Ataques a un Sistema de Detección de Intrusiones mediante Redes Generativas Adversarias

Attacks on an Intrusion Detection System by Adversary Generative Networks

TRABAJO FIN DE GRADO
GRADO EN INGENIERÍA INFORMÁTICA
CURSO 2022–2023

Aitor Esteban Núñez
Roberto Portillo Torres
Marcos Matute Fernández

Directores

Luis Javier García Villalba
Luis Alberto Martínez Hernández

Departamento de Ingeniería del Software e Inteligencia Artificial
Facultad de Informática
Universidad Complutense de Madrid

Madrid, Junio de 2023

Agradecimientos

A nuestros directores del trabajo y a todas las personas del Grupo de Análisis, Seguridad y Sistemas de la Universidad Complutense de Madrid que nos han ayudado y explicado cualquier duda en todo momento. En especial a Luis Alberto Martínez Hernández que ha estado disponible para nosotros todo este tiempo y nos ha resuelto los problemas que teníamos tanto en las reuniones como en horarios diferentes.

Índice General

Índice de Figuras

Índice de Tablas

Lista de Códigos

Lista de Acrónimos

AA *Aprendizaje Automático*

AI *Artificial Intelligence*

ANN *Artificial Neural Network*

AP *Aprendizaje Profundo*

API *Application Programming Interface*

CNN *Convolutional Neural Networks*

DL *Deep Learning*

DoS *Denial of Service*

GAN *Generative Adversarial Networks*

IA *Inteligencia Artificial*

IDS *Intrusion Detection System*

ML *Machine Learning*

RNN *Recurrent Neural Networks*

RNP *Redes Neuronales Profundas*

Abstract

The artificial intelligence is experiencing a boom in the society, the power of these algorithms is growing and new applications are constantly appearing: image classifiers, voice recognizers, chatbots, etc. Artificial intelligence, like all other branches of computer science, must be concerned about the security of its algorithms and data processing. This work is an analysis of adversarial attacks, which use the tools of artificial intelligence against itself. If a model can be trained with data to detect patterns and make predictions it can also be trained to generate data that can falsify these predictions. Throughout this project different ways of performing adversarial attacks are analyzed focusing mainly on attacking malware classifiers. In addition, an example implementation of an adversarial attack on an intrusion detection system is given and the results will be analyzed.

Keywords: Adversarial Attack, Adversarial Network, Arificial Intelligence, Deep Learning, Generative Malware, Machine Learning, Neural Network.

Resumen

La inteligencia artificial está experimentando un fuerte impacto en la sociedad, la potencia de estos algoritmos no para crecer y no paran de aparecer nuevas aplicaciones: clasificadores de imágenes, reconocedores de voz, chatbots, etc. La inteligencia artificial, como el resto de ramas de la informática, debe preocuparse por la seguridad de sus algoritmos y del tratamiento de datos. Este trabajo es un análisis de los ataques adversarios, los cuales utilizan las herramientas de la inteligencia artificial contra sí misma. Si un modelo puede entrenarse con datos para detectar patrones y hacer predicciones también se puede entrenar para generar datos que puedan falsear estas predicciones. A lo largo de este proyecto se analizan distintas formas de realizar ataques adversarios centrándose principalmente en ataques a clasificadores de software malicioso. Además se da un ejemplo de implementación de ataque adversario a un sistema de detección de intrusos del que se analizarán los resultados.

Palabras clave: Aprendizaje Profundo, Aprendizaje Automático, Ataque Adversario, Red Generativa Adversaria, Malware, Redes Neuronales, Inteligencia Artificial.

Capítulo 1

Introducción

1.1. Motivación

En la actualidad, la *Inteligencia Artificial* (IA) es uno de los campos de la informática con más desarrollo e inversiones a nivel global. Los avances están permitiendo crear multitud de herramientas útiles para el ser humano en contextos muy diferentes. Entre ellas se encuentran los usos de la IA para la seguridad de otros campos como por ejemplo clasificadores de malware o detectores de tráfico malicioso en redes. Sin embargo, la IA también tiene vulnerabilidades y puede recibir ataques que perturben su correcto funcionamiento.

Los ataques adversarios son la mayor preocupación si se habla de seguridad en IA puesto que pueden engañar de diferentes maneras a algoritmos de *Machine Learning* (ML) entre otras cosas. En en 2014, Ian Goodfellow propuso las Redes Generativas Adversarias conocidas en inglés como *Generative Adversarial Networks* (GAN) en su trabajo [GPAM+20]. Estas son a día de hoy, la forma más estudiada y utilizada para realizar ataques adversarios. Este trabajo se centrara en el estudio de las vulnerabilidades y de la seguridad de los algoritmos de ML mediante la investigación y posterior experimentación de ataques que están a la orden del día en el mundo cada vez más grande de la IA. Para ello se llevará a cabo la implementación de una red GAN que se utilizara para realizar un ataque adversario contra algún algoritmo de ML.

1.2. Contexto

El presente Trabajo Fin de Grado se enmarca dentro de un proyecto de investigación titulado Platform for Analysis of Resilient and Secure Software – LAZARUS, aprobado por la Comisión Europea dentro del Programa Marco Horizonte (convocatoria HORIZON-CL3-2021-CS-01) en virtud del acuerdo de subvención número 101070303 y

en el que participa el Grupo GASS de la Universidad Complutense de Madrid (Grupo de Análisis, Seguridad y Sistemas, https://gass.ucm.es, grupo 910623 del catálogo de grupos de investigación reconocidos por la UCM).

Además de la Universidad Complutense de Madrid participan en LAZARUS las siguientes entidades: Athena Research Center – ARC (Grecia), The University of Padua (Italia), Infotrend Innovations Company Limited (Chipre), Data Centric Services SRL (Rumanía), Luxembourg Institute of Science and Technology (Luxemburgo), Motivian EOOD (Bulgaria), Binare Oy (Finlandia), Fundación APWG European Union Foundation (España), Maggioli Spa (Italia).

Tienen más información en:

https://cordis.europa.eu/project/id/101070303

https://lazarus-he.eu

1.3. Objeto de la Investigación

El objetivo de este trabajo es realizar un estudio e investigación sobre la seguridad en la IA, centrándose en los ataques adversarios y más en concreto en las Redes Generativas Adversarias o en inglés Generative Adversarial Networks (GAN). Tras tener una base sólida de conocimientos sobre el tema, se tratará de diseñar e implementar una GAN que sea capaz de engañar a algún algoritmo de ML. El objetivo es obtener una implementación eficiente y probarla de diferentes maneras.

1.4. Plan de Trabajo

El desarrollo de este trabajo se ha realizado en tres fases distribuidas según el diagrama de Gantt de la Figura 1.1 y descritas a continuación:

1. **Investigación**:

   Para comenzar, mediante la realización de una reunión general, se explicó el punto inicial o de partida del proyecto así como los objetivos principales y los conocimientos que se iban a necesitar para llevarlos a cabo. Posteriormente, se acordó reunirse semanalmente con la idea de realizar un seguimiento de los avances y progreso en la investigación y para resolver las posibles dudas que pudiesen surgir. Otra razón por la cual se concertaron dichas reuniones fue para poder desarrollar y tratar conceptos básicos sobre los distintos temas o campos que concierne este trabajo, los cuales serán tratados en los siguientes apartados. Esta fase comenzó a mediados de septiembre de 2022. Durante los primeros cuatro meses hubo que adaptarse al contexto del

trabajo y fue necesario adquirir conocimientos para dar pie al posterior desarrollo. Para ello, se realizaron cursos sobre diferentes temas relacionados y lecturas de artículos que eran referencias base del trabajo. Cada integrante del grupo realizaba resúmenes y presentaciones de los artículos sobre los que investigaba cada semana para realizar poco a poco un repositorio de conocimiento común sobre el trabajo en *Google Drive* y para avanzar conjuntamente hacia un objetivo común. En el último mes de investigación, el equipo ya tenía una base lo suficientemente fuerte como para empezar a plantearse a revisar código y pensar en la etapa de desarrollo. La investigación en sí se seguía llevando a cabo constantemente mediante la lectura de artículos con código vinculado para referenciarse y observar implementaciones antes de comenzar con la propia.

2. **Desarrollo**: En el momento en el que se tubieron suficientes conocimientos, se acordó dedicar menos tiempo a la investigación y empezar a codificar la propuesta para elaborar una versión sólida del proyecto. La fase de investigación, como ya se ha señalado, continuó pero pasó a un segundo plano, ya que se investigó sobre nociones concretas que surgieron durante el desarrollo. Por ello, en esta fase se estudiaron principios avanzados del lenguaje de programación *Python* y de librerías necesarias para la implementación. Se estudiaron diferentes propuestas sobre redes GAN. Durante esta fase se procesaron diferentes conjuntos de datos para entenderlos mejor y se implementaron las funcionalidades necesarias para llevar a cabo el proyecto. Fue en esta fase donde se decidió que se utilizaría nuestra implementación de la GAN para atacar un Sistema de Detección de Intrusos (IDS, del inglés *Intrusion Detection System*). Se realizó la implementación de este mediante un algoritmo de ML y procesando un conjunto de datos específico.

3. **Experimentación**: Una vez que se empezaron a fabricar los primeros prototipos del concepto, se puso en marcha el proceso de experimentación. Durante este tiempo, el equipo analizó los resultados y llegó a ciertas conclusiones. Cabe mencionar que el modelo se comparó continuamente con otros y se optimizó durante la fase de experimentación, continuando así el proceso de desarrollo.

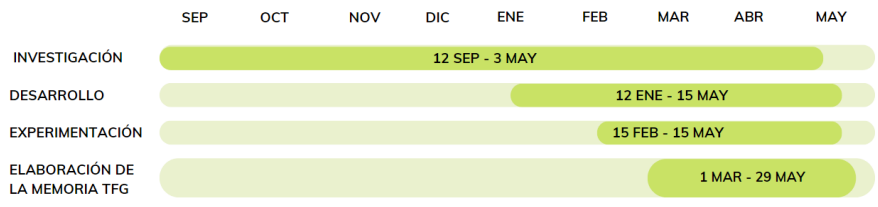

Figura 1.1: Diagrama de Gantt del Proyecto.

1.5. Estructura del Trabajo

El presente trabajo está dividido en 8 capítulos de los cuales 6 capítulos están en castellano y 2 en inglés. Seguidamente se presenta su estructura:

El Capítulo 2 se realiza una introducción de los conceptos base para comprender el desarrollo del trabajo. Se introducen bases de *Aprendizaje Automático* (AA), *Aprendizaje Profundo* (AP), seguridad y el objeto principal de nuestro trabajo: las GAN. También se aporta algún ejemplo de ataque adversario.

El Capítulo 3 se realiza una recopilación de trabajos relacionados con el tema de estudio, centrada en las diferentes amenazas adversarias existentes y se proponen ejemplos de artículos que realizan ataques adversarios.

En el Capítulo 4 se desarrolla la metodología y diseño del trabajo, explicando la fases que tuvieron las implementaciones realizadas para llevar a cabo este trabajo y cómo se realizaron.

El Capítulo 5 describe los experimentos realizados para evaluar la efectividad la GAN implementada y desarrollada en el capítulo 4 y presenta los resultados obtenidos.

El Capítulo 6 presenta las contribuciones personales de cada miembro del grupo en este trabajo.

El Capítulo 7 muestra conclusiones extraídas en este trabajo y las posibles futuras investigaciones.

Los Capítulos 8 y 9 son las traducciones al inglés de la Introducción y de las Conclusiones.

Capítulo 2

Marco Teórico

En este capítulo se explica el contexto en el que se basa la investigación de este Trabajo Fin de Grado. En el punto 2.1, se expone una pequeña base sobre conceptos de Aprendizaje Automático profundizando el los árboles de decisión y en el punto 2.2 sobre Aprendizaje Profundo. A continuación, en la sección 2.3 se introducen conceptos importantes sobre seguridad de información y en la 2.4 se profundizará en seguridad en inteligencia artificial introduciendo los ataques adversarios. En la sección 2.5 se presentan las GAN como concepto clave del trabajo. Por último se van a explicar conceptos importantes que se van a utilizar a lo largo del trabajo como son los ataques DoS en la sección 2.6, las GAN tabulares en la 2.7, la librería *Kafka* en la 2.8 y por último los IDS en la sección 2.9.

2.1. Aprendizaje Automático

Machine Learning es la traducción en inglés de *Aprendizaje Automático* (AA), una rama de la IA en la que los sistemas informáticos utilizan ciertos algoritmos que les aportan la capacidad de aprender y adaptarse a nuevos datos sin necesidad de intervención humana [ML123]. Así, a partir de los datos de entrada, el sistema es capaz de dar una respuesta a problemas de clasificación, regresión, recomendaciones etc. El ML está estrechamente relacionado con la minería de datos ya que los algoritmos mejoran su rendimiento a medida que se entrenan con nuevos datos. También está relacionado con la estadística ya que se emplean modelos estadísticos y probabilísticos a la hora realizar predicciones.

Según la tarea que se quiera realizar y los datos disponibles para el entrenamiento de los modelos necesarios para cubrirla, los algoritmos de ML se dividen en tres tipos de aprendizaje:

- **Aprendizaje supervisado**: se le suministra al sistema datos ya clasificados por el ser humano. El algoritmo tratará de descubrir las relaciones entre los datos con el objetivo de obtener predicciones para nuevas entradas. Este tipo de aprendizaje se

utiliza para problemas de clasificación en los que se busca etiquetar con una clase entre varias clases limitadas y de regresión en los que el resultado es un número a predecir. Algunos ejemplos de algoritmos son árboles de decisión que son los que se van a utilizar en este trabajo y por ello se profundiza en ellos en la subsección 2.1.1, clasificador de Naïve Bayes, regresión logística, regresión polinomial, máquinas de soporte vectorial, KNN, Bosques Aleatorios o redes neuronales artificiales.

- **Aprendizaje no supervisado**: no hay información sobre la clase a la que pertenecen los ejemplos en los datos por lo que el objetivo es descubrir patrones en el conjunto de entrenamiento que permitan agrupar y diferenciar unos ejemplos de otros. La técnica más empleada es el *clustering* con diferentes algoritmos como *k-means* o *DBSCAN* en los que se emplean conceptos matemáticos como las p-distancias, vecindades de Voronói y probabilidad. En *k-means* se agrupan los datos en k vecindades eligiendo puntos centroides aleatorios y realizando vecindades de Voronói iterativamente hasta alcanzar un umbral de error establecido. *DBSCAN* se basa en la densidad de los elementos definida como la cantidad de puntos cercanos a uno mismo según la p-distancia elegida.

- **Aprendizaje por refuerzo**: el sistema recibe algún tipo de recompensa positiva o negativa cada vez que produce una respuesta, adaptando su comportamiento a partir de ella.

Independientemente del tipo de aprendizaje, uno de los retos en ML es la representación de los individuos o datos así como su preprocesamiento para que los algoritmos funcionen de manera óptima. Para ello hay diferentes técnicas y pasos a realizar antes de comenzar con el entrenamiento.

2.1.1. Árboles de decisión

Un árbol de decisión es un algoritmo de ML de aprendizaje supervisado que puede ser utilizado tanto para tareas de clasificación como para regresión. Su estructura es de árbol jerárquica, con su nodo raíz, ramas, nodos internos y hojas. Este permite el aprendizaje de conceptos de forma inductiva: a partir de un conjunto de entrenamiento, construye un árbol donde cada nodo pregunta por el valor de una variable o característica. Así, una vez construido el árbol, se pueden clasificar ejemplos no etiquetados respondiendo las preguntas de cada nodo y llegando hasta las hojas donde está la clasificación. El árbol se construye recursivamente hacia abajo mediante algoritmos como *ID3* que tratan de disminuir una función definida de entropía en los nodos hijos. Se emplean estrategias de divide y vencerás mediante la realización de una búsqueda voraz para identificar los puntos de división óptimos dentro de un árbol [DT23].

2.2. Aprendizaje Profundo

Machine Learning (ML) engloba muchos algoritmos y técnicas diferentes. Sin embargo, los métodos convencionales sufren limitaciones a la hora de procesar datos en bruto. El *Deep Learning* o *Aprendizaje Profundo* (AP) es una rama dentro de ML que trata de aprender la representación interna o características de los datos para procesarlos. Los métodos de *Deep Learning* (DL) tienen varios niveles de representación que se obtienen componiendo módulos simples no lineales que transforman la representacion en un nivel (comenzando por la entrada bruta) en una representación a un nivel superior ligeramente más abstracta. Componiendo suficientes transformaciones de esta forma se pueden aprender funciones muy complejas [LBH15]. La implementación de estos métodos se lleva a cabo mediante redes neuronales que pueden ser útiles tanto para aprendizaje supervisado como no supervisado. Por ende, es necesario comprender las redes neuronales para poder llevar a cabo un proyecto de DL.

A continuación se explican conceptos básicos para entender los tres principales tipos de redes neuronales.

2.2.1. Redes neuronales artificiales

Las Redes neuronales artificiales o *Artificial Neural Network* (ANN) en inglés están basadas en el funcionamiento de las neuronas en el cerebro humano y se emplean en aprendizaje supervisado [HGMH+00]. En este caso, el perceptrón simple es el elemento básico que va a servir para formar las diferentes capas de la red neuronal. Un ejemplo de red es el perceptrón multicapa. En la Figura 2.1 se observa un ejemplo de perceptrón simple que recibe un vector analógico de entrada. En él se asignan pesos a las diferentes variables del vector y se realiza una suma ponderada, se resta un umbral y se pasa el resultado a la función no lineal que se llama función de activación y que en este caso es de tipo escalón. Así el perceptrón divide el hiperplano en dos clasificando por clases los elementos dados.

Sin embargo, el perceptrón simple tiene ciertas limitaciones a la hora de clasificar ya que es posible que las clases no sean linealmente separables. Por ello el perceptrón multicapa combina perceptrones simples en diferentes capas y permite crear hiperplanos que al combinarse linealmente en la siguiente capa crean hipersuperficies no lineales que separan las clases. El perceptrón multicapa tiene que tener mínimo tres capas como se observa en la Figura 2.2, una capa de entrada con tantas neuronas como variables de entrada, una o varias capas ocultas con un número variable de neuronas y cada una de ellas con una función de activación y una capa de salida con tantas neuronas como salidas. Este se entrena con un algoritmo que minimiza el error para poder obtener buenos resultados de clasificación.

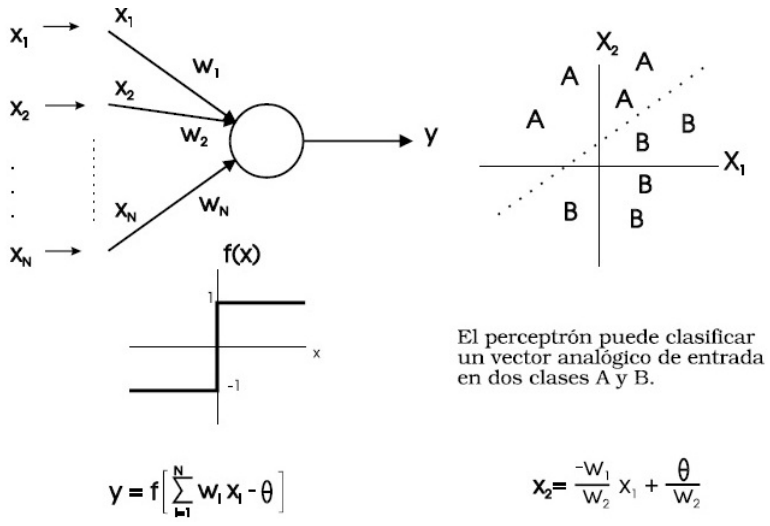

Figura 2.1: Perceptrón simple [HGMH+00].

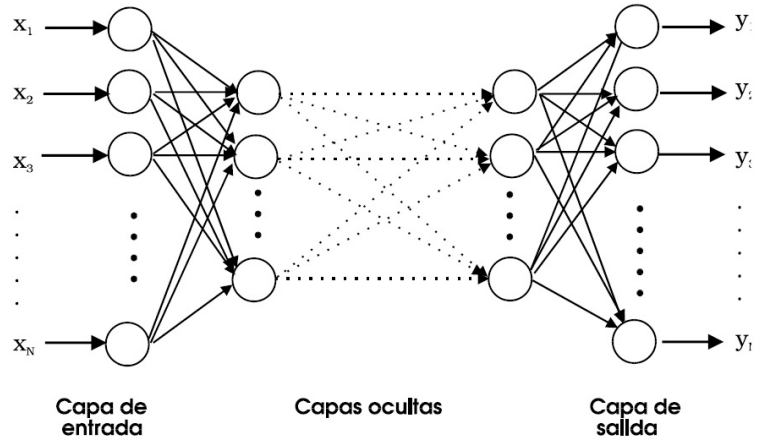

Figura 2.2: Perceptrón multicapa [HGMH+00].

2.2.2. Redes neuronales convolucionales

Las Redes neuronales convolucionales o *Convolutional Neural Networks* (CNN) en inglés son capaces de extraer las características de los datos mediante estructuras llamadas de convolución. Su arquitectura está basada en la percepción visual y no en las neuronas y sus núcleos representan diferentes receptores que pueden responder a diversas características. La función de activación simula que sólo las señales eléctricas neuronales que superan cierto umbral pueden transmitirse a la neurona siguiente y aprender lo que se quiere mediante una función de pérdida. Las ventajas de este tipo de redes neuronales son las conexiones locales, es decir, las neuronas no están conectadas a todas las neuronas de las capas anteriores, solo a algunas, también la compartición de pesos y sobre todo la reducción de la dimensionalidad, reduciendo la cantidad de datos o parámetros conservando en cada

capa la información útil.

En la Figura 2.3 se muestra el procedimiento básico de una red neuronal convolucional de dimensión dos. El paso fundamental a realizar es la convolución junto con la aplicación de una función de activación. Antes de ello se realiza el relleno o *padding* que se encarga de alargar la entrada con ceros en los bordes para no perder la información de los bordes al establecer un núcleo de convolución. El *stride* se emplea para controlar la densidad de convolución haciendo que cuanto más grande sea, menor será la convolución. El resultado obtenido es un mapa de características al que se aplica una puesta en común o *pooling* para eliminar redundancia [LLY+22].

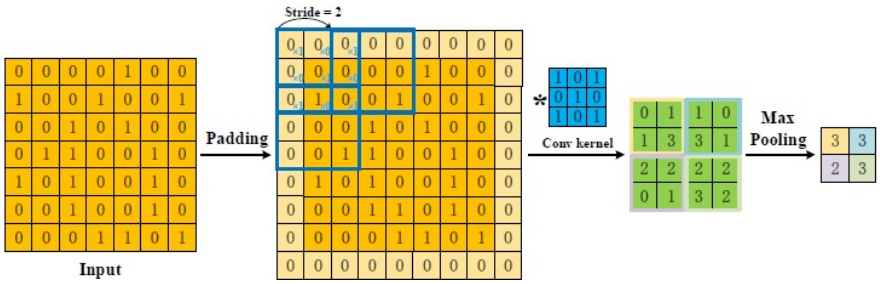

Figura 2.3: Procedimiento básico de una red neuronal convolucional [LLY+22].

Las redes neuronales convolucionales están formadas por varias capas. La capa de entrada, las capas ocultas que realizan el procedimiento explicado y las capas de salida que se pueden conectar a otros algoritmos de ML para interpretar las características extraídas o a otras funcionalidades dependiendo de la finalidad.

2.2.3. Redes neuronales recurrentes

Las Redes neuronales recurrentes son conocidas en inglés como *Recurrent Neural Networks* (RNN). Las ANN son muy útiles y según el número de capas ocultas en la red se pueden obtener representaciones buenas de las características de los datos. Sin embargo, no son buenas si se busca tratar datos secuencialmente o secuencias muy largas entre otras cosas. Las ANN con conexiones recurrentes a las se llaman RNN y son capaces de modelar datos secuenciales para el reconocimiento y predicción de secuencias. Están formadas por estados ocultos de dimensiones altas con dinámica no lineal. La estructura de esos estados ocultos funciona como la memoria de la red y el estado de la capa oculta en un momento dado está condicionado su estado anterior. Esta estructura permite a las RNN almacenar, recordar y procesar señales complejas pasadas durante largos periodos de tiempo. Las RNN pueden asignar una secuencia de entrada a la secuencia de salida actual y predecir la secuencia en el siguiente paso temporal [SBS+17]. En la Figura 2.4se observa un ejemplo simple de una RNN con tres capas.

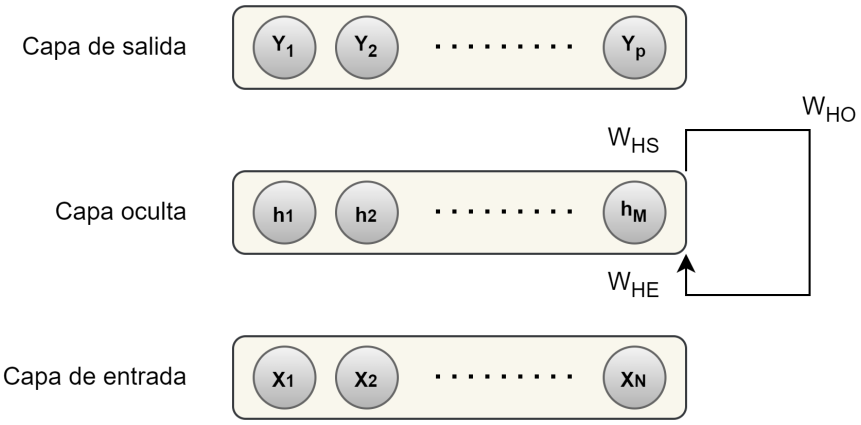

Figura 2.4: Red neuronal recurrente de tres capas [SBS+17].

2.3. Ciberseguridad

La ciberseguridad es el área relacionada con la informática y la telemática que se enfoca en la protección de los sistemas y dispositivos informáticos contra el acceso no autorizado, el robo de datos, la manipulación de la información y otros ataques cibernéticos. Es un tema crucial en la era digital actual, ya que la dependencia de la tecnología ha aumentado exponencialmente en todos los ámbitos de la sociedad, desde la industria y el comercio hasta la vida cotidiana.

Las amenazas son muchas, aunque a continuación se exponen algunas como ejemplo que pueden dañar un sistema informático:

- **Ataques de puerta trasera**: una puerta trasera en un sistema informático es cualquier método secreto para eludir la autenticación normal o los controles de seguridad. Un algoritmo o una red neuronal con una puerta trasera funcionan de forma correcta para la mayoría de entradas, para que sea más difícil de detectar, y se activa con una o varías entradas específicas. Pueden existir por muchas razones, incluido el diseño original o una mala configuración. Pueden haber sido agregados por una parte autorizada para permitir algún acceso legítimo, o por un atacante por motivos maliciosos, pero independientemente de los motivos de su existencia, crean una vulnerabilidad. Las puertas traseras pueden ser muy difíciles de detectar y, por lo general, las descubre alguien que tiene acceso al código fuente de la aplicación o un conocimiento íntimo del sistema operativo de la computadora.

- **Ataques de denegación de servicios**: los ataques de denegación de servicio, del inglés DoS, están diseñados para hacer que una máquina o recurso de red no esté disponible para sus usuarios previstos. Los atacantes pueden denegar el servicio a víctimas individuales o pueden sobrecargar las capacidades de una máquina o red y

bloquear a todos los usuarios a la vez. Se profundizará sobre ellos en la sección 2.6.

- **Troyanos**: Se hace referencia a un software malicioso que se hace pasar por un elemento valido del sistema, este software brinda a un atacante acceso a un equipo infectado para realizar extracción o suplantación de información, abrir puertas traseras, tomar control del equipo víctima, entre otras.

- **Virus Informáticos**: son las amenazas mas conocidas por el público, se trata de código con el que se infecta un programa ejecutable y que se propaga copiándose a si mismo afectando a otros programas, archivos o sistemas.

2.4. Seguridad en Inteligencia Artificial

La seguridad en la Inteligencia Artificial es un tema cada vez más importante y relevante en el mundo de la tecnología. A medida que la IA se vuelve más común y se integra en una amplia gama de sistemas y aplicaciones, la necesidad de asegurar que estos sistemas sean seguros y confiables se vuelve crítica.

Como se menciona en el Capítulo 1, la IA sufre vulnerabilidades principalmente ante amenazas adversarias que son capaces de confundir algoritmos y perturbar su funcionamiento. En el Capítulo 3 se realizará un estudio sobre el modelo de amenaza adversaria actual, explicando conceptos y posibles ataques que se realizan contra la IA.

Cabe mencionar también que una de las principales preocupaciones es la privacidad de los datos. A medida que se recopilan grandes cantidades de datos para entrenar los algoritmos de IA, existe el riesgo de que estos datos se utilicen de manera inapropiada o se compartan con terceros sin el consentimiento de los usuarios. Es importante implementar medidas de seguridad y privacidad efectivas para garantizar que los datos se manejen de manera responsable y segura.

2.5. Redes Generativas Adversarias

En esta sección se introduce un concepto clave en el trabajo que se va a utilizar para llevar a cabo las pruebas de ataques adversarios: las Redes Generativas Adversarias, también llamadas GAN y que fueron propuestas en 2014 por Ian Goodfellow y descritas en su trabajo [GPAM+20]. Estas consisten en un modelo formado por dos redes neuronales que conforman un juego de suma cero. Una de las redes tratará de discriminar un conjunto de datos mientras que la otra tiene el objetivo de aumentar todo lo posible el índice de error del discriminador. El objetivo final del modelo es ser capaz de generar muestras adversarias con las que se pueda engañar al discriminador.

Para una mejor comprensión del las GAN, en la Figura 2.5 se observa una red neuronal que recibe fotografías y decide para cada ejemplo si se trata de una foto de un gato o de otra cosa, este será nuestro discriminador. A la entrada y salida del discriminador se conecta una red generativa que introduzca alteraciones en las muestras tratando de producir un fallo en el resultado del discriminador.

Figura 2.5: Estructura básica de una GAN.

El resultado de este modelo desemboca en un ataque adversario, en caso de éxito el generador proporcionará muestras adversarias capaces de engañar al discriminador y por tanto a otros modelos de ML como un clasificador como se ve en la siguiente subsección con un ejemplo de ataque a un clasificador de malware.

2.5.1. Ataque Adversario a un Clasificador de Malware

Realizar un ataque adversario a un clasificador que trata un conjunto de datos de software añade la problemática de alterar un fichero ejecutable sin dejarlo inservible. Al momento de generar un ejemplo adversario nos interesa que la muestra siga siendo un software funcional además de engañar al clasificador.

La entrada del generador consta de un fichero de malware en forma de un vector característico m de tamaño M y de un vector de ruido z. El vector m es binario y cada dígito implica la existencia de una característica. z es un vector de tamaño Z donde Z es un hiperparámetro del modelo. La función de z es permitir al generador producir varios ejemplos adversarios para un solo vector característico.

El generador consiste en una red neuronal *feedforward* multicapa que recibe y devuelve un vector característico. La última capa de la red tiene M neuronas con la función de activación sigmoide, con esto se consigue que la salida de la red Θ sean M características con valores entre 0 y 1. Después se aplica una normalización binaria a la salida para obtener un vector de unos y ceros, a este vector se le llamara Θ'.

Para evitar dejar el malware inservible se añade alguna característica irrelevante, nunca desechar una característica que estuviese en el vector de entrada. Para este fin se aplica el operador OR lógico bit a bit, así solo se añadirán los unos de Θ' que no estén ya en m, al vector resultante de esta operación se le llamara m' ($m' = m|\Theta'$).

La naturaleza binaria del vector m' no permite la propagación hacia atrás del error hasta el

generador. Se define una función G que permita al generador acceder a la información del gradiente del detector sustituto en cada iteración del entrenamiento. Está función aplicará la función max bit a bit entre m y Θ, los bits a 1 de m se mantendrán intactos y el resto de bits se modificarán en función del gradiente.

$$G(m, z) = max(m, \Theta).$$

La función de perdida del generador será inversa a la del clasificador sustituto, de esta forma, a medida que el clasificador sustituto vaya aprendiendo a clonar al clasificador de caja negra, el generador heredará el conocimiento para generar vectores malignos que el clasificador de caja negra no pueda etiquetar como *Malware*.

2.6. Ataque de Denegación de Servicio

Un ataque DoS es un tipo de ataque cibernético cuyo objetivo es saturar un sistema, red o servicio en línea, con el fin de impedir el acceso o uso por parte de usuarios legítimos.

Durante un ataque DoS, los perpetradores emplean una red de múltiples dispositivos comprometidos para enviar una gran cantidad de tráfico falso o solicitudes maliciosas hacia el objetivo deseado. Estos dispositivos comprometidos pueden incluir computadoras, servidores u otros dispositivos conectados a Internet previamente infectados con malware y controlados por el atacante.

El objetivo del ataque consiste en sobrecargar el sistema objetivo con un volumen excesivo de tráfico o solicitudes, lo cual agota los recursos del sistema, tales como el ancho de banda de la red, la capacidad de procesamiento o las conexiones disponibles. Como resultado, el sistema objetivo se vuelve inaccesible o experimenta un rendimiento degradado de manera significativa, lo que impide a los usuarios legítimos utilizarlo normalmente.

2.7. Redes Generativas Adversarias Tabulares

Las GAN tabulares son un tipo de modelo adversario utilizado para generar datos tabulares de forma sintética. Este enfoque se basa en las GAN, las cuales están compuestas como sabemos por dos redes neuronales: el generador y el discriminador, que compiten entre sí.

En el contexto de datos tabulares, que son conjuntos de datos estructurados en forma de tablas con filas y columnas, las GAN tabulares se utilizan para crear nuevas muestras de datos similares al conjunto de datos original. Estas redes resultan especialmente útiles cuando se pretende mantener la estructura y las relaciones entre las diferentes características presentes en los datos tabulares. El generador en una GAN tabular recibe una entrada aleatoria llamada ruido latente y la transforma en una muestra de datos

tabulares sintéticos. Por otro lado, el discriminador evalúa si una muestra es real o sintética. Ambas redes se entrenan de manera adversaria: el generador intenta engañar al discriminador generando muestras más realistas, mientras que el discriminador busca distinguir entre muestras reales y generadas. Durante el proceso de entrenamiento, el generador y el discriminador se mejoran continuamente mientras compiten entre sí. Idealmente, al final del entrenamiento, el generador es capaz de generar muestras que resultan indistinguibles para el discriminador.

2.8. *Kafka*

Kafka es una biblioteca en *Python* que implementa un consumidor y productor para Apache Kafka, un sistema de mensajería distribuida. *Kafka* es una plataforma de transmisión de datos de alto rendimiento y baja latencia utilizada para enviar datos en tiempo real entre aplicaciones y sistemas distribuidos.

La biblioteca *Kafka* brinda una interfaz fácil de usar para interactuar con clústeres de *Kafka* desde *Python*. Permite a los desarrolladores escribir aplicaciones de consumidores y productores que envían y reciben mensajes a través de temas de *Kafka*. Algunas características destacadas de *Kafka* incluyen:

1. **Productor**: Permite crear un productor en *Python* para enviar mensajes a los temas de *Kafka*. Esto posibilita la publicación de datos en tiempo real para que otros consumidores los procesen.

2. **Consumidor**: *Kafka* también permite crear consumidores en *Python* que se suscriben a los temas de *Kafka* y reciben mensajes en tiempo real. Los consumidores pueden procesar y analizar los datos enviados por los productores.

2.9. Sistema de Detección de Intrusos basado en Aprendizaje Automático

Un IDS es un software de seguridad cuya función es detectar accesos no autorizados en un sistema o una red de ordenadores, y en base a ello, generar algún tipo de alerta o log para que posteriormente pueda ser gestionado por el administrador de sistemas correspondiente [IDS23]. Su objetivo principal por tanto es revisar y clasificar el tráfico de red como normal o malicioso. Así, ML es de gran utilidad para clasificar el tráfico basándose en las características del mismo [LSX22]. Los árboles de decisión son uno de los posibles algoritmos utilizados para implementarlos.

Capítulo 3

Estado del Arte

En la actualidad, hay diversos estudios sobre las múltiples amenazas adversarias y trabajos en los que se ponen a prueba y se analizan metodologías y objetivos muy diferentes. En la sección 3.1 se hablará del modelo de amenaza adversario actual analizando artículos en los que se exponen formas de crear ejemplos adversarios en contextos variados y dando una clasificación de los ataques adversarios según diferentes características. En la sección 3.2 se hablara sobre los ataques adversarios según el conocimiento del atacante enfocándose ya en el punto de vista de entrenar una GAN. A continuación se ven ataques de envenenamiento en la sección 3.3 y ataques de evasión en la sección 3.4.

3.1. Modelo de Amenaza Adversaria

Un componente clave del análisis de seguridad de los sistemas o componentes de un sistema es el modelado de amenazas. Los objetivos principales de este es establecer objetivos de seguridad mediante la identificación de amenazas y potenciales vulnerabilidades permitiendo desarrollar políticas para prevenir o minimizar el impacto de dichas amenazas en los sistemas. Esta etapa permite maximizar la seguridad de un sistema.

Los modelos de ML sufren vulnerabilidades ante ataques adversarios. En concreto, en el ámbito de seguridad de red se realizan muchos más ataques que en otros campos debido a la naturaleza de las aplicaciones propias de este campo como detección de malware, intrusos o filtrado de spam. En [IAKMS19] se presenta una taxonomía para generar ataques adversarios contra un modelo de ML implementado en plataformas de seguridad que supervisan el tráfico en la red. Además se presentan dos enfoques de clasificación para los ataques adversarios sugiriendo algunas defensas para estos ataques.

3.1.1. Ejemplos Adversarios

El primer estudio en el que se habla de conceptos relacionados con los ejemplos adversarios es [SZS+13]. En él se argumenta que una pequeña perturbación en la forma de una entrada cuidadosamente elaborada podía confundir a una red neuronal de DL. En ese artículo hacen pruebas con imágenes y hay muchos estudios que investigan ese campo.

Los ejemplos adversarios son el componente principal de los ataques adversarios y consiste en una entrada perturbada de un algoritmo de ML. Sea por ejemplo en un dataset concreto con características x y con etiquetas clasificadas y, se trata de obtener un ejemplo $\tilde{x} = x+\delta$, siendo δ la perturbación, que sea prácticamente indistinguible de x pero que sea clasificado con una etiqueta distinta a y. Si se llama c al clasificador, se busca una perturbación pequeña tal que $c(x) \neq c(\tilde{x})$. Estos ejemplos adversarios se crean usando métodos de optimización que buscan la mínima perturbación que maximice la función de pérdida de la red neuronal.

Como se ha mencionado, en este trabajo se van a generar ejemplos adversarios mediante el uso de una GAN. Sin embargo, hay muchos otros métodos interesantes para crearlos. A continuación se enumeran algunos de ellos. En [KGB16] se exponen varios métodos para generarlos enfocados a datasets de imágenes y añaden fórmulas matemáticas para explicarlo. En [IAKMS19] se enumeran con una explicación breve.

- **Método del Gradiente de Signo Rápido del inglés *Fast Gradient Sign Method (FGSM)***: es un método simple y computacionalmente eficiente comparado con otros métodos para generar ejemplos adversarios. Los ejemplos se crean calculando la pérdida de la propragación hacia delante, a continuación el gradiente respecto a la entrada y modificando los datos en la dirección de los gradientes que maximiza la pérdida [FGS23].
- **Método Iterativo básico o *Basic Iterative Method (BIM)***: realiza el cálculo del gradiente en multiples iteraciones. Esto resuelve el problema de otros métodos como el FGSM de asumir que pueden introducirse los ejemplos directamente en el modelo de aprendizaje automático. Esto dista mucho de ser práctico, ya que la mayoría de los atacantes a lo hacen a través de dispositivos como sensores. Hay variantes de este método escogiendo la clase menos probable predicha por la red para aplicar el BIM [BIM23].
- **Ataque basado en el Jacobiano y Mapa de Atención o *Jacobian-based Saliency Map Attack (JSMA)***: se usa una matriz jacobiana de la entrada para encontrar las características que más afectan a la salida. El ataque adversario se genera añadiendo pequeñas perturbaciones basándose en esas características.

También hay otros trabajos que se enfocan en generar ejemplos adversarios pero sobre

datos distintos a imágenes. En [ZDS17] se expone que las perturbaciones a menudo no son naturales, no tienen significado semántico y no son aplicables a dominios complicados como el lenguaje y se propone un marco para generar ejemplos contradictorios naturales y legibles que se encuentran en la variedad de datos. En [SM17] se trata la generación de ejemplos adversarios en textos con un algoritmo que modifica la entrada original (reemplazando, insertando y quitando palabras del texto).

3.1.2. Clasificación de Amenazas Adversarias

Dentro del modelo de amenaza adversaria, en [IAKMS19] se consideran diferentes tipos de ataque teniendo en cuenta el conocimiento del atacante, sus objetivos, el momento del ataque, la frecuencia con la que se actualiza u optimiza el ejemplo y la falsificación del ataque. Seguidamente se da una explicación breve de todo ello:

- **Conocimiento del atacante**: a la hora de atacar, no solo hay que conocer la función que cumple el modelo que se quiere atacar, su implementación o el tipo de datos de entrada sino que además se debe tener en cuenta el conocimiento que el atacante tiene sobre la arquitectura y la implementación del modelo. Se explica la clasificación más detalladamente en la sección 3.2 y enfocado a las GAN.

- *Objetivo del atacante*: ataques dirigidos o ataques de fiabilidad. Los primeros buscan inducir una predicción específica y definitiva en el modelo de ML mientras que los segundos buscan maximizar el error cometido en la predicción.

- **Momento del ataque**: ataques de evasión frente a ataques de envenenamiento. En la evasión, también conocidos como ataques exploratorios o ataques a tiempo de decisión, el atacante pretende confundir la decisión del modelo de aprendizaje automático después de que se haya aprendido como se muestra en la Figura 3.1. En la Sección 3.4 lo veremos con más detalle. Esto contrasta con los ataques de envenenamiento que implica corrupción adversaria de los datos de entrenamiento antes del entrenamiento para inducir una predicción errónea del modo de aprendizaje como se muestra en la Figura 3.2. Profundizaremos sobre ellos en la Sección 3.3.

- **Frecuencia**: ocasiones en la que el ejemplo adversario es actualizado u optimizado. En los *one-shot* se realiza una vez. En los iterativos se realiza varias veces, mejorando el rendimiento pero agrandando el coste.

- **Falsificación**: los falsos positivos hacen que se clasifique mal un ejemplo adversario negativo como uno positivo como por ejemplo un marware clasificado como benigno y un falso negativo al revés.

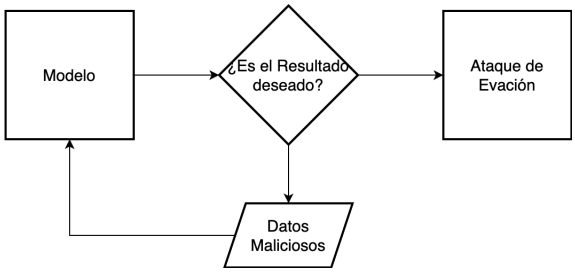

Figura 3.1: Ataque de evasión

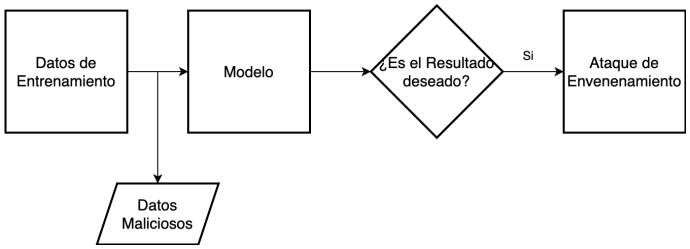

Figura 3.2: Ataque de envenenamiento

3.2. Clasificación de Ataques Según el Conocimiento del Atacante

A partir de ahora se pone el foco en las redes GAN que se introdujeron en el Capítulo 2 y se va a desarrollar la clasificación de los ataques adversarios según el conocimiento del atacante. A parte de la función que cumple el modelo que se quiere atacar, su implementación o el tipo de datos de entrada, se debe tener en cuenta el conocimiento que un atacante tiene sobre la arquitectura y la implementación del modelo. El generador necesita obtener conocimiento del modelo que se está atacando para producir los ejemplos adversarios. Esta puede ser una tarea se complica a medida que se tiene menos información sobre la arquitectura, los parámetros de entrenamiento y los pesos resultantes del modelo atacado. En este paradigma se pueden dividir los ataques adversarios en tres categorías principales: los ataques de caja negra, de caja gris y de caja blanca.

3.2.1. Ataques de caja blanca

En este tipo de ataque, el perpetrador tiene un conocimiento completo del modelo de destino, incluidos los datos de entrenamiento, la estructura y los parámetros. Para lograr la falla del modelo, un atacante debe estudiar su comportamiento y comprender las vulnerabilidades del modelo, por ejemplo, qué datos de entrada producen una salida falsa. Conociendo toda esta información es mucho más sencillo decidir la implementación para el generador y la transferencia de conocimiento [ML222] desde el clasificador.

Este tipo de ataques es ampliamente utilizado. En [WLC+21b] propusieron un ataque

adversario de caja blanca que permite realizar ataques de modelos de ML con perturbaciones sutiles y se rompen modelos de entrenamiento adversario TRADES [ZYJ+19] con la tasa de éxito más alta. Los ataques generados logran reducir la precisión de los modelos entre un 16 % y un 31 % en los ataques de transferencia de caja negra. Además, en Wang et al. [WLC+21a] se propone un método para reducir la cantidad de información innecesaria para la generación de ejemplos negativos integrados (IWA) basados en gradientes: Algoritmo de ataque de punto finito integrado (IFPA) y Algoritmo de ataque de universo integrado (IUA). IFPA se aplica a un número predeterminado de puntos de perturbación e IUA se aplica a un número indeterminado de puntos de perturbación para un ejemplo más controvertido. Los experimentos pueden verificar que se pueden generar ejemplos contradictorios con menos interferencia y mejor velocidad de generación en todo tipo de conjuntos de datos.

3.2.2. Ataques de caja negra

A diferencia de las taxonomías anteriores, los ataques de caja negra no requieren conocimiento del modelo, su configuración o las tecnologías que lo soportan [YHZL19]. La única información que se tiene del modelo atacado es la salida que produce para una entrada dada. El modelo de red adversaria que mejores resultados ha dado en este tipo de ataques es la estructura MalGAN [HT17b] que vemos en la Figura 3.3. El primer paso en la implementación de la red MalGAN será construir un clasificador sustituto, este clasificador será entrenado con muestras de un dataset previamente etiquetado por el clasificador de caja negra. El objetivo es lograr que el clasificador sustituto consiga clonar los resultados de la caja negra. Cuando se logre esto se puede realizar la transferencia de conocimiento al generador utilizando la misma estructura de red GAN que se ha visto en el resto de ejemplos.

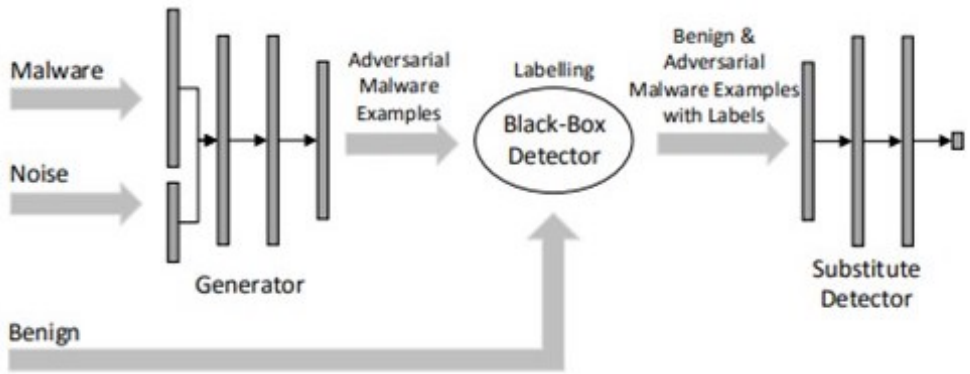

Figura 3.3: Estructura MalGAN [HT17b]

En el trabajo [LSX22] se utiliza también una GAN para realizaar un ataque de caja negra a un IDS. Hoy en día, algunas soluciones permiten el uso de algoritmos de IA

para realizar tareas críticas, como detectar malware en una organización. Este tipo de soluciones son las más vulnerables, por lo que las muestras de malware modificadas se consideran muestras válidas. En [HT17a], se propone un algoritmo para generar ejemplos secuenciales antagónicos para atacar sistemas basados en RNN, la red generativa se basa en un modelo secuencia por secuencia, y el RNN proxy correspondiente al RNN de la víctima se entrena utilizando Gumbel-Softmax sobre las muestras aproximadas generadas. Por lo tanto, la mayoría de las muestras maliciosas generadas por recomendaciones no pueden ser detectadas por algoritmos basados en RNN.

Además, Zhao et al. [GZL+21] propuso ataques de caja negra en algoritmos de detección de tráfico de red basados en ML sospechosos. Esta propuesta, al igual que las anteriores, se basa en la comprensión de patrones en la generación de imágenes e IDS al extender el algoritmo BIM con un método de reemplazo de patrones aprendido objetivamente utilizando los conjuntos de datos KDD99 y CSE-CIC-IDS2018. Las muestras generadas por la herramienta emiten detección de evasión de patrón objetivo.

3.2.3. Ataques de caja gris

A diferencia de los dos anteriores, los ataques de caja gris entrenan un modelo generativo con la intención de generar muestras adversarias asumiendo que se tiene acceso al modelo atacado tan solo durante la fase de entrenamiento [XZJL21]. La metodología de entrenamiento es similar a la de los ataques de caja negra pero, al tener acceso al modelo atacado durante el entrenamiento no es necesario mandarle tantas consultas.

La ventajas del ataque de caja gris incluyen: mayor eficiencia de tiempo, la no asunción del conocimiento del modelo atacado durante la fase de ataque y son más fáciles de integrar en la defensa de ataques adversarios.

3.3. Ataques de Envenenamiento

Los ataques de envenenamiento en el aprendizaje automático buscan manipular el conjunto de entrenamiento de manera que el modelo resultante pierda precisión. Estos ataques tienen dos objetivos principales que son degradar la precisión del modelo y crear una puerta trasera en el modelo.

Hay dos maneras de envenenar el conjunto de entrenamiento:

- La primera consiste en etiquetar mal ciertos datos para que el modelo aprenda información errónea, lo que a su vez afecta la frontera de decisión y produce predicciones incorrectas.
- La segunda es mediante la creación de datos confusos, donde el atacante crea datos

con características especiales que el modelo aprende a explotar, lo que puede permitir al atacante acceder al modelo de forma no autorizada.

3.3.1. Ejemplo de un Ataque de Envenenamiento

Un ejemplo de uso del ataque de envenenamiento a los datos de un aprendizaje colaborativo utilizando una GAN que aparece en [ZCW+19] se expone a continuación.

3.3.2. Funcionamiento del Aprendizaje Colaborativo

Para entender este ejemplo de ataque primero es necesario saber en que consiste el aprendizaje colaborativo. Sin intercambiar explícitamente muestras de datos, el aprendizaje federado intenta entrenar un algoritmo de AA, como las *Redes Neuronales Profundas* (RNP), en numerosos conjuntos de datos locales presentes en nodos locales. En el aprendizaje colaborativo, los usuarios solo cargan gradientes generados por los datos y modelos de entrenamiento locales, lo que reduce el riesgo de filtración de la privacidad. Además, en el aprendizaje colaborativo, los modelos globales en el lado del servidor comparten la misma estructura con los modelos locales del cliente, lo que permite que los usuarios conserven el control sobre sus datos. En cada iteración, los usuarios descargarán los parámetros y el modelo global del lado del servidor y luego entrenarán el modelo con los conjuntos de datos locales en cada cliente. Los gradientes cargados se promediarán y acumularán en el modelo global actual, lo que permitirá que el modelo mejore con cada iteración.

3.3.3. Ejecución del Ataque

El atacante tiene como objetivo corromper el modelo global en la fase de entrenamiento haciéndose pasar por un participante benigno y engañar al modelo global para clasificar incorrectamente las entradas correctas como la etiqueta incorrecta objetivo en la etapa de inferencia.

El proceso para envenenar los datos por parte de un atacante es el siguiente, utilizando una arquitectura GAN en el procedimiento de aprendizaje colaborativo, el atacante genera muestras similares a las de otros participantes benignos y las inserta en los conjuntos de datos locales de entrenamiento. Estas muestras están envenenadas y se les ha asignado la etiqueta incorrecta. El modelo local se entrena con estos datos envenenados, lo que da lugar a una actualización local envenenada. Después de que se hace el promedio del modelo en el lado del servidor, los resultados de predicción en algunas entradas de los participantes benignos cambian a las clases elegidas por el atacante.

Para llevar a cabo este ataque, se deben seguir los siguientes pasos:

- El protocolo de aprendizaje federado se ejecuta durante numerosas rondas para actualizar el modelo global hasta que la precisión alcanza un nivel determinado, suponiendo que hay dos participantes en el sistema de aprendizaje federado, P y A, con conjuntos de datos de entrenamiento privados separados (clase a y b).
- Los parámetros del modelo local del participante P se sustituyen por los del modelo global, que se descarga. Una vez entrenado el modelo en el conjunto de datos local, se publica una actualización local en el servidor principal.
- El atacante A descarga el modelo global para sustituir los parámetros del modelo local. Cuando D *(discriminador)* y G *(generador)* se aplican a D para simular muestras dirigidas a la clase b del participante P, el modelo local resultante se duplica. Las muestras de clase a creadas se clasifican entonces como muestras de clase b, y las muestras de datos envenenados se añaden al conjunto de datos local A. Basándose en el conjunto de datos combinado, se entrena el modelo local y se produce un modelo local envenenado actualizado.

Los pasos 2 y 3 se repiten hasta que el modelo global converge, en el paso 4.

3.4. Ataques de Evasión

Los ataques de evasión son métodos para engañar o manipular los sistemas de AA, como las redes neuronales, cambiando los datos de entrada para producir resultados no deseados o evitar ser detectados. Estos ataques aprovechan los defectos y deficiencias de la capacidad de los algoritmos de aprendizaje automático para extrapolar patrones a partir de los datos de entrenamiento.

En un ataque de evasión, un oponente, también conocido como adversario, altera a propósito los datos de entrada con pequeñas perturbaciones en un intento de engañar al modelo para que proporcione resultados falsos o no deseados. Aunque estas perturbaciones pueden no ser perceptibles para los humanos, son lo suficientemente significativas como para cambiar la clasificación o el comportamiento del modelo. Los ataques que eluden los sistemas de detección de malware, evitan la tecnología de reconocimiento facial o engañan a los sistemas de recomendación en línea pueden tener graves repercusiones.

Un ejemplo básico de ataque de evasión sería la generación de imágenes adversarias utilizando una GAN. En este caso, el atacante entrena una GAN para generar imágenes que sean similares a los datos de entrada originales, pero con pequeñas modificaciones diseñadas para engañar al modelo objetivo. Estas modificaciones pueden ser casi imperceptibles para los humanos, pero pueden ser suficientes para confundir al modelo entrenado.

Capítulo 4

Diseño e Implementación

En este capítulo se trata de mostrar el proceso que se ha seguido para desarrollar el proyecto con el fin de explicar todos los elementos esenciales que lo forman. Como se menciona en el Capítulo 1, se decidió que se llevaría a cabo la implementación de una GAN para atacar de forma adversaria un algoritmo de ML. En este trabajo se realizó la implementación de un IDS mediante árboles de decisión para poder poner a prueba la GAN en un contexto en el que el algoritmo de ML se encuentra en un sistema muy utilizado hoy en día. Los IDS son sistemas fundamentales en cualquier empresa. Este detecta gran parte de los ataques por lo que un mal funcionamiento podría tener consecuencias catastróficas. Para comenzar, en la sección 4.1 se describe el conjunto de datos y su preprocesamiento para las posteriores implementaciones y en la 4.2 se muestra un diagrama del esquema general de nuestra implementación y cómo colaboran los componentes. Se recuerda que en la sección 2.9 se explicó qué es un IDS y en la 2.1.1 cómo funcionan los árboles de decisión. Se va a explicar la implementación del IDS basado en árboles de decisión en la sección 4.3 y a continuación, en la sección 4.4, la metodología seguida para implementar la GAN así como el algoritmo de entrenamiento de la misma. En la sección 4.5 se explica como también se trató de implementar una GAN tabular utilizando diferentes métodos y basándose en distintas referencias para atacar el IDS mediante un ataque degenerado de denegación de servicio.

4.1. Conjunto de datos y Preprocesamiento

En esta sección se va a describir qué conjunto de datos se ha utilizado en este trabajo así como todos los pasos que se han llevado a cabo para procesarlo de manera que se pueda usar para el objetivo final y se optimicen los resultados. Esta parte se llevó a cabo en un *notebook* de la herramienta *Google Colab* que permite programar y ejecutar *Python* en el navegador sin necesidad de configuraciones previas, con acceso a *GPUs* de *Google* y con

gran facilidad de colaboración.

El conjunto de datos que se utiliza para poner a prueba el IDS es el conocido como *NLS-KDD*. Se trata de el conjunto de datos de referencia para evaluar IDS a día de hoy. Este comprende un conjunto de entrenamiento *KDDTrain+* y un conjunto de prueba *KDDTest+*. Los datos son extraídos de un entorno de red real y contienen tráfico normal y cuatro categorías principales de tráfico malicioso que incluyen *Probing (Probe), Denial of Service (DoS), User to Root (U2R) y Root to Local (R2L).* Los registros de tráfico en NSL-KDD se extraen en las secuencias de características, como la descripción abstracta del tráfico de red normal y malicioso. Cada elemento de la secuencia representa una característica del tráfico. Esta información más ampliada así como el descargable del conjunto de datos se encuentra en [NLS23].

Todo el preprocesamiento se va a realizar tanto en el conjunto de entrenamiento como en el de prueba. Lo primero que se hace en el *notebook* es observar cómo es el dataset. Para el tratamiento de datos se van a utilizar las librerías *Pandas y Numpy*. Se añaden los nombres de las columnas/características y se observa la distribución de los ataques para conocer mejor cómo está distribuido el conjunto de datos. A continuación se buscan las variables categóricas entre las que se encuentra la variable objetivo que se ha llamado *attack*. Estas es necesario convertirlas en numéricas para poder aplicar los algoritmos. Para ello se utilizará *LabelEncoder* y a continuación se utilizará *OneHotEncoding* que nos aportan esta funcionalidad. El método de *LabelEncoder* transforma las variables categóricas en enteros asociando uno a cada etiqueta y tiene la ventaja de ser sencillo de implementar. Sin embargo, tiene un inconveniente: algunos algoritmos pueden malinterpretar los valores numéricos. Por ejemplo, si se codifican varias etiquetas con los valores 0, 1, 2 y 3, ¿significa eso que la etiqueta que recibe el valor 3 es, según algunas normas, tres veces mayor que la etiqueta que recibe el valor 1? Naturalmente, la respuesta es no. Por ello se utiliza el *OneHotEncoding* que soluciona este problema creando una columna para cada valor distinto que exista en la variable categórica que se está codificando y, para cada registro, marcar con un 1 la columna a la que pertenezca dicho registro y dejar las demás con 0.

Ahora ya se puede observar la correlación entre las variables con el objetivo de eliminar las más correladas. Esto se hace ya que pueden perjudicar el correcto funcionamiento de los algoritmos de ML. La correlación de las principales características se observan en el mapa de calor de la Figura 4.1 creado con la librería *seaborn*.

Se localizan las variables con más de un 0.95 de correlación y se eliminan: *num_root, srv_error_rate, srv_rerror_rate, dst_host_serror_rate, dst_host_srv_serror_rate, dst_host_srv_rerror_rate y flag_s0.*

Antes de entrenar el IDS, es importante dividir el conjunto de datos (únicamente el de entrenamiento) para entrenar IDS y GAN para evitar el sobreajuste de modelos con el objetivo de poder realizar el ataque que se busca con la GAN.

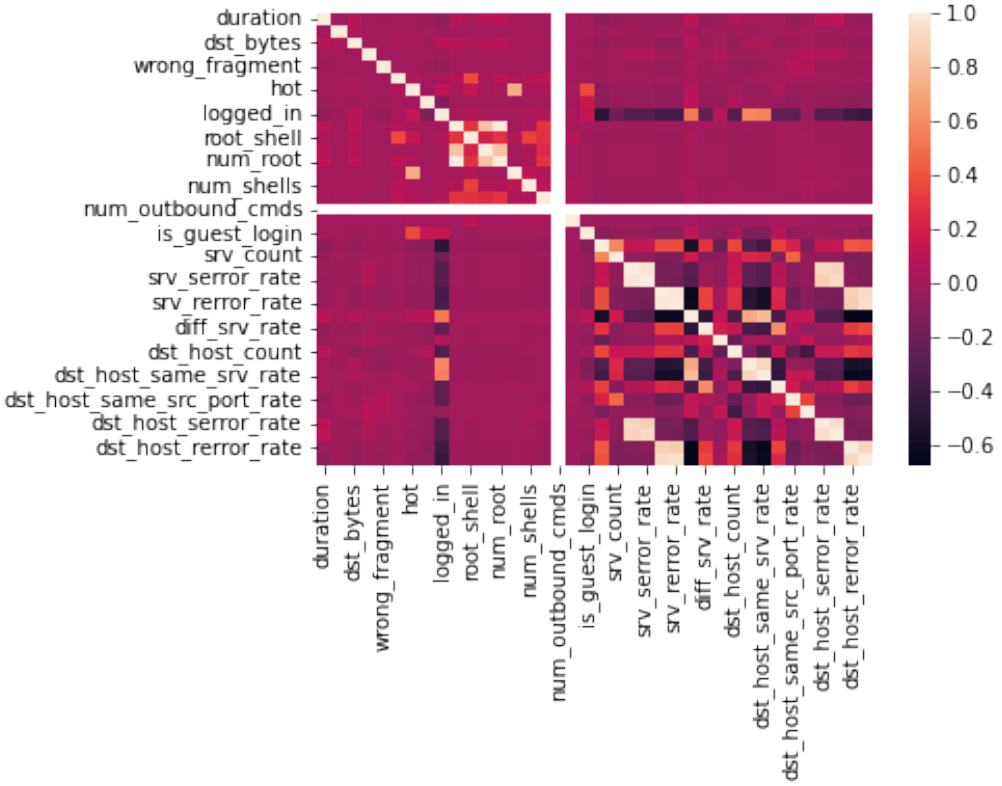

Figura 4.1: Mapa de calor de correlación entre las principales características del conjunto de datos.

A continuación se etiqueta el tráfico de la siguiente manera en la característica *attack: normal(0), DoS(1), Probe(2), R2L(3) y U2R(4).*

Por último, se separa el conjunto de datos en X e Y siendo X el dataframe con las características e Y el dataframe con las salidas. También se realizó un proceso de normalización de las entradas de datos. El motivo de la normalización fue el hecho de que las características que se miden en diferentes escalas no contribuyen por igual al entrenamiento del modelo y son propensas a crear sesgos. Para hacer frente a este problema potencial, se utilizó la normalización de características mediante *MinMaxScaling*, antes de ajustar el modelo. Al hacerlo, todas las características se transformaron en el rango $[0, 1]$.

4.2. Componentes del Sistema

En el diagrama que se ve en la Figura 4.2 se observan todos los componentes de la implementación y cómo interactuan entre ellos. El ataque a realizar va a ser de caja negra, es decir, el modelo de la GAN va a tener como única información del IDS su salida que se va a usar para entrenar el discriminador. El generador recibe datos maliciosos modificados con ruido y va a generar tráfico adversario. En ese momento el IDS realiza

una predicción tanto para el tráfico adversario como para tráfico normal generando las salidas que el discriminador va a usar para aprender los parámetros del IDS. En todo momento, el discriminador y el generador se van entrenando utilizando las funciones de pérdida.

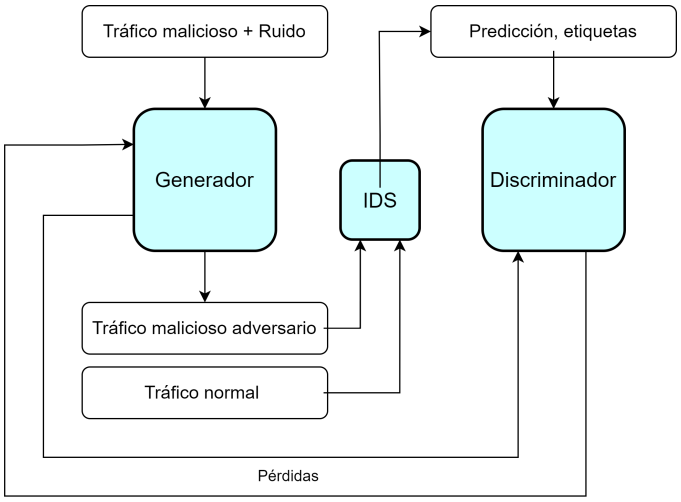

Figura 4.2: Estructura con los componentes del sistema.

4.3. Implementación del Sistema de Detección de Intrusos y metodología

Se procede ahora a desarrollar cómo se llevó a cabo la implementación del IDS basado en árboles de decisión. Ya se ha introducido el conjunto de datos empleado y descrito su preprocesamiento. A continuación, se detallará la implementación del sistema en sí, Esta parte del trabajo se llevó a cabo también en un *notebook* de la herramienta *Google Colab*.

4.3.1. Entrenamiento

El procedimiento para el entrenamiento del clasificador es el que se describe a continuación. En él se ha tratado de dar con una parametrización que aprenda "lo justo", sin quedarse corto, ni sobreaprender. El algoritmo de ML utilizado es como se ha adelantado árboles de decisión y se utilizará el módulo *DecisionTreeClassifier* de la librería *sklearn* para implementarlo.

Se realizaron pruebas con distintos árboles variando el número mínimo de ejemplos para dividir un nodo (parámetro *min_samples_split*) y usando validación cruzada en 10 partes. La validación cruzada en k partes es una forma de estimar la precisión del modelo dividiendo el conjunto de datos en k subconjuntos iguales, entrenando el modelo con $k-1$

y validando con el restante usando en este caso la medida de exactitud que se explicará en el Capítulo 5. Esto se repite k veces cambiando el conjunto de datos de validación.

En la gráfica de la Figura 4.3 se muestra el resultado. Para *min_samples_split* = 4 se obtiene el mejor valor de exactitud para el conjunto de test. A partir de ese valor la métrica empeora para el conjunto de test. Si se reduce este valor la métrica mejora para el conjunto de entrenamiento y empeora para el conjunto de test lo que indica que hay sobreaprendizaje, generalizando peor.

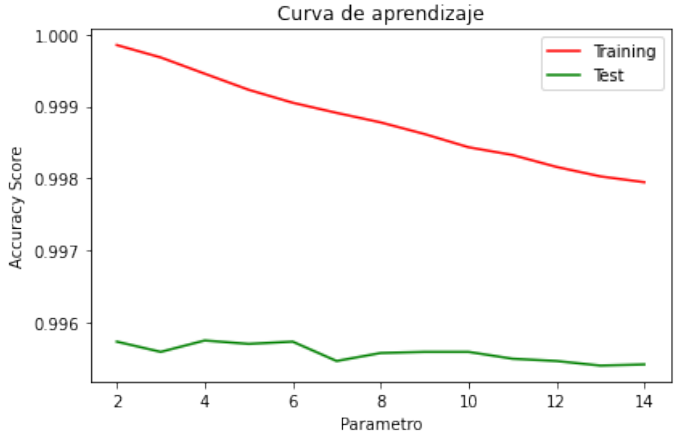

Figura 4.3: Curva de aprendizaje del árbol variando el parámetro *min_samples_split*.

Entrenando el árbol con ese parámetro optimizado y mostrando un máximo de profundidad de 1, se obtiene el árbol de la Figura 4.4. En él se observa que la primera característica por la que se pregunta es *same_srv_rate*. Esta es por tanto la característica con más importancia a la hora de clasificar los ataques.

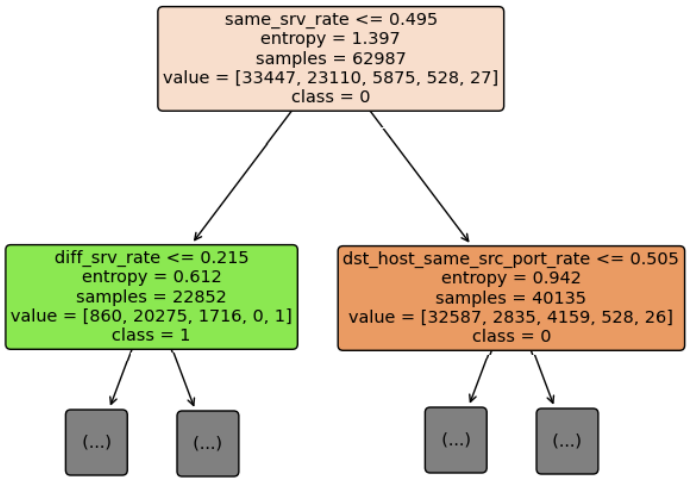

Figura 4.4: Árbol de decisión del IDS.

Se tiene por tanto ya el IDS entrenado y listo para recibir nuevos ejemplos y clasificarlos. En el Capítulo 5 de experimentos y resultados se analiza el error del IDS según las métricas habituales.

4.4. Red Generativa Adversaria para Atacar un Sistema de Detección de Intrusos

En esta sección se describe la implementación de la GAN con el objetivo de atacar el IDS de la sección anterior. El ataque va a ser de caja negra, es decir, la GAN va a tener como única información sobre el IDS su salida y va a tratar de aprender de ella. El trabajo se llevó a cabo de nuevo en otro *notebook* de *Google Colab*. En él, se importa la parte del conjunto de datos que corresponde a la GAN y que previamente se había guardado en un archivo *.csv* y el modelo ya entrenado de IDS mediante la funcionalidad de la librería *pickle*.

4.4.1. Preprocesamiento del Conjunto de Datos

Como se mostró anteriormente se realizó la división del conjunto de datos de entrenamiento extraído de *NLS-KDD* una vez codificado las variables categóricas y eliminado las más correladas. Ahora, como entrenar una GAN es un proceso largo y costoso se dividirá el conjunto de datos en subconjuntos según el tipo de ataque y otro conjunto para tráfico benigno. A continuación se separan los subconjuntos en otros diferentes X_tipo_ataque e Y_tipo_ataque y también otro conjunto para el tráfico normal de la misma forma que se mostró para el clasificador. En los experimentos se realizará el entrenamiento de la red GAN únicamente con los datos de ataques DoS y se usarán también el tráfico normal para realizar la predicción con el IDS y comparar con la predicción de los datos adversarios. Se realizó la normalización y algunos ajustes para que el tamaño de los conjuntos de datos sea el óptimo para el entrenamiento de la GAN.

4.4.2. Desarrollo de la Red Generativa Adversaria

La estructura de la red GAN propuesta y su entrenamiento se presentan en las siguientes secciones.

4.4.2.1. Estructura

Como se sabe, la implementación de la GAN incluye un generador y un discriminador. El generador es responsable de generar los ejemplos adversarios a partir de vectores de ruido y de los datos en sí. En este caso, se ha definido una función llamada *build_generator*

que construye un generador básico con tres capas densas y dos capas de normalización por lotes empleando las funcionalidades *Dense* y *BatchNormalization* de la librería *Keras*. Este generador toma como entrada un tensor de forma $(124,)$ y produce un tensor de salida de forma $(115,)$ que es el tamaño que deben de tener los datos ya que el tráfico tiene 115 características en el conjunto de datos procesado.

La primera capa del generador es una capa densa con 124 unidades de entrada y una función de activación lineal rectificada (ReLU) con una pendiente negativa de 0.2. A continuación, se agrega una capa de normalización por lotes con un factor de momento de 0.8 para mejorar la estabilidad y la velocidad de convergencia de la red. Posteriormente, se agregan dos capas densas más, cada una con una función de activación ReLU con una pendiente negativa de 0.2 y una capa de normalización por lotes con el mismo factor de momento que la capa anterior. Finalmente, se agrega una última capa de activación ReLU con una pendiente negativa de 0.2.

En el Código 4.1 se observa el código de la función *build_generator*. Para el discriminador es una función similar.

```
def build_generator():
    model = keras.Sequential()
    model.add(keras.layers.Dense(124, input_shape=(124,)))
    model.add(keras.layers.BatchNormalization(momentum = 0.8))
    model.add(keras.layers.LeakyReLU(0.2))

    model.add(keras.layers.Dense(119))
    model.add(keras.layers.BatchNormalization(momentum = 0.8))
    model.add(keras.layers.LeakyReLU(0.2))

    model.add(keras.layers.Dense(115))
    model.add(keras.layers.LeakyReLU(0.2))

    model.summary()
return model
```

Código 4.1: Código de *build_generator*.

A continuación se realiza la configuración del discriminador que es el responsable de distinguir entre tráfico real y falso. La función *build_discriminator* define la arquitectura básica del discriminador, que incluye cuatro capas densas, tres capas de normalización por lotes y una capa de activación *softmax*.

El discriminador toma como entrada un tensor de forma $(115,)$, que representa una instancia de tráfico, y produce una salida unidimensional que representa la probabilidad de que la imagen sea real o sintética. Por tanto, la primera capa es una capa densa con 115 unidades de entrada y una función de activación ReLU, seguida de una capa de

normalización por lotes con un factor de momento de 0.8. A continuación, se agrega una capa de activación ReLU con una pendiente negativa de 0.2 y otra capa de normalización por lotes con el mismo factor de momento que la capa anterior. Luego se agregan dos capas densas más, cada una con una función de activación ReLU y una capa de normalización por lotes con el mismo factor de momento. Finalmente, se agrega una capa densa con una función de activación *softmax*, que se utiliza para producir una salida unidimensional que representa la probabilidad de que la imagen sea real o no.

Para ambos modelos se utiliza una llamada a la función *summary* que imprime una representación gráfica del modelo y una descripción de sus parámetros. Esto permite tener una visión general de la arquitectura del generador y discriminador y de sus valores de los parámetros utilizados.

Para crear ambos componentes de la GAN se utiliza la función *createIDSGANDOScomponents* que devuelve el generador, el discriminador y el modelo GAN completo. El discriminador se compila con una la función de pérdida *sparse_categorical_crossentropy*, el optimizador *sgd* y la métrica de precisión. El discriminador se congela en este punto, lo que significa que sus pesos no se actualizarán durante el entrenamiento.

El modelo GAN se crea combinando el generador y el discriminador.Este tiene como entrada el tensor de ruido y como salida la clasificación del tráfico generado como real o falso por el discriminador. El modelo se compila con la misma función de pérdida, otimizador métrica que el discriminador.

4.4.2.2. Entrenamiento

Para entrenar la GAN se utiliza el IDS entrenado. El objetivo es minimizar la pérdida del generador mientras se maximiza la del discriminador. Para ello, se generarán ejemplos adversarios con ruido en el generador, se clasifican junto a los datos reales en el IDS, y se entrena el discriminador teniendo en cuenta esos datos para que aprenda a distinguir entre real y falso.

Se realiza la definición de dos funciones auxiliares. La función *getAdversarialSample* recibe como entrada los datos originales de ataque, genera un vector de ruido uniformemente distribuido con valores entre 0 y 1 de forma $(9,)$ y los concatena. Se observa su código en el Código 4.2. Se realiza esto para cada instancia de los datos de ataque. Por otro lado, la función *getBatch* se encarga de obtener un lote de datos a partir de un conjunto de datos, el tamaño del lote y el momento del algoritmo en el que se encuentra. Se realiza el entrenamiento por lotes ya que se ha descubierto que el modelo se entrena más rápido y se requiere menos memoria.

```
1 def getAdversarialSample(X_attack):
```

```
    batch = []
    for sample in X_attack:
        noise = np.random.uniform(0,1,(9,))
        batch.append(np.concatenate((sample,noise)))
    return np.array(batch)
```

Código 4.2: Código de *getAdversarialSample*.

Se define la función que va a llevar a cabo el entrenamiento en sí de la red. Esta es *trainIDSGAN* y recibe el generador, el discriminador, el modelo GAN, el modelo del IDS, los datos de entrenamiento de ataque y normales y los pasos que se dan en cada época del bucle del algoritmo. Estos pasos se calculan teniendo en cuenta el tamaño elegido de los lotes. El entrenamiento consiste en un bucle por épocas con otro bucle interno de los pasos por cada lote. En cada uno de ellos se realizan las siguientes operaciones que se enumeran a continuación y que se observa el código que lo implementa en el Código 4.3:

1. Se obtiene el lote de datos de ataque correspondiente al paso actual mediante el uso de *getBatch*.
2. Se construye la muestra adversaria del lote mediante el uso de *getAdversarialSample*.
3. Se utiliza la función *predict* del generador para generar el tráfico malicioso adversario.
4. Se recuperan de nuevo los datos reales del ataque mediante *getBatch*.
5. Se realiza la clasificación de los datos reales y los falsos utilizando el *predict* del IDS.
6. Se marca el discriminador como entrenable y lo se entrena basandose en los resultados del IDS.
7. Se actualizan los parámetros del discriminador y se marca una vez mas como no entrenable.
8. Es genera de la muestra adversaria y se entrena el generador el generador a través del modelo GAN.
9. Y por ultimo se actualizan los parámetros del generador.

```
def trainIDSGAN(generator,discriminator,gan,ids_model,
X_train_attack,X_traffic_train,steps_per_epoch):
    for epoch in range(epochs):
        for step in range(steps_per_epoch):

        X_attack = getBatch(batch_size,step,X_train_attack)
        adversarial_batch = getAdversarialSample(X_attack)
        X_fake = generator.predict(adversarial_batch).round(decimals = 4)
```

```
9
10          X_real = getBatch(batch_size,step,X_traffic_train)
11
12          d_target_real = ids_model.predict(X_real)
13          d_target_fake = ids_model.predict(X_fake)
14
15          discriminator.trainable = True
16
17          d_loss_real = discriminator.train_on_batch(X_real, d_target_real)
18          d_loss_fake = discriminator.train_on_batch(X_fake, d_target_fake)
19          d_loss = 0.5*np.add(d_loss_real, d_loss_fake)
20
21          discriminator.trainable = False
22
23          adversarial_batch = getAdversarialSample(X_attack=X_attack)
24          g_loss = gan.train_on_batch(adversarial_batch, normal_values)
25
26      print(f'Epoch: {epoch} \t Discriminator Loss: {d_loss}
27      \t Generator Loss: {g_loss}')
```

Código 4.3: Código de *trainIDSGAN*.

El entrenamiento se llevó a cabo con 15 épocas y tamaño de lote 38 para que el tamaño de los datos fuese divisible por ese número.

4.5. Síntesis de Ataques Distribuidos de Denegación de Servicio Utilizando Redes Adversarias Generativas Tabulares

Con el objetivo de conseguir engañar a una red neuronal que detecta ataques DoS de los que se habló en la sección 2.6, se decidió realizar ataques de DDoS usando una GAN tabular en el IDS. Para realizar estos se utilizarán varios conceptos que vienen ya explicados y junto con el código en [HHAE22] y que se introducen en las secciones 2.7 y 2.8.

4.5.1. Ejecución del código

Para empezar a ejecutar este proyecto primero se instala una distribución de *Linux* en el ordenador y se realiza la configuración base del entorno. Después se realiza la instalación de *Conda* ya que es uno de los requisitos necesarios para la ejecución de *Makefile*, dentro del cual hay varias opciones, a nosotros nos van a interesar dos una que crea el entorno de ejecución en conda y otra que instala una lista de requerimientos. Se realiza la ejecución del entorno porque para lanzar el proyecto y después se instalan los requerimientos dentro del entorno. Con esto terminado se instala *Kafka* para ello hay que instalar *Docker* y ejecutar

el *Script* que viene en el proyecto para instalar *Kafka*. En este punto se puede ejecutar el *attack* y el *consume* para generar el tráfico de red con la GAN tabular y también si se ejecuta el se puede observar que esta prediciendo y ver si es posible confundirla o no.

Capítulo 5

Experimentos y Resultados

En este capítulo se realizará el análisis de los experimentos realizados a lo largo del trabajo y los resultados. Para comenzar, analizaremos en la sección 5.1 el error del IDS entrenado en el Capítulo 4 mediante validación cruzada utilizando diferentes métricas. En la siguiente sección 5.2 se discutirán los resultados obtenidos al utilizar la GAN implementada.

5.1. Análisis del Error cometido por el Sistema de Detección de Intrusos

Para medir el error en problemas de clasificación, se deben comparar los resultados obtenidos por el modelo propuesto con los resultados observados en el conjunto de datos. Para ello se utilizan diferentes métricas de las cuales se calcularan dos. Para explicarlas se supone que el clasificador es binario, y se intuye que clasifica un dato de tráfico o como normal o como ataque. Se llaman Verdaderos Positivos (VP) a los datos normales clasificados correctamente como la normales. Los Verdaderos Negativos (VN) son los clasificados como ataque y que son ataque. Así, se llaman Falsos Positivos (FP) a los clasificados como normales siendo ataques y Falsos Negativos (FN) a los clasificados como ataque siendo normales. La métricas son las siguientes:

- Exhaustividad o *recall* en inglés. Es la Tasa de Verdaderos Positivos (TVP):

$$Exhaustividad = \frac{VP}{VP + FN} = \frac{VP}{P} \tag{5.1}$$

- Exactitud o *accuracy* en inglés. Es una medida buena muy utilizada:

$$Exactitud = \frac{VP + VN}{VP + VN + FP + FN} = \frac{VP}{P} \tag{5.2}$$

- Precisión o Valor Predictivo Positivo (VPP):

$$Precisión = \frac{VP}{VP + FP} \tag{5.3}$$

- Medida F1. Realiza una media armónica entre la precisión y la exhaustividad

$$F1 = 2 * \frac{VPP * TVP}{VPP + TVP} = \frac{2 * VP}{2 * VP + FP + FN} \tag{5.4}$$

Utilizando la librería *metrics* de *sklearn* se puede calcular las diferentes métricas de error de el clasificador utilizando validación cruzada. Realizando validación cruzada en 10 partes variando la profundidad del árbol y se puede observar que el error disminuía al aumentar la profundidad de forma muy exagerada hasta profundidad 8. Esto es lógico ya que a más profundidad del árbol, más preguntas se hacen y más características de los datos se tienen en cuenta pudiendo clasificar mejor. Calculando las métricas con profundidad del árbol 8, de forma óptima como se muestra en el Capítulo 4 y realizando validación cruzada en 10 partes se obtuvieron los siguientes medias ponderadas de las métricas redondeando a cuatro valores decimales:

```
scoring_metrics = ['precision_weighted', 'recall_weighted','f1_weighted']

clf_opt = DecisionTreeClassifier(criterion="entropy", max_depth=8,
        min_samples_split=4, random_state=333)


scores = cross_validate(clf_opt,X_train, Y_train,
        scoring=scoring_metrics, cv=10, return_train_score=False)

print('PPM:',np.mean(scores['test_precision_weighted']))
print('EPM:',np.mean(scores['test_recall_weighted']))
print('F1 PM:',np.mean(scores['test_f1_weighted']))
```

Código 5.1: Extraccion de Métricas de Error

- Precisión media ponderada: 0,9879.
- Exhaustividad media ponderada: 0,9882.
- F1 media ponderada: 0.9880.

En todas las métricas se obtiene un error muy pequeño lo que significa que el clasificador etiqueta correctamente la mayor parte del tráfico confundiéndose de forma muy rara habiendo sido entrenado con el conjunto de datos de prueba.

5.2. Análisis de Resultados de la Red Generativa Adversaria

El entrenamiento de la GAN se llevó a cabo para generar ejemplos adversarios únicamente en el caso de ataques DoS para simplificar ya que el tiempo de entrenamiento es muy largo. A continuación se obtiene la exactitud al hacer *predict* con el IDS utilizando el conjunto de datos de prueba original o pasado por la GAN entrenada para obtener un conjunto de datos de prueba adversarios. Con esta métrica es posible calcular el número de aciertos del modelo respecto a la cantidad total. También se utiliza *confusion_matrix y ConfusionMatrixDisplay* para crear matrices de confusión y poder visualizar mejor los resultados.

En la matriz de la Figura 5.1 se muestran los resultados de hacer *predict* con el IDS con el conjunto de datos de prueba para el ataque DoS etiquetados como 1. El trafico benigno está etiquetado como 0. La métrica de exactitud es del 73.59 %, que no es muy alta pero se observa que la mayoría de los fallos son Falsos Positivos, es decir, tráfico benigno clasificado como malo, lo cual no supondría riesgo a la seguridad. El modelo acierta 12636 de 17171 predicciones en el conjunto de prueba.

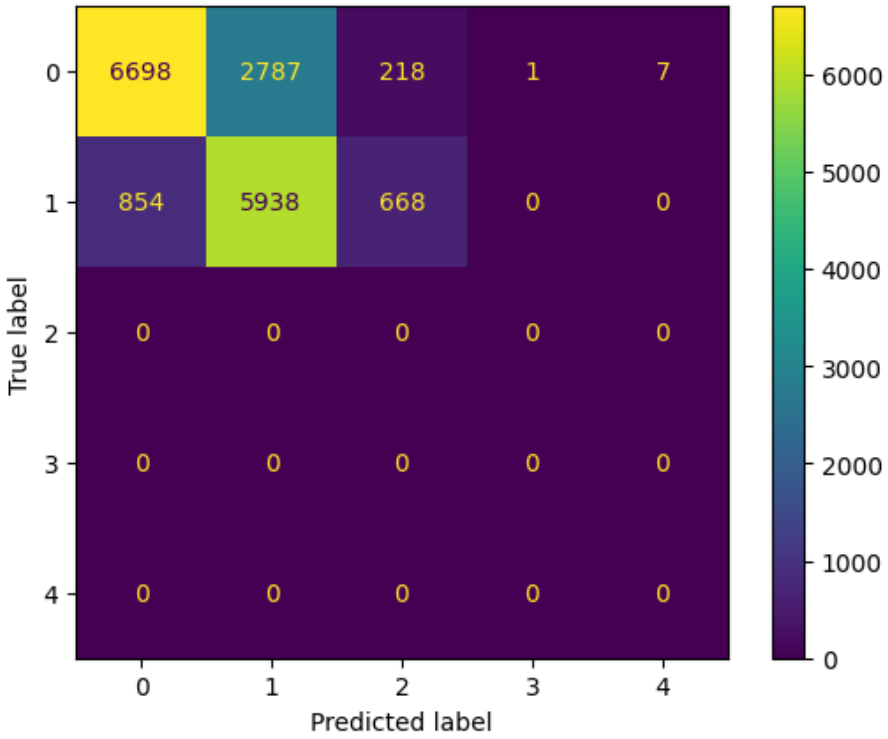

Figura 5.1: Matriz de confusión sobre el conjunto de prueba.

En la Figura 5.2 se muestra la matriz de confusión tras hacer *predict* con los datos generados por la GAN. La exactitud ha bajado a un 55.84 % por lo que ya se puede deducir que la GAN ha surtido efecto al hacer fallar al IDS. Si se observa la matriz, 7404

datos de tráfico que son DoS los clasifica como benigno, es decir, se obtienen muchos Falsos Negativos por lo que el ataque podría generar fallos importantes en la seguridad al haber hecho que el IDS no haya sido capaz de detectar el tráfico malicioso. El modelo acertó 9589 de 17171 predicciones en el conjunto de prueba pero como se comentó, la mayoría es trafico benigno. Simplemente acierta 7 instancias de tráfico maligno. En la Tabla 5.1 se muestra una comparativa resumida de todo lo anterior.

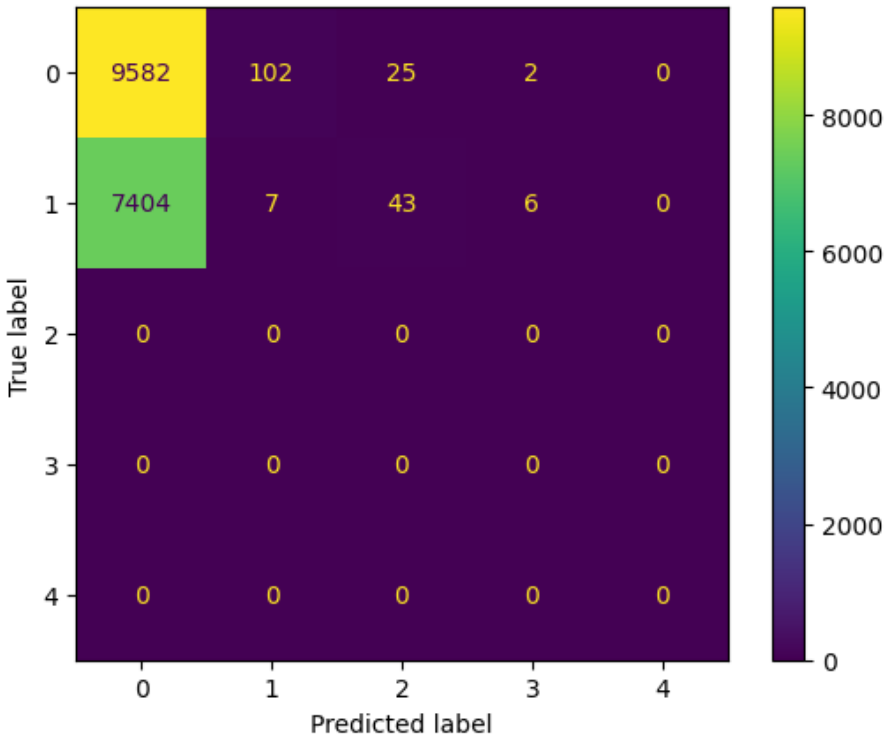

Figura 5.2: Matriz de confusión sobre el conjunto de prueba adversario.

Tabla 5.1: Tabla comparativa entre predicciones del IDS entre datos originales y modificados por la GAN para el ataque DoS.

	Datos originales	Datos modificados
Exactitud	73.59 %	55.84 %
Aciertos Totales	12636	9589
Aciertos DoS	5938	7
Falsos Positivos	2787	102
Falsos Negativos	854	7404

Capítulo 6

Contribuciones

Antes de pasar a los experimentos y resultados de nuestro trabajo, se van a mostrar en este capítulo las aportaciones que se han realizado cada uno de los miembros del equipo al mismo. El capítulo está dividido en tres secciones que corresponden a los tres integrantes.

6.1. Aitor Esteban Núñez

Tras la reunión inicial en la que se habló de las bases del trabajo y de la metodología que se iba a seguir durante el curso, investigué sobre *DevSecOps* leyendo diferentes referencias y páginas de empresas que llevan a cabo este marco de trabajo en constante evolución. Estudié todas las ventajas que tiene así como los retos que se plantean. También me informé sobre la seguridad de la información aprendiendo los conceptos más importantes sobre *InfoSec* y ciberseguridad. A continuación realicé dos cursos en *Coursera* para recordar la programación en *Python* y los conceptos de *Machine Learning y Deep Learning*. Al mismo tiempo realicé un repaso del temario de la asignatura de IA que tuve durante el curso anterior.

En este momento y ya habiendo entrado de lleno en el contexto de investigación, me centré en las GAN así como en algunas mejoras de las mismas, leyendo numerosos artículos que explicaban su estructura y su funcionamiento. Uno de ellos hablaba sobre sus implicaciones en la seguridad ya que las GAN aportan grandes resultados tanto en defensa como en ataque.

Las GAN son útiles para realizar ataques adversarios ya que generan ejemplos adversarios. Sin embargo, quise informarme sobre el modelo de amenaza adversaria actual, informándome sobre las diferentes formas de generar ejemplos adversarios en diferentes contextos como por ejemplo en textos o en imágenes y los tipos de ataques que se pueden realizar. En concreto, analicé diferentes artículos que hablaban sobre las vulnerabilidades de ML ante estos ataques. También investigué que se pueden inyectar ejemplos adversarios

en los conjuntos de entrenamiento de modelos para mejorar su rendimiento.

Con todo este conocimiento decidí investigar a fondo los ataques a algoritmos de ML utilizando redes GAN. En concreto, me llamó la atención un estudio en el que atacaban a un IDS y me centré en ello. Leí artículos sobre ataques de caja negra a IDS, mejoras de IDS mediante la inyección de ejemplos adversarios en el conjunto de entrenamiento, ataques de denegación de servicio adversarios, etc. Durante esta parte, traté de observar qué conjuntos de datos empleaban en cada trabajo, leyendo sobre cada uno de ellos con el objetivo de seleccionar el que mejor cuadrase para nuestro proyecto.

El proyecto pasó a la fase de desarrollo pero mi investigación no cesó en ningún momento ante la necesidad de seguir informandome sobre estos ataques. Sin embargo mi atención se centró en revisar código de diferentes repositorios y *notebooks* para entender ahora la implementación de las redes GAN y de los ataques realizados con ellas así como las librerías utilizadas. Acudí varias veces a la documentación de ciertas de ellas para saber los diferentes métodos que se pueden emplear y cómo.

A continuación comencé a realizar pruebas de código en un *notebook* de *Google Colab*. Lo primero que hice fue elegir el conjunto de datos *NLS-KDD* para llevar a cabo una implementación de IDS y posteriormente de una GAN que utilice el IDS para entrenarse y atacarlo.

LLevé a cabo el estuduio del conjunto de datos así como su preprocesamiento. Realicé una implementación de un IDS basado en ML y decidí usar árboles de decisión como ejemplo. Tras obtener una buena implementación y haber analizado su error cometido, me puse con la implementación de la GAN. Tras varios intentos y días de depuración en el que el código no funcionaba ya que había que realizar algunos cambios en los datos y elegir bien las capas de los componentes de la GAN, obtuve una implementación que funcionaba y la puse a entrenar en mi ordenador durante 3 horas. Tras ese tiempo pude realizar experimentos con el IDS y observar y analizar los resultados.

Mientras se llevaba a cabo el desarrollo, comenzó también la redacción de la memoria del trabajo y gran parte del tiempo empleado a partir de ese momento fue dedicada a realizar una redacción clara y buena.

6.2. Roberto Portillo Torres

El primer paso de este proyecto fue revisar todos los conceptos de *Machine Learning* y *Deep Learning* para ello completé un curso online en el que se revisaban los conceptos a través de ejemplos prácticos en *Google Collab*. Con estos conocimientos asentados, se inicia con la fase de investigación, la cual ha consumido la mayor parte del tiempo de proyecto.

Lo primero fue investigar sobre redes y ataques adversarios, se realizó la revisión numerosos

artículos en los que se proponían formas de aplicar las GAN para distintos ataques. El tutor nos facilitó la web *Papers With Code* que nos daba acceso a implementaciones de los artículos, esto nos sirvió para familiarizarnos con algunas *Application Programming Interface* (API) como *Kafka* o *Pandas*.

Algunas de estas implementaciones estaban en formato *Jupyter Notebook* pero otras estaban desarrolladas en código *Python*. Muchos de los repositorios a los que se tuvieron acceso estaban formados por ficheros y scripts diseñados para Linux. Se ejecutan estos scripts desde una máquina virtual pero esto nos trajo un montón de problemas, tanto de rendimiento como a la hora de instalar dependencias. Por ello opté por instalar un sistema operativo basado en linux (Ubuntu) de forma nativa, nos sirvió para solventar muchos errores y se pudieron ejecutar varios modelos de GAN.

También le dediqué un tiempo a investigar sobre redes de aprendizaje federado y como realizar ataques adversarios a este tipo de redes, sin embargo, la naturaleza descentralizada de estas redes nos iba a complicar conseguir una implementación. Por este motivo se descartó esta línea de investigación y se enfoca la investigación en los ataques adversarios a los clasificadores de *Malware*.

Durante todo este proceso se realizó una reunión semanal para la que se preparaban unos resúmenes de los artículos que se habían leído durante la semana. Nuestro tutor, Luis, nos daba indicaciones de en que línea se debía seguir la investigación y que cosas se podían descartar. Muchos de los resúmenes realizados para estas reuniones nos han servido ahora para redactar esta memoria.

A la hora de ponerme a redactar este documento también ha sido necesario repasar algunos conceptos de LaTeX, la herramienta que se han utilizado utilizado para redactar y maquetar la memoria.

6.3. Marcos Matute Fernández

En la primera reunión se nos contó sobre que iba a ir el proyecto, desde ese momento estuve haciendo cursos sobre *Python* para ponerme al día ya que se decidió realizar el desarrollo en este lenguaje de programación, en este momento de aprendizaje todos exponían sus dudas en reuniones periódicas que se organizaban los jueves.

Finalizados los cursos sobre *Python*, se comenzaron los cursos de seguridad, que personalmente me resultaron más difíciles ya que durante la carrera no he visto nada sobre esto. En las reuniones de los jueves también se comentaban las dudas sobre esto.

Con los cursos ya terminados, se inció con la búsqueda de información sobre el proyecto. Se inició buscando cosas referentes a las GAN ya que nuestro trabajo lo tiene como base. Después cada uno comenzó a buscar información sobre para que se podía utilizar una

GAN, en mi caso encontré información sobre los ataques de envenenamiento que usaban GAN y fue lo que más me gusto. Busqué bastante información sobre esto y también estuve probando código de diferentes repositorios y notebooks pero el código no me convenció.

Finalmente encontré información sobre los IDS con GAN y su respectivo código y encontré por fin algo que me gustaba. Empecé la etapa de desarrollo finaliza con lo que se explica en la sección 4.5.1 pero hasta llegar a ese punto fue todo a base de prueba y error, por ejemplo uno de los fallos se tuvieron fue ejecutar el código en una máquina virtual en vez de en un ordenador con *Linux*, otro fue que no se sabían los requerimientos de *Docker* para *kafka* y así poco a poco a base de prueba y error y fueron resueltos conforme avanzaba el proyecto. Mientras realizaba el desarrollo en ningún momento pare de seguir buscando información y también de escribir la memoria.

Capítulo 7

Conclusiones y Trabajo Futuro

7.1. Conclusiones

La Inteligencia Artificial tiene a día de hoy una gran importancia en prácticamente todos los ámbitos de la vida. Las empresas utilizan algoritmos de IA en infinidad de cosas pero uno de los usos más importantes en la implementación de los Sistemas de Detección de intrusos o IDS. Estos son una parte importante de la seguridad en muchos aspectos ya que tinene la función de alertar en caso de percibir tráfico sospechoso en la red. Es por ello que un fallo en ellos puede traer graves consecuencias como por ejemplo, ser víctima de un ataque de Denegación de Servicio.

En este trabajo se han estudiado las vulnerabilidades de los algoritmos de *Machine Learning* dentro de la IA. Para ello, se ha llevado a cabo una investigación de los diferentes ataques adversarios y amenazas de la IA. En concreto, se ha investigado sobre las redes GAN y establecido el objetivo de implementar y experimentar con una. Se decidió implementar un IDS tanto por su importancia como porque es un ejemplo de uso de ML en la vida real y al que se puede poner a prueba usando ataques adversarios. Por ello, se ha estudiado y procesado el conjunto de datos de tráfico de red más utilizado que es *NLS-KDD*, preparándolo para un correcto entrenamiento del IDS y de la GAN. Se ha implementado un IDS utilizando árboles de decisión como ejemplo de algoritmo de ML aunque se podría haber utilizado otro algoritmo e investigar sobre las diferencias en los resultados y cuales sufren más o menos a el ataque realizado. Se ha implementado una red GAN y entrenado para engañar al IDS modificando datos maliciosos de tráfico y utilizando las predicciones del IDS en el discriminador tratándolo como un ataque de caja negra en el que esas predicciones es lo único que se sabe del IDS como atacantes. En concreto, se ha conseguido hacer pasar tráfico de un ataque DoS como tráfico benigno obteniendo resultados muy significativos en cuanto a que con los datos modificados, el IDS es prácticamente incapaz de detectar el tráfico relacionado con DoS.

Sin embargo, se ha realizado el proyecto con unos parámetros muy concretos y para un ataque específico, además de que hay muchas otras formas de generar ejemplos adversarios como se ha visto. Es por ello que nuestro trabajo podría tener varias mejoras que requerirían de mejores medios y más tiempo para realizarlas pero que se reservan como posible trabajo futuro en la siguiente sección.

7.2. Trabajo Futuro

Como posibles estudios futuros pueden señalarse los siguientes:

- **Analizar los resultados con otros tipos de ataques u otros conjuntos de datos**: la red GAN de este trabajo fue entrenada con datos de tráfico normal y del ataque DoS. Un avance interesante sería entrenarla con otro tipo de ataques como por ejemplo los que conforman el propio conjunto de datos usado en este trabajo (*User to Root, Root to Local y Probing*).
- **Utilizar más épocas en el entrenamiento de la red**: en el caso de tener más tiempo o un ordenador con más recursos, se podría entrenar la misma red GAN utilizando un número mayor de épocas. Esto debería minimizaar en cada época aún más la pérdida del generador y maximizar la del discriminador que es el objetivo del entrenamiento de la GAN. Seguramente se obtendrían mejores resultados.
- **Probar nuestra red con otros algorimos de *Machine Learning* y con otros conjuntos de datos**: en este trabajo se ha atacado a un IDS basado en árboles de decisión pero se podría probar el rendimiento ante otros algoritmos y realizar una comparativa tanto del rendimiento del IDS como de la GAN a la hora de engañarlo. Además, se podría probar el funcionamiento utilizando otros conjuntos de datos, tanto de tráfico como de cualquier otra cosa.
- **Introducir mejoras en la red GAN**: se podrían utilizar otros parámetros o incluso otras estructuras en las capas del generador y del discriminador. Cambiar las librerias utilizadas para comparar cuales obtienen mejores resultados sería otra posibilidad. También se podría experimentar con mejoras de las propias GAN ya estudiadas como por ejemplo las *Wassertein GAN*.
- Entrenar el algoritmo de ML con los datos adversarios: se ha probado que mejora el rendimiento del algoritmo ante ataques adversarios si se entrena. Se podría por tanto intentar entrenar nuestro IDS con los datos falsos para que aprendiese a distinguirlos.

Capítulo 8

Introduction

8.1. Motivation

Nowadays , *Artificial Intelligence* (AI) is one of the fields of computer science with the most development and investment globally. Advances are enabling the creation of a multitude of useful tools to humans in many different contexts. These include uses of *Artificial Intelligence* (AI) for security in other fields such as malware classification or malicious network traffic detecttion. However, AI also has vulnerabilities and can be subject to attacks that disrupt its proper functioning. Adversarial attacks are a major concern when it comes to AI security, as they can fool Machine Learning algorithms in different ways. Ian Goodfellow proposed Generative Adversarial Networks (GAN) in 2014 in his paper [GPAM+20]. These are the most studied and used way to carry out adversarial attacks. In this work we will focus on the study of vulnerabilities and security of Machine Learning algorithms by investigating and then experimenting with attacks that are the order of the day in the ever-growing world of AI. For this purpose, we will implement a GAN network that we will use to perform an adversarial attack against a ML algorithm.

8.2. Context

This Final Degree Project is part of a research project entitled Platform for Analysis of Resilient and Secure Software - LAZARUS, approved by the European Commission within the Horizon Framework Programme (HORIZON-CL3-2021-CS-01) under grant agreement number 101070303 and in which the GASS Group of the Universidad Complutense de Madrid is taking part (Grupo de Analisis, Seguridad y Sistemas, https://gass. ucm.es, group 910623 of the catalogue of research groups recognised by the UCM). In addition to the Complutense University of Madrid, the following entities participate in LAZARUS: Athena Research Center - ARC (Greece), The University

of Padua (Italy), Infotrend Innovations Company Limited (Cyprus), Data Centric Services SRL (Romania), Luxembourg Institute of Science and Technology (Luxembourg), MotivianEOOD (Bulgaria), Binare Oy (Finland), APWG European Union Foundation (Spain), Maggioli Spa (Italia).

You can find more information here:

https://cordis.europa.eu/project/id/101070303

https://lazarus-he.eu

8.3. Object of the Investigation

The aim of our work is to carry out a study and research on AI security, focusing on adversarial attacks and more specifically on Generative Adversarial Networks (GAN). After having a solid base of knowledge on the subject, we will try to design and implement a GAN that is able to fool some ML algorithm. The aim is to obtain an efficient implementation and to test it in different ways.

8.4. Workplan

The development of this work has been carried out in three phases which are described bewow and they can be seen in Figure 8.1:

1. **Research**:

   We started the work doing a general meeting in order to explain the starting point of the work, the objectives were achieve and the knowledge that would be needed. After this, weekly meetings were arranged to monitor the progress of the research and to resolve any doubts that might arise. Another reason for arranging these meetings was to explain basic concepts about the different fields covered by this work, which will be discussed in the following sections. This phase started in September 2022. During the first four months, a period of adaptation to the work context and acquisition of the necessary knowledge to start the further development was carried out. For this purpose, we took courses on diferents fields and we read basic articles. Each member of the group made summaries and presentations of the articles they researched each week in order to gradually build up a common knowledge repository of the work on Google Drive and to progress together towards a common goal. By the last month of research, the team had a strong enough foundation to start thinking about revising code and thinking about the development stage. The research itself continued to be carried out steadily by reading articles with linked code to benchmark and observe implementations before starting our own.

2. **Development**:

   Once we had enough knowledge to start creating a solid version of the proyect, we decided to use less time on research and to begin the code of our proposal. As we mentioned before, researhc did not stop, it just took a less importance, but we still searched information about concepts that we needed during the developement. Therefore, during this phase, advanced concepts of the Python programming language and libraries necessary for the implementation were investigated. Besides, different proposals on Generative Adversarial Networks were studied. During this phase, different datasets were processed to better understand them and the necessary functionalities to carry out the project were implemented. It was in this phase when we decided that we would use our implementation of the GAN to attack an Intrusion Detection System (IDS). This was implemented using a Machine Learning algorithm and processing a specific dataset.

3. **Experimentation**: Once prototypes of the initial idea began to be produced, the experimentation process began, in which the team analysed the results obtained and drew conclusions. It should be noted that the development phase did not cease during the experimentation phase, as the model was constantly compared with others and modified for optimisation.

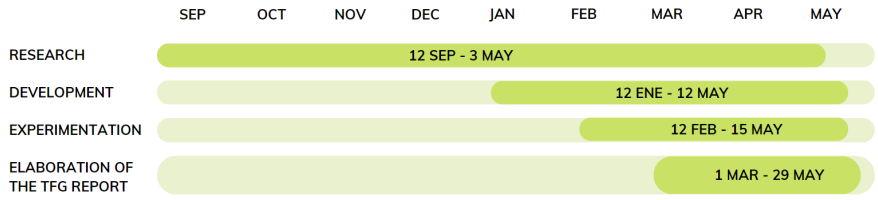

Figura 8.1: Project Gantt Chart.

8.5. Struture of the Work

The rest of the work is organised in 6 chapters in Spanish and 2 in English with the following structure:

Chapter 2 introduces some concepts that are elementary to understand the development of the work. We introduce the basics of Machine Learning (ML), Deep Learning (DL), security and the main object of our work: GANs.

Chapter 3 is a compilation of works related to the subject of our study, focusing on the different existing adversarial threats and giving examples of articles that carry out adversarial attacks.

Chapter 4 develops the methodology of the work, explaining the phases of the

implementations carried out in our work and how they were done.

Chapter 6 presents the personal contributions of each member of the group in this work.

Chapter 5 describes the experiments carried out to evaluate the effectiveness of the GAN implemented and developed in Chapter 4 and presents the results obtained.

Chapter 7 shows the main conclusions of this work and the future lines of research.

Chapter 9 is the English translations of Conclusions.

Capítulo 9

Conclusions and Future Work

9.1. Conclusions

Artificial Intelligence is nowadays of great importance in almost all areas of life. Companies use AI algorithms for a multitude of things, but one of the most important uses is in the implementation of Intrusion Detection Systems or IDS. IDSs are an important part of security in many respects as they have the function of alerting in case they perceive suspicious traffic on the network. That is why a failure in them can lead to serious consequences such as, for example, falling victim to a Denial of Service attack.

This paper has studied the vulnerabilities of Machine Learning algorithms within AI. For this purpose, an investigation of the different adversarial attacks and AI threats has been carried out. In particular, it has investigated GANs and set the goal of implementing and experimenting with one. It was decided to implement an IDS both because of its importance and because it is an example of real-life use of ML that can be tested using adversarial attacks. Therefore, the most widely used network traffic dataset, NLS-KDD, has been studied and processed, preparing it for a correct training of the IDS and the GAN. An IDS has been implemented using decision trees as an example of an ML algorithm, although another algorithm could have been used and the differences in the results and which ones suffer more or less from the attack could have been investigated. A GAN has been implemented and trained to fool the IDS by modifying malicious traffic data and using the IDS predictions in the discriminator treating it as a black-box attack where those predictions are the only thing we know about the IDS as attackers. In particular, it has been possible to pass off DoS attack traffic as benign traffic with very significant results in that with the modified data, the IDS is practically unable to detect DoS-related traffic.

However, we have done the project with very specific parameters and for a specific attack, and there are many other ways to generate adversarial examples as we have seen.

That is why our work could have several improvements that will require more means and time to carry them out, but we leave them as possible future work in the next section.

9.2. Future Work

As possible future work, we identify the following ones:

- Analyse the results with other types of attacks or other datasets: the GAN of our work was trained with data from normal traffic and the DoS attack. An interesting advance would be to train it with other types of attacks such as those that make up the dataset used in this work (User to Root, Root to Local and Probing).
- Use more epochs in the training of the network: in the case of more time or a computer with more resources, the same GAN network could be trained using a larger number of epochs. This should minimise at each epoch the loss of the generator and maximise the loss of the discriminator, which is the goal of GAN training. Surely, we would obtain better results
- Test our network with other Machine Learning algorithms and with other datasets: in our work we have attacked an IDS based on decision trees, but we could test the performance against other algorithms and compare both the performance of the IDS and the GAN when it comes to cheating. In addition, performance could be tested using other datasets, both traffic and other kind of data.
- Improvements to the GAN network: other parameters or even other structures could be used in the generator and discriminator layers. Another possibility would be to change the libraries used to compare which ones give better results. Wo could also experiment with improvements to the GANs already studied, such as the Wassertein GANs.
- Train the ML algorithm with adversarial data: it has been proven to improve the performance of the algorithm against adversarial attacks if it is trained. We could therefore try to train our IDS with the false data so that it learns to distinguish them.

Bibliografía

[BIM23] Generate Untargeted and Targeted Adversarial Examples for Image Classification. https://es.mathworks.com/help/deeplearning/ug/generate-adversarial-examples.html, April 2023.

[DT23] Árboles de decisión. https://www.ibm.com/es-es/topics/decision-trees, May 2023.

[FGS23] Adversarial attacks with FGSM (Fast Gradient Sign Method). https://pyimagesearch.com/2021/03/01/adversarial-attacks-with-fgsm-fast-gradient-sign-method/, April 2023.

[GPAM+20] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

[GZL+21] Sensen Guo, Jinxiong Zhao, Xiaoyu Li, Junhong Duan, Dejun Mu, and Xiao Jing. A black-box attack method against machine-learning-based anomaly network flow detection models. *Security and Communication Networks*, 2021:1–13, 2021.

[HGMH+00] José Ramón Hilera González, Víctor José Martínez Hernando, et al. *Redes neuronales artificiales: fundamentos, modelos y aplicaciones*. 2000.

[HHAE22] Abdelmageed Ahmed Hassan, Mohamed Sayed Hussein, Ahmed Shehata AboMoustafa, and Sarah Hossam Elmowafy. Synthesis of adversarial ddos attacks using tabular generative adversarial networks, 2022.

[HT17a] Weiwei Hu and Ying Tan. Black-box attacks against rnn based malware detection algorithms, 2017.

[HT17b] Weiwei Hu and Ying Tan. Generating Adversarial Malware Examples for Black-Box Attacks Based on GAN. *CoRR*, abs/1702.05983, 2017.

[IAKMS19] Olakunle Ibitoye, Rana Abou-Khamis, Ashraf Matrawy, and M Omair Shafiq. The Threat of Adversarial Attacks on Machine Learning in Network Security–A Survey. *arXiv preprint arXiv:1911.02621*, 2019.

[IDS23] Qué es un IDS o Intrusion Detection System. https://www.clavei.es/blog/que-es-un-ids-o-intrusion-detection-system/, May 2023.

[KGB16] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial machine learning at scale. *arXiv preprint arXiv:1611.01236*, 2016.

[LBH15] Yann LeCun, Y. Bengio, and Geoffrey Hinton. Deep Learning. *Nature*, 521:436–44, 05 2015.

[LLY+22] Zewen Li, Fan Liu, Wenjie Yang, Shouheng Peng, and Jun Zhou. A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects. *IEEE Transactions on Neural Networks and Learning Systems*, 33(12):6999–7019, 2022.

[LSX22] Zilong Lin, Yong Shi, and Zhi Xue. Idsgan: Generative adversarial networks for attack generation against intrusion detection. In *Advances in Knowledge Discovery and Data Mining: 26th Pacific-Asia Conference, PAKDD 2022, Chengdu, China, May 16–19, 2022, Proceedings, Part III*, pages 79–91. Springer, 2022.

[ML123] What is Machine Learning? https://www.geeksforgeeks.org/ml-machine-learning/, March 2023.

[ML222] ¿Qué es el *transfer learning?* https://datascientest.com/es/que-es-el-transfer-learning, January 2022.

[NLS23] NSL-KDD dataset. https://www.unb.ca/cic/datasets/nsl.html, May 2023.

[SBS+17] Hojjat Salehinejad, Julianne Baarbe, Sharan Sankar, Joseph Barfett, Errol Colak, and Shahrokh Valaee. Recent Advances in Recurrent Neural Networks. *ArXiv*, abs/1801.01078, 2017.

[SM17] Suranjana Samanta and Sameep Mehta. Towards crafting text adversarial samples. *arXiv preprint arXiv:1707.02812*, 2017.

[SZS+13] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[WLC+21a] Yixiang Wang, Jiqiang Liu, Xiaolin Chang, Jelena Mišić, and Vojislav B. Mišić. Iwa: Integrated gradient-based white-box attacks for fooling deep neural networks. *Int. J. Intell. Syst.*, 37(7):4253–4276, oct 2021.

[WLC+21b] Yixiang Wang, Jiqiang Liu, Xiaolin Chang, Jianhua Wang, and Ricardo J. Rodríguez. Di-aa: An interpretable white-box attack for fooling deep neural networks, 2021.

[XZJL21] Ying Xu, Xu Zhong, Antonio Jimeno-Yepes, and Jey Han Lau. Grey-box adversarial attack and defence for sentiment classification. *CoRR*, abs/2103.11576, 2021.

[YHZL19] Xiaoyong Yuan, Pan He, Qile Zhu, and Xiaolin Li. Adversarial examples: Attacks and defenses for deep learning. *IEEE transactions on neural networks and learning systems*, 30(9):2805–2824, 2019.

[ZCW+19] Jiale Zhang, Junjun Chen, Di Wu, Bing Chen, and Shui Yu. Poisoning attack in federated learning using generative adversarial nets. In *2019 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, pages 374–380, 2019.

[ZDS17] Zhengli Zhao, Dheeru Dua, and Sameer Singh. Generating natural adversarial examples. *arXiv preprint arXiv:1710.11342*, 2017.

[ZYJ+19] Hongyang Zhang, Yaodong Yu, Jiantao Jiao, Eric P. Xing, Laurent El Ghaoui, and Michael I. Jordan. Theoretically principled trade-off between robustness and accuracy, 2019.