UNIVERSIDAD DE GRANADA

TRABAJO FIN DE GRADO

GRADO DE INGENIERÍA INFORMÁTICA

Generación sintética de tráfico de red con Deep Learning

La *botnet* Neris como caso de estudio

Autor

Francisco Álvarez Terribas

Directores

Roberto Magán Carrión

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

Granada, 8 de julio de 2022

Generación sintética de tráfico de red con Deep Learning

La *botnet* Neris como caso de estudio

Autor
Francisco Álvarez Terribas

Directores
Roberto Magán Carrión

Generación sintética de tráfico de red con Deep Learning: La *botnet* Neris como caso de estudio

Francisco Álvarez Terribas

Palabras clave: Generación sintética de datos, Deep Learning, Sistemas de detección de intrusos en red, Autoencoder variacional

Resumen

Sin duda vivimos en un entorno cambiante e interconectado donde personas y cosas intercambian información heterogénea y de forma continua. Si bien este escenario anima a la aparición de nuevos servicios y aplicaciones, desde el punto de vista de la seguridad el panorama no es tan bueno. Para la protección frente amenazas y ataques de seguridad son los sistemas de detección de intrusiones los que hoy en día se utilizan y han sido utilizados ampliamente, especialmente los sistemas de detección de intrusiones en red. Usualmente, estos sistemas de apoyan en el uso de conjuntos de datos predefinidos para su entrenamiento y evaluación pero no todos los conjuntos de datos son adecuados para ello y utilizar uno un otro tiene un impacto notable sobre el rendimiento y robustez de estos sistemas. En este trabajo se propone una metodología basada en la utilización de un VAE (Variational Autoencoder) para la generación de flujos de red sintéticos que determine roles de los nodos implicados así como la relación entre ellos en forma de topología de red para diferentes ataques en redes de comunicaciones. En concreto, en el presente trabajo, se caracterizarán una serie de ataques del conjunto de datos de red UGR'16 para luego centrarse en la replicación de la topología de red de uno en concreto: la *botnet* Neris.

Synthetic network traffic generation with Deep Learning. The Neris botnet: a case of study

Francisco Álvarez Terribas

Keywords: Synthetic data generation, Deep Learning, Network Intrusion Detection System, Variational Autoencoder

Abstract

Nowadays, we are living in a highly dynamic scenario, where people and things are continuously communicating and sharing heterogeneous information. This context encourages the development of new applications and services. However, from the point of view of security, it is a very concerning environment. To protect networks and systems against new security threats and attacks intrusion detection systems has been and still being used in general and the network intrusion detection systems in particular. Such systems rely on the use of pre-defined dataset most of them useless due to their characteristics like freshness, duration, representation, etc. Using useless network datasets reduces the performance of the detection systems making them also useless to be deployed in a real production environment. For that, in this work, we propose a Deep Learning based methodology for data augmentation where we will mimic realistics attack topologies as part of the generation of traffic network samples. Actually, our solution relies on the use of a VAE (Variational Autoencoder) which has been used in Literature in other contexts like the image generation problem. Concretely, we focus on the replication of the Neris botnet attack included in the well known dataset UGR'16.

Yo, **Francisco Álvarez Terribas**, alumno de la titulación Grado de Ingeniería Informática de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 77447876E, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Francisco Álvarez Terribas

Granada a 8 de julio de 2022.

D. **Roberto Magán Carrión**, Profesor Departamento de Teoría de la Señal, Telemática y Comunicaciones de la Universidad de Granada.

Informa:

Que el presente trabajo, titulado ***Generación sintética de tráfico de red con Deep Learning: La botnet Neris como caso de estudio***, ha sido realizado bajo su supervisión por **Francisco Álvarez Terribas**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 8 de julio de 2022.

El director:

Roberto Magán Carrión

Agradecimientos

A mi familia e Inés por haberme acompañado en este largo camino, tanto en lo bueno como en lo malo. A Roberto por haber sido mi guía y consejero.

Índice general

Índice de figuras

Índice de cuadros

Acrónimos

DL Deep Learning

DoS Denial of Service

HTTP Hyper Text Transfer Protocol

ID2T Intrusion Detection Dataset Toolkit

IDS Intrusion Detection Systems

IoT Internet of Things

IP Internet Protocol

ISP Internet Service Provider

LSTM Long Short-Term Memory

ML Machine Learning

MNIST Modified National Institute of Standards and Technology database

MSE Mean Squared Error

NIDS Network Intrusion Detection Systems

NIMS Network Information Management and Security Group

NMAP Network Mapper

PCA Principal Component Analysis

PCAP Packet Capture

SDN Software Defined Network

SMOTE Synthetic Minority Oversampling Technique

SMTP Simple Mail Transfer Protocol

SSH Secure Shell

TCP Transmission Control Protocol

ToS Type of Service

UDP User Datagram Protocol

VAE Variational Autoencoder

Capítulo 1

Introducción

1.1. Motivación

Cisco, en su informe Annual Internet Report (2018-2023) [2], prevé un notable incremento en el número total de dispositivos conectados a Internet. Se estima que habrá entorno a 29 mil millones de dispositivos conectados, más de tres veces la población mundial actual. Además, se espera que esto se vea impulsado por el despliegue real de las nuevas tecnologías de comunicaciones *p.e.*, el 5G, permitiendo que dispositivos heterogéneos como los encontrados en diferentes ecosistemas de Internet of Things (IoT) tengan un acceso a Internet fácil y asequible. A pesar de los beneficios que esta hiperconectividad aporta a la sociedad, esta también es un arma de doble filo. Desde el punto de vista de la seguridad, este escenario incrementa la superficie de ataque y, por tanto, también el riesgo de sufrir dichos ataques. Según el informe ENISA Threat Landscape 2020 (ETL) [3], la sofisticación de las técnicas empleadas y las capacidades de los atacantes aumentó seriamente en 2019, habiéndose detectado más de 200.000 nuevas variantes diarias de *malware* dirigidas a diversos objetivos.

Por los motivos anteriores, se hacen necesarias medidas de seguridad adicionales para hacer frente a todo tipo de amenazas de seguridad tanto conocidas como desconocidas (ataques *zero-day*). Para ello, tradicionalmente, se han utilizado sistemas Intrusion Detection Systems (IDS) apoyados en diferentes tecnologías, técnicas y algoritmos [4]. Los IDS se basan en el uso de conjuntos de datos previamente recogidos para su entrenamiento, validación y prueba. En particular, los Network Intrusion Detection Systems (NIDS) se basan en el uso de conjuntos de datos de tráfico de red para diferentes fines, como la clasificación de ataques o la detección de anomalías. Sin embargo, el principal inconveniente de estos sistemas, basados mayormente en técnicas de Machine Learning (ML), es que requieren de conjuntos de datos adecuados y fiables para su entrenamiento, en los que la existencia de diferencias notables entre la distribución de la clase positiva, tráfico de ataque,

y la clase negativa, tráfico de fondo (*background*), siguiendo esta última un comportamiento normal. Este hecho, junto con el uso de conjuntos de datos inadecuados, la mayoría de ellos obsoletos, generados sintéticamente y con una duración insuficiente [4], tienen un impacto notable en el rendimiento y limitan la aplicación práctica y despliegue de los NIDS.

Es por todo lo anterior que en el presente proyecto se propone el desarrollo de modelos basados en Deep Learning (DL) con el objetivo de generar flujos de red complejos. Más en concreto, nos centraremos en la generación de topologías de red de ataques a partir de muestras de conjuntos de datos de red. Para ello nos centraremos en la utilización información categórica presente en conjuntos de datos de red como son las direcciones Internet Protocol (IP). Por un lado, nuestro trabajo aborda la utilización de este tipo de información para la generación de muestras de tráfico de red que no se tiene en cuenta en las soluciones del estado del arte y, por otro lado, esta tarea se implementa con técnicas o modelos de DL, en concreto con Variational Autoencoder (VAE). El fin último de la generación de topologías de red de ataques y posteriormente la generación de flujos de tráfico de red completos, es el de mejorar la robustez, fiabilidad y resiliencia de sistemas NIDS ante ataques dirigidos a ellos mismos, como los de evasión, o para la detección de muestras no conocidas o ataques *zero-day*.

1.2. Objetivos

Los objetivos principales del proyecto son los siguientes:

- **OB1**. Estudio del estado estado del arte en lo referente a generación de muestras sintéticas de tráfico de red y topologías de red.
- **OB2**. Caracterización de las diferentes tipologías de ataque que componen el conjunto de datos UGR'16 [1] que se usará en el presente trabajo.
- **OB3**. Desarrollo de modelos basados en DL para la generación de flujos de red sintéticos.

1.3. Contribuciones científicas

Parte del presente trabajo ha sido publicado en las VII Jornadas Nacionales de Investigación y Ciberseguridad celebradas el pasado junio. Además, ha sido aceptado para su publicación en la décimo quinta edición del congreso internacional CISIS (Conference on Computational Intelligence in Security for Information Systems) que se celebrará el próximo mes de septiembre. Dichas contribuciones, por orden, son:

- F. Álvarez-Terribas, R. Magán-Carrión, G. Maciá-Fernández y A. Mora-García, *"Generación sintética de topologías de red con deep learning: la botnet neris como caso de estudio."* en VII Jornadas Nacionales de Investigación en Ciberseguridad (JNIC2022), pp. 34-37, junio 2022, Bilbao (España).
- F. Álvarez-Terribas, R. Magán-Carrión, G. Maciá-Fernández y A. Mora-García, *"A Deep Learning-based approach for Mimicking Network Topologies: the Neris Botnet as a Case of Study"* en 15th Conference on Computational Intelligence in Security for Information Systems (CISIS2022), septiembre 2022, Salamanca (España).

1.4. Organización

Este trabajo está organizado en seis capítulos. En este primer Capítulo 1 se ha presentado la motivación y los objetivos deseados.

En el Capítulo 2 hablaremos sobre la planificación seguida para realizar el trabajo. En él se detallan los requisitos necesarios para su realización, las tareas a desarrollar, los recursos tanto técnicos como humanos empleados y el presupuesto estimado.

En el Capítulo 3 se detalla el estudio del estado del arte en el tema abordado. Por un lado, en la Sección 3.1 nos centraremos en aquellos trabajos enfocados a la generación de topologías de red para, después, en la Sección 3.2, centrarnos en aquellas propuestas que se dedican a la generación de muestras sintéticas.

A lo largo del Capítulo 4 se detallan los métodos, técnicas y herramientas utilizadas para la generación de flujos de red sintéticos. Así, en primer lugar, en la Sección 4.1 se realiza un estudio del conjunto de datos de red UGR'16 para posteriormente caracterizar sus diferentes tipologías de ataque en la Sección 4.2. A continuación, en la Sección 4.3 se expone la arquitectura del modelo propuesto así como la metodología seguida para la generación de flujos de red sintético basadas en técnicas de DL.

En el Capítulo 5 se analizan en detalle los resultados obtenidos de forma numérica y visual.

Para finalizar, se exponen las conclusiones alcanzadas, así como posibles líneas de trabajo futuro en el Capítulo 6.

Capítulo 2

Tareas, recursos y planificación.

A partir de los objetivos definidos en el proyecto, en este capítulo se detallan las tareas y planificación temporal de estas a lo largo del tiempo para llevar a cabo dichos objetivos. Así mismo, se expondrán los recursos utilizados y una estimación de costes asociados al proyecto.

2.1. Tareas a desarrollar

Al empezar el proyecto se llevó a cabo un metódico proceso para la definición de las principales tareas necesarias para la consecución de los objetivos del proyecto. Estas son:

- **Estado del arte.** Consiste en la búsqueda e investigación de la información relativa a nuestro problema. Esta tarea deriva directamente del objetivo OB1 visto en la Sección 1.2.
- **Evaluación de soluciones.** Después de haber hecho una selección de posibles soluciones será necesario estudiar en profundidad la formulación e implementación de estas con el fin de poder probarlas a pequeña escala. Una vez implementadas las soluciones se evaluarán los resultados obtenidos con el fin de elegir la más prometedora. Esta tarea deriva del objetivo OB1.
- **Estudio del conjunto de datos de red UGR'16.** La solución elegida necesitará de un conjunto de datos adecuado y de calidad para su entrenamiento. Por tanto llevaremos a cabo un análisis exploratorio del dataset UGR'16 caracterizando las diferentes tipologías de ataque que lo comprenden con el fin de seleccionar cual será, o serán, los ataques que se usarán con la solución elegida. Esta tarea deriva del objetivo OB2.

- **Estudio de representaciones de datos.** El preprocesado de los datos con el fin de que estos sean adecuados para los modelos de ML es una de las partes más importante en la implementación de dichos modelos. Una correcta representación de los datos puede marcar una gran diferencia en los resultados que genere nuestra implementación. Esta tarea deriva del objetivo OB2.

- **Implementación final de la solución.** Una vez implementado el preprocesado el siguiente paso es realizar la implementación final de la solución elegida puesto que en la tarea de Evaluación de soluciones ya se ha realizado una aproximación a la misma. Esta tarea deriva del objetivo OB3.

- **Evaluación de resultados.** Tras la implementación del modelo hay que generar un conjunto de datos sintético con el fin de realizar un análisis exploratorio sobre el mismo. A partir de este se valorá la correctitud de nuestra solución. Esta tarea deriva del objetivo OB3.

2.2. Recursos humanos y técnicos

Los recursos necesitamos para llevar a cabo el proyecto son:

- **Hardware**:

  - Ordenador portátil Lenovo Thinkpad E15.
  - Servidor del Laboratorio de Ciberseguridad de la UGR denominado como *Ripper*. Cuenta con un procesador Intel Xeon Silver 4208, a 2.10GHz y 32 núcleos; 32GB de memoria ECC y tres tarjetas gráficas Nvidia RTX 2080ti de 12GB.

- **Software**:

  - **Ubuntu 20.04 LTS.** Será la distribución Linux usada tanto en el ordenador portátil como en el servidor.
  - **Jupyter Hub.** Será el entorno en el cual se desarrollarán las Jupyter Notebooks relacionadas con el proyecto. Estará desplegado en el servidor.
  - **SSH.** Para establecer una conexión segura al servidor se crea un tunel SSH redirigiendo el puerto por el cual se sirve Jupyter Hub.
  - **Gephi.** Esta herramienta permite realizar análisis de topologías de red en forma de grafo de conectividad.

- **Librerías Python.** Para la realización de este proyecto se ha utilizado el lenguaje de programación Python junto con las siguientes librerías:
  - Tensorflow
  - Numpy
  - Pandas
  - Matplotlib
  - Ipaddress

- **Recursos humanos**. En la Tabla 2.1 se muestra una aproximación de las horas de trabajo empleadas en las diferentes tareas realizadas, tanto por el alumno como por el tutor, incluyendo las reuniones asociadas a cada tarea.

Tarea	**Tiempo empleado por el estudiante (horas)**	**Tiempo empleado por el tutor (horas)**
Estado del arte	50	7
Evaluación de soluciones	90	8
Estudio de UGR'16	60	10
Estudio de las diferentes representaciones de datos	30	2
Implementación final de la solución	60	8
Evaluación de resultados	10	10
Total	**300**	**45**

Cuadro 2.1: Resumen recursos humanos.

2.3. Planificación

Es muy importante llevar a cabo una correcta planificación temporal de las tareas a realizar cuando se plantea un proyecto.

En este caso las tareas planteadas se debían hacer en el orden expuesto en la Sección 2.1 puesto que, era muy complicado abordar una nueva tarea sin haber completado la tarea anterior. Esto es debido a que en la mayoría de los casos el contexto de una tarea no podía ser definido totalmente sin el trabajo previo de las tareas anteriores.

La planificación temporal de las tareas se ha hecho mediante un diagrama de Gantt, la asignación de tareas se ha realizado de forma semanal desde la primera semana de diciembre de 2021 hasta la segunda semana de julio de 2022, véase *Fig. 2.1.* Como se puede ver las primeras tareas fueron divididas en subtareas, dichas tareas presentaban un contexto menos concreto y necesitaron de algo más de planificación. Conforme el proyecto fue avanzando

las tareas planteadas tenían contextos mucho más concretos haciendo que su planificación fuera menos exhaustiva.

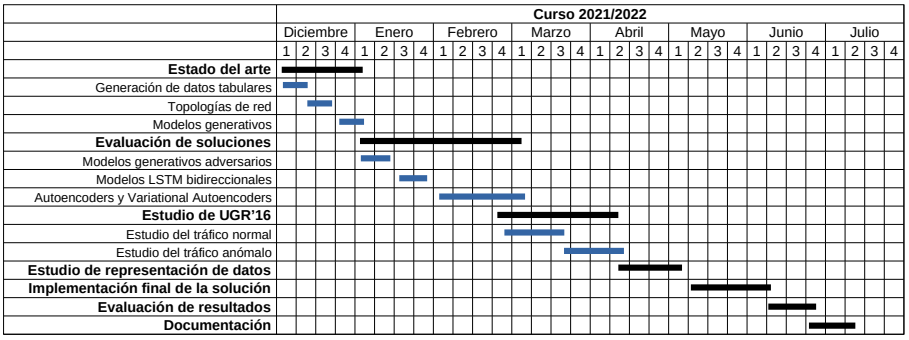

Figura 2.1: Diagrama de Gantt del proyecto.

El control de la realización de estas tareas se ha realizado durante las reuniones semanales entre el alumno y el tutor. Durante estas reuniones se exponía el trabajo realizado así como problemas que bloquearan la realización de dichas tareas y posibles siguientes pasos a tomar.

2.4. Presupuesto

Para presupuestar el proyecto hemos de tener en cuenta el tiempo empleado por parte de los recursos humanos en las tareas, así como los recursos materiales empleados en estas.

2.4.1. Recursos humanos

Por un lado el alumno tendrá un coste de 25€ por hora mientras que la supervisión realizada por el tutor estará valorada en 70€ por hora. Dicha supervisión se ha realizado en reuniones semanales de entre media hora hasta una hora y media. Si observamos la Tabla 2.2 se puede ver que el monto referente a recursos humanos asciende a 10.650€.

2.4.2. Recursos materiales

La estimación de costes materiales se ha hecho realizando directamente una exploración del mercado actual, principalmente debido a que el coste de los componentes del servidor se ha mantenido en el tiempo y el portátil utilizado es de nueva adquisición. Dicho esto podemos ver como en la Tabla 2.2 el coste aproximado del servidor de 5.650€ mientras que el portátil tiene un coste de 650€.

Tarea	**Coste del estudiante (25€/h)**	**Coste del tutor (70€/h)**	**Coste total (€)**
Estado del arte	1250	490	1740
Evaluación de soluciones	2250	560	2810
Estudio de UGR'16	1500	700	2200
Estudio de representaciones de datos	750	140	890
Implementación final de la solución	1500	560	2060
Evaluación de resultados	250	700	950
Total			**10.650**

Cuadro 2.2: Coste de recursos humanos.

Recurso	**Coste (€)**
Servidor	5650
Ordenador portátil Lenovo Thinkpad E15	650
Total	**6.300**

Cuadro 2.3: Coste de los recursos Hardware.

Coste Recursos humanos (€)	10.650
Coste Recursos hardware (€)	6300
Coste Recursos software (€)	0
Total (€)	**16.950**

Cuadro 2.4: Presupuesto total estimado.

2.4.3. Recursos software

Al haber utilizado solamente software libre el coste de total de este apartado es de 0€.

2.4.4. Coste total

Tal y como se puede ver en la Tabla 2.4 el presupuesto del proyecto asciente a 16.950€.

Capítulo 3

Estado del arte

A lo largo del presente capítulo se introducen algunos de los principales trabajos de la Literatura que abordan problemas similares al de nuestro proyecto.

3.1. Generación de topologías de red

Los conjuntos de datos de tráfico de red contienen variables categóricas, *e.g.* nodos o dispositivos finales representados por IP y puertos. Estas variables son cruciales para caracterizar de forma adecuada una red y sus flujos de comunicaciones siendo capaces de representar una determinada topología de red a partir de dicha información.

El estudio, evaluación y generación sintética tanto de topologías físicas como lógicas ha sido ampliamente abordado en la Literatura durante los últimos años. Debido a la proliferación del paradigma de las Software Defined Network (SDN) [5], las redes han empezado a comportarte como sistemas autónomos los cuales han dejado de requerir de intervención de operadores humanos para su reconfiguración haciendo que dichas infraestructuras sean más resilientes y tolerantes a fallos, esto hace que las organizaciones tengan la necesidad de poder validar de forma adecuada sus SDN en entornos controlados requiriendo el uso de herramientas de simulación de topologías de red como *TOPOGEN* [6]. Junto con este tipo de herramientas también hay que hacer especial hincapié en la validación e inferencia de las topologías generadas, para esto desde la Literatura [7] principalmente se propone la evaluación de las diferentes métricas del grafo generado *e.g.* grados medios, distribución de grados, densidad, etc. Dentro del campo del análisis de topología físicas [8] se puede ver como, de igual forma que con las SDN, para medir la resiliencia de la red también se analizan las métricas generadas por el grafo de la topología. Sin embargo, a pesar de su gran utilidad para la generación de topologías de red estas herramientas están pensadas específicamente solo para la generación de topologías y, por tanto, la generación de

trazas de red completas queda fuera de su campo de aplicación.

3.2. Generación de muestras sintéticas

El desbalanceo de clases en conjuntos de datos es un problema recurrente en el desarrollo de sistemas de clasificación basados en modelos de aprendizaje supervisado [9]. Dicho problema cobra especial relevancia a la hora de implementar y evaluar NIDS basados en ML, dichos NIDS necesitan conjuntos de datos adecuados y fiables para su entrenamiento. En la mayoría de los casos la utilización de conjuntos de datos inadecuados durante el desarrollo de los NIDS afecta negativamente a sus capacidades de detección de la clase minoritaria. De igual forma la evaluación del rendimiento de estos puede resultar problemática si no se escogen las métricas adecuadas para ello. En este sentido, sin abordar eficazmente este problema, se torna difícil generar sistemas de detección fiables y robustos para ser desplegados en entornos productivos. A continuación vamos a revisar algunos de los trabajos más relevantes en el campo de la generación de muestras sintéticas de tráfico de red en el contexto de los NIDS.

El algoritmo Synthetic Minority Oversampling Technique (SMOTE) [10], junto a todos los algoritmos derivados de este [11, 12], ha sido el método más utilizado para la realización de *oversampling* o sobre-muestreo en conjuntos de datos desbalanceados. Principalmente se basan en seleccionar una instancia de la clase minoritaria, calcular cuales son sus vecinos más cercanos e interpolar muestras sintéticas entre dicha instancia y sus vecinos.

Gracias a la irrupción del DL durante estos últimos años, se ha podido ver como diferentes autores empiezan a aplicar este tipo de técnicas para la generación de muestras sintéticas de tráfico de red. Por ejemplo, Vu *et al.* [13] propusieron la aplicación de *deep generative adversarial models* sobre el dataset Network Information Management and Security Group (NIMS) [14], obteniendo un mejor rendimiento de clasificación respecto a algoritmos derivados de SMOTE. Posteriormente Engelmann *et al.* [15] plantearon una arquitectura de modelos generativos adversarios más robusta y capaz de trabajar con datos categóricos que normalmente se obvian en la Literatura. Para ello aplicaron técnicas de pre-procesado *one-hot encoding* es decir, representando cada valor categórico como una *feature* con valor 0 (*low*) o 1 (*high*). Cabe destacar que dichos modelos, junto con los VAE [16] están siendo ampliamente utilizados para la generación de imágenes sintéticas [17].

Otra metodología a destacar, propuesta en los trabajos [18, 19], es la codificación de las muestras en un espacio latente. Una vez en este espacio, los autores aplican SMOTE u otro algoritmo derivado para, posteriormente, decodificarlas en el espacio de muestras original.

Todas estas soluciones funcionan muy bien a la hora de trabajar con variables continuas o aplicadas a la generación de imágenes sintéticas. Sin

embargo, soluciones en las cuales se abordan problemas que involucran variables categóricas, como es el caso de las IP en los conjuntos de datos de tráfico de red, no siempre pueden aplicarse cuando existen variables que presentan una gran dimensionalidad. En este contexto, la utilización de aproximaciones como *one-hot encoding* se hacen inviables debido a la gran demanda de recursos de memoria que necesitan.

Por todo lo anterior, el presente trabajo tratará obtener de forma automática muestras sintéticas de tráfico de red haciendo uso de conjuntos de datos predefinidos que incluyen todo el conjunto de variables contempladas tanto numéricas (*e.g. bytes* transmitidos) como categóricas (*e.g.* IP. Para ello se utilizarán técnicas DL, como los VAE para, primero, replicar los más fielmente topologías, roles, comunicaciones y comportamientos de ciertos ataques de red en forma de grafos de topología; y, segundo, obtener trazas completas de red que puedan ser utilizadas para obtener sistemas NIDS robustos y fiables.

Capítulo 4

Métodos, técnicas y herramientas

A continuación se introduce el conjunto de datos a utilizar así como la metodología propuesta para la generación sintética de topologías de red mediante la utilización de VAE.

4.1. Conjunto de datos: UGR'16

El conjunto de datos UGR'16 [1] está formado por flujos de tráfico de red (*NetFlow*) anonimizados capturados durante 4 meses en las instalaciones de un Internet Service Provider (ISP) español de capa 3. Este se divide a su vez en dos: CAL y TEST. El primero de ellos solo contiene tráfico normal generado y visto en la red durante tres meses mientras que al segundo se le añaden ataques generados de forma sintética con herramientas actuales (*Denial of Service (DoS) low rate*, *Scan (Port Scanning)* o *Botnet*) y aquellos ataques que fueron identificados por varios detectores (*User Datagram Protocol (UDP) port scan*, *Secure Shell (SSH) scan* y campañas de *Spam*). En las *Fig. 4.1 y 4.2* se muestra la evolución del tráfico de red para los conjuntos CAL y TEST respectivamente. Adicionalmente hay algunos flujos marcados como *Blacklist*, dichos flujos pertenecen a direcciones IP públicas dentro de listas *blacklist* conocidas aunque no necesariamente representan tráfico malicioso. La distribución de las clases previamente descritas se puede apreciar en detalle en la *Fig. 4.3*

La red del ISP donde se ha realizado la captura de flujos de red sigue una arquitectura de zona desmilitarizada simple siendo sus principales componentes los siguientes:

- Dos *router* frontera redundantes, $BR1$ y $BR2$. Dichos *router* proveen a la infraestructura de acceso a internet y a una subred frontal con máquinas atacantes ($A_1 - A_2$). Las sondas encargadas de capturar los

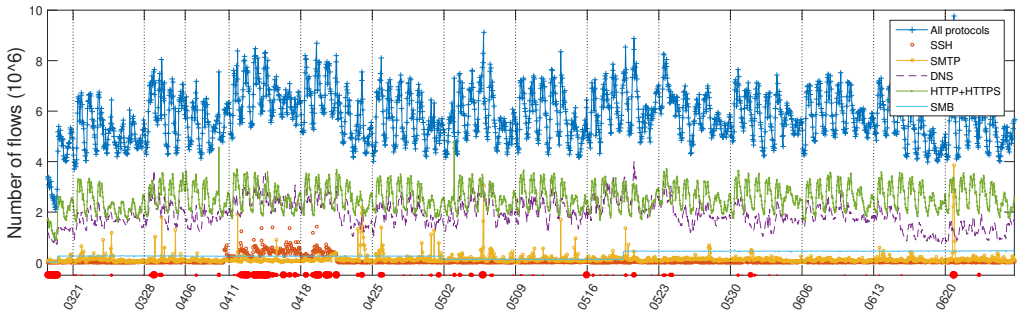

Figura 4.1: Flujos de red de la partición CAL de UGR'16 [1].

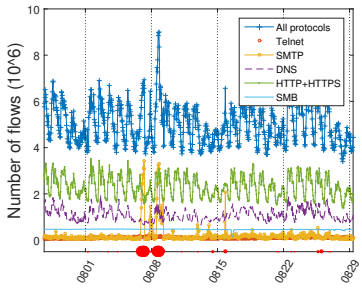

Figura 4.2: Flujos de red de la partición TEST de UGR'16 [1].

NetFlow del conjunto de datos se colocaron en las interfaces de red de estos *router* frontera.

- Una red frontal principal (*core network*) donde se ubica la subred de víctimas V_1 (máquinas $V_{11} - V_{15}$).
- Una red interna (*inner network*) conectada a la red frontal mediante dos interfaces de red redundantes, ambas protegidas por un *firewall*. Dentro de esta red encontramos las subredes de víctimas V_2 (máquinas $V_{21} - V_{25}$), V_3 (máquinas $V_{31} - V_{35}$) y V_4 (máquinas $V_{41} - V_{45}$).

4.2. Caracterización de ataques contenidos en UGR'16.

Para el caso que nos atañe una tipología de ataque de las que dispone UGR'16 ha de ser elegida, por tanto es necesario realizar una caracterización de estas. En primer lugar, para esta elección, vamos a dejar fuera a los ataques de escaneo debido a que estos ataques pueden ser fácilmente generados con herramientas como Network Mapper (NMAP) [20]. De igual forma hay herramientas como Intrusion Detection Dataset Toolkit (ID2T) [21] las cuales están pensadas para realizar un estudio estadístico de archivos Packet

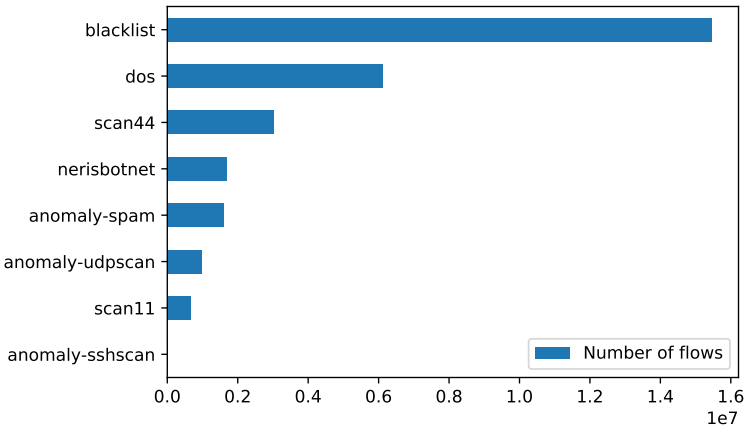

Figura 4.3: Distribución de ataques en UGR'16, tanto generados como detectados. Para diferenciarlos, se le añade el sufijo *anomaly-* a aquellos que fueron detectados.

Capture (PCAP) para posteriormente añadirles de forma sintética flujos de red maliciosos englobados dentro del contexto del archivo aportado. ID2T funciona muy bien para inyectar tipologías de ataques muy simples, como los anteriormente mencionados ataques de escaneo. Sin embargo, a la hora de probar esta herramienta para la generación de tipologías de ataque más complejas nos hemos encontrado con que carece de estabilidad durante su ejecución y, por tanto, desaconsejamos su uso para estos casos más complejos. Por todo esto caracterizaremos los ataques *DoS Low rate*, *Spam* y *Botnet*.

A continuación vamos a analizar la cantidad de flujos de red que presentan cada uno de los ataques a lo largo del tiempo junto con la topología que representan.

4.2.1. *DoS*

En la *Fig. 4.4* se puede ver como el ataque *DoS*, genera ráfagas de tráfico de forma periódica, así mismo, según los autores de UGR'16 [1] dentro de dichas ráfagas se engloban 3 subtipologías de ataque:

- **DoS11.** Es un ataque DoS uno a uno $A_1 \rightarrow V_{21}$ con una duración de 3 minutos.
- **DoS53s.** En esta tipología de ataque cinco actores maliciosos atacan de forma sincronizada a tres víctimas siguiendo el esquema $(A_1,A_2) \rightarrow V_{21}$, $(A_3,A_4) \rightarrow V_{31}$ y $A_5 \rightarrow V_{41}$.
- **DoS53a.** Esta tipología de ataque sigue el mismo esquema que *DoS53s* pero la ejecución de ataques no se realiza de forma sincronizada. A ca-

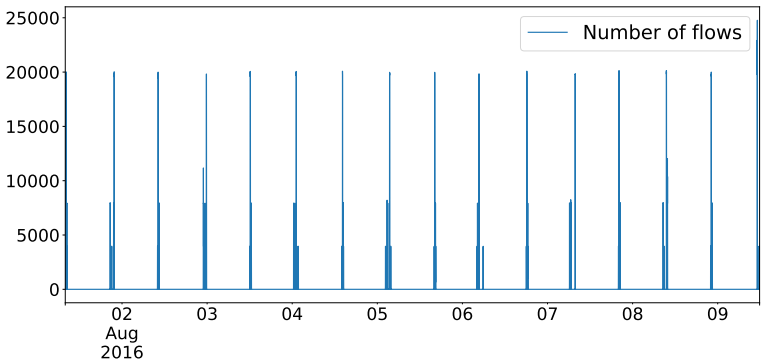

Figura 4.4: Flujos *DoS* capturados desde 01-08-2016 hasta 09-08-2016.

da víctima se le ataca durante 3 minutos con un periodo de inactividad entre ataques de 30 segundos.

Si analizamos en detalle los flujos de una de las ráfagas de este ataque (véase *Fig. 4.5*) se pueden apreciar claramente las 3 subtipologías descritas. *DoS11* se correspondería con los flujos de red detectados entre las 12:00 y las 12:03, *DoS53s* se corresponde con los flujos de red detectados entre las 12:09 y las 12:12, por último flujos de red detectados entre las 12:19 y las 12:29 se corresponden con *DoS53a*.

Para analizar cuales son los nodos de red implicados en esta tipología de ataque se ha representado su topología de red en la *Fig. 4.6*. En dicho gráfico el tamaño del nodo representa la cantidad de conexiones que este tiene mientras que el grosor de las aristas representa la cantidad de conexiones entre dos nodos. Analizando la topología nos damos cuenta de que, al igual que en el análisis de la serie temporal del agregado de flujos, las diferentes subtipologías de ataque DoS son fácilmente caracterizables. Véase en el gráfico que junto a las direcciones IP se ha incluido una etiqueta indicando que papel cumple dicho nodo siguiendo el esquema introducido anteriormente de cada subtipología.

Como hemos visto, la topología de red, relaciones y roles del ataque DoS hace fácil su caracterización e interpretabilidad. Así, este tipo de ataques se pueden implementar o replicar de forma sencilla con herramientas del estado del arte como ID2T vista anteriormente.

4.2.2. *Spam*

A continuación analizaremos *Spam*, dicha tipología de ataque viene definida por flujos de red Simple Mail Transfer Protocol (SMTP) anómalos generados desde 5 direcciones IP públicas dirigidos hacia servidores de correo de Yahoo.

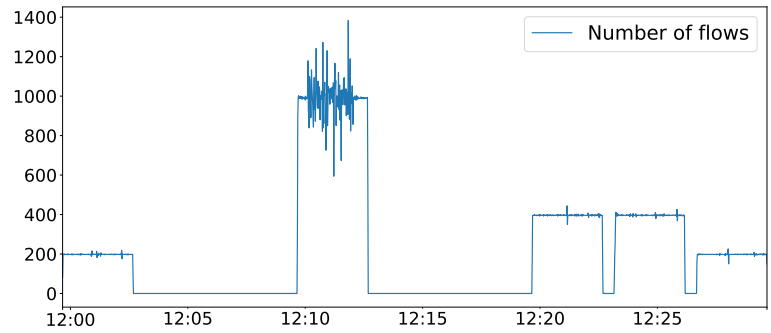

Figura 4.5: Detalle de ráfaga de ataque *DoS*.

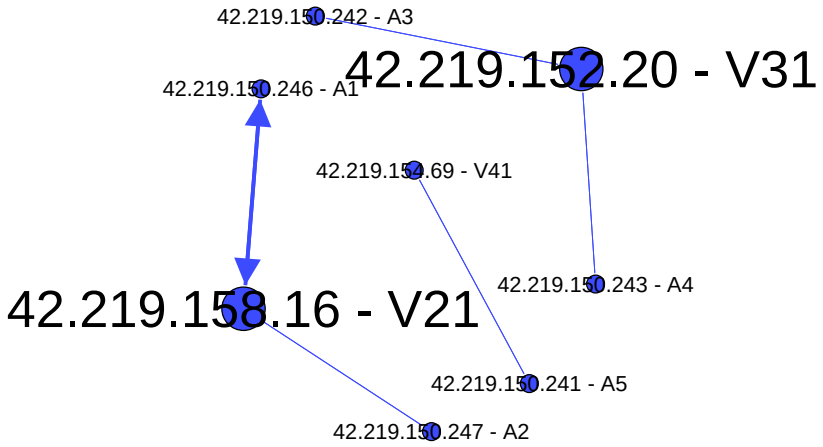

Figura 4.6: Topología de red de *DoS*.

La distribución en el tiempo de los flujos *Spam*, al contrario que *DoS*, no presenta un comportamiento estacionario. En la *Fig. 4.7* se pueden ver como se detectaron 3 pequeñas anomalías *Spam* entre el día 1 y el día 17. Adicionalmente, desde el día 20 hasta el 24 se detecta una importante anomalía de una magnitud muy superior a las anteriores. Analizando en detalle esta última anomalía, en la *Fig. 4.8* se observa que en el periodo comprendido entre el 20 de mayo hasta el 24 de mayo la distribución de flujos generados por la ráfaga de ataque no sigue un patrón preestablecido como la presentada por los flujos *DoS*. Esta diferencia de comportamiento se debe a que, al contrario que *DoS*, este no es un ataque generado de forma sintética, fue tráfico detectado como anómalo mediante el empleo de algoritmos y técni-

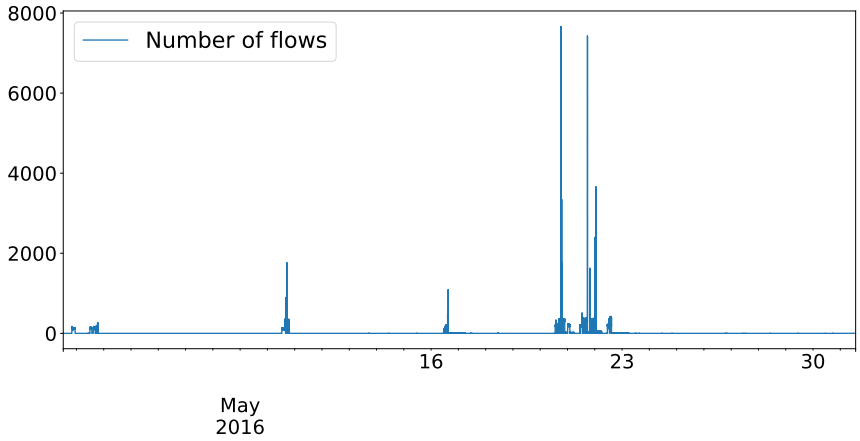

Figura 4.7: Flujos *Spam* capturados desde 02-05-2016 hasta 31-05-2016.

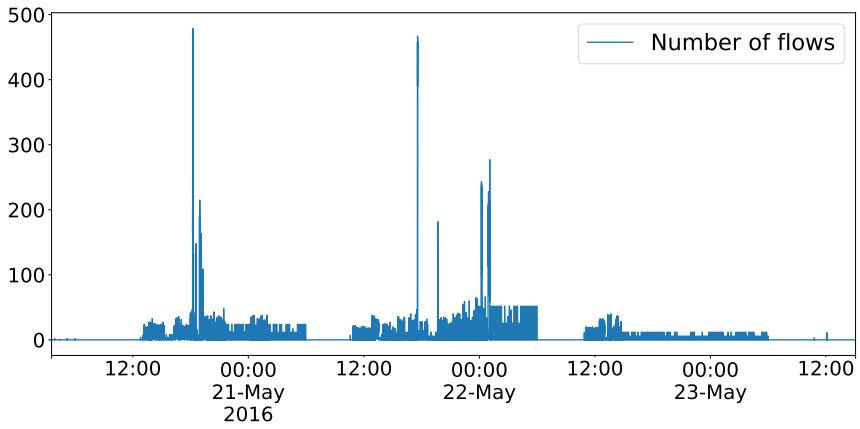

Figura 4.8: Flujos *Spam* capturados desde 20-05-2016 hasta 24-05-2016.

cas no supervisadas de detección de anomalías, dos de ellas basadas en el análisis estadístico multivariante [22].

En la *Fig. 4.9* se expone la topología del ataque en donde el tamaño de los nodos es proporcional a su grado de salida. Se pueden ver perfectamente los cinco nodos que realizan este ataque previamente descritos (representados en color rojo en la figura). Estos nodos vienen caracterizados principalmente por presentar altos grados de salida, mientras que víctimas del ataque son los nodos caracterizados por presentar menor grado de salida. Adicionalmente se puede ver como ningún nodo víctima (representados en color verde en la figura) se comunica directamente con otro nodo víctima, solo reciben comunicadores de los nodos que actúan como actores maliciosos. Dentro del grupo de los atacantes podemos ver que existe un nodo con mayor grado de salida, dicho nodo lo podríamos considerar como el *actor principal* del

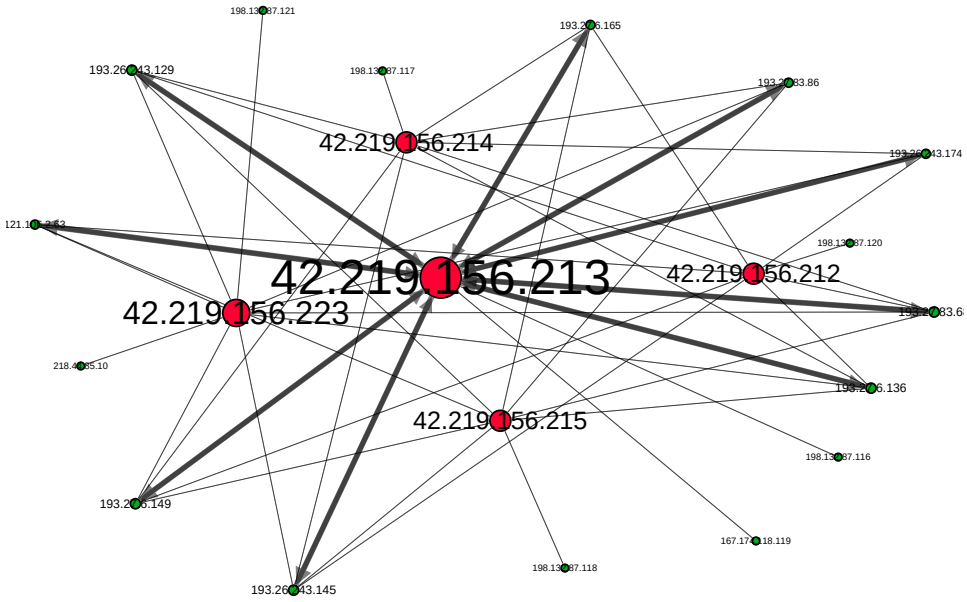

Figura 4.9: Topología de red de *Spam* donde el tamaño de los nodos representa su grado de salida. En verde las víctimas y en rojo los atacantes.

ataque ya que es que el genera la mayor cantidad de flujos de red.

Tras el análisis de *Spam* podemos concluir que esta tipología de ataque presenta un comportamiento y topología de red más complejo en comparación con el ataque *DoS* siendo mayor el número de nodos implicados.

4.2.3. *Botnet*

Pasemos a analizar *Botnet*, esta tipología de ataque está conformada por flujos de red de la famosa Neris Botnet. Dicha *botnet* posee una estructura jerárquica en donde existe un *botmaster* denominado como *Saruman*, servidores C&C y los propios *bots* controlados por cada uno de los servidores C&C. Los *bots* se conectan a los servidores C&C mediante Hyper Text Transfer Protocol (HTTP) para enviar *spam* y realizar *ClickFraud* (más detalles en el trabajo [23]).

En este caso las ráfagas de ataque tienen un carácter estacionario comprendido desde el 28 de julio hasta el 8 de agosto tal y como se muestra en la *Fig. 4.10*. Esta distribución de ráfagas de ataque fue generada de forma sintética, sin embargo, la generación de dichas trazas fue realizada en un entorno controlado, el laboratorio de *Stratosfere Lab* [24]. En la *Fig. 4.11* se detalla el tráfico generando durante una ráfaga específica del ataque. En dicha figura se observa la gran complejidad que presenta *Botnet* en términos

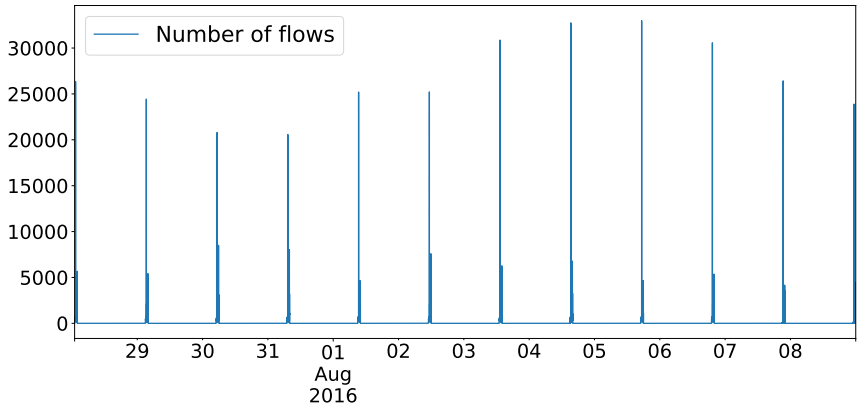

Figura 4.10: Flujos *Botnet* capturados desde 28-07-2016 hasta 08-08-2016.

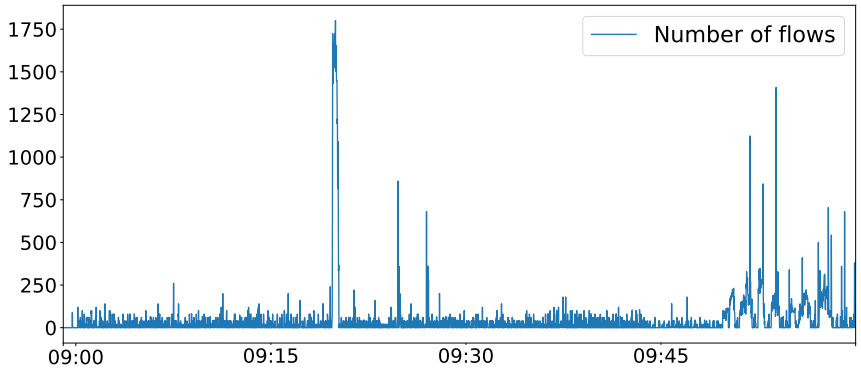

Figura 4.11: Flujos *Botnet* capturados durante 02-08-2022.

del patrón de tráfico generado.

La topología de red de Neris *Botnet* se representa en la *Fig. 4.12*. Se puede ver como esta presenta un tamaño y una complejidad considerablemente superior al mostrado por las tipologías previamente estudiadas. En la figura podemos identificar a *Saruman* como el nodo central de color amarillo, los servidores C&C son representados como los nodos rojos que rodean a *Saruman*. El resto de nodos visibles en la figura son los *bots* controlados por la *Botnet*.

Una vez más este ataque en concreto, en comparación con los anteriores , presenta una especial complejidad tanto en los patrones de tráfico que genera como en la relación y roles de los nodos implicados. Por lo tanto, replicar este tipo de ataques presenta todo un reto tanto científico como ingenieril que es abordado en el presente trabajo mediante la utilización de modelos de DL.

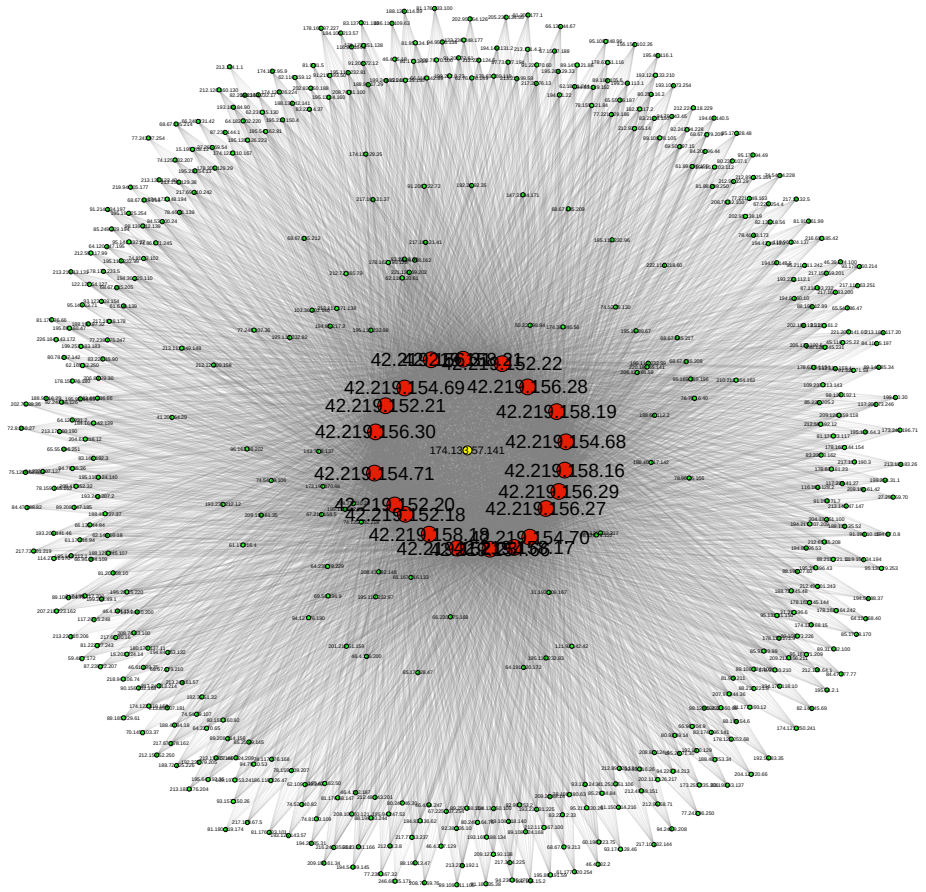

Figura 4.12: Topología de red de Neris *Botnet*, el tamaño de los nodos representa su grado de salida. Se muestra el nodo botmaster en amarillo rodeado por los nodos C&C coloreados en rojo, el resto de nodos coloreados en verde representarn los bots.

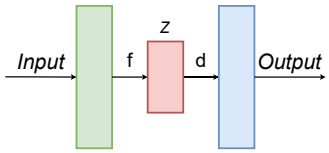

Figura 4.13: Arquitectura básica de un *autoencoder*.

4.3. Metodología propuesta

En primer lugar definiremos y justificaremos el uso de las arquitecturas VAE estudiando su utilización en otros ámbitos, *e.g.* la generación de imágenes. Posteriormente se describirá en detalle el modelo implementado junto con el procesamiento de datos realizado antes y después de la utilización de dicho modelo.

4.3.1. Introducción a los VAE

Empecemos sentando las bases de los VAE definiendo, primero, qué son los *autoencoders* para, después, introducir las características generales de los modelos generativos.

En la *Fig. 4.13* se muestra un esquema típico general de un *autoencoder*. Los *autoencoders* son modelos de redes neuronales que codifican una entrada *input* reduciendo su dimesionalida, dicha representación de dimensionalidad reducida es denominadada como espacio latente (Z). Posteriormente se decodifica Z con el fin de obtener una salida, *output*, con el menor error de reconstrucción posible con respecto a la entrada original, *input* [25, 26]. En la *Eq. 4.1* se muestra la descripción matemática de Mean Squared Error (MSE), esta función de pérdida [27] modela el propósito de un *autoencoder* básico. En dicha función se busca minimizar el error de reconstrucción al decodificar una determinada muestra *input* de tamaño n donde f y d representan la función de codificación y de decodificación respectivamente.

$$minE(input, d(f(input))) = min\frac{1}{n}\sum_{i=1}^{n}(input_i - d(f(input_i)))^2 \qquad (4.1)$$

Estos modelos han sido ampliamente utilizados para diversas tareas. En primer lugar, cabe destacar el gran desempeño que presentan en tareas de reducción de dimensionalidad [28] produciendo menor error de reconstrucción que técnicas como Principal Component Analysis (PCA) [29]. Esta reducción de dimensionalidad permite incrementar el rendimiento de tareas de clasificación [30] tanto de imágenes como de documentos de texto. Otro importante campo de aplicación de los *autoencoders* está en la detección de anomalías [31]. Para ello se entrena el modelo solo con muestras de la clase

normal. La detección de dichas anomalías viene dada por los altos errores de reconstrucción que presentan las muestras anómalas al ser decodificadas por el modelo. Una utilidad muy interesante de los *autoencoders* y que abordaremos en un futuro, es su capacidad de manipulación de imágenes [32, 33, 34]. En la mayoría de los trabajos encontrados en la Literatura, se estudia como modificar una foto que contiene una cara añadiendo o modificando aspectos de dicha fotografía como pueden ser rasgos faciales o el peinado.

Dentro del campo del DL [35, 36] los modelos generativos son aquellos modelos capaces de aprender una distribución a partir de un conjunto de entrenamiento con el fin de generar muestras a partir de dicha distribución.

A partir de lo anterior, podemos definir a los VAE [16] como aquellos modelos basados en *autoencoders* capaces de generar muestras a partir de un espacio latente definido por una distribución de probabilidad en concreto.

Los primeros trabajos relativos a la generación de muestras con VAE [37] demostraron que dichos modelos eran capaces de generar muestras de conjuntos de imágenes simples como Modified National Institute of Standards and Technology database (MNIST) [38]. Posteriormente empezaron a surgir trabajos en los cuales aplicaban dichos modelos a conjuntos de datos con muestras más complejas como imágenes de alta resolución a color [39]. Las capacidades de generación de estos modelos también se han utilizado para la generación de piezas musicales, vease, *Magenta* [40]. Esta aplicación del VAE es también muy interesante en el contexto de nuestro problema ya que demuestra la capacidad del modelo para procesar una serie de valores teniendo en cuenta el contexto que lo engloba. Posteriormente a este trabajo también surgieron modelos especializados en la generación de vídeo como puede ser *VideoGPT* [41].

En este trabajo buscamos principalmente utilizar la capacidad de los *autoencoders* para la modificación de muestras junto con la capacidad de generación probabilística de los VAE. Para la generación de flujos de red sintéticos además es necesario que el modelo sea capaz de reconocer el contexto en el que englobar las muestras a generar. Por ello, para la implementación del codificador y el decodificador se va a hacer uso de capas Long Short-Term Memory (LSTM) [42] ya que dichas capas son capaces de mantener la información de contexto capturando las dependencias temporales a largo plazo de información secuencial como, por ejemplo, la contenida en una serie de flujos de tráfico de red [43, 44]. De esta manera, se pretende que el modelo sea capaz de generar flujos de red teniendo en cuenta el contexto y temporalidad inherente de cualquier conjunto de datos de red para así replicar lo más fielmente posible el funcionamiento, topología, características y roles que definen el comportamiento de la *botnet* Neris en particular y, a priori, de cualquier ataque de forma general.

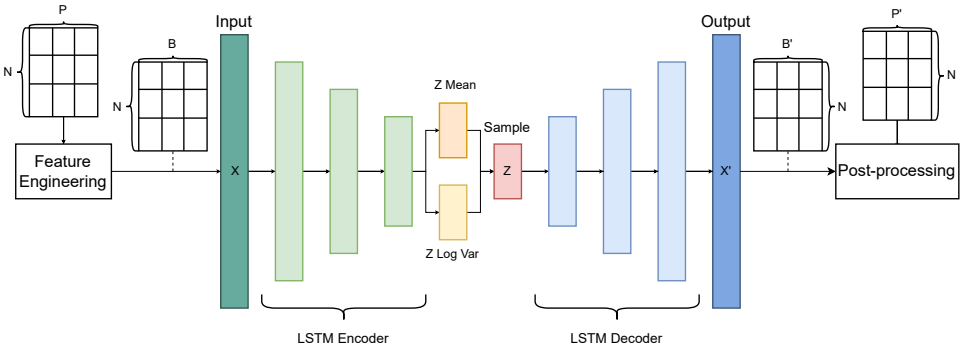

Figura 4.14: Arquitectura del VAE con capas LSTM.

4.3.2. Implementación de la solución

Una vez puesto en contexto la solución que buscamos queda describir en detalle su implementación. En la *Fig. 4.14* se puede apreciar la arquitectura de la solución propuesta. Esta está conformada por varias etapas o módulos generales que son: una etapa previa de *Feature Engineering*, un VAE el cual principalmente hace uso de capas LSTM en sus capas de codificación y decodificación. Como etapa final la salida del modelo debe ser procesada para obtener muestras, entendibles o *human-readable*.

En la etapa de *Feature Engineering* (ver detalles sobre su codificación en la Sección *A.3*) los flujos de red son pre-procesados para, por un lado, codificar direcciones IP, puertos, protocolos, flags Transmission Control Protocol (TCP) y Type of Service (ToS) en formato binario. Por otro lado, las variables continuas como el tiempo de conexión, el número de paquetes y la cantidad de bytes son normalizadas entre 0 y 1. Esto hace que la matriz de datos original $T = N \times P$, siendo N el número de observaciones y P el número de variables, se transforme en $X = N \times B$ con $B > P$ siendo esta la entrada a la siguiente etapa VAE.

La segunda etapa la conforma el VAE (ver detalles sobre su codificación en las Secciones *A.1 y A.2*). Este codifica las muestras originales, en nuestro caso un subconjunto de muestras UGR'16, el correspondiente a la *botnet* Neris, en un espacio latente en donde dichas muestras siguen una distribución gaussiana $G(ZMean, ZLogVar)$ con media $ZMean$ y desviación estándar $ZLogVar$. A partir de la distribución generada, se extraen muestras aleatorias (representadas por Z). Una vez allí, dichas muestras son decodificadas presentando características similares a las muestras originales pero siempre con ciertas diferencias. Este modelo usa como función de pérdida un error de reconstrucción, el anteriormente descrito MSE, más la *Kullback–Leibler divergence* (D_{KL}) [45] *Eq. 4.1*.

$$min(E(input, d(f(input))) + D_{\mathrm{KL}}(ZMean \parallel ZLogVar)) =$$
$$min(\frac{1}{n}\sum_{i=1}^{n}(input_i - d(f(input_i)))^2 + \sum_{g\in\mathcal{G}} ZMean \log\left(\frac{ZMean}{ZLogVar}\right)). \quad (4.2)$$

En la última etapa las muestras obtenidas como salida del VAE ($X' = B' \times N$, siendo B' las variables obtenidas del VAE), son post-procesadas (ver detalles sobre su codificación en la Sección *A.4*) con el fin de tener una representación de los datos entendible o *human-readable*.

Para nuestra experimentación hemos establecido $N = 75$ y un *learning rate* de $1e-5$ durante 10 *epochs*. Así mismo, el ajuste de este modelo lo realizamos con una partición de entrenamiento conformada por dos tercios del conjunto *botnet* Neris. Por otra parte la generación la realizamos utilizando una partición de test con el resto de flujos de red.

Capítulo 5

Resultados y evaluación

Para evaluar el rendimiento del sistema propuesto, hemos propuesto la comparación analítica y visual entre el conjunto original de datos y el generado por el sistema propuesto. Podemos observar, en la *Fig. 5.1*, como los grados medios, de entrada y de salida de los nodos son menores en el conjunto de datos generado en comparación con el original. Sin embargo la densidad del grafo generado en ambos casos es exactamente igual *Figs. 5.2* por otro lado, la topología de red original y la generada, difieren principalmente en el número de nodos generados y no en tanta medida en el rol de estos. Dicho esto, analíticamente, podemos afirmar que la generación se realiza de forma correcta. Este hecho se observa en las *Figs. 5.3 y 5.4* para el conjunto de datos original y generado, respectivamente, dónde el número de *bots* (nodos coloreados en verde) y servidores C&C (coloreados en rojo) son claramente menores. Más en detalle, a través de las *Figs. 5.5 y 5.6*, vemos el papel *botmaster* (coloreado en amarillo) y sus conexiones principales con los nodos C&C. Concluimos así, como el conjunto de datos generado y, por ende el sistema en sí, es capaz de caracterizar los tres principales actores de la *botnet* Neris: *bots*, servidores C&C y el/los *botmasters*.

Analizando las variables continuas, comparando tanto la media como la desviación estándar de las muestras generadas con las muestras originales *Cua. 5.1* podemos ver como la solución implementada no genera de una forma adecuada las datos continuos. Gran parte de los valores tienen valor 0, este comportamiento se repite con variables que representan valores binarios como pueden ser los puertos, los *flags* TCP y el protocolo.

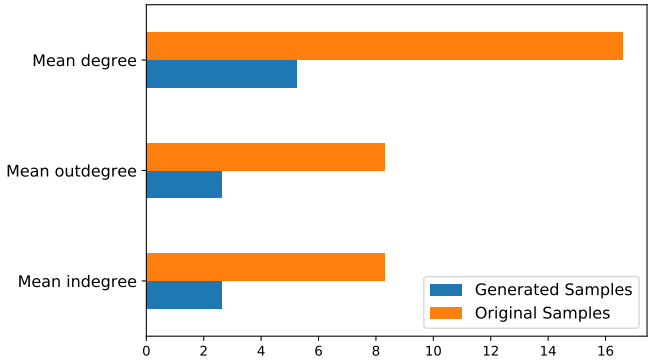

Figura 5.1: Grado medio de los nodos de red (tanto de la topología original como de la generada).

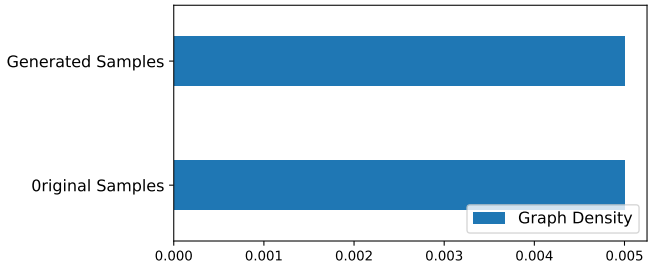

Figura 5.2: Densidad del grafo de red (tanto de la topología original como de la generada).

	Original Samples			**Generated Samples**		
	td	pkt	byt	td	pkt	byt
count	67260	67260	67260	1614225	1614225	1614225
mean	12.028684	3.305798	493.203999	0	0	0.6256972
std	97.02572	6.612446	6224.418826	0	0	7.222679
min	0	1	28	0	0	0
25 %	0	1	70	0	0	0
50 %	0	1	86	0	0	0
75 %	8.914	6	288	0	0	0
max	2691.891	256	322938	0	0	84

Cuadro 5.1: Comparación del conteo, media, desviación estándar y cuartiles de los flujos originales y los generados.

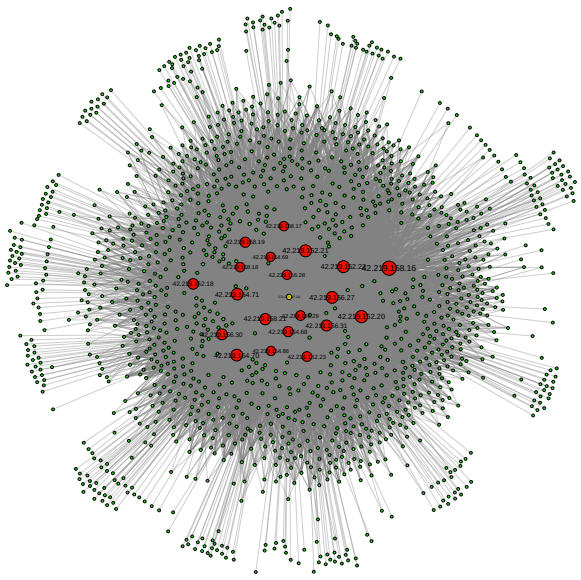

Figura 5.3: Topología de la partición test de *botnet* Neris. Se muestra el nodo botmaster en amarillo rodeado por los nodos C&C coloreados en rojo, el resto de nodos coloreados en verde representarn los bots.

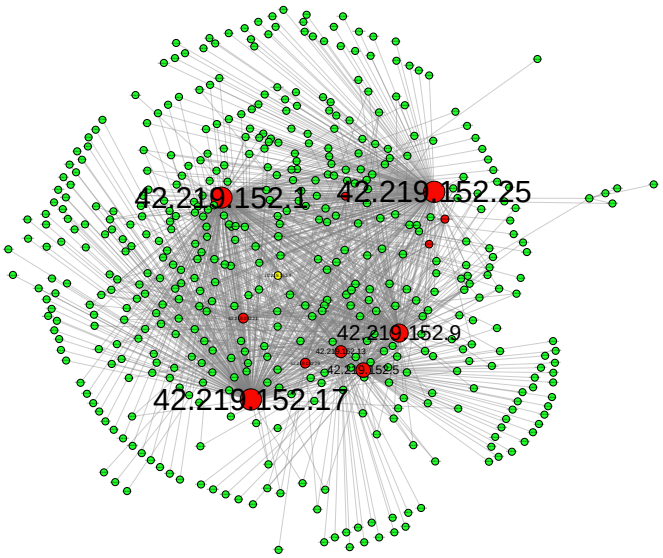

Figura 5.4: Topología generada completa. Se muestra el nodo botmaster en amarillo rodeado por los nodos C&C coloreados en rojo, el resto de nodos coloreados en verde representarn los bots.

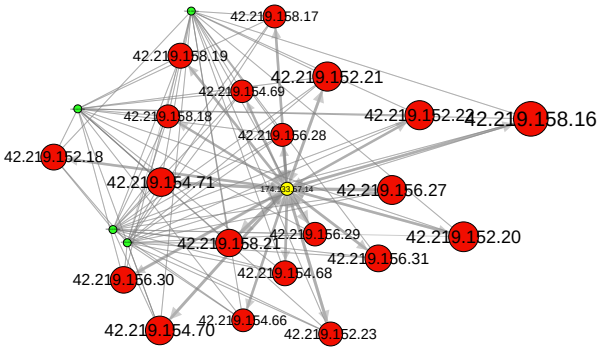

Figura 5.5: Detalle de la topología de test. Se muestra el nodo *botmaster* en amarillo rodeado por los nodos C&C coloreados en rojo. El tamaño de los nodos representa su grado: a mayor tamaño mayor grado.

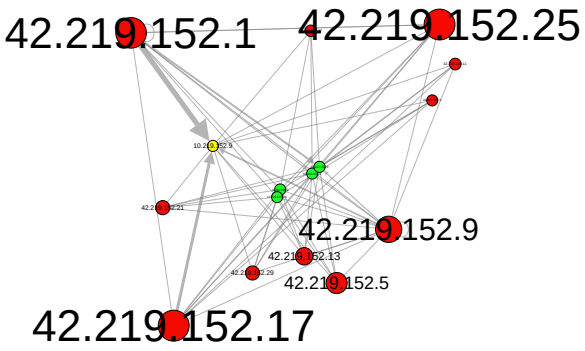

Figura 5.6: Detalle de la topología generada. Se muestra el nodo *botmaster* en amarillo rodeado por los nodos C&C coloreados en rojo. El tamaño de los nodos representa su grado: a mayor tamaño mayor grado.

Capítulo 6

Conclusiones y trabajo futuro

Hoy en día, los dispositivos, las cosas y las personas están interconectas e intercambian grandes volúmenes de información. Desde el punto de vista de la seguridad, este escenario promueve la generación de nuevas vulnerabilidades que pueden ser explotadas por atacantes debido a debido al continuo aumento de la superficie de ataque. Por ello, es necesario idear soluciones inteligentes, robustas y automáticas como defensa ante amenazas y ataques a la seguridad conocidos como no conocidos, especialmente para estos últimos. Son muchas las técnicas y algoritmos propuestos en la Literatura, destacando los sistemas NIDS basados en modelos ML o DL. Sin embargo, dichos modelos dependen del uso de conjuntos de datos predefinidos que, en su mayoría, están obsoletos, son poco realistas o no tienen una duración suficiente para representar con precisión el contexto o escenario de uso.

En este trabajo hemos propuesto una solución basada en VAE que es capaz de imitar con precisión y en gran medida la estructura y topología de los flujos de red de, a priori, diferentes tipologías de ataque. En particular, se ha reproducido el comportamiento de la *botnet* Neris, en la que los roles de los nodos implicados también están bien caracterizados. En comparación con las soluciones del estado del arte, nuestro trabajo maneja inteligentemente las direcciones IP, como variables categóricas, obviadas normalmente en la Literatura. Eludir el uso de información categórica de los conjuntos de datos de red reduce los datos contextuales del entorno útiles para la detección y clasificación de ataques.

Como trabajo futuro, se debe realizar más experimentación para concluir firmemente la relevancia de usar VAE o algunas otras técnicas basadas en DL para imitar con precisión topologías de red. Adicionalmente esta experimentación debe venir acompañada de una re-evaluación de la representación de los datos que se usan como *input* para el modelo. Esto viene dado por el objetivo de generar observaciones de red completas que incluyan todas

las variables originales y no sólo las que definen la topología de la red, como las direcciones IP. De esta forma, también sería interesante el estudio y la aplicación de modelos de DL más complejos. Por ejemplo, el uso de las sofisticadas *attention layers* para la creación de modelos *transformers* [46] ha demostrado ser una alternativa a las capas LSTM debido a su mayor facilidad de entrenamiento [47]. Finalmente, se evaluará el impacto en el en el rendimiento, robustez y fiabilidad de los sistemas NIDS cuando son entrenados con conjuntos de datos aumentados con esta y otros técnicas.

Apéndice A

Código desarrollado

En esta sección se muestra la implementación de la función de pérdida del modelo así como la clase que define las métricas y las funciónes de inferencia, *train step* y *test step*.

A.1. Implementación de la clase del modelo

```
@tf.function()
def VAE_loss(data, reconstruction, z_mean, z_log_var, z):

    mae = tf.keras.losses.MeanSquaredError()
    reconstruction_loss = tf.math.reduce_sum(mae(data,
    reconstruction))

    kl = tf.keras.losses.KLDivergence(reduction=tf.keras.losses
    .Reduction.SUM)
    kl_loss = kl(z_mean, z_log_var)

    total_loss = reconstruction_loss + kl_loss

    return reconstruction_loss, kl_loss, total_loss

class VAE(keras.Model):

    def __init__(self, encoder, decoder, **kwargs):
        super(VAE, self).__init__(**kwargs)
        self.encoder = encoder
        self.decoder = decoder
        self.total_loss_tracker = keras.metrics.Mean(name="
    total_loss")
        self.reconstruction_loss_tracker = keras.metrics.
            Mean(name="reconstruction_loss")

        self.kl_loss_tracker = keras.metrics.Mean(name="kl_loss
    ")
        self.accuracy_tracker = keras.metrics.
```

```
26              BinaryAccuracy(name="accuracy")
27
28      def pred(data):
29          z_mean, z_log_var, z = self.encoder(data)
30          return self.decoder(z)
31
32      @property
33      def metrics(self):
34          return [
35              self.total_loss_tracker,
36              self.reconstruction_loss_tracker,
37              self.kl_loss_tracker,
38              self.accuracy_tracker,
39          ]
40
41      @tf.function()
42      def train_step(self, data):
43          with tf.GradientTape() as tape:
44              z_mean, z_log_var, z = self.encoder(data)
45              reconstruction = self.decoder(z)
46
47              reconstruction_loss, kl_loss, total_loss = VAE_loss
    (
48                  data, reconstruction, z_mean, z_log_var, z)
49
50
51          grads = tape.gradient(total_loss, self.
    trainable_weights)
52          self.optimizer.apply_gradients(zip(grads, self.
    trainable_weights))
53
54          self.total_loss_tracker.update_state(total_loss)
55          self.reconstruction_loss_tracker.update_state(
    reconstruction_loss)
56          self.kl_loss_tracker.update_state(kl_loss)
57          self.accuracy_tracker.update_state(data,reconstruction)
58
59          return {
60              "loss": self.total_loss_tracker.result(),
61              "reconstruction_loss": self.
    reconstruction_loss_tracker.result(),
62              "kl_loss": self.kl_loss_tracker.result(),
63              "accuracy": self.accuracy_tracker.result(),
64          }
65
66      def test_step(self,data):
67          z_mean, z_log_var, z = self.encoder(data)
68          reconstruction = self.decoder(z)
69          reconstruction_loss, kl_loss, total_loss = VAE_loss(
70              data, reconstruction, z_mean, z_log_var, z)
71
72          self.total_loss_tracker.update_state(total_loss)
73          self.reconstruction_loss_tracker.update_state(
    reconstruction_loss)
```

```
74          self.kl_loss_tracker.update_state(kl_loss)
75          self.accuracy_tracker.update_state(data,reconstruction)
76          return {
77              "loss": self.total_loss_tracker.result(),
78              "reconstruction_loss": self.
    reconstruction_loss_tracker.result(),
79              "kl_loss": self.kl_loss_tracker.result(),
80              "accuracy": self.accuracy_tracker.result(),
81          }
```

A.2. Implementación de la capa de muestreo, codificador y decodificador VAE

En esta sección se muestra, por un lado, la implementación de la función que obtiene muestras del espacio latente probabilístico y, por otro, la implementación de la estructura del codificador y del decodificador del VAE.

```
1
2  class Sampling(keras.layers.Layer):
3      """Uses (z_mean, z_log_var) to sample z, the vector
    encoding a digit."""
4      """https://keras.io/examples/generative/vae/"""
5
6      def call(self, inputs):
7          z_mean, z_log_var = inputs
8          epsilon = tf.keras.backend.random_normal(
9              shape=(tf.shape(z_mean)[0], tf.shape(z_mean)[1])
10         )
11         return (tf.exp(z_log_var * 0.5) * epsilon) + z_mean
12
13 try:
14     with tf.device('/device:GPU:0'):
15
16         cols = 122
17         latent_dim = int(cols//2.5)*50
18
19         # CODIFICADOR
20
21         encoder_inputs = keras.Input(shape=(75,cols))
22         x = Dense(cols, activation="relu")(encoder_inputs)
23         x = LSTM(cols, activation='relu', return_sequences=True
    , name='lstm1')(x)
24         x = LSTM(int(cols//1.3), activation="relu",
    return_sequences=True, name='lstm2')(x)
25         x = LSTM(int(cols//2), activation="relu",
    return_sequences=True, name='lstm3')(x)
26
27         x = Dense(int(cols//2.5), activation="relu", name='
    Dense1')(x)
28         x = tf.keras.layers.Flatten(name='Netflow_burst')(x)
```

```
29
30          z_mean = Dense(latent_dim, name="z_mean")(x)
31          z_log_var = Dense(latent_dim, name="z_log_var")(x)
32          z = Sampling()([z_mean, z_log_var])
33
34          VAE_encoder = keras.Model(encoder_inputs, [z_mean,
    z_log_var, z], name="encoder")
35          VAE_encoder.summary()
36
37          # DECODIFICADOR
38
39          latent_inputs = keras.Input(shape=(latent_dim,))
40          x = Dense(latent_dim, activation="relu", name='Dense1')
    (latent_inputs)
41          #x = tf.keras.layers.LeakyReLU()(x)
42          x = Dense(75*int(cols//2.5), activation="relu", name='
    Dense2')(x)
43
44          x = tf.keras.layers.Reshape((75,int(cols//2.5)))(x)
45
46
47          x = LSTM(int(cols//2), activation="relu",
    return_sequences=True, name='lstm1')(x)
48          x = LSTM(int(cols//1.3), activation="relu",
    return_sequences=True, name='lstm2')(x)
49          x = LSTM(cols, activation='relu', return_sequences=True
    , name='lstm3')(x)
50
51          decoder_outputs = Dense(cols, activation="sigmoid")(x)
52
53          VAE_decoder = keras.Model(latent_inputs,
    decoder_outputs, name="decoder")
54          VAE_decoder.summary()
55  except RuntimeError as e:
56      print(e)
```

A.3. Funciones de preprocesado

En esta sección de muestran las funciones con las cuales se realiza el preprocesado de las muestras.

```
# Normalizacion entre 0 y 1
#  Devuelve los valores m ximos
#  y m nimos para poder reconstruirlos
def min_max_normalization(column):
    min_max = {
        'min': column.min(),
        'max': column.max()
    }

    return (column-column.min())/(column.max() - column.min()),
     min_max


# Deshace la normalizacion entre 0 y 1
def denormalize(column, min_max):
    return column*(min_max['max']- min_max['min'])+min_max['min
    ']


# Funcion auxiliar, devuelve
#  un string n veces con el
#   sufijo i (string+'i')
def get_names(name, n):
    names=[]
    for i in range(n):
        names.append(name+str(i))
    return names


# Convierte un puerto en el formato
#  decimal a su formato de 16 bits
def bin_port(df, name):
    df = pd.DataFrame(df.apply(lambda port: bin(int(port))[2:].
    zfill(16)))
    df = pd.DataFrame(df.to_numpy().astype('U16').view('U1').
    astype(int))

    names=[]
    for i in range(16):
        names.append(name+str(i))

    df.columns = names
    print("bin_port " + name + " completed")
    return df


# Convierte una direccion IPv4 en el formato
#  decimal a su formato de 32 bits
```

```
45 def bin_addresses(df, name):
46     df = pd.DataFrame(df.apply(lambda ip: bin(int(ipaddress.
       ip_address(ip)))[2:].zfill(32) ) )
47     df = pd.DataFrame(df.to_numpy().astype('U32').view('U1').
       astype(int))
48
49     names=[]
50     for i in range(32):
51         names.append(name+str(i))
52
53     df.columns = names
54     print("bin_adress " + name + " completed")
55     return df
56
57
58 # Convierte el valor de un protocolo
59 #  de hexadecimal a binario
60 def get_bin_pr(hex_value):
61     return bin(hex_value)[2:].zfill(8)
62
63
64 # Convierte un string de un protocolo
65 #  determinado a binario
66 def bin_pr(df):
67     df =  df.replace({ 'ICMP': get_bin_pr(0X01),
68                        'TCP': get_bin_pr(0X06),
69                        'UDP': get_bin_pr(0X11), })
70     df = pd.DataFrame(df.str.split('', n=8, expand=True).drop(
       columns=[0]))
71
72     names=[]
73     for i in range(8):
74         names.append('pr'+str(i))
75
76     df.columns = names
77     print("bin_pr completed")
78     return df
79
80
81 # Convierte el string de TCP flags
82 #  a binario
83 def bin_flag(df): ### UAPRSF dividir en 6 columnas, si es un
       punto 0 si es otra cosa 1
84     df = pd.DataFrame(df.str.split('',n=6,expand=True)).drop(
       columns=[0]
85         ).replace({'.': 0, 'U': 1, 'A': 1, 'P': 1, 'R': 1, 'S':
        1, 'F': 1,  })
86     df.columns = ['U','A','P','R','S','F']
87     print("bin_flag completed")
88     return df
89
90
91 # Convierte ToS de decimal
92 #  a binario
```

```
93 def bin_ToS(df, name):
94     df = pd.DataFrame(df.apply(lambda ToS: bin(int(ToS))[2:].
       zfill(8)))
95     df = pd.DataFrame(df.to_numpy().astype('U8').view('U1').
       astype(int))
96
97     names=[]
98     for i in range(8):
99         names.append(name+str(i))
100
101    df.columns = names
102    print("bin_ToS completed")
103    return df
104
105
106 # Funcion principal de preprocesado
107 def df_preprocessing(df):
108
109    df = df.reset_index()
110
111    df['td'], td_min_max = min_max_normalization(df['td'])
112    print("td normalized")
113    df['pkt'], pkt_min_max = min_max_normalization(df['pkt'])
114    print("pkt normalized")
115    df['byt'], byt_min_max = min_max_normalization(df['byt'])
116    print("byt normalized")
117
118    min_max = {}
119
120    min_max.update({'td': td_min_max})
121    min_max.update({'pkt': pkt_min_max})
122    min_max.update({'byt': byt_min_max})
123
124    return pd.concat([
125                      df['te'],
126                      df['td'],
127                      bin_addresses(df['sa'],'sa'),
128                      bin_addresses(df['da'], 'da'),
129                      bin_port(df['sp'],'sp'),
130                      bin_port(df['dp'],'dp'),
131                      bin_pr(df['pr']),
132                      bin_flag(df['flg']),
133                      df['fwd'],
134                      bin_ToS(df['stos'],'stos'),
135                      df['pkt'],
136                      df['byt'],
137                      df['label']],
138                      axis='columns'), min_max
```

A.4. Funciones de post-procesado

En esta sección se muestra el post-procesado de la salida del VAE para la obtención de muestras *human-readable*.

```
# Convierte una IPv4 binaria
#  en su formato human-readable
def bin_addresses_deprocess(df, name):
    names = get_names(name,32)
    intAdresses = pd.DataFrame()

    intAdresses[1] = df[names[0:8]].apply(lambda row: int(''.
    join(row.values.astype(str)),2), axis=1)
    intAdresses[2] = df[names[8:16]].apply(lambda row: int(''.
    join(row.values.astype(str)),2), axis=1)
    intAdresses[3] = df[names[16:24]].apply(lambda row: int(''.
    join(row.values.astype(str)),2), axis=1)
    intAdresses[4] = df[names[24:32]].apply(lambda row: int(''.
    join(row.values.astype(str)),2), axis=1)
    Adresses = pd.DataFrame()
    Adresses[name] = intAdresses.apply(lambda row: '.'.join(row
    .values.astype(str)), axis=1)

    print(name + "deprocess completed")

    return Adresses


# Convierte un puerto de binario
#  a formato human-readable
def bin_port_deprocess(df, name):
    intPort = pd.DataFrame()
    intPort[name] = df[get_names(name,16)].apply(lambda row:
    int(''.join(row.values.astype(str)),2), axis=1)

    print(name + "deprocess completed")

    return intPort


# Convierte un protocolo de binario
#  a formato human-readable
def bin_pr_deprocess(df):
    names = get_names('pr',8)

    intPr = pd.DataFrame()
    intPr['pr'] = df[names].apply(lambda row: int(''.join(row.
    values.astype(str)),2), axis=1)

    print("pr deprocess completed")

    return intPr['pr'].replace({ 1: 'ICMP', 6: 'TCP', 17: 'UDP'
    , })
```

```
41
42
43 # Convierte las flags TCP binarias
44 #  a su formato human-readable
45 def bin_flg_deprocess(df):
46     return (df['U'].replace({ 1: 'U', 0: '.', }) +
47             df['A'].replace({ 1: 'A', 0: '.', }) +
48             df['P'].replace({ 1: 'P', 0: '.', }) +
49             df['R'].replace({ 1: 'R', 0: '.', }) +
50             df['S'].replace({ 1: 'S', 0: '.', }) +
51             df['F'].replace({ 1: 'F', 0: '.', }) )
52
53
54 # Convierte ToS de formato binario
55 #  a su formato human-readable
56 def bin_ToS_deprocess(df):
57
58     df =  pd.DataFrame( df[get_names('stos',8)].apply(lambda
       row: int(''.join(row.values.astype(str)),2), axis=1),
       columns=['stos'])
59
60     print("stos deprocess completed")
61
62
63     return df
64
65
66 # Funcion principal de deprocesado
67 def df_deprocessing(df, min_max):
68
69     df['td'] = denormalize(df['td'],min_max['td'])
70     df['pkt'] = denormalize(df['pkt'],min_max['pkt'])
71     df['byt'] = denormalize(df['byt'],min_max['byt'])
72
73
74     return pd.concat([
75             df['td'],
76             bin_addresses_deprocess(df.astype(int), 'sa'),
77             bin_addresses_deprocess(df.astype(int), 'da'),
78             bin_port_deprocess(df.astype(int), 'sp'),
79             bin_port_deprocess(df.astype(int), 'dp'),
80             bin_pr_deprocess(df.astype(int)),
81             bin_flg_deprocess(df.astype(int)),
82             df['fwd'],
83             bin_ToS_deprocess(df.astype(int)),
84             df['pkt'],
85             df['byt'],
86              ], axis=1)
```

Bibliografía

[1] G. Maciá-Fernández, J. Camacho, R. Magán-Carrión, P. García-Teodoro, and R. Therón, "UGR'16: A new dataset for the evaluation of cyclostationarity-based network IDSs," *Comput. Secur.*, vol. 73, pp. 411–424, 2018.

[2] Cisco, "Cisco Annual Internet Report (2018–2023) White Paper," https://bit.ly/3jpAgNx, 2020.

[3] ENISA, "ENISA Threat Landscape (2020) White Paper," https://www.enisa.europa.eu/news/enisa-news/enisa-threat-landscape-2020, 2020, [Online; Accessed **?today?**].

[4] R. Magán-Carrión, D. Urda, I. Diaz-Cano, and B. Dorronsoro, "Towards a reliable comparison and evaluation of network intrusion detection systems based on machine learning approaches," *Applied Sciences*, vol. 10, no. 5, 2020.

[5] K. Benzekki, A. El Fergougui, and A. Elbelrhiti Elalaoui, "Software-defined networking (sdn): a survey," *Security and Communication Networks*, vol. 9, no. 18, pp. 5803–5833, 2016. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/sec.1737

[6] A. Laurito, M. Bonaventura, M. E. Pozo Astigarraga, and R. Castro, "Topogen: A network topology generation architecture with application to automating simulations of software defined networks," in *2017 Winter Simulation Conference (WSC)*, 2017, pp. 1049–1060.

[7] H. Haddadi, M. Rio, G. Iannaccone, A. Moore, and R. Mortier, "Network topologies: inference, modeling, and generation," *IEEE Communications Surveys Tutorials*, vol. 10, no. 2, pp. 48–69, 2008.

[8] J. P. Sterbenz, E. K. Çetinkaya, M. A. Hameed, A. Jabbar, S. Qian, and J. P. Rohrer, "Evaluation of network resilience, survivability, and disruption tolerance: analysis, topology generation, simulation, and experimentation," *Telecommunication systems*, vol. 52, no. 2, pp. 705–736, 2013.

[9] M. Mohri, A. Rostamizadeh, and A. Talwalkar, *Foundations of machine learning*. MIT press, 2018.

[10] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[11] H. He, Y. Bai, E. A. Garcia, and S. Li, "Adasyn: Adaptive synthetic sampling approach for imbalanced learning," in *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, 2008, pp. 1322–1328.

[12] S. Barua, M. M. Islam, X. Yao, and K. Murase, "Mwmote–majority weighted minority oversampling technique for imbalanced data set learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 2, pp. 405–425, 2014.

[13] L. Vu, C. T. Bui, and Q. U. Nguyen, "A deep learning based method for handling imbalanced problem in network traffic classification," in *Proceedings of the Eighth International Symposium on Information and Communication Technology*, ser. SoICT 2017. New York, NY, USA: Association for Computing Machinery, 2017, p. 333–339. [Online]. Available: https://doi.org/10.1145/3155133.3155175

[14] R. Alshammari and A. N. Zincir-Heywood, "Can encrypted traffic be identified without port numbers, ip addresses and payload inspection?" *Computer networks*, vol. 55, no. 6, pp. 1326–1350, 2011.

[15] J. Engelmann and S. Lessmann, "Conditional wasserstein gan-based oversampling of tabular data for imbalanced learning," *Expert Systems with Applications*, vol. 174, p. 114582, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417421000233

[16] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[17] V. A. Fajardo, D. Findlay, C. Jaiswal, X. Yin, R. Houmanfar, H. Xie, J. Liang, X. She, and D. Emerson, "On oversampling imbalanced data with deep conditional generative models," *Expert Systems with Applications*, vol. 169, p. 114463, 2021.

[18] S. K. Lim, Y. Loo, N.-T. Tran, N.-M. Cheung, G. Roig, and Y. Elovici, "Doping: Generative data augmentation for unsupervised anomaly detection with gan," in *2018 IEEE International Conference on Data Mining (ICDM)*, 2018, pp. 1122–1127.

[19] D. Dablain, B. Krawczyk, and N. V. Chawla, "Deepsmote: Fusing deep learning and smote for imbalanced data," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–15, 2022.

[20] "Nmap network scanning," https://nmap.org, accessed: 2022-06-06.

[21] C. G. Cordero, E. Vasilomanolakis, N. Milanov, C. Koch, D. Hausheer, and M. Mühlhäuser, "Id2t: A diy dataset creation toolkit for intrusion detection systems," in *2015 IEEE Conference on Communications and Network Security (CNS)*, 2015, pp. 739–740.

[22] J. Camacho, A. Pérez-Villegas, P. García-Teodoro, and G. Maciá-Fernández, "Pca-based multivariate statistical network monitoring for anomaly detection," *Computers Security*, vol. 59, pp. 118–137, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404816300116

[23] S. Lab, "CTU-13 Dataset. Captura 42. Neris botnet." 2011.

[24] S. R. Laboratory, "Stratosphere research laboratory," https://www.stratosphereips.org/.

[25] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[26] P. Baldi, "Autoencoders, unsupervised learning, and deep architectures," in *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*, ser. Proceedings of Machine Learning Research, I. Guyon, G. Dror, V. Lemaire, G. Taylor, and D. Silver, Eds., vol. 27. Bellevue, Washington, USA: PMLR, 02 Jul 2012, pp. 37–49. [Online]. Available: https://proceedings.mlr.press/v27/baldi12a.html

[27] S. Raschka. Birmingham: Packt Publishing, Limited, 2019.

[28] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.1127647

[29] A. Maćkiewicz and W. Ratajczak, "Principal components analysis (pca)," *Computers Geosciences*, vol. 19, no. 3, pp. 303–342, 1993. [Online]. Available: https://www.sciencedirect.com/science/article/pii/009830049390090R

[30] R. Salakhutdinov and G. Hinton, "Learning a nonlinear embedding by preserving class neighbourhood structure," in *Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research,

M. Meila and X. Shen, Eds., vol. 2. San Juan, Puerto Rico: PMLR, 21–24 Mar 2007, pp. 412–419. [Online]. Available: https://proceedings.mlr.press/v2/salakhutdinov07a.html

[31] C. Zhou and R. C. Paffenroth, "Anomaly detection with robust deep autoencoders," in *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*. New York, NY, USA: Association for Computing Machinery, 2017. [Online]. Available: https://doi.org/10.1145/3097983.3098052

[32] A. Heljakka, A. Solin, and J. Kannala, "Towards photographic image manipulation with balanced growing of generative autoencoders," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, March 2020.

[33] K. Preechakul, N. Chatthee, S. Wizadwongsa, and S. Suwajanakorn, "Diffusion autoencoders: Toward a meaningful and decodable representation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2022, pp. 10 619–10 629.

[34] Y. Zhou and B. E. Shi, "Photorealistic facial expression synthesis by the conditional difference adversarial autoencoder," in *2017 Seventh International Conference on Affective Computing and Intelligent Interaction (ACII)*, 2017, pp. 370–376.

[35] H. GM, M. K. Gourisaria, M. Pandey, and S. S. Rautaray, "A comprehensive survey and analysis of generative models in machine learning," *Computer Science Review*, vol. 38, p. 100285, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1574013720303853

[36] D. Foster, *Generative Deep Learning*. O'Reilly Medias, 2019.

[37] K. Gregor, I. Danihelka, A. Graves, D. Rezende, and D. Wierstra, "Draw: A recurrent neural network for image generation," in *Proceedings of the 32nd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, F. Bach and D. Blei, Eds., vol. 37. Lille, France: PMLR, 07–09 Jul 2015, pp. 1462–1471. [Online]. Available: https://proceedings.mlr.press/v37/gregor15.html

[38] L. Deng, "The mnist database of handwritten digit images for machine learning research," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[39] A. Razavi, A. van den Oord, and O. Vinyals, "Generating diverse high-fidelity images with vq-vae-2," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer,

F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32. Curran Associates, Inc., 2019. [Online]. Available: https://proceedings.neurips.cc/paper/2019/file/5f8e2fa1718d1bbcadf1cd9c7a54fb8c-Paper.pdf

[40] A. Roberts, J. Engel, and D. Eck, Eds., *Hierarchical Variational Autoencoders for Music*, 2017. [Online]. Available: https://nips2017creativity.github.io/doc/Hierarchical_Variational_Autoencoders_for_Music.pdf

[41] W. Yan, Y. Zhang, P. Abbeel, and A. Srinivas, "Videogpt: Video generation using VQ-VAE and transformers," *CoRR*, vol. abs/2104.10157, 2021. [Online]. Available: https://arxiv.org/abs/2104.10157

[42] B. Bakker, "Reinforcement learning with long short-term memory," in *Advances in Neural Information Processing Systems*, T. Dietterich, S. Becker, and Z. Ghahramani, Eds., vol. 14. MIT Press, 2001.

[43] K. Greff, R. K. Srivastava, J. Koutnik, B. R. Steunebrink, and J. Schmidhuber, "LSTM: A search space odyssey," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 10, pp. 2222–2232, oct 2017. [Online]. Available: https://doi.org/10.1109%2Ftnnls.2016.2582924

[44] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to Forget: Continual Prediction with LSTM," *Neural Computation*, vol. 12, no. 10, pp. 2451–2471, 10 2000. [Online]. Available: https://doi.org/10.1162/089976600300015015

[45] S. Kullback and R. A. Leibler, "On Information and Sufficiency," *The Annals of Mathematical Statistics*, vol. 22, no. 1, pp. 79 – 86, 1951. [Online]. Available: https://doi.org/10.1214/aoms/1177729694

[46] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[47] A. Zeyer, P. Bahar, K. Irie, R. Schlüter, and H. Ney, "A comparison of transformer and lstm encoder decoder models for asr," in *2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, 2019, pp. 8–15.