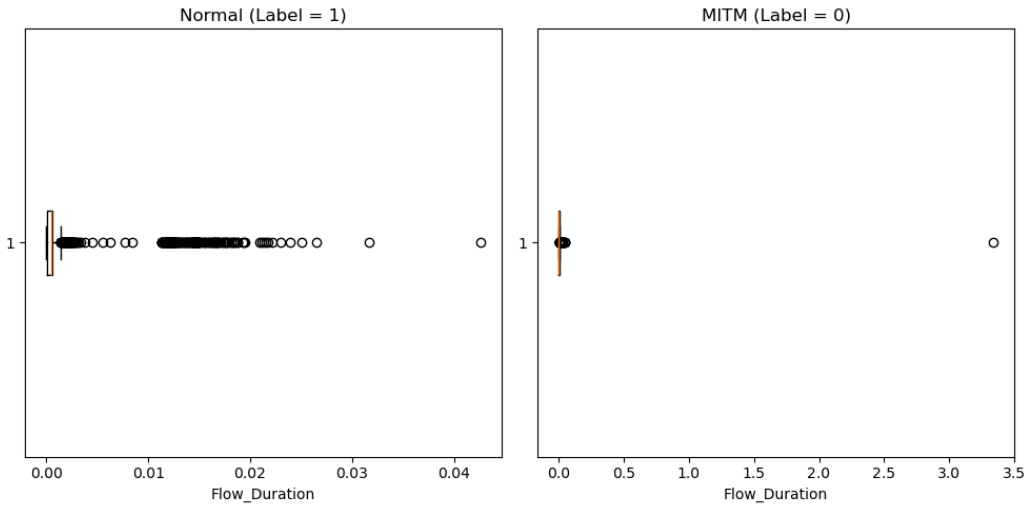


Figura 12. Distribución Flow:Duration para flujo MITM y Normal

Ahora, vamos a representar la gráfica del *Flow_Byts/s* y ver que está ocurriendo con esta característica, Figura 13.

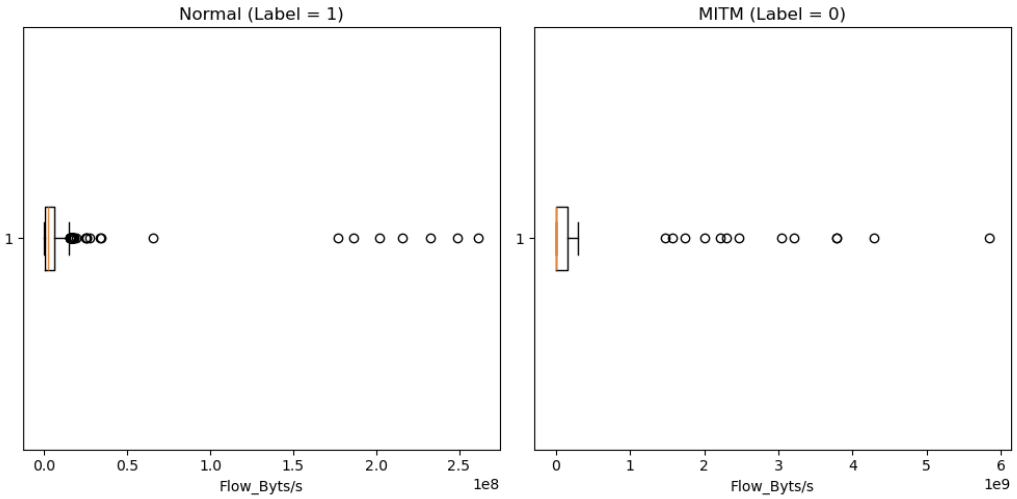


Figura 13. Distribución Flow_Byts/s para flujo MITM y Normal

La gráfica del *Flow_Byts/s* (Figura 10) nos muestra valores muy parecidos para ambos casos, en donde para el caso de MITM la mayoría de sus valores vuelven a ser 0, y para el caso Normal, también son muy cercanos a cero, pero con una leve diferencia respecto al caso MITM de 0.001.

Como podemos ver, el comportamiento de ambas características para los casos de MITM y Normal son prácticamente iguales; aunque los valores del Flujo Normal sí que presentan una leve diferencia tomando valores un tanto alejados del 0 (en torno a 0.001), lo que al parecer es suficiente para que el modelo pueda clasificar entre los casos del MITM y Normal con un gran acierto y precisión como así lo demuestra su *accuracy* del 83%.

Hilando las conclusiones obtenidas anteriormente, di con una idea que podría mejorar bastante el enfoque.

Hasta ahora, hemos estado trabajando y evaluando el modelo paquete a paquete, pero ¿y si trabajamos en tanto a conjunto de paquetes ?, es decir, si tomamos como punto de avaluación todo el conjunto de paquetes que pertenecen a una transacción.

Mirándolo desde un punto teórico, las 2 características con las que venimos trabajando, podrán tomar un mayor sentido, valor y peso a la hora de su evaluación, por ejemplo para *Flow_Duration*, no estaríamos trabajando con valores que presentan diferencias en tanto a milisegundos, sino que podríamos estar tratando datos correspondientes a valores de segundos o décimas de segundos, además, con este enfoque, podría utilizar en un futuro todas las características importantes que destacamos al plantear los experimentos, ya que la mayoría de ellas no las podíamos usar en el enfoque de paquete a paquete

De esta forma, nos surge otro experimento de estudio, el cual analizaremos y trabajaremos en el experimento 3.

5.3 EXPERIMENTO 3

Este experimento parte con un cambio de enfoque respecto a los experimentos anteriores, aquí, el enfoque a seguir será el análisis y evaluación en tanto a conjunto de paquetes pertenecientes a la misma transacción, es decir, tomaremos todos los paquetes que pertenecen a una misma transferencia entre Cliente y Servidor, en este caso la transferencia en cuestión se trata de una imagen.

5.3.1 Adquisición de datos y preprocesado

Para este experimento, crearemos otro *dataset*, el cual llamaremos, DatasetUPV3. Dicho *dataset*, estará formado por valores provenientes de una comunicación entre Cliente y Servidor, en el cual el Cliente pasará 1994 imágenes al Servidor tanto de forma Normal como cuando existe un MITM, si recordamos, el segundo *dataset* del experimento 2 también se basaba en esta transferencia de imágenes, pero para ese caso acortamos los datos en 3000 paquetes de MITM y 3000 paquetes de flujo Normal; para este tercer experimento, tomaremos todos los paquetes de la comunicación , es decir, aproximadamente los 23000 paquetes de MITM y los 9900 paquetes del flujo Normal.

Como tenemos un enfoque totalmente distinto, también tendremos que cambiar la extracción de las características que realizaban mis ficheros de Python, ahora deberían ser capaces de analizar las características en tanto a un conjunto de paquetes y no a un solo paquete.

Pero ¿Como podemos detectar los paquetes pertenecientes a una transacción?

Teniendo en cuenta nuestros conocimientos en tanto a redes, telemática y los protocolos implicados, analizamos el flujo de la comunicación entre Cliente y Servidor a través de Wireshark y nos dimos cuenta de que todo envió de una imagen, comenzaba por un paquete que contenía los *Flags* PSH y ACK activados, es decir, Flas PSH y ACK con valor de 1, y finaliza cuando el servidor contesta con un mensaje de *Status Code*, en nuestro caso un 200 OK, que quiere decir que todo está bien. Así, todos los paquetes que están en medio pertenecen a la misma transacción.

No.	Time	Source	Src_Port	Destination	Dst_Port	Protocol	Length	Info
64	0.197177	192.168.250.122	51795	192.168.250.146	80	TCP	312	51795 → 80 [PSH, ACK] Seq=197480 Ack=1837 Win=2102016 Len=258 [TCP segment of a reassembled PDU]
65	0.197668	192.168.250.146	80	192.168.250.122	51795	HTTP	79	HTTP/1.1 100 Continue
66	0.197779	192.168.250.122	51795	192.168.250.146	80	HTTP	20259	POST / HTTP/1.1 (application/x-www-form-urlencoded)
67	0.198062	192.168.250.146	80	192.168.250.122	51795	TCP	60	80 → 51795 [ACK] Seq=1862 Ack=205038 Win=317696 Len=0
68	0.198062	192.168.250.146	80	192.168.250.122	51795	TCP	60	80 → 51795 [ACK] Seq=1862 Ack=212338 Win=332288 Len=0
69	0.198062	192.168.250.146	80	192.168.250.122	51795	TCP	60	80 → 51795 [ACK] Seq=1862 Ack=217943 Win=343424 Len=0
70	0.198672	192.168.250.146	80	192.168.250.122	51795	HTTP	233	HTTP/1.1 200 OK
71	0.213007	192.168.250.122	51795	192.168.250.146	80	TCP	313	51795 → 80 [PSH, ACK] Seq=217943 Ack=2041 Win=2101760 Len=259 [TCP segment of a reassembled PDU]
72	0.213462	192.168.250.146	80	192.168.250.122	51795	HTTP	79	HTTP/1.1 100 Continue
73	0.213539	192.168.250.122	51795	192.168.250.146	80	HTTP	20278	POST / HTTP/1.1 (application/x-www-form-urlencoded)
74	0.213688	192.168.250.146	80	192.168.250.122	51795	TCP	60	80 → 51795 [ACK] Seq=2066 Ack=222582 Win=355200 Len=0
75	0.213688	192.168.250.146	80	192.168.250.122	51795	TCP	60	80 → 51795 [ACK] Seq=2066 Ack=234262 Win=378496 Len=0
76	0.213688	192.168.250.146	80	192.168.250.122	51795	TCP	60	80 → 51795 [ACK] Seq=2066 Ack=238426 Win=386816 Len=0
77	0.214824	192.168.250.146	80	192.168.250.122	51795	HTTP	233	HTTP/1.1 200 OK
78	0.226189	192.168.250.122	51795	192.168.250.146	80	TCP	313	51795 → 80 [PSH, ACK] Seq=238426 Ack=2245 Win=2101760 Len=259 [TCP segment of a reassembled PDU]
79	0.226592	192.168.250.146	80	192.168.250.122	51795	HTTP	79	HTTP/1.1 100 Continue
80	0.226663	192.168.250.122	51795	192.168.250.146	80	HTTP	19563	POST / HTTP/1.1 (application/x-www-form-urlencoded)
81	0.226815	192.168.250.146	80	192.168.250.122	51795	TCP	60	80 → 51795 [ACK] Seq=2270 Ack=244525 Win=401408 Len=0
82	0.226815	192.168.250.146	80	192.168.250.122	51795	TCP	60	80 → 51795 [ACK] Seq=2270 Ack=251825 Win=416000 Len=0
83	0.226815	192.168.250.146	80	192.168.250.122	51795	TCP	60	80 → 51795 [ACK] Seq=2270 Ack=256205 Win=424832 Len=0
84	0.226815	192.168.250.146	80	192.168.250.122	51795	TCP	60	80 → 51795 [ACK] Seq=2270 Ack=258194 Win=428800 Len=0
85	0.227507	192.168.250.146	80	192.168.250.122	51795	HTTP	233	HTTP/1.1 200 OK
86	0.247866	192.168.250.122	51795	192.168.250.146	80	TCP	312	51795 → 80 [PSH, ACK] Seq=258194 Ack=2449 Win=2101504 Len=258 [TCP segment of a reassembled PDU]
87	0.248343	192.168.250.146	80	192.168.250.122	51795	HTTP	79	HTTP/1.1 100 Continue
88	0.248431	192.168.250.122	51795	192.168.250.146	80	HTTP	8880	POST / HTTP/1.1 (application/x-www-form-urlencoded)
89	0.248539	192.168.250.146	80	192.168.250.122	51795	TCP	60	80 → 51795 [ACK] Seq=2474 Ack=262832 Win=440448 Len=0
90	0.248654	192.168.250.146	80	192.168.250.122	51795	TCP	60	80 → 51795 [ACK] Seq=2474 Ack=267278 Win=449280 Len=0
91	0.249091	192.168.250.146	80	192.168.250.122	51795	HTTP	233	HTTP/1.1 200 OK

Figura 14. Captura del flujo de datos perteneciente a una comunicación entre Cliente y Servidor

Siguiendo así este patrón de comportamiento, utilizamos dicha información para filtrar los paquetes y detectar las transferencias, pudiendo de esta manera ajustar la extracción de las características que teníamos en nuestros archivos de Python.

En dichos archivos, iré leyendo los paquetes del pcap uno a uno, pero solo analizaré las características cuando tenga el conjunto de paquetes que pertenecen a una misma transferencia; cuando detecte una transferencia se calculara todas las características y se guardaran en un diccionario, el cual tendrá una estructura de: {Numero de transferencia: X, *Flow_Duration:* valor, *Flow_Byts/s*: valor}

De eta forma, igual que hicimos en experimentos pasados, leeré 2 pcaps, una de las tramas MITM y otro con las tramas de flujo Normal, cada pcap me generará un archivo CSV y los juntaré para generar así nuestro DatasetUPV3 Recordemos que no basta con solo juntar ambos archivos, debemos mezclar los datos para no tener un *dataset* dividido, también debemos reestructurar la forma que presenta el *dataset*, en donde pongamos como columnas las características evaluadas y como filas los datos obtenidos y, por último, renombrar las columnas con el nombre adecuado. Ahora si tenemos nuestro DatasetUPV3 terminado y listo para operar con él, el cual cuenta con 4000 transacciones, lo que corresponde a las 2000 imágenes enviadas de Cliente a Servidor con influencia del MITM y las otras 2000 bajo un flujo Normal.

5.3.2 IA

El entrenamiento del modelo de IA para el experimento 3, sí que coincide con el entrenamiento realizado en los experimentos pasados, ya que las características son las misma (*Flow_Duration y Flow_Byts/s)*, tan solo habría que cambiar la fuente de datos, es decir, poner como fuente de datos nuestro DatasetUPV3. De esta forma, creamos el modelo con la herramienta de *Cross-Validation* para poder jugar con todos los datos correspondientes al nuestro *dataset*, teniendo en cada Split un 80% de los datos para el entrenamiento y un 20% para test, luego entrenamos el modelo y guardamos tanto el modelo como el *Scaler*.

Al entrenar el modelo, obtuvimos un valor de *accuracy* del 92%.

5.3.3 Validación

Como todo modelo que estamos validando, cabría generar unos pcap con datos nuevos para el modelo, por lo que realizaremos otra transferencia de 1994 imágenes de Cliente a Servidor, tanto para un ambiente de MITM como para un ambiente Normal.

Al tener estas capturas, pasamos a validar el comportamiento de nuestro modelo frente a ese conjunto de datos, por ejemplo, para la captura con datos provenientes de un ambiente Normal, obteníamos un *accuracy* del 92,6% lo que en un principio parece ser muy buen modelo y que se ajusta bastante al *accuracy* mostrado en el entrenamiento. Pero, sin embargo, al mirar el comportamiento sobre los datos provenientes de un ambiente con MITM el *accuracy* obtenido es del 48,9%, lo que es realmente malo y a la vez extraño.

Si miramos las capturas del tráfico de red que estamos analizando cuando se realiza la trasferencia de las imágenes de Cliente a Servidor, obtenemos que en el caso de MITM se obtienen más del doble de paquetes que en el caso del flujo Normal, en donde diferenciamos una gran cantidad de paquetes ACKs entre un caso y el otro. También descubrimos, que el traspaso de las imágenes por MITM se realizan de forma segmentada, no sabemos si este fenómeno ocurre de forma natural al hacer el MITM la encriptación de los datos o si se está produciendo otro fenómeno que desconocemos.

Lo que nos lleva enseguida a hacernos estas preguntas: ¿Por qué el modelo clasifica bien el flujo Normal y no el MITM? ¿Por qué tenemos una bajada tan grande en la *accuracy* del MITM si en principio este tercer experimento es una mejora al experimento 2, y así lo demuestra la validación de este experimento con el flujo Normal? ¿Experimento a experimento hemos estado mejorando la *accuracy* y por tanto el criterio de clasificación del modelo, porque ahora empeora, hemos cometido un error? ¿Por qué tenemos más del doble de paquetes para el caso de MITM que para el flujo Normal, es este fenómeno normal?

Dichas preguntas, son cuestiones que a día de hoy desafortunadamente no puedo resolver. Lo que sí puedo aportar es el trabajo realizado hasta la fecha, en donde hemos realizado varios experimentos para intentar detectar cualquier tipo de amenaza de carácter MITM, en donde experimento a experimento pudimos aportar conocimientos e ideas tras las cuales pudimos avanzar según el objetivo, en donde cada paso que lográbamos hacer, nos mantenía con la motivación de seguir planteando experimentos y llegar al objetivo de aportar un diseño e implementación de un algoritmo de Inteligencia Artificial capaz de detectar un posible ataque MITM basado en un ataque *ARP Spoofing*. De esta forma, concluimos nuestro último experimento, dejando una ventana de mejora y seguramente una futura línea de trabajo.

CAPÍTULO 6. CONCLUSIÓN Y TRABAJO FUTURO

6.1 CONCLUSIÓN

En el desarrollo de este TFG se ha trabajado fundamentalmente con dos sectores de gran escala e importancia tecnológica como lo son la Ciberseguridad y la Inteligencia Artificial, en donde pudimos unir y conceptualizar ambos sectores para llevar a cabo el diseño e implementación de un sistema de inteligencia artificial capaz de detectar ataques MITM basados en *ARP poisoning*; lo que supone un gran avance y punto de colaboración para el continuo estudio y desarrollo que se está realizando día a día por partes de identidades privadas y públicas para la detección y mitigación de ataques informáticos.

Tras el planteamiento y los experimentos desarrollados a lo largo del TFG, hemos ido cumpliendo de forma satisfactoria todos los objetivos listados en el punto 1.2 del Capítulo 1, lo que nos llevó a realizar varios modelos de IA, en donde pude desarrollar varios planteamientos de diseño, características y enfoques, hasta conseguir un modelo que cumpliera con las expectativas y objetivos propuestos, y de esta forma, poder presentar una solución final. Durante este proceso, obtuvimos que la mejor estrategia y concepto desarrollado, es el modelo que engloba el Experimento 2, que tras conseguir implementar nuestro propio *dataset* con un total de 4000 datos, hemos conseguido diseñar un modelo de inteligencia artificial capaz de detectar con una precisión aproximada del 83% si se está produciendo una actividad anómala correspondiente a la existencia de un MITM en la comunicación. Dejando así al experimento 2 como nuestro modelo estrella, el cual aporta incluso, mejores prestaciones que el experimento 3, lo cual es sorprendente, porque este tercer experimento corresponde, en un principio, a una mejora de concepto y técnica en tanto a la detección del supuesto MITM, donde se planteaba un análisis sobre la trasferencia completa de archivos, es decir, analizar el envío de una imagen, en este caso, y poder identificar si la transferencia cuanta con la presencia de un atacante MITM en la comunicación.

También es importante destacar que nuestra solución final, fue desarrollada con tan solo 2 de las 5 características más importantes que había destacado en un primer momento, y de esta forma, obtenemos muy buenas prestaciones, por lo que no es necesario analizar un modelo con 5 características, ya que esto supondría un mayor consumo de recursos informáticos, lo que se traduce como un mayor tiempo de respuesta, que para nuestro estudio, es fundamental mantener dicho tiempo lo más bajo posible, ya que queremos informar al Cliente, de la existencia de un MITM en la comunicación y una respuesta rápida puede ser clave para la protección de los datos.

De esta forma, satisfactoria y orgullosamente puedo aportar a través de mi TFG, un modelo de Inteligencia Artificial capaz de detectar ataques MITM, el cual fue sometido a diversas pruebas de diseño, enfoque y validación de los resultados sobre un entorno real.

6.2 IMPACTO GLOBAL Y ALCANCE

A lo largo del TFG se desarrolla una solución para cubrir un problema del sector de la ciberseguridad, que si bien no es nuevo, se encuentra en continuo crecimiento, en donde todavía queda mucho por recorrer en su campo de estudio y su amplio marguen de mejora; día a día el reto que presenta la ciberseguridad se vuelve más desafiante, debido a la innumerable cantidad de nuevos ataques que surgen el en sector informático; no solo surgen nuevos sistemas maliciosos, sino también, se está consiguiendo romper los protocolos de seguridad que ya han sido diseñados, generando así una brechas en el sistema de seguridad y por consiguiente su vulnerabilidad, lo que obliga mantener a este sector en un continuo desarrollo, evaluación e investigación de cara a nuevas soluciones, en donde el objetivo es brindar a los usuarios la confianza y seguridad de mantener sus datos a salvo.

De esta forma, el TFG busca atacar y aportar en un sector especifico de la ciberseguridad, en donde involucrando el uso de la inteligencia artificial, puedo brindar con total seguridad y orgullo, que he sido capaz de implementar y diseñar un modelo de IA capaz de detectar un posible ataque de MITM basado en *ARP Spoofing*, en donde además, junto al proyecto GUARDIAN, podemos aportar la primera línea de trabajo sobre el área de ataques *ransomware* sobre datos en movimiento, lo que se alinea a la perfección con los Objetivos de Desarrollo Sostenible de la Unión Europea, en donde el TFG desarrollado encaja a la perfección con el punto Nº9 Industria, Innovación e Infraestructura, el cual se base en mejorar el desarrollo de un país promoviendo el desarrollo de las infraestructuras, el sector industrial y la innovación, para de esta forma lograr la recuperación económica, la calidad de vida y nuevas oportunidades de empleo.

En tanto a esta idea, la Unión Europea presenta 5 metas las cuales quieren desarrollar y cumplir (Ministerio de Derechos Sociales, Consumo y Agenda 2030, 2021), en donde nuestro proyecto colabora con el cumplimiento de la siguiente meta: (Figura 15)

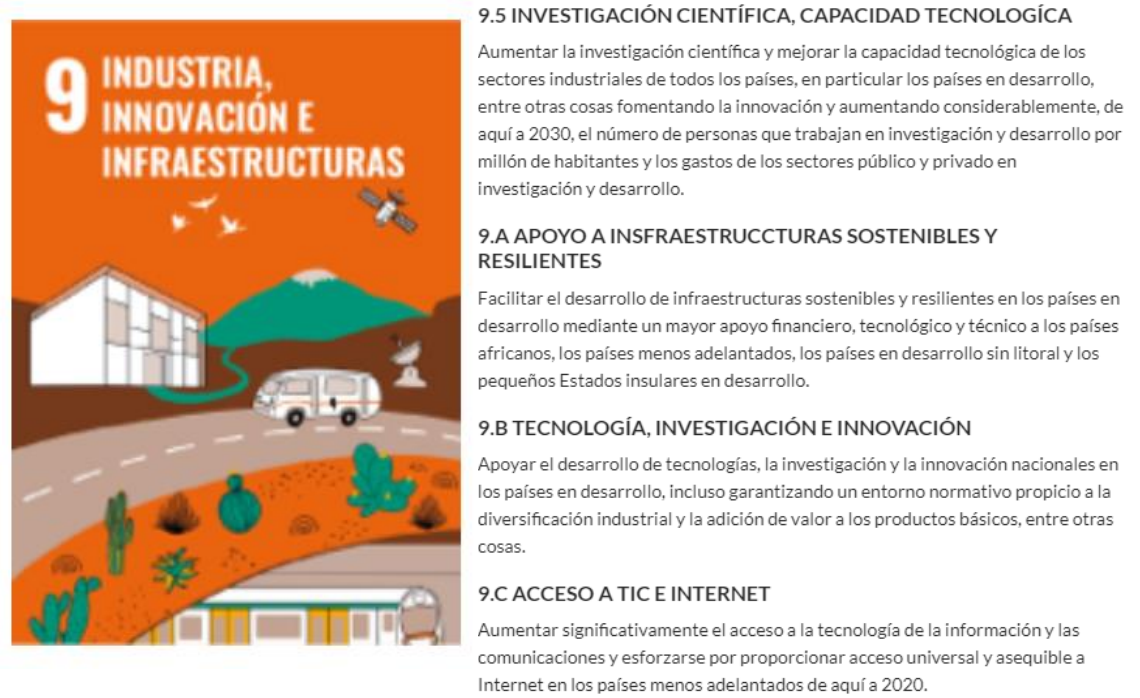


Figura 15. Objetivo Nº9 de Desarrollo Sostenible

6.3 TRABAJO FUTURO

Tomando como punto de partida el TFG realizado para la detección de ataques MITM, podemos plantear 2 líneas de trabajos futuros en las cuales convendría investigar y desarrollar:

1- **Experimento 3:** cabría continua con el estudio desarrollado del experimento 3, en donde la base del experimento consiste en el análisis de las trasferencias de imágenes y detectar a través de estas transferencias si se está produciendo un MITM o no. Hasta donde hemos llegado, obtenemos buenas prestaciones a la hora de caracterizar el flujo Normal, pero el modelo se comporta realmente mal cuando intenta clasificar los flujos anómalos de MITM.; desgraciadamente, no pude encontrar el motivo a este caso, pero de todos modos, desde mi punto de vista, el experimento planteado es muy interesante y valdría la pena continuarlo, ya que si se logra solucionar la cuestión de la malísima interpretación de los casos de MITM, creo sinceramente que el enfoque planteado presentara una mejora en tanto a la precisión de clasificación frente al experimento 2 que pude desarrollar.

2- **Experimento 4:** Como segunda línea de trabajo, me gustaría plantear este cuarto experimento, el cual se basaría en continuar con el estudio de transferencia de archivos, pero no solo basarnos en imágenes, sino también en otro tipo de archivos, como pueden ser archivos pdf, Word, txt, mp4, etc. Si puntualizamos, seria prácticamente la misma implementación realizada para el experimento 3, solo que extrapolando el análisis a otros tipos de archivos. Con esta propuesta y solución, podríamos presentar un modelo de IA capaz de detectar un MITM sobre cualquier tipo de dato que se trasmitan durante una comunicación.

Bibliografía

Braue, D. (10 de Septiembre de 2021). *El gasto Mundial en Ciberseguridad superará los 1.7 billones de dolares entre 2021 y 2025*. Recuperado el 10 de Junio de 2024, de CyberCrime Magazine: https://cybersecurityventures.com/cybersecurity-spending-2021-2025/

F.Pedregosa. (2011). Scikit-learn: Machine Learning in Python. *The Journal of Machine Learning Research, 12*, 2825-2830. Recuperado el 11 de Junio de 2024, de https://dl.acm.org/doi/10.5555/1953048.2078195

KimiNewt. (Dicimebre de 2023). *Pyshark*. Recuperado el 12 de junio de 2024, de GitHub: https://github.com/KimiNewt/pyshark

Michell, T. (1997). Does machine learning really work? *AI Magazine, 18*(3). Recuperado el 10 de Junio de 2024, de https://ojs.aaai.org/aimagazine/index.php/aimagazine/article/view/1303

Ministerio de Derechos Sociales, Consumo y Agenda 2030. (2021). *Objetivos de Desarrollo Sostenible*. Obtenido de Ministerio de Derechos Sociales, Consumo y Agenda 2030: https://www.mdsocialesa2030.gob.es/agenda2030/index.htm

U. Banerjee, A. V. (210). Evaluation of the Capabilities of WireShark as a tool for Intrusion Detection. *International Journal of Computer Applications, 6*(7), 1-5. Recuperado el 10 de Junio de 2024

Ullah, I. &. (2020). A scheme for generating a dataset for anomalous activity detection in iot networks. *In Canadian conference on artificial intelligence*, 508-520. Recuperado el 11 de Junio de 2024, de https://link.springer.com/chapter/10.1007/978-3-030-47358-7_52

WatchGuard. (8 de Enero de 2024). *Cada 39 segundos se produjo un ciberataque en 2023*. Recuperado el 10 de Junio de 2024, de WatchGuard: https://www.watchguard.com/es/wgrd-news/blog/cada-39-segundos-se-produjo-un-ciberataque-en-2023

ANEXOS

En este apartado se adjuntará todos los archivos que utilizamos para el desarrollo de la solución, los cuales pertenecen al Experimento 2.

EXTRACCIÓN DE LAS CARACTERÍSTICAS IMPORTANTES DE WIRESHARK A TRAVÉS DE LA LIBRERÍA PYSHARK EN PYTHON

```python
#-----------------------------------------------------------------------------------------------------------------------------------------
#                                                EXTRACCIÓN DE LAS CARACTERÍSTICAS
#                                                        IMPORTANTES
#-----------------------------------------------------------------------------------------------------------------------------------------

# FLOW_DURATION:Duración del flujo de datos en milisegundos.
# Para obtener Flow_Duration , bastará con identificar el tiempo de inicio y fin de un flujo , para luego restarlos

def calculate_flow_duration(capture_file, numero):
    cap = pyshark.FileCapture(capture_file, keep_packets=True) # True para guaradar los paquetes en memoria y poderlos usar luego
    pkt_next = None
    # Hacemos un blucle for sobre el paquete que se quiere analizar y también sobre el siguiente
    # así , tenemos el tiempo de inicio y fin del paquete requerido

    for i, pkt in enumerate(cap): # ponemos enumerate , pq tenemos 2 valores índices de iteracion , los paketes y su índice i

        if 'TCP' in pkt:
            if i == numero:
                pkt_1 = pkt
            elif i == numero + 1:
                pkt_next = pkt
                break # Salimos del bluce una vez obtengamos los 2 paquetes
        else:
            print(f'Los paquetes {numero} y {numero + 1} no son del protocolo TCP')


    # Calculamos el tiempo de inicio y fin del paqute requerido
    start_time = datetime.fromtimestamp(float(pkt_1.frame_info.time_epoch))
    end_time = datetime.fromtimestamp(float(pkt_next.frame_info.time_epoch))

    # Para calcular el flo_duration restamos el tiempo de inicio y fin
    flow_duration = end_time - start_time
    flow_duration_ms = flow_duration.total_seconds() #* 1000 # Convertir a milisegundos

    return flow_duration_ms
```

```python
#-----------------------------------------------------------------------------------------------------------------------------------------
# Flow_Pkts/s: Tasa de paquetes por segundo

def calculate_flow_byts_perSecond(capture_file, numero):

    cap = pyshark.FileCapture(capture_file, keep_packets=True)


    for i ,pkt in enumerate(cap):
        if 'TCP' in pkt:
            if i == numero:
                    pkt_1 = pkt
                    long_1 = len(pkt_1)
            elif i == numero + 1:
                    pkt_next = pkt
                    break # Salimos del bluce una vez obtengamos los 2 paquetes
        else:
            print(f'Los paquetes {numero} y {numero + 1} no son del protocolo TCP')

    # Calculamos el tiempo de inicio y fin del paquete requerido
    start_time = datetime.fromtimestamp(float(pkt_1.frame_info.time_epoch))
    end_time = datetime.fromtimestamp(float(pkt_next.frame_info.time_epoch))

    # Para calcular el flo_duration restamos el tiempo de inicio y fin
    flow_duration = (end_time - start_time).total_seconds() #*1000
    if flow_duration > 0:
        Flow_bytes_per_second = long_1 / flow_duration
    else:
        print(f' Debido a que el flow Duration es 0 , en el paquete {numero} no se puede calcular el Flow_bytes_per_second')
        Flow_bytes_per_second = 0

    return Flow_bytes_per_second
```

```python
# Down/Up_Ratio: Relación entre los bytes enviados desde el origen al destino y los bytes enviados desde el destino al origen.

from collections import defaultdict

def calculate_down_up_ratio(capture_file):

    cap = pyshark.FileCapture(capture_file, keep_packets=False)
    flow_count = {}
    Down_up_ratio = [] # Array donde guardo los ratios de cada flujo
    ip_cliente = '192.168.250.122'
    ip_server = '192.168.250.146'
    for pkt in cap:
        try:
            if 'TCP' in pkt:
                src_ip = pkt.IP.src
                dst_ip = pkt.TCP.srcport

                flow_key = tuple(sorted([src_ip, dst_ip]))
                packet_size = int(pkt.length) # Tamaño del paquete en bytes
            else:
                continue # Saltamos si el paquete no es TCP ni UDP
        except KeyError:
                # Caso en que la capa IP no está presente en el paquete
                print(f"El paquete {packet_count} contienen IPv6.")


        if flow_key not in flow_count:
            flow_count[flow_key] = {'up': 0, 'down': 0}

        # Determinar si el paquete es de subida o bajada
        if pkt.IP.src == ip_cliente:
            flow_count[flow_key]['up'] += packet_size
        elif pkt.IP.src == ip_server:
            flow_count[flow_key]['down'] += packet_size
```

```python
    # Calculamos el Dow/Up_Ratio para cada flujo
    for flow_key, flow_info in flow_count.items():
        total_data = flow_info['up'] + flow_info['down']
        print(f'total datos {total_data}')
        print(f'total paquetes {packet_size}')
        if total_data > 0: # Evitar la división por cero
            down_up_ratio = flow_info['down'] / total_data
            Down_up_ratio.append(down_up_ratio)
            print(f"Flujo {flow_key}: Dow/Up_Ratio = {down_up_ratio}")
            print(f'flujo {flow_count[flow_key]}')
        else:
            print(f"Flujo {flow_key}: No se pudo calcular el Dow/Up_Ratio")
            print(f'flujo {flow_count[flow_key]}')

    print(Down_up_ratio)
    return Down_up_ratio
```

```python
# Flow_IAT_Mean: Media del intervalo de tiempo entre paquetes para el flujo completo.

def calculate_flow_iat_mean (capture_file):

    cap = pyshark.FileCapture(capture_file, keep_packets=False)
    flow_count = {}
    Flow_IAT_Mean = []


    for pkt in cap:
        try:
            if 'TCP' in pkt:
                flow_key = (pkt.IP.src, pkt.TCP.srcport, pkt.IP.dst, pkt.TCP.dstport)
            else:
                continue # Saltamos si el paquete no es TCP ni UDP
        except KeyError:
                # Caso en que la capa IP no está presente en el paquete
                print(f"El paquete {packet_count} contienen IPv6.")

        # Convertimos el tiempo del paquete a datetime.datetime
        current_time = datetime.fromtimestamp(float(pkt.frame_info.time_epoch))
        # Para el primer paquete
        if flow_key not in flow_count:
            flow_count[flow_key] = {'packet_count': 1, 'start_time': current_time, 'end_time': current_time , 'inter_arrival_times': []}

        else:
            flow_count[flow_key]['packet_count'] += 1
            flow_count[flow_key]['end_time'] = current_time
```

```python
        # Solo calculamos el intervalo de tiempo entre paquetes si hay más de un paquete en el flujo
        if flow_count[flow_key]['packet_count'] > 1:
            # Ahora, antes de intentar acceder al último elemento, verificamos que la lista no esté vacía
            if flow_count[flow_key]['inter_arrival_times']:
                # convertimos last_packet en un objeto datatime , para luego poder hacer la resta en iat , donde restamos cuurent time (tipo datatime) con last_packet(tipo datatime)
                last_packet_time = flow_count[flow_key]['start_time'] + timedelta(seconds=flow_count[flow_key]['inter_arrival_times'][-1])
                iat = (current_time - last_packet_time).total_seconds()
                flow_count[flow_key]['inter_arrival_times'].append(iat)
            else:
                start_time = flow_count[flow_key]['start_time']
                # Si es el primer intervalo de tiempo, simplemente añadimos el tiempo actual
                flow_count[flow_key]['inter_arrival_times'].append((current_time - start_time).total_seconds())

    # Calculamos Flow_IAT_Mean para cada flujo
    for flow_key, flow_info in flow_count.items():
        if flow_info['packet_count'] > 1:
            flow_IAT_Mean = sum(flow_info['inter_arrival_times']) / len(flow_info['inter_arrival_times']) # Aqui guardamos nuestra caracteristica
            Flow_IAT_Mean.append(flow_IAT_Mean)
            print(f"Flow {flow_key}: Flow_IAT_Mean = {flow_IAT_Mean} seconds")
        else:
            print(f"Flow {flow_key}: Not enough packets to calculate Flow_IAT_Mean")
    return Flow_IAT_Mean
```

CREACIÓN DE NUESTRO DATASET

```python
import json
import pandas as pd
import pickle

# Cargamos nuestro modelo entrenado y nuestro saclaer
loaded_model = pickle.load(open('features_UPV2.sav', 'rb'))
loaded_scaler = pickle.load(open('Scalaer_UPV2.sav', 'rb'))

features = {} # Diccionario donde cada clave es un número de paquete y cada valor es otro diccionario con las características del paquete:
capture_file= 'C:/Users/Jose A Martinez/Desktop/GUARDIAN/Wireshark/Capturas/animales/http_mitm_animales.pcapng'
predictions = [] # array donde guardaré todas las predicciones
for packet_count, packet in enumerate(capture):

    # Hacemos una comprobacion en cierto punto del dataset para verificar que se estaba generando correctamente
    if packet_count == 500:
        with open('EXP2_animales_mitm_MITAD.json', 'w') as f:
            json.dump(features, f)

    packet_features ={}

    # Obtenemos las características deseadas
    packet_features['flow_duration'] = calculate_flow_duration(capture_file, packet_count)
    packet_features['flow_byts_perSecond'] = calculate_flow_byts_perSecond(capture_file, packet_count)
    packet_features['label'] = 'MITM'
    # Almacenar las características del paquete en el diccionario principal
    features[packet_count] = packet_features # le asociamos a cada paquete el valor de las caracteristicas


    #Solo incorporamos a nuestro Dataset 3000 paquetes
    if packet_count == 3000:
        print(f'Diccionario {features}')
        break

# Cerrar la captura de archivo
capture.close()
```

```python
# GENERAMOS LOS 2 DATAFRAME DE PANDAS , PARA LUEGO JUNTARLOS Y CREAR ASI
# EL DATASET_UPV2 QUE VISUALIZAREMOS EN JUPYTER NOTEBOOK
df = pd.DataFrame(features)
df.to_csv('dt_EXP2_animales_mitm.csv', index=False) # Guardamos el dataset en CSV
#-----------------------------------------------------------------------------------------------------------------------
#-----------------------------------------------------------------------------------------------------------------------
# Indicamos la ruta de mis diccionarios
ruta1 = 'C:/Users/Jose A Martinez/Desktop/GUARDIAN/Wireshark/Código/EXP2/dt_EXP2_animales_mitm.csv'
ruta2 = 'C:/Users/Jose A Martinez/Desktop/GUARDIAN/Wireshark/Código/EXP2/dt_EXP2_animales_normal.csv'

# Convertir los diccionarios en DataFrames
dt1 = pd.read_csv(ruta1)
dt2 = pd.read_csv(ruta2)

# Combinar los DataFrames
Dataset_UPV = pd.concat([dt1, dt2], axis=1)

#-----------------------------------------------------------------------------------------------------------------------
# REORGANIZAMOS EL DATASET
# Crear un nuevo diccionario para almacenar los datos reorganizados
reorganized_data = {}


for packet_number, packet_data in Dataset_UPV.items():

    for feature, value in packet_data.items():
        # Si la característica no está en el diccionario reorganizado, añadirla con una lista
        if feature not in reorganized_data:
            reorganized_data[feature] = []
        # Añadir el valor de la característica a la lista correspondiente
        reorganized_data[feature].append(value)

# Convertir el diccionario reorganizado a un DataFrame de pandas
Dataset_UPV2 = pd.DataFrame(reorganized_data)

#Renombramos las columnas
Dataset_UPV2 = Dataset_UPV2.rename(columns={ 0:'Flow_Duration', 1:'Flow_Byts/s', 2:'Label'})


# GUARDAMOS EL DATASET_UPV
# Guardar el DataFrame modificado en un archivo CSV
Dataset_UPV2.to_csv('Dataset_UPV2.csv', index=False)
#-----------------------------------------------------------------------------------------------------------------------
```

VALIDACIÓN DEL MODELO DE IA

```python
#-----------------------------------------------------------------------------------------------------------------------------------------
#                                                VALIDAMOS NUSTRO MODELO DE IA
#                                                      YA ENTRENADO
#-----------------------------------------------------------------------------------------------------------------------------------------

import json
import pandas as pd
import pickle

# Cargamos nuestro modelo entrenado y nuestro saclaer
loaded_model = pickle.load(open('features_UPV2.sav', 'rb'))
loaded_scaler = pickle.load(open('Scalaer_UPV2.sav', 'rb'))

features = {} # Diccionario donde cada clave es un número de paquete y cada valor es otro diccionario con las características del paquete:
capture_file= 'C:/Users/Jose A Martinez/Desktop/GUARDIAN/Wireshark/Capturas/animales/http_mitm_animales.pcapng'
predictions = [] # array donde guardaré todas las predicciones
count_mitm = 0
count_normal = 0

for packet_count, packet in enumerate(capture):


    print(f" Paquete {packet.number}")
    packet_features ={}

    # Obtenemos las características deseadas
    packet_features['flow_duration'] = calculate_flow_duration(capture_file, packet_count)
    packet_features['flow_byts_perSecond'] = calculate_flow_byts_perSecond(capture_file, packet_count)

    # Almacenar las características del paquete en el diccionario principal
    features[packet_count] = packet_features # le asociamos a cada paquete el valor de las características
```

```python
    # EVALUAMOS EL MODELO
    X= [[packet_features['flow_duration'],packet_features['flow_byts_perSecond']]]

    # cargamos el scaler
    X = loaded_scaler.transform(X)
    y_pred = loaded_model.predict(X)
    predictions.append(y_pred[0]) # añadimos la prediccion comenzando en la posicion 0


    # 1=Normal   0=MITM
    # marcamos en la etiqueta label la predicción del modelo si es MITM o Normal
    if y_pred[0] == 1:
        packet_features['label'] = 'Normal'
        count_normal +=1
    else:
        packet_features['label'] = 'MITM'
        count_mitm +=1


    if packet_count == 1000:
        porcentaje = count_mitm*100
        Accuracy = (porcentaje/packet_count)
        print(f'Diccionario {features}')
        print (f'Accuracy : {Accuracy}') # MITM 61.4%
        print(f'Paquetes normales {count_normal}') # MITM------- 387 PAQUETES
        print(f'Paquetes MITM {count_mitm}') # MITM------------- 614 PAQUETES
        break

# Guardamos el diccionario en un csv para su visualizacion
with open('Validacion_EXP2_animales_mitm.json', 'w') as f:
    json.dump(features, f)
# Cerrar la captura de archivo
capture.close()
```

Entrenamiento del modelo de IA

DATASET CREADO CON LA TRASFERENCIA DE 50 IMÁGENES NORMALES Y 50 MITM

[1]:
```
import pandas as pd
df = pd.read_csv(r'C:\Users\Usuario.DESKTOP-KE22E9A\Desktop\GUARDIAN\Wireshark\Código\EXP2\Dataset_UPV.csv')
df.head(10)
```

[1]:

	Flow_Duration	Flow_Byts/s	Label
0	0.000133	2.300752e+06	MITM
1	0.011611	5.412109e+05	MITM
2	1.028981	2.264376e+02	MITM
3	0.000126	2.246032e+06	MITM
4	0.003123	1.883445e+06	MITM
5	1.012809	2.300532e+02	MITM
6	0.000122	2.319672e+06	MITM
7	0.003624	2.248896e+06	MITM
8	1.011864	2.302681e+02	MITM
9	0.000108	2.620370e+06	MITM

[2]:
```
# Cambiamos el orden aleatoriamente de las filas
df= df.sample(frac=1).reset_index(drop=True)
df.head(10)
```

[2]:

	Flow_Duration	Flow_Byts/s	Label
0	0.000134	2.111940e+06	MITM
1	1.024865	2.273470e+02	Normal
2	0.000114	1.328070e+07	MITM
3	0.000117	2.427350e+06	MITM
4	0.000827	1.314027e+07	Normal
5	1.012776	2.300607e+02	MITM
6	1.014641	2.296379e+02	Normal
7	0.000110	2.572727e+06	MITM
8	0.003634	2.423225e+06	MITM
9	0.000175	1.617143e+06	MITM

```
VEMOS LA NATURALEZA DE LOS DATOS
```

[3]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 308 entries, 0 to 307
Data columns (total 3 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Flow_Duration  308 non-null    float64
 1   Flow_Byts/s    308 non-null    float64
 2   Label          308 non-null    object
dtypes: float64(2), object(1)
memory usage: 7.3+ KB
```

```
Dejamos la columna Label en 1 o 0
De cara a la evaluación del modelo es mejor tener números que strings
```

[4]:
```
#LABEL ENCODING
from sklearn.preprocessing import LabelEncoder

# Codificador
encoder = LabelEncoder()

# Ajustamos y transformamos la columna
df['Label'] = encoder.fit_transform(df['Label'])
```

[5]:
```
df.head(5)
# 1 = NORMAL    0 = MITM
```

[5]:

	Flow_Duration	Flow_Byts/s	Label
0	0.000134	2.111940e+06	0
1	1.024865	2.273470e+02	1
2	0.000114	1.328070e+07	0
3	0.000117	2.427350e+06	0
4	0.000827	1.314027e+07	1

[6]:
```
df.shape
```

[6]:
```
(308, 3)
```

ENTRENAMOS EL MODELO UTILIZANDO CROSS-VALIDATION

[7]:
```
import pandas as pd
from sklearn.model_selection import KFold
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix


k = 4
kfold = KFold(n_splits=k, shuffle=True , random_state=42)

X = df[['Flow_Duration', 'Flow_Byts/s']] # Eje x
Y = df['Label']

# Para guardar los valores obtenidos en cada division
accuracy_scores = []
confusion_matrices = []

for train, test in kfold.split(X):
    X_train = X.iloc[train]
    X_test = X.iloc[test]
    y_train = Y.iloc[train]
    y_test = Y.iloc[test]

    # Normalizamos los datos
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test) # fit_transform


    # Creamos el modelo
    model = RandomForestClassifier(n_estimators=4, random_state=42)

    # Entrenamos el modelo
    model.fit(X_train, y_train)
```

```
    # Predecir en el conjunto de prueba para luego evaluar el modelo
    y_pred = model.predict(X_test)

    # Evaluamos el rendimiento
    accuracy = accuracy_score(y_test, y_pred)
    accuracy_scores.append(accuracy)
    confusion_matrices.append(confusion_matrix(y_test, y_pred))

    # f-string, construcción de cadenas de texto dinámicas sea más sencilla y legible
    print(f'train: {train}, test: {test}')
    print("Accuracy:", accuracy)
    print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))


# Calculamos el rendimiento promedio
avg_accuracy = sum(accuracy_scores) / k
print("Average Accuracy:", avg_accuracy)
```

```
Average Accuracy: 0.922077922077922
```

NECESITAMOS UN DATASET MAS GRANDE PARA PODER VALIDAR DE FORMA CORRECTA

```
import pandas as pd
df3 = pd.read_csv(r'C:\Users\Usuario.DESKTOP-KE22E9A\Desktop\GUARDIAN\Wireshark\Código\EXP2\Dataset_UPV2.csv')
df3.head(10)
```

	Flow_Duration	Flow_Byts/s	Label
0	0.000742	8.894879e+04	MITM
1	0.000062	1.064516e+06	MITM
2	0.000849	6.360424e+04	MITM
3	0.000658	4.574468e+05	MITM
4	0.000443	1.354402e+05	MITM
5	0.000061	1.295082e+06	MITM
6	0.001789	3.242035e+04	MITM
7	0.018297	1.273433e+04	MITM
8	0.001515	2.019802e+05	MITM
9	0.000083	9.518072e+05	MITM

```
# Cambiamos el orden aleatoriamente de las filas
df3 = df3.sample(frac=1).reset_index(drop=True)
df3.head(10)
```

[10]:

	Flow_Duration	Flow_Byts/s	Label
0	0.000276	2.329493e+08	NORMAL
1	0.000265	2.316000e+08	NORMAL
2	0.000904	6.637168e+04	NORMAL
3	0.000603	9.950249e+04	MITM
4	0.000279	1.100860e+08	NORMAL
5	0.000509	5.842829e+06	MITM
6	0.000357	1.680672e+05	MITM
7	0.000518	2.031081e+07	NORMAL
8	0.000582	5.109966e+06	MITM
9	0.000242	1.208843e+08	NORMAL

[11]:
```
#LABEL ENCODING
from sklearn.preprocessing import LabelEncoder

# Codificador
encoder = LabelEncoder()

# Ajustamos y transformamos la columna
df3['Label'] = encoder.fit_transform(df3['Label'])
```

[12]:
```
df3.head(5)
```

[12]:

	Flow_Duration	Flow_Byts/s	Label
0	0.000276	2.329493e+08	1
1	0.000265	2.316000e+08	1
2	0.000904	6.637168e+04	1
3	0.000603	9.950249e+04	0
4	0.000279	1.100860e+08	1

ENTRENAMOS EL MODELO UTILIZANDO CROSS-VALIDATION

[14]:
```
import pandas as pd
from sklearn.model_selection import KFold
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix


k = 4
kfold = KFold(n_splits=k, shuffle=True , random_state=42)

X = df3[['Flow_Duration', 'Flow_Byts/s']] # Eje x
Y = df3['Label']

# Para guardar los valores obtenidos en cada division
accuracy_scores = []
confusion_matrices = []

for train, test in kfold.split(X):
    X_train = X.iloc[train]
    X_test = X.iloc[test]
    y_train = Y.iloc[train]
    y_test = Y.iloc[test]
```

```
    # Normalizamos los datos
    scaler2 = StandardScaler()
    X_train = scaler2.fit_transform(X_train)
    X_test = scaler2.transform(X_test) # fit_transform


    # Creamos el modelo
    model2 = RandomForestClassifier(n_estimators=4, random_state=42)

    # Entrenamos el modelo
    model2.fit(X_train, y_train)

    # Predecimos en el conjunto de prueba para luego evaluar el modelo
    y_pred = model2.predict(X_test)

    # Evaluamos el rendimiento
    accuracy = accuracy_score(y_test, y_pred)
    accuracy_scores.append(accuracy)
    confusion_matrices.append(confusion_matrix(y_test, y_pred))

    # f-string, construcción de cadenas de texto dinámicas sea más sencilla y legible.
    print(f'train: {train}, test: {test}')
    print("Accuracy:", accuracy)
    print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))


# Calculamos el rendimiento promedio
avg_accuracy = sum(accuracy_scores) / k
print("Average Accuracy:", avg_accuracy)
```

```
train: [   0    1    2 ... 5853 5854 5855], test: [   8   12   14 ... 5847 5848 5851]
Accuracy: 0.8271857923497268
Confusion Matrix:
 [[650 105]
 [148 561]]
train: [   1    2    3 ... 5851 5852 5854], test: [   0    6   19 ... 5850 5853 5855]
Accuracy: 0.8183060109289617
Confusion Matrix:
 [[617 112]
 [154 581]]
train: [   0    2    3 ... 5853 5854 5855], test: [   1    7   11 ... 5837 5839 5845]
Accuracy: 0.8230874316939891
Confusion Matrix:
 [[651 102]
 [157 554]]
train: [   0    1    6 ... 5851 5853 5855], test: [   2    3    4 ... 5849 5852 5854]
Accuracy: 0.7991803278688525
Confusion Matrix:
 [[639 119]
 [175 531]]
Average Accuracy: 0.8169398907103826
```

GRÁFICAS DE LAS CARACTERÍSTICAS

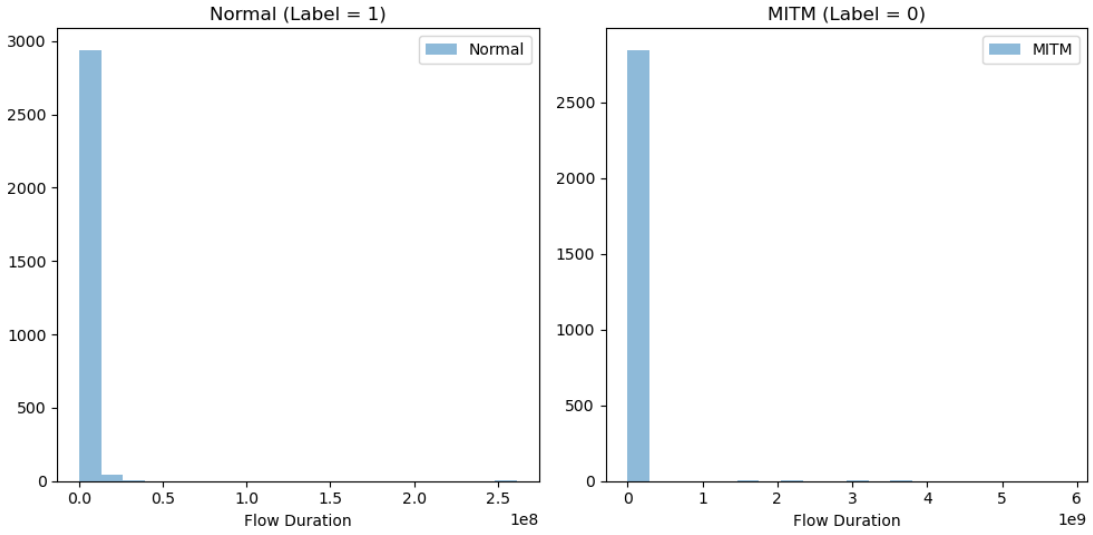
[16]:
```
import matplotlib.pyplot as plt
mitm_data = df3[df3['Label'] == 0]['Flow_Byts/s']
normal_data = df3[df3['Label'] == 1]['Flow_Byts/s']

# Creamos un gráfico de caja para cada categoría
plt.figure(figsize=(10, 5))

# Creamos un histograma para cada categoría
plt.subplot(1, 2, 1)
plt.hist(mitm_data, bins=20, alpha=0.5, label='Normal')
plt.title('Normal (Label = 1)')
plt.xlabel('Flow Duration')
plt.legend()

plt.subplot(1, 2, 2)
plt.hist(normal_data, bins=20, alpha=0.5, label='MITM')
plt.title('MITM (Label = 0)')
plt.xlabel('Flow Duration')
plt.legend()
```

```
plt.tight_layout()
plt.show()
```



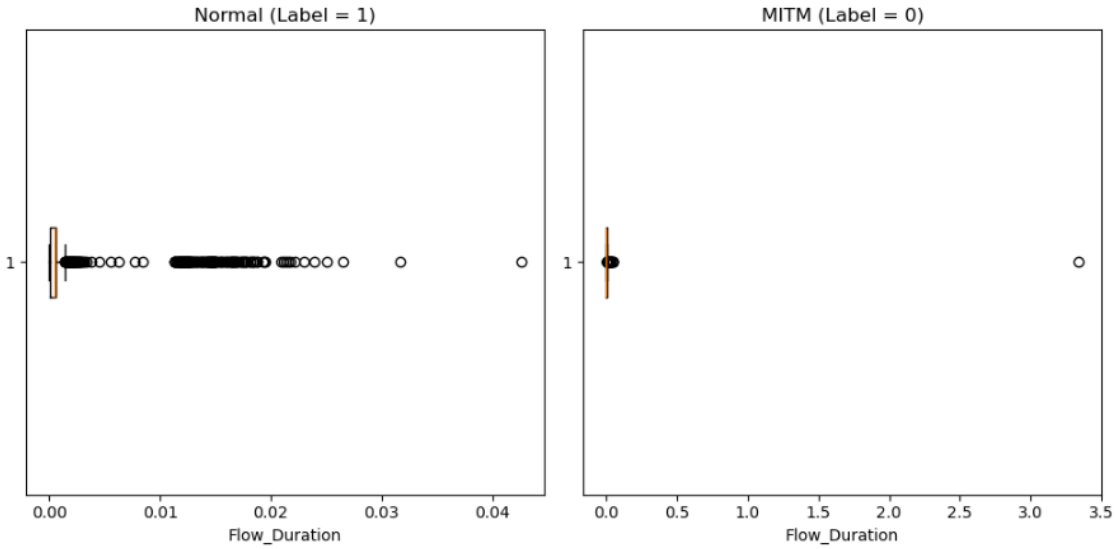
[17]:
```
mitm_data = df3[df3['Label'] == 0]['Flow_Duration']
normal_data = df3[df3['Label'] == 1]['Flow_Duration']

# Creamos un gráfico de caja para cada categoría
plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
plt.boxplot(mitm_data, vert=False)
plt.title('Normal (Label = 1)')
plt.xlabel('Flow_Duration')

plt.subplot(1, 2, 2)
plt.boxplot(normal_data, vert=False)
plt.title('MITM (Label = 0)')
plt.xlabel('Flow_Duration')

plt.tight_layout()
plt.show()
```



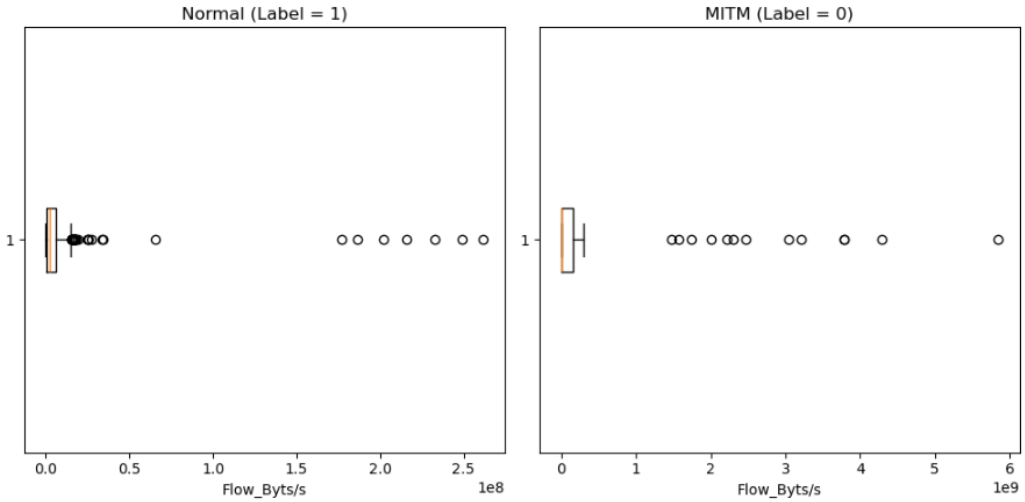
[18]:
```
mitm_data = df3[df3['Label'] == 0]['Flow_Byts/s']
normal_data = df3[df3['Label'] == 1]['Flow_Byts/s']

# Creamos un gráfico de caja para cada categoría
plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
plt.boxplot(mitm_data, vert=False)
plt.title('Normal (Label = 1)')
plt.xlabel('Flow_Byts/s')

plt.subplot(1, 2, 2)
plt.boxplot(normal_data, vert=False)
plt.title('MITM (Label = 0)')
plt.xlabel('Flow_Byts/s')

plt.tight_layout()
plt.show()
```



GUARDAMOS EL MODELO

[19]:
```
import pickle
pickle.dump (model2 , open('features_UPV2.sav', 'wb'))

pickle.dump (scaler2 ,open('Scalaer_UPV2.sav', 'wb'))
```