UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

Ingeniería del Software

SecurAI
Una aplicación inteligente para la detección de ataques en tu red

SecurAI
A smart app to detect attacks on your network

Realizado por
Álvaro Valencia Villalón

Tutorizado por
Gabriel Jesús Luque Polo

Departamento
Lenguajes y Ciencias de la Computación
UNIVERSIDAD DE MÁLAGA

MÁLAGA, (Junio 2025)

RESUMEN

En este trabajo fin de grado (TFG) se plantea desarrollar una aplicación que incorpora un IDS (Sistema de Detección de Intrusos) que se ejecutará localmente en tiempo real para varios sistemas operativos.

De hecho, un objetivo de este TFG es facilitar la instalación para todos los usuarios, generando instaladores para que el usuario final no tenga que tener conocimientos avanzados de informática.

El sistema planteado analizará el tráfico de la red para estimar la probabilidad de que estemos siendo atacados. Para ello se estudiará el uso de algunas técnicas de inteligencia artificial y se implementarán diversos algoritmos para notificar al usuario del ataque.

Otro elemento importante es proporcionar un diseño modular que facilite para incorporación de nuevos componentes para detectar ataques adicionales.

SecurAI es el resultado de esta investigación: un software modular, escalable e innovador que incorpora funcionalidades clave como módulos preestablecidos de detección de amenazas, recopilación de estadísticas de la red y simulación de ataques sobre el propio equipo para comprobar en tiempo real el rendimiento de los módulos.

Palabras clave:

Seguridad, Machine Learning, IDS, Sistema Modular

ABSTRACT

In this Final Degree Project (TFG), the goal is to develop an application that incorporates an IDS (Intrusion Detection System) running locally and in real time across multiple operating systems.

One of the main objectives of this project is to simplify installation for all users by generating installers so that the end user does not need advanced computer knowledge.

The proposed system will analyze network traffic to estimate the likelihood of an ongoing attack. To achieve this, the use of certain artificial intelligence techniques will be explored, and various algorithms will be implemented to notify the user of potential threats.

Another important aspect is to provide a modular design that facilitates the addition of new components for detecting additional types of attacks.

SecurAI is the result of this research: a modular, scalable, and innovative software solution that integrates key features such as preconfigured threat detection modules, network statistics collection, and the simulation of attacks on the local system to test module performance in real time.

Keywords:

Security, Machine Learning, IDS, Modular System

ÍNDICE

CAPÍTULO 1
INTRODUCCIÓN

En esta memoria se explican todas las características de SecurAI. Aquí se documenta su funcionamiento, su propósito, sus fortalezas...

También podrá encontrarse aquí toda la información necesaria para desarrollar nuevos módulos para SecurAI, cubriendo ámbitos como requisitos técnicos de los módulos, o el flujo de trabajo recomendado.

Puede descargar SecurAI, y consultar toda su documentación, guías de usuario y código fuente desde https://github.com/Valeaal/SecurAI

1.1 MOTIVACIÓN

En un contexto social en el que los sistemas informáticos cada vez son más frecuentes y manejan información cada vez más importante, es crucial tener un escudo ante, por lo menos, los ataques más comunes que podemos recibir.

Es poco común que los hogares o pequeñas empresas tengan una protección avanzada de sus redes. El uso de herramientas IDS/IPS (Sistema de Detección/Prevención de Intrusos) no es trivial y requieren de un conocimiento avanzado para su instalación y utilización.

Destacar también que la complejidad de los ataques cibernéticos está en claro aumento [1][2], por lo que las técnicas de protección y prevención que hasta ahora han funcionado están en camino de ser ineficaces.

Por ello, es muy importante mantener el desarrollo de tecnologías punteras, accesibles y al nivel de los atacantes, para garantizar que la infraestructura digital de nuestro hogar o negocio no se verá comprometida.

1.2 OBJETIVOS

El objetivo principal de este Trabajo Fin de Grado (TFG) es construir un IDS, un sistema de detección de intrusos capaz de operar en tiempo real, analizando el tráfico de red local para identificar posibles amenazas. Nuestro sistema está diseñado para ejecutarse de forma autónoma en distintos sistemas operativos, con el fin de proporcionar una herramienta accesible que mejore la seguridad en entornos domésticos o profesionales.

El segundo objetivo, por nivel de importancia, es que se usará inteligencia artificial (o IA por sus siglas en ingles). La IA permite superar las limitaciones de los sistemas tradicionales basados exclusivamente en reglas o firmas, proporcionando un enfoque más robusto y dinámico frente a un panorama de ciberamenazas cada vez más complejo.

Más allá de los requisitos de detección de amenazas, queremos un sistema altamente escalable para poder adaptarlo a las necesidades de todos los usuarios, y crear un proyecto colaborativo que no quedara desfasado rápidamente. Nuestro propósito final es que SecurAI fuera útil de verdad.

SecurAI no puede ser útil si la gente no puede usarlo, así que necesitábamos que la aplicación fuera multiplataforma y accesible.

La utilidad real no se define únicamente por la capacidad técnica, sino también por su usabilidad y adaptabilidad.

Desde el principio, entendimos que por muy sofisticado que fuera el sistema de detección de amenazas, el proyecto no alcanzaría su propósito si no era utilizable por cualquier persona interesada en mejorar la seguridad de su red.

1.2.1 DEFENSA DE LA RED

Esta es la principal utilidad de SecurAI: detectar amenazas a nivel de red en tiempo real.

Nuestro proyecto tiene que ser capaz de analizar en directo los paquetes de red que llegan al equipo, analizarlos sin ignorar ninguno, y avisar al usuario si se ha detectado una amenaza.

Esto implica un procesamiento eficiente, continuo y autónomo, sin interferir con el rendimiento general del sistema ni requerir intervención constante del usuario. SecurAI (véase Fig. 1) es un escudo, un sistema detector de ataques personalizable que analiza cada mensaje que le llega a tu equipo y supervisa que todo esté correcto.

Fig 1: SecurAI monitorizando la red

1.2.2 USO DE LA IA

Queremos que SecurAI emplee técnicas de inteligencia artificial para poder suplir una necesidad del mercado. Gracias a su uso, nuestra plataforma podría detectar patrones de ataques más complejos.

Por ejemplo, el artículo de Suri-Oculus [3] destaca cómo la integración de técnicas de inteligencia artificial, como el aprendizaje automático y el aprendizaje profundo, puede mejorar significativamente las capacidades de los sistemas de detección de intrusiones. En este artículo se comenta cómo al aplicar modelos de inteligencia artificial es posible analizar patrones complejos en el tráfico de red, identificar anomalías y detectar amenazas desconocidas que podrían pasar desapercibidas mediante enfoques tradicionales basados en firmas.

Otros artículos científicos, como [4], también destacan cómo la integración de sistemas de detección de intrusiones con la inteligencia artificial puede transformar la ciberseguridad al permitir la creación de sistemas autónomos capaces de aprender y adaptarse rápidamente. Esta combinación proporciona múltiples capas de protección, mejorando la capacidad de detectar amenazas en tiempo real y reduciendo la posibilidad de brechas de seguridad. Además se enfatiza que, al combinar IDS con IA, las organizaciones pueden construir sistemas de defensa dinámicos que se adaptan continuamente a las necesidades cambiantes de seguridad, proporcionando una protección más robusta y eficiente contra ciberataques.

Además del respaldo científico a incorporar la IA este tipo de sistema, también era de especial interés el aprendizaje que se podía conseguir del uso de estas técnicas innovadoras y con gran proyección de futuro, ya que durante el grado se enseñan muy someramente.

1.2.3 SISTEMA MODULAR

La modularidad del proyecto es clave para su escalabilidad. Al hacer cada algoritmo de detección independiente entre sí y del núcleo de SecurAI fomentamos el desarrollo de nuevos módulos.

En una arquitectura como la que tienen este proyecto, el núcleo del sistema representa la parte central, estable y común de la aplicación. Proporciona la lógica fundamental, la infraestructura de comunicación, la gestión de recursos, y los puntos de extensión que permiten que otros componentes (los módulos) se conecten a él.

Este núcleo actúa como plataforma base: define cómo deben estructurarse e interactuar los módulos, orquesta su ciclo de vida y mantiene el control general del sistema, sin contener por sí mismo las funcionalidades específicas de cada uno. La estructura de SercurAI se detallará en el apartado 4.1 Estructura y flujo de la aplicación.

Este enfoque no solo favorece el desarrollo interno, sino que prepara el terreno para la colaboración externa. Como plataforma de código abierto, sabemos que no podemos aspirar a ser tan hábiles como la potencial comunidad de desarrolladores, creando módulos personalizados a diferentes necesidades, y actualizados a nueva amenazas.

Será esa comunidad la que, inspirada por desafíos reales, extienda nuestro alcance con soluciones que aún no imaginamos. Mientras nosotros trazamos el camino inicial, ellos lo bifurcan, lo expanden y lo elevan.

1.2.4 APLICACIÓN MULTIPLATAFORMA

Un problema habitual que se encuentran los usuarios de MAC (entre los que se encuentra el desarrollador de este TFG) es la compatibilidad entre programas, dependencias mal resueltas o plataformas que simplemente no pensaban en usuarios fuera del entorno Windows.

Habitualmente para resolver ese tipo de problemáticas, se proponen alternativas de bajo nivel. Compilar bibliotecas, modificar rutas, instalar herramientas adicionales, todo eso rompe con la experiencia que debería ofrecer cualquier herramienta tecnológica moderna.

Por ello, desde el inicio del desarrollo de SecurAI, un objetivo importante es que sus usuarios no tuvieran que enfrentarse a ese tipo de barreras. Por ello, es una funcionalidad importante que la instalación y uso de la herramienta sea lo más fluido posible, sin importar el sistema operativo que utilicen.

Actualmente, SecurAI está disponible para Windows y macOS, abarcando para este último tanto arquitecturas Intel x86_64 como ARM64 (por ejemplo, en los nuevos chips Apple Silicon). En todos los casos, la aplicación se ejecuta de forma nativa, sin necesidad de emulación ni soluciones intermedias.

1.2.5 APLICACIÓN ACCESIBLE SIN CONOCIMIENTOS

Relacionado con el punto anterior, uno de los pilares fundamentales del diseño de SecurAI es la accesibilidad tecnológica, un objetivo importante era que todo el mundo pudiera usar SecurAI independientemente de su experiencia técnica o sus conocimientos previos en ciberseguridad. SecurAI es un proyecto comprometido con la accesibilidad digital y la inclusión tecnológica.

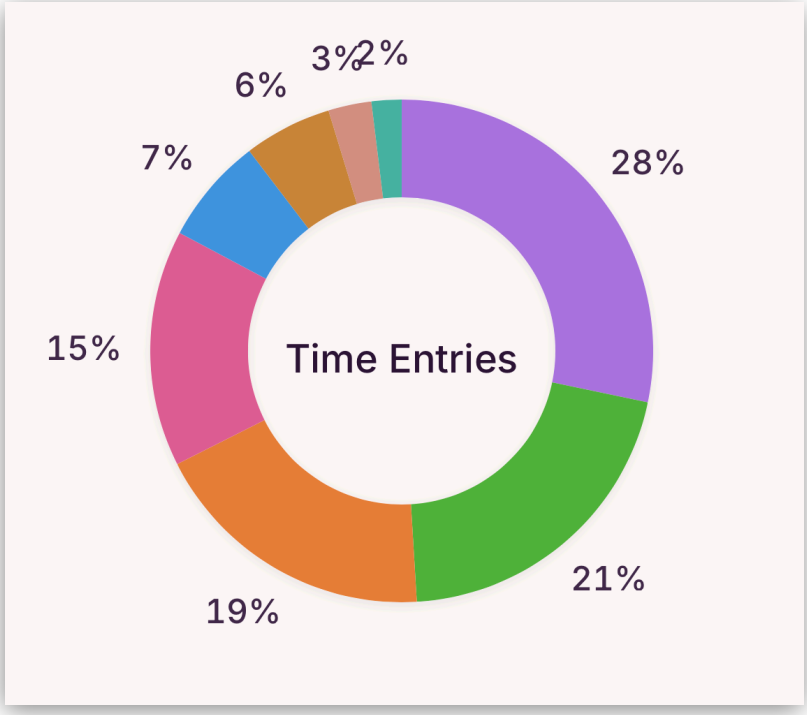
El hecho de que las plataformas actuales no dispongan de interfaz gráfica, y en su mayoría basen su configuración y uso en comandos de terminal [5] aleja de estas herramientas a todo aquel que no tiene experiencia en administración de sistemas informáticos y redes.

1.3 METODOLOGÍA

Para la realización de este proyecto, hemos seguido una metodología iterativa adaptada para una sola persona donde el tutor tuvo el rol de Product Owner.

Siguiendo la filosofía de las metodologías ágiles, las etapas de desarrollo se adaptaron a las necesidades del proyecto: hicimos periodos de desarrollo de módulos de defensa y luego periodos de prueba de forma que nos íbamos asegurando de que el proyecto progresaba de forma adecuada. Por ejemplo, la idea de integrar módulos de ataque surgió de la necesidad de llevar a cabo dichas pruebas, y la interfaz gráfica se desarrolló cuando veíamos necesidad de monitorizar la hebra limpiadora a la vez que observábamos el comportamiento de nuestro primer módulo, arpFlooding.

Además, hemos hecho un seguimiento del tiempo dedicado al proyecto, separado por categorías.



Desglose de porcentaje de tiempo invertido en el proyecto, de mayor porcentaje a menor:

- Desarrollo de algoritmos de defensa
- Memoria y documentación
- Despliegue de la herramienta
- Desarrollo de la WebApp, interfaz e integración
- Otros

Fig 2: Desglose de tiempo

1.4 ORGANIZACIÓN DE LA MEMORIA

En esta memoria se ha plasmado todo lo necesario para permitir que la comunidad open source pueda desarrollar nuevos módulos y ampliar SecurAI. Para nosotros era muy importante plasmar la filosofía del proyecto y la interconexión de todas sus partes, para que el día de mañana su modificación sea sencilla, y el proyecto sea útil.

En este primer capítulo se ha descrito el contexto del proyecto, los objetivos a conseguir y por qué.

En el segundo capítulo se ha hecho un breve repaso al estado del arte, para entender qué fundamentos tecnológicos y herramientas hemos usado en el desarrollo del proyecto.

En el tercer capítulo se encuentra un desglose de los requisitos funcionales y no funcionales a completar para considerar como completo este Trabajo de Fin de Grado.

El cuarto capítulo es bastante extenso, ya que se explica todo el funcionamiento de SecurAI. Por ejemplo se detalla el funcionamiento del *BackEnd* o cómo se limpia la cola de mensajes.

El capítulo se centra en los módulos que vienen implementados ya con SecurAI. Mientras que en el capítulo anterior se explicaba la estructura de la plataforma, aquí se indice en los módulos preinstalados, con esperanza de ayudar a quien quiera desarrollar módulos nuevos.

El capítulo seis está destinado a cómo funciona la funcionalidad de recopilación de estadísticas.

Por último, en el capítulo siete se ubican las conclusiones del proyecto y posibles ampliaciones del mismo.

Se añaden además al final de esta memoria dos apéndices: el manual de usuario de SecurAI en español y en inglés.

Durante toda la memoria se ha utilizado diferentes estilos de escritura para hacer más clara y dinámica la lectura. Por ejemplo, cuando queramos expresar una conclusión, resultado o idea clave, usaremos un párrafo en ***cursiva y en color azul claro***. Si queremos referirnos a un segmento de código o al nombre de un archivo, usaremos un estilo `monoespaciado de color azul oscuro`.

CAPÍTULO 2
FUNDAMENTOS Y TECNOLOGÍAS

En este capítulo se detallarán tanto los conceptos fundamentales necesarios para comprender nuestro desarrollo como las tecnologías clave que lo sustentan. Se explicarán los principios sobre los que se apoya SecurAI, así como las herramientas seleccionadas y su papel dentro del sistema.

2.1 FUNDAMENTOS

Para la realización de SecurAI se han empleado fundamentos tecnológicos bien conocidos, y en auge. Este proyecto ha sido ideado con vistas a futuro, y queríamos que estuviera a la vanguardia. Por ello la integración de técnicas como el aprendizaje supervisado que ahora se explican en más detalle.

2.1.1 CONCEPTOS INICIALES

Primero, ¿qué es la inteligencia artificial (IA)? En la informática, la inteligencia artificial es una disciplina cuyo objetivo es imitar la inteligencia humana para realizar tareas como pueden ser la toma de decisiones, el reconocimiento de patrones o la detección de anomalías.

Para lograr esto, la IA necesita aprender a partir de datos, lo que se conoce como aprendizaje automático o machine learning. En lugar de programar reglas específicas para cada situación, se entrena a la inteligencia artificial con ejemplos para que aprendan a generalizar comportamientos o identificar patrones por sí mismos.

Existen principalmente dos enfoques de aprendizaje dentro del machine learning: El **aprendizaje supervisado** (véase Fig. 3), en el que el modelo aprende a partir de un conjunto de datos (*dataset*)

etiquetado, es decir, donde se conoce el resultado esperado, y el **aprendizaje no supervisado**, en el que los datos no están etiquetados y el modelo debe encontrar patrones o agrupaciones por sí mismo.

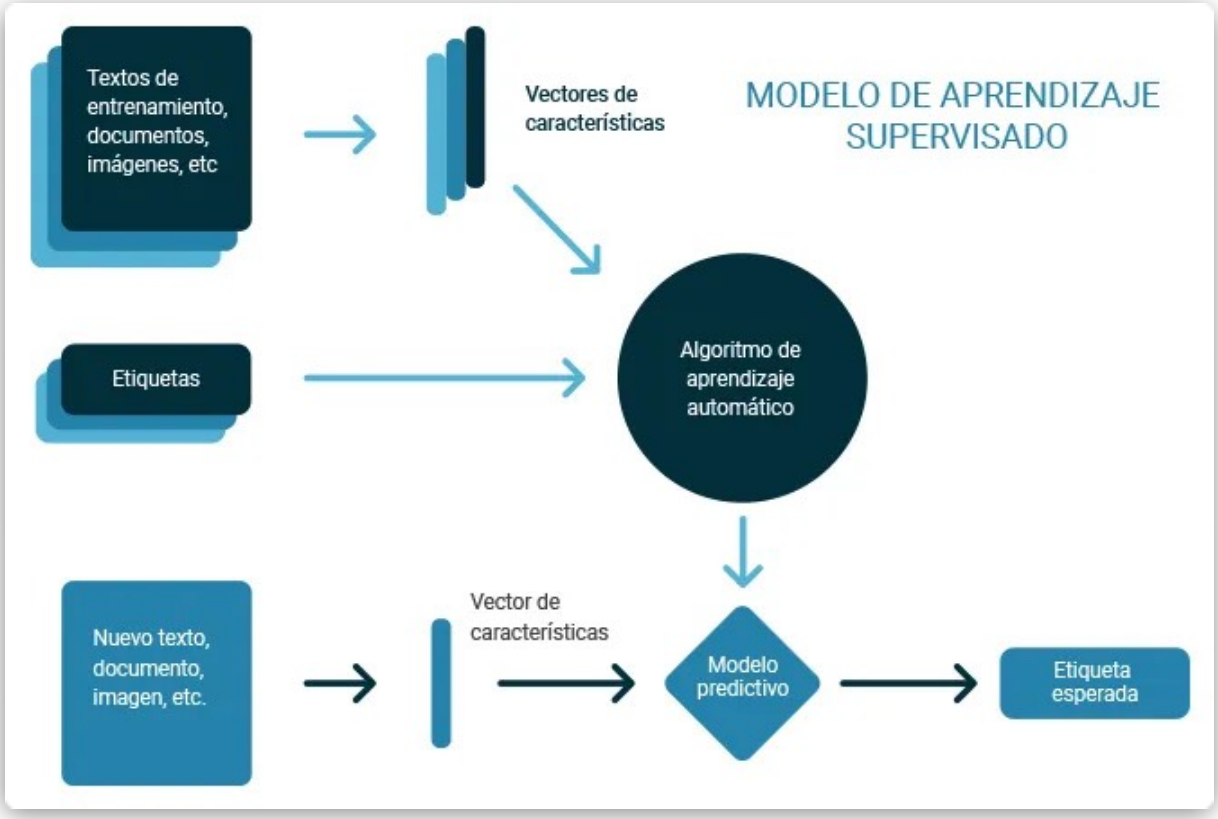


Fig 3: Esquema del modelo de lenguaje supervisado. Imagen de P. Rodríguez.

Concretamente usaremos el primer enfoque, ya que contamos con datos etiquetados que permiten entrenar a los modelos para reconocer patrones específicos de amenazas. Usaremos redes neuronales y máquinas de vectores de soporte (SVM).

Mientras que las primeras son modelos inspirados en el funcionamiento del cerebro humano, las SVM son muy usadas en la clasificación (como es nuestro caso, que clasificamos entre tráfico benigno y maligno) y útiles con datos con muchas variables distintas, como es nuestro caso. Profundizaremos en estos modelos en los apartados 2.1.2 Redes neuronales y 2.1.3 SVM.

Este enfoque de aprendizaje supervisado es ideal para nuestro caso porque nos proporciona un control preciso sobre el entrenamiento, facilitando la generación de modelos con alta capacidad de detección y baja tasa de falsos positivos, pudiendo validar y ajustar el rendimiento del sistema de forma efectiva.

2.1.2 REDES NEURONALES

Para comprender el concepto de red neuronal, es importante comprender el concepto de neurona artificial. Una neurona artificial es una unidad computacional que devuelve un valor cuando la combinación de sus entradas y sus pesos sinápticos (ponderaciones de las entradas) supera cierto umbral, determinado en la llamada función de activación.

Este comportamiento simula de forma abstracta el funcionamiento de una neurona biológica.

Este modelo fue formalizado inicialmente en el perceptrón simple (véase Fig. 4), propuesto por Rosenblatt (1958), que consistía en una única neurona con función de activación escalón [6].

La función escalón actúa como un interruptor: si el resultado de la suma ponderada supera un cierto umbral (habitualmente 0), la salida es 1; en caso contrario, es 0 (o -1).

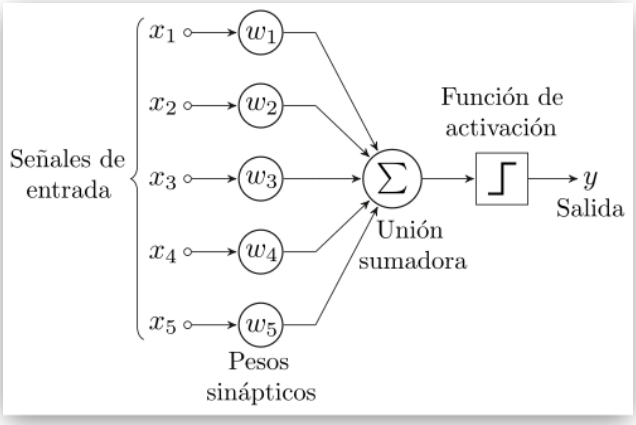


Fig 4: Perceptrón simple

A pesar de su sencillez, sentó las bases para el desarrollo de redes neuronales más complejas, como las redes densas multicapa, permitiendo que la red aprenda a reconocer patrones mucho más complejos.

Una red neuronal está compuesta por muchas neuronas artificiales organizadas en capas, que son capaces de aprender patrones tras un entrenamiento.

¿Qué es el entrenamiento? Entrenar una red neuronal significa enseñarle cómo dar buenas respuestas. Las redes neuronales aprenden procesando varios conjuntos grandes de datos ya etiquetados (clasificados) o sin etiquetar. Lo que se realiza exactamente es el ajuste de los pesos correspondientes a las entradas de cada una de las neuronas de entrada.

La respuesta en las neuronas de la capa de salida debe ser lo más parecida posible a los valores esperados y así conseguir que el error cometido sea el mínimo. El proceso de entrenamiento implica el ajuste iterativo de los pesos sinápticos para conseguirlo, y para ello se usa un algoritmo llamado retropropagación del error (*backpropagation*), que permite distribuir el error cometido en la salida

hacia las capas anteriores y calcular cómo modificar los pesos de manera eficiente, utilizando técnicas de optimización como el descenso del gradiente [7].

Al principio todas las entradas tienen valores iniciales sin ajustar y la respuesta de la red puede no tener mucho sentido. Durante el entrenamiento el modelo va aprendiendo y proporcionará respuestas cada vez más adecuadas [8].

2.1.2.1 REDES NEURONALES DENSAS

Una red neuronal densa está compuesta por muchas neuronas artificiales conectadas entre capas (véase Fig. 5). Cada neurona de una capa oculta tiene como entrada todas las neuronas de la capa anterior y su salida se conecta con todas las neuronas de la siguiente capa.

Para cada entrada en una capa densa, se calcula una operación lineal a partir de la entradas y pesos, generalmente seguida de una función de activación no lineal, como la activación o no a partir del umbral definido.

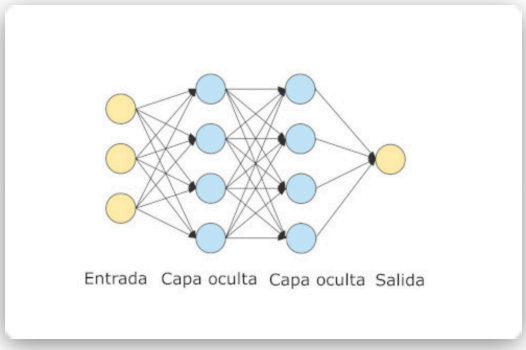


Fig 5: Red neuronal densa

El número de conexiones entre dos capas densas consecutivas es el número de neuronas de la primera por el número de neuronas de la siguiente capa. En consecuencia, el número de parámetros puede ser extremadamente alto para que el ordenador entrene el modelo en un período de tiempo razonable.

Otro problema común que encuentran las capas densas es que son propensas a sobreajustar los datos, es decir, el hecho de que los modelos de aprendizaje automático tienden a funcionar peor en los nuevos datos que en sus datos de entrenamiento. Sobreentrenar la red causa poca precisión en las nuevas entradas, ya que los pesos y sesgos se han ajustado excesivamente cerca de los datos de entrenamiento, lo que hace que las siguientes actualizaciones sean inflexibles y menos precisas [9].

2.1.2.2 LSTM

Las redes LSTM (Long Short-Term Memory) son un tipo especial de red neuronal que es capaz de recordar información durante más tiempo, algo que las redes normales no hacen bien. Están diseñadas para evitar que el modelo "olvide" datos importantes a medida que procesa secuencias (como frases o series temporales). Para lograrlo, usan un "canal de memoria" que guarda lo

importante a largo plazo, y lo van actualizando con tres mecanismos: la puerta de olvido, que decide qué borrar; la puerta de entrada, que elige qué nueva información guardar; y la puerta de salida, que genera la salida de la red en ese momento (véase Fig. 6).

Gracias a esta estructura, las LSTM funcionan muy bien en tareas como traducción automática, análisis de sentimientos o generación de texto, donde es clave entender el contexto y el orden de las palabras [10].

Numerosa literatura científica respalda el uso de modelos LSTM en sistemas de detección de intrusiones como SecurAI, especialmente en contextos donde el tráfico de red se presenta como una secuencia temporal de eventos. Estos modelos, gracias a su capacidad para capturar dependencias a corto y largo plazo, han demostrado ser altamente efectivos para diferenciar entre tráfico legítimo y comportamientos maliciosos. Por ejemplo, estudios como el de Almseidin et al. (2024) [11] evidencian que la combinación de LSTM y SMOTE (Synthetic Minority Over-sampling Technique, una técnica que genera nuevas muestras sintéticas de la clase minoritaria para balancear las clases) permite detectar con mayor precisión intrusiones poco frecuentes, manteniendo la interpretabilidad y eficiencia del sistema.

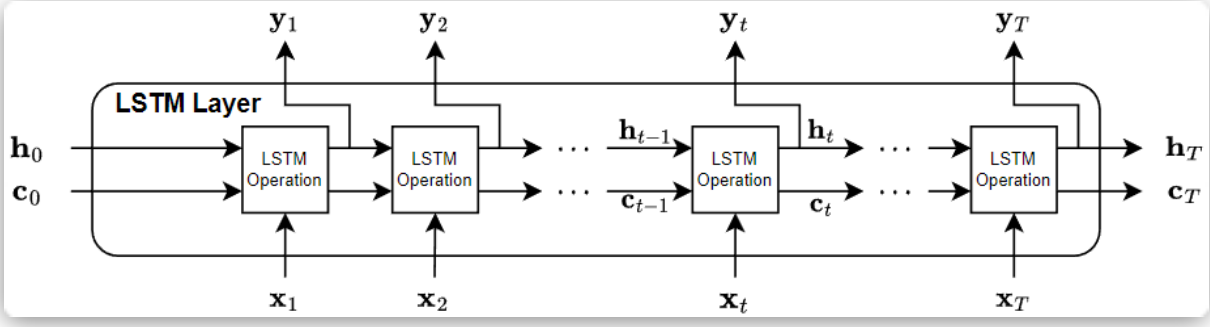


Fig 6: Capa LSTM

2.1.3 SVM

Las SVM (máquinas de vectores de soporte) son uno de los algoritmos de clasificación y regresión más potentes y robustos en múltiples campos de aplicación. Las SVM han estado desempeñando un papel importante en el reconocimiento de patrones, que es un área de investigación ampliamente popular y activa entre los investigadores [12].

El objetivo principal en la clasificación de patrones es obtener un modelo que, para cada par de entrada-salida clasifica correctamente dentro de la clase a la que pertenece (por ejemplo en nuestro caso, como ataque o benigno). Sin embargo, si el clasificador es demasiado apto para los

datos de entrenamiento, el modelo comienza a memorizar los datos de entrenamiento en lugar de aprender a generalizar, degradando la capacidad de generalización del clasificador [13].

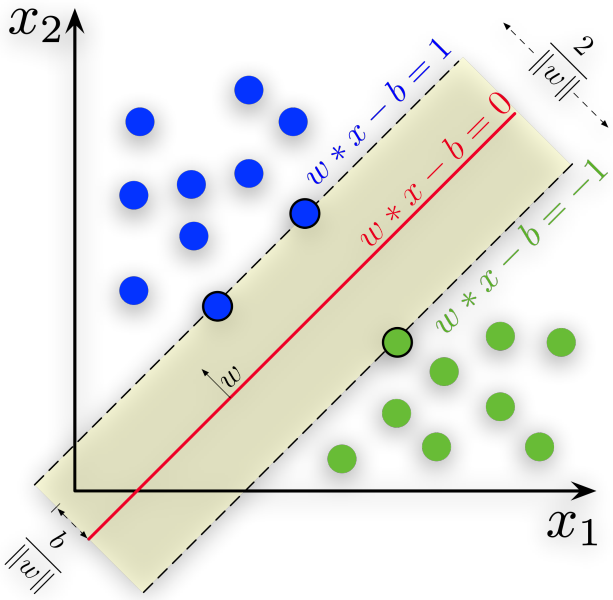


Fig 7: SVM

Las máquinas de vectores de soporte son modelos que aprenden a separar clases de datos buscando una frontera que deje el mayor espacio posible entre ellas (véase Fig. 7). Esta separación se calcula usando los datos de entrenamiento, de forma que se reduzcan los errores y el modelo pueda generalizar bien a datos nuevos. Durante el proceso, la SVM identifica solo algunos puntos clave llamados vectores de soporte, que son los más cercanos a la frontera y los que realmente determinan cómo se clasifican los datos. Esto hace que el modelo sea eficiente y funcione bien incluso cuando no hay muchos datos.

2.2 TECNOLOGÍAS

SecurAI no podría haberse desarrollado sin ayuda, tenemos que agradecer a la comunidad de desarrolladores, a las plataformas open-source, a los programas free-ware y a todos aquellos que comentan en GitHub y Stack Overflow.

Sin tecnologías como Python, Node.js o Electron no podríamos siquiera haber planteado SecurAI.

Se explican en los siguientes apartados estas tecnologías que hemos utilizado:

2.2.1 PYTHON

Python es la tecnología principal de *BackEnd*. Fue elegida por la gran cantidad de herramientas existentes para esta plataforma orientadas a la inteligencia artificial y al análisis de la red. En concreto se usa la versión 3.9.

Aunque ya habíamos trabajado antes con este lenguaje, no nos considerábamos ni mucho menos unos expertos en él. La gestión de paquetes y dependencias ha sido un reto, al igual que la gestión de múltiples procesos y concurrencia. Todas las tecnologías que han sido utilizadas apoyadas en Python están escritas en el archivo `requirements.txt`, siguiendo buenas prácticas de programación y gestión de paquetes con el gestor de paquetes pip.

2.2.1.1 FLASK

Flask levanta el servidor en el lado del *BackEnd*. Envía la información del estado de la cola de mensajes al *FrontEnd* y resuelve sus peticiones. Ha sido elegida por que ya estábamos familiarizados con él, y por su sencillez.

2.2.1.2 SCAPY

Scapy es una piedra angular de SecurAI, el corazón del análisis de red: Se encarga de capturar y en parte manipular los paquetes de red. Ha sido elegida por su sencillez, documentación, flexibilidad y bajo nivel en el control sobre los paquetes.

2.2.1.3 TENSORFLOW

TensorFlow es la librería encargada de cargar y ejecutar los modelos de inteligencia artificial entrenados previamente. Esta herramienta ha sido elegida por su madurez, rendimiento y buena integración con Keras, la API de alto nivel para construir y entrenar modelos de redes neuronales. TensorFlow ha permitido definir y reutilizar redes neuronales con facilidad tanto en el entrenamiento como en la fase de detección dentro de la aplicación.

2.2.1.4 PYINSTALLER

PyInstaller se ha utilizado para empaquetar todo el *BackEnd* del proyecto en un archivo ejecutable que pueden ser distribuidos fácilmente sin requerir que el usuario tenga Python instalado, fundamental para cumplir el objetivo de que SecurAI sea accesible por todos los usuarios sin conocimientos avanzados de informática.

Este empaquetado incluye el servidor Flask, el modelo de inteligencia artificial, y todas sus dependencias (como TensorFlow, Scikit-learn y Pandas). Al generar un *BackEnd* autocontenido,

facilitamos su integración con el entorno de escritorio proporcionado por Electron, asegurando que la aplicación pueda ejecutarse de forma sencilla en sistemas MacOS y Windows sin configuraciones adicionales.

2.2.2 JAVASCRIPT

JavaScript es la tecnología principal del *FrontEnd*. Ha sido elegida por ser el lenguaje de referencia para el desarrollo web moderno y por su integración natural con tecnologías como React y Next.js, que nos han permitido construir una interfaz interactiva, modular y reactiva.

La decisión de desarrollar el *FrontEnd* a modo de webapp en vez de una aplicación de escritorio fue tomada para así hacerlo más portable a varios sistemas operativos. Debido a este enfoque, realmente el motor del *FrontEnd* es Chromium, un navegador de código abierto en el que se basan navegadores como Google Chrome, Microsoft Edge y otros, proporcionado por el empaquetador que usamos, Electron.

Aunque realmente la aplicación se muestre en un navegador, para el usuario final esto es invisible y SecurAI se muestra como una aplicación independiente.

2.2.2.1 NODE.JS

Node.js se usa como entorno de ejecución para JavaScript del lado del servidor. Aunque en ese proyecto no se use como servidor principal (ese rol lo tiene Flask en el *BackEnd*), Node.js es fundamental para el funcionamiento del *FrontEnd*, ya que es el entorno necesario para ejecutar Next.js.

2.2.2.2 NEXT.JS Y REACT

Next.js no genera una web estática, sino que es un framework completo para aplicaciones web modernas, construido sobre React que se ejecuta en tiempo real en el servidor Node.js. Next.js genera y sirve páginas dinámicamente. Para eso necesita interpretar módulos de React, generar rutas dinámicas, ejecutar middleware y lógica previa al renderizado, gestionar estados de sesión...

En este proyecto, React es el lenguaje de construcción de interfaces. Se usa para definir componentes como botones, gráficas, tarjetas de estado, etc., y Next.js se encarga de organizar estos componentes en páginas reales y gestionarlas como una aplicación web estructurada.

Además, se ha utilizado Shadcn UI, una librería de componentes para React que se basa en Radix UI y TailwindCSS. Shadcn no es una librería empaquetada tradicional, sino un conjunto de componentes que se copian directamente al proyecto, lo que permite una personalización completa sin depender de actualizaciones externas. Se ha elegido esta librería por su estética y diseño moderno, el uso de TailwindCSS y la modularidad que nos proporciona al solo incorporar los componentes que realmente usamos.

2.2.2.3 ELECTRON

Electron es la tecnología que se ha utilizado para convertir SecurAI en una aplicación de escritorio multiplataforma (Windows y macOS), sin necesidad de que el usuario tenga conocimientos técnicos para instalar y ejecutar el sistema.

No se usa en el *FrontEnd* como tal, pero sí que se ejecuta sobre un servidor Node.js para orquestar la aplicación: abre ventanas, lanza el *BackEnd* de Python, lanza Next.js, gestiona errores, etc.

Una app hecha con Electron puede renderizar componentes React como en un navegador, pero también tiene acceso a funcionalidades de bajo nivel propias de una aplicación nativa de escritorio.

En la estructura de SecurAI, Electron es un proyecto independiente que está en una carpeta separada del *FrontEnd* (Next.js) y del *BackEnd* (Python). Se podría decir que es el tercer servidor de la aplicación y su función es levantar los otros dos (*FrontEnd* y *BackEnd*) como procesos hijos, y luego abrir una ventana que carga la interfaz web generada por Next.js. En el Apartado 4.1.3.3 Empaquetado final se cuenta en detalle cómo se utiliza esta tecnología específicamente en SecurAI, y cómo se resolvieron los problemas que surgieron.

Electron se usa ampliamente en el sector profesional, en aplicaciones de renombre como Visual Studio Code, Slack, Discord o Postman.

Aún así, siendo una herramienta altamente probada y utilizada, durante el desarrollo y empaquetado con electron-builder, nos encontramos con varios problemas técnicos que requirieron un esfuerzo adicional, como la compatibilidad con arquitecturas ARM (Apple Silicon) al no empaquetar correctamente al compilar para arm64 debido a problemas con los symlinks. Otro problema que surgió fue la inclusión de archivos de node_modules al no usar ASAR debido al orden de copia de las carpetas.

Los archivos ASAR en Electron empaquetan todos los recursos de una aplicación en un único archivo `.asar`, similar a un archivo `.zip` pero sin compresión. Esto mejora la organización del

código y puede acelerar el arranque de la app, pero no era posible usarlos en SecurAI debido, entre otros problemas, a la incompatibilidad con symlinks y a que impedían la correcta ejecución de procesos externos como el backend en Python y bibliotecas nativas como TensorFlow.

Estos problemas no solo fueron resueltos internamente, sino que compartimos nuestras soluciones en foros y comunidades como GitHub [14–17], contribuyendo así al conocimiento colectivo de otros desarrolladores que se enfrentaban a problemas similares.

CAPÍTULO 3
ESPECIFICACIÓN

En esta sección se detallan los aspectos clave que definen el alcance funcional y técnico del proyecto. Aquí se establecen una serie de requisitos funcionales y no funcionales que garantizan la viabilidad, mantenibilidad y escalabilidad del sistema.

Asimismo, se presentan los casos de uso principales que ilustran cómo los diferentes usuarios interactuarán con SecurAI en escenarios reales.

3.1 RF

- SecurAI analizará la red en búsqueda de ataques.
- SecurAI analizará la red en tiempo real.
- SecurAI avisará si encuentra un ataque.
- SecurAI mostrará en pantalla los módulos de detección actualmente activos.
- El desarrollador podrá añadir o personalizar módulos de detección conforme a sus necesidades.
- El usuario podrá activar o desactivar módulos de detección dinámicamente.
- El usuario podrá ver estadísticas de la red que mejorará su conocimiento de la misma.
- El usuario podrá ver el estado de la cola de mensajes en tiempo real.
- El usuario podrá activar módulos de ataque predefinidos para probar el rendimiento de la aplicación.
- El usuario podrá consultar una página de ayuda que resolverá sus dudas.
- El usuario podrá consultar una página de créditos del proyecto.

3.2 RNF

- El usuario podrá instalar fácilmente SecurAI.
- SecurAI utilizará técnicas de inteligencia artificial.
- SecurAI cargará los modelos de inteligencia artificial completamente en memoria al inicio para minimizar latencia durante el análisis.
- SecurAI limpiará automáticamente los paquetes de red ya analizados, reduciendo su impacto en la memoria del equipo.
- SecurAI basará la comunicación entre componentes deberá realizarse mediante protocolos HTTP o WebSocket, con bajo acoplamiento entre *FrontEnd* y *BackEnd*.
- SecurAI tendrá una interfaz de usuario estará construida con tecnologías web modernas
- SecurAI será empaquetado como aplicación de escritorio usando Electron, para facilitar su distribución multiplataforma.
- SecurAI estará disponible para Windows, MacOS x86 y MacOS ARM.

3.3 PERFILES DE USUARIO

Con el objetivo de asegurar que SecurAI sea una herramienta útil y accesible para distintos tipos de usuarios, se han definido varios perfiles representativos. Estos perfiles ayudan a entender cómo la plataforma puede integrarse en diferentes contextos y cuáles son las necesidades específicas que cubre en cada caso.

A continuación, se describen algunos de estos perfiles potenciales, junto con un flujo básico de uso. Esto permite ilustrar tanto la interacción esperada con la herramienta como los beneficios concretos que ofrece.

3.3.1 PERFIL DE USUARIO 1

- Actor principal: Propietario de una PYME preocupado por la seguridad del equipo, el cual tiene conexiones la red extrañamente lentas.

- Descripción: El usuario puede instalar fácilmente SecurAI y el sistema detecta automáticamente patrones de tráfico sospechosos que indican diferentes tipos de ataque y alerta al usuario en tiempo real.
- Flujo principal:
    1. El usuario instala SecurAI en su sistema.
    2. El usuario activa los módulos de defensa.
    3. SecurAI detecta los posibles ataques.
    4. SecurAI avisa al usuario de los posibles ataques.

3.3.2 PERFIL DE USUARIO 2

- Actor principal: Empleado experto en ciberseguridad se propone emplear SecurAI para detectar un tipo de ataque concreto que ya han sufrido con anterioridad.
- Descripción: El usuario quiere poder detectar un ataque ya documentado rápidamente. Es un ataque muy específico en una situación muy concreta por lo que las tecnologías convencionales no lo detecta, pero la empresa tiene datos de su funcionamiento.
- Flujo principal:
    1. El empleado experto en ciberseguridad instala SecurAI en el sistema.
    2. El empleado experto en ciberseguridad construye *datasets* adecuados para el entrenamiento de la inteligencia artificial, inspirado en los existentes en SecurAI.
    3. El empleado experto en ciberseguridad entrena una inteligencia artificial siguiendo los procedimientos de esta memoria, y siguiendo los ejemplos existentes en SecurAI.
    4. El empleado experto en ciberseguridad desarrolla un módulo de detección siguiendo los procedimientos de esta memoria, y siguiendo los ejemplos existentes en SecurAI.
    5. El empleado experto activa el nuevo módulo desarrollado.
    6. SecurAI monitoriza la red, y avisa si detecta un ataque.

3.3.3 PERFIL DE USUARIO 3

- Actor principal: Usuario común interesado en aprender sobre seguridad de red.

- Descripción: El usuario instala SecurAI para familiarizarse con conceptos básicos de seguridad y observar estadísticas en tiempo real de su red local, así como consultar la documentación y ayuda integrada.
- Flujo principal:
    1. El usuario instala SecurAI en su sistema.
    2. El usuario abre la aplicación y accede a la sección de estadísticas de red.
    3. SecurAI muestra gráficos y datos en tiempo real sobre el tráfico de red, como volumen de paquetes y protocolos usados.
    4. El usuario vuelve al inicio y activa un módulo de ataque.
    5. El usuario vuelve a la sección de estadísticas de red y observa el cambio en las gráficas.
    6. El usuario navega por la página de ayuda integrada para entender conceptos básicos y el funcionamiento de la aplicación.
    7. El usuario aprende a interpretar las estadísticas y alertas que muestra SecurAI, mejorando su conocimiento en seguridad de red.

3.4 CASOS DE USO

Los casos de uso son una herramienta fundamental en el análisis y diseño de sistemas, pues permiten describir de forma clara y estructurada las interacciones entre los usuarios (actores) y el sistema. Su objetivo principal es detallar cómo se espera que el sistema responda ante diferentes situaciones o acciones realizadas por los usuarios, facilitando así la comprensión de los requisitos funcionales y la planificación del desarrollo.

A continuación se muestran 3 casos de uso típicos de la aplicación a modo de ejemplo de acciones relevantes en la aplicación:

3.4.1 CASO DE USO 1

- Descripción: El usuario quiere detectar ataques de tipo ARP Flooding.
- Pre-condición:

1. El usuario ya tiene instalado SecurAI.
2. El usuario ha ejecutado SecurAI en su sistema.
3. El usuario está en la pantalla "Inicio" de SecurAI.

- Post-condición: El usuario observa en la pantalla "Inicio" si está siendo o no atacado.
- Escenario principal:
    1. El usuario activa como mínimo uno de los módulos relacionado con el ataque ARP Flooding (arpFlooding, arpFloodingSW, arpFloodingSVM) de la sección de módulos en la parte derecha de la pantalla.

3.4.2 CASO DE USO 2

- Descripción: El usuario quiere asegurarse del funcionamiento del módulo tcpSYN.
- Pre-condición:
    1. El usuario ya tiene instalado SecurAI.
    2. El usuario ha ejecutado SecurAI en su sistema.
    3. El usuario está en la pantalla "Inicio" de SecurAI.
- Post-condición: El usuario observa en la pantalla "Inicio" si que está siendo atacado, y el ataque está siendo detectado por el módulo tcpSYN.
- Escenario principal:
    1. El usuario activa el módulo tcpSYN de la sección de módulos en la parte derecha de la pantalla.
    2. El usuario activa el ataque tcpSYN de la sección "Simulaciones de ataque" desde la barra de menús.

3.4.2 CASO DE USO 3

- Descripción: El usuario quiere comprobar el nivel de saturación de la red.
- Pre-condición:
    1. El usuario ya tiene instalado SecurAI.

2. El usuario ha ejecutado SecurAI en su sistema.
3. El usuario está en la pantalla "Inicio" de SecurAI.

- Post-condición: El usuario observa un gráfico en tiempo real del número de paquetes que recibe su equipo.
- Escenario principal:
    1. El usuario accede a la pantalla "Número de paquetes" desde el menú de "Estadísticas de la red" desde la barra de menús.
    2. El usuario espera unos segundos para observar la construcción del gráfico.

CAPÍTULO 4
ESTRUCTURA DE LA SOLUCIÓN

En este capítulo se describen en profundidad los fundamentos y la estructura interna de SecurAI. Pero más allá de una simple lista de herramientas o definiciones técnicas, este capítulo representa un profundo análisis de SecurAI mostrando las decisiones estructurales que le dan forma, propósito y coherencia a la solución que se ha desarrollado.

Aquí no solo se explica cómo funciona cada componente, sino por qué existe, qué propósito cumple dentro del sistema y cuál fue la intención detrás de cada decisión de diseño.

Desde la división entre *FrontEnd* y *BackEnd* hasta la elección de un núcleo extensible basado en plugins, todo responde a una lógica que busca mantener el sistema modular, escalable, mantenible y accesible.

4.1 ESTRUCTURA Y FLUJO DE LA APLICACIÓN

La aplicación que se ha desarrollado sigue en su parte central la arquitectura cliente-servidor: Se ha separado el *BackEnd* del *FrontEnd* y los dos son totalmente independientes entre sí, facilitando el desarrollo, mantenimiento, escalabilidad y portabilidad.

Por otro lado, el *BackEnd* de SecurAI sigue a su vez una arquitectura modular; los módulos de detección funcionan también de forma independiente entre sí, y aunque se detectan, validan y cargan al iniciar la aplicación.

Su estructura permite que sean añadidos o eliminados sin tocar el código principal. Los módulos podrían considerarse plugins porque están desacoplados del núcleo y solo requieren una implementación concreta que especificaremos más adelante en este capítulo.

En este tipo de arquitectura, el núcleo representa el componente central que coordina, comunica y controla. Su misión es orquestar la lógica esencial que da sentido al sistema, no es la parte más extensa, pero sí la más fundamental, con procesos clave y comunes a todos los módulos como la hebra limpiadora o la comunicación con el *FrontEnd*.

Los módulos además una vez cargados, pueden desactivarse sin necesidad de reiniciar el programa. Teniendo esto en cuenta, podríamos definir el *BackEnd* como un Sistema Modular Extensible basado en Plugins con Carga Semi-Dinámica.

La aplicación, tanto el *FrontEnd* como el *BackEnd*, es autocontenida es decir, no es necesario acceder a ningún archivo ajeno a ella.

4.1.1 BACKEND

Es la parte más importante de la aplicación, ya que alberga los módulos de detección propiamente dichos, la gestión de la cola de mensajes, la captura de paquetes, las simulaciones de ataques, las estadísticas y, en parte, la comunicación con el *FrontEnd*.

El BackEnd está diseñado para ser modular y extensible, y actúa como coordinador de las funcionalidades críticas de la aplicación.

El BackEnd inicia los módulos de detección, captura y procesa los paquetes y los añade a la cola de mensajes. Luego los módulos se encargan de analizarlos individualmente y una vez hecho, el BackEnd limpia la cola. Además, notifica al FrontEnd los módulos cargados y el estado de la cola de mensajes. Por si fuera poco, aquí en el BackEnd tiene lugar la simulación de ataques y la recopilación de estadísticas.

También alberga los archivos necesarios para entrenar los modelos, como los *datasets*, y otros archivos como pruebas de rendimiento de los mismos. Se espera que cuando un desarrollador ajeno a este proyecto desarrolle un nuevo módulo de detección, aloje los archivos necesarios para su correcto funcionamiento en las carpetas destinadas a ello dentro del *BackEnd*, ampliaremos esta información en el Apartado 4.2.2 Cómo implementar un módulo.

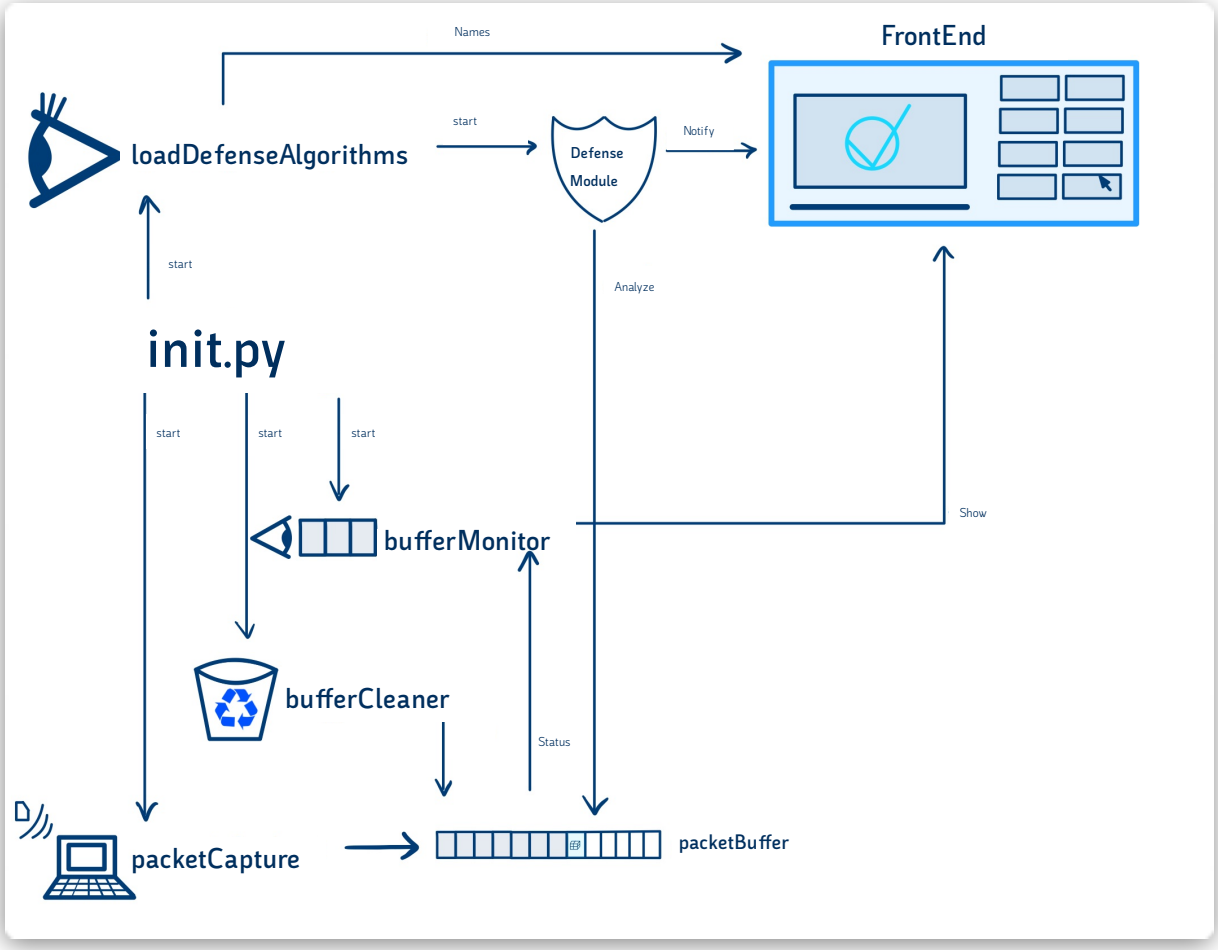


Fig 8: Esquema básico de funcionamiento de SecurAI

4.1.1.1 ARCHIVOS DE ARRANQUE Y CONFIGURACIÓN

Se engloban en este grupo a los archivos `run.py`, `__init__.py` y otros archivos típicos de Python como `requirements.txt`.

Al ejecutar `run.py` este importa el módulo `app`, que incluye la inicialización de la aplicación (definida en `BackEnd/app/__init__.py`).

En `__init__.py` se definen los detalles de configuración de Flask, se crea el objeto notificador de ataques, arranca el hilo de capturar de paquetes, monitorización y limpieza del buffer. Además, carga entre otros los algoritmos de defensa y ataques.

Con ánimo de seguir buenas prácticas de programación para el desarrollo de aplicaciones en Python, se han usado entornos virtuales con venv y gestionado sus dependencias con pip. El archivo `requirements.txt` registra todas las dependencias instaladas en el entorno.

4.1.1.2 PACKETCAPTURE

El archivo de inicialización `__init__.py` arranca un proceso `packetCapture` que se encarga de la captura de paquetes, empaquetamiento de los mismos en una estructura `PacketIndexed` que le añade más información y posterior anexión a la estructura de datos principal de nuestro programa (véase Fig. 9).

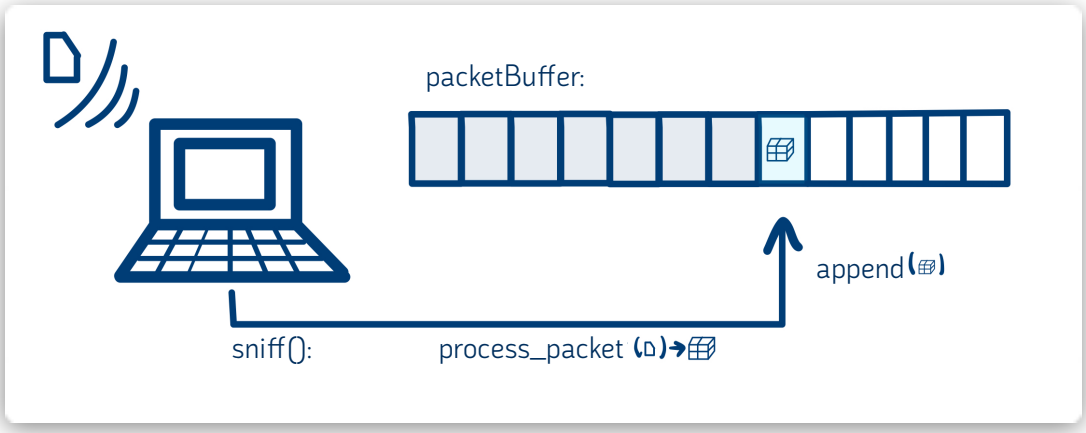


Fig 9: Esquema de funcionamiento de packetCapture

Para la captura en este proyecto se utiliza Scapy, una librería de código abierto que usa la función `sniff()` que permite interceptar y procesar paquetes directamente desde interfaces de red (NIC) [18].

Scapy se apoya en libpcap (Linux/macOS) o Npcap/WinPcap (Windows) para acceder directamente al tráfico de red, instalando drivers de captura a bajo nivel que permiten interceptar paquetes en bruto desde las interfaces de red sin intervención del sistema operativo.

La principal información que se añade al paquete de red capturado es un diccionario de clave los módulos de defensa cargados en la aplicación y valor si dicho paquete ha sido o no filtrado por dicho módulo. Esta estructura permite el diseño modular del programa, ya que procuraremos que no sea eliminado hasta que sea procesado por todos los módulos cargados.

SecurAI es un programa modular y si un desarrollador añade un módulo, no tiene que manipular este diccionario de control, sino que se hace de forma automática. El desarrollador puede así centrarse exclusivamente en la lógica de detección.

Cada módulo de defensa se encarga de forma independiente de marcar si ha procesado el paquete, para ello accederán a la estructura de datos principal del programa, compartida entre todos los procesos del mismo.

4.1.1.3 BUFFERMONITOR Y BUFFERCLEANER

Estos dos archivos se encargan de la gestión de la cola de mensajes del programa, tanto de su monitorización para la visualización de su estado en el *FrontEnd* como de su vaciado.

La cola de mensajes es la estructura de datos compartida principal de la aplicación. Su buen funcionamiento es crucial para tanto el análisis del tráfico de la red, como para el rendimiento del sistema.

`bufferMonitor.py` es iniciado como proceso independiente para enviar constantemente el estado de la cola de mensajes al *FrontEnd* mediante Web Sockets.

`bufferCleaner.py` es el proceso encargado de revisar qué paquetes han sido analizados por todos los módulos cargados, y de eliminarlos para evitar que la cola aumente en exceso (véase Fig. 10). Este proceso revisa la cola de mensajes y accede a los objetos `PacketIndexed` que contiene. Para cada uno, accede a su diccionario y comprueba si ha dicho paquete ha sido analizado por todos los filtros cargados en el programa. Si el paquete ha sido completamente analizado, se elimina. Si no, el proceso deja el paquete en la cola y deja de analizar hasta su próxima iteración con el fin de mejorar el rendimiento.

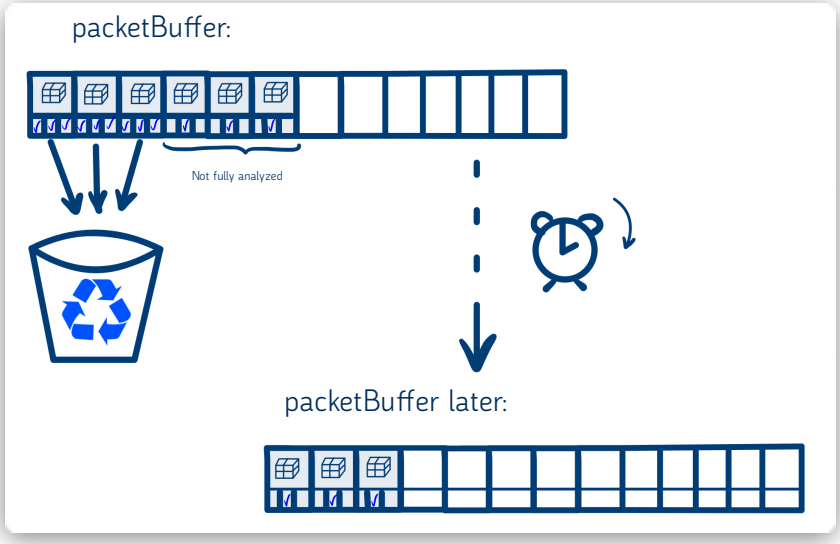


Fig 10: Esquema de funcionamiento de bufferCleaner

La comprobación y manipulación del diccionario y en general de la cola de mensajes por supuesto siguen el principio de exclusión mutua.

4.1.1.4 ARCHIVOS DE CARGA

En este grupo se engloban los archivos `loadDefenseAlgorithms.py` y a `loadAttackTests.py`. Se encargan de buscar en las carpetas adecuadas qué módulos de defensa y de ataques existen, para luego mostrarlos en el *FrontEnd* (véase Fig. 11). También se encargan de activar o desactivar cada módulo, de forma que no es necesario reiniciar el programa cada vez que el usuario quiera cambiar las opciones de SecurAI.

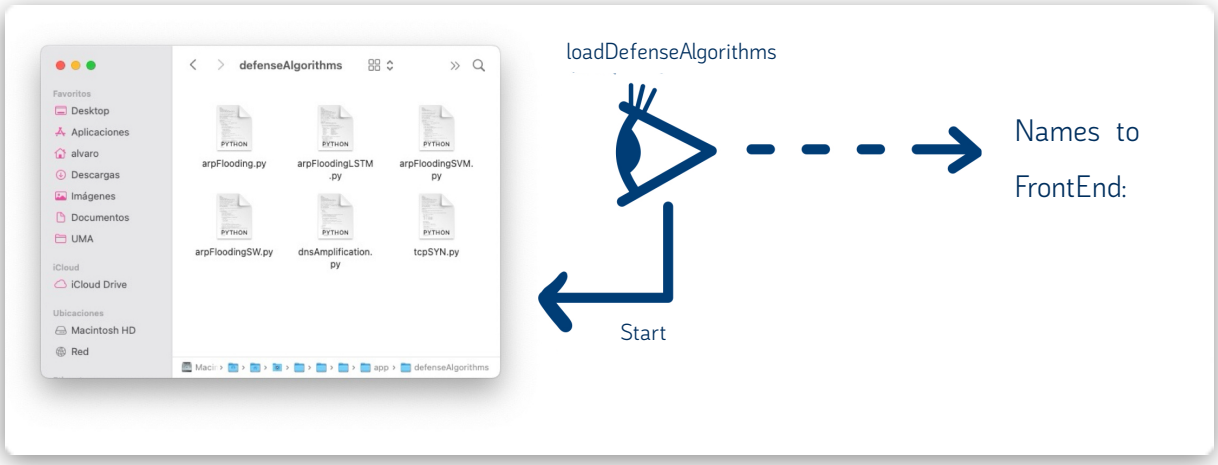


Fig 11: Esquema de funcionamiento de loadDefenseAlgorithms

Para la correcta activación y desactivación dinámica del módulo, este debe de tener una variable `running` que será accedida por estos archivos. Este y otros aspectos de especificación a la hora de crear un módulo serán detallados en el Apartado 4.2.2. Cómo implementar un módulo.

4.1.1.5 OBJETO ATTACKNOTIFY

El archivo `attackNotify.py` define dicho objeto que, al invocarlo en un módulo, permite notificar al *FrontEnd* que dicho módulo está siendo atacado. Su uso está también detallado en la Sección *Cómo implementar un módulo*.

La existencia de este objeto permite abstraer al desarrollador la lógica de comunicación con el *FrontEnd*, ya que simplemente usando su método `notifyAttack` el usuario recibirá la alerta adecuada del módulo de defensa en cuestión.

Como con el uso de técnicas de inteligencia artificial es normal que los resultados de la predicción tengan cierto margen de error es posible que durante un ataque se marquen algunos paquetes maliciosos como benignos (falso negativo). Para evitar confusión al usuario, el aviso de ataque del

FrontEnd tiene una duración de algunos segundos, para que el aviso no aparezca y desaparezca en cuestión de milésimas de segundo. Este comportamiento es transparente tanto para el usuario como para el desarrollador y gestionado automáticamente por el núcleo de SecurAI.

4.1.1.6 DIRECTORIO ATTACKTESTS

Dentro de esta carpeta, se ubicarán los archivos de simulaciones de ataques que se presentarán en el *FrontEnd*. Los archivos en formato `.py` de su interior son predefinidos, ya que simplemente sirven de ejemplo para que el usuario vea el correcto funcionamiento de los módulos proporcionados, y para que el desarrollador pueda estudiar el funcionamiento de los mismos.

La conexión del *FrontEnd* con el *BackEnd* para la integración de módulos de ataque se realiza de forma automática siempre que el botón del NavBar del FrontEnd se llame igual que el módulo de ataque en el *BackEnd*. El *FrontEnd* construye automáticamente la petición HTTP que el *BackEnd* espera recibir para reanudar o pausar el funcionamiento de un módulo de ataque.

Al igual que pasa con los algoritmos de defensa, los módulos de ataque tienen una variable running que activan o desactivan el ataque una vez el módulo ha sido cargado. También tienen un método obligatorio `attack()` que se encarga de implementar la lógica en cuestión del ataque. La existencia de este método se comprueba al cargar el módulo en el inicio del programa.

4.1.1.7 DIRECTORIO DEFENSEALGORITHMS

Esta carpeta sigue la filosofía del directorio anterior y alberga los módulos de defensa que se mostrarán en el *FrontEnd*.

A diferencia de las simulaciones de ataques, los módulos de defensa son modificables por el usuario, de manera este que puede añadir o retirar módulos de esta carpeta. Cuando se inicie el programa, este cargará los archivos `.py` con un método `detect()` y los mostrará en el *FrontEnd* de la aplicación, pudiendo activarse o desactivarse dinámicamente gracias a la variable global `running` que deben de incluir.

Cuando un usuario desee añadir o retirar módulos de defensa, simplemente tiene que acceder a la carpeta defenseAlgorithms y añadir o quitar el .py que quiera. El programa se encargará de gestionar la integración con el resto del mismo, incluyendo la interfaz.

La especificación para crear nuevos módulos de defensa compatibles con SecurAI se encuentra en la Sección *Cómo implementar un módulo*, y la documentación de los módulos existentes proporcionados están en el capítulo *Módulos implementados*.

4.1.1.8 DIRECTORIO MACHINEMODELS

Esta es una de las carpetas más grandes del programa, ya que engloba y centraliza todo el proceso de entrenamiento de modelos de inteligencia artificial, así como aloja los modelos ya entrenados y los *datasets* usados para ello.

A excepción del directorio `/machineModels/models`, esta carpeta no es accedida en la ejecución normal del programa y sus archivos están pensados para ejecutarlos individualmente cuando sea necesario, por ejemplo para el entrenamiento o prueba puntual de un modelo.

machineModels es el corazón de la IA de SecurAI, de manera que el desarrollador que quiera entrenar un modelo, hacer pruebas con el mismo o simplemente acceder a él en su módulo de defensa debe de hacerlo usando los respectivos directorios dentro de machineModels.

machineModels contiene como decimos todos los directorios relacionados con los entrenamientos y pruebas de modelos de inteligencia artificial que serán usados por los módulos de detección. Estos directorios son a su vez:

- `/dataSetsOriginals` – Lugar donde se alojarán los *datasets* originales en formato `.csv` que se usarán para entrenar a los modelos de inteligencia artificial. Por legibilidad se recomienda usar nombres iguales o parecidos a los algoritmos que los usarán, y al algoritmo de defensa que usará el modelo. Aun así, como los *datasets* pueden servir para diversos modelos de defensa, pueden tener nombres más generales.
- `/dataSetsTransformed` – Aquí se alojarán los *datasets* que han sido modificados para su uso final en el entrenamiento de un modelo en formato `.csv`. Si bien los *datasets* sin modificación se encuentran en el directorio anterior, en este se encuentra, si procede, el mismo *dataset* pero en una versión modificada para optimizar el entrenamiento de un modelo. Por legibilidad se recomienda usar nombres iguales o parecidos a su *dataset* original, a no ser que el *dataset* original fuera general para varios módulos y este específico. Por ejemplo en los archivos proporcionados el *dataset* original `arpFlooding+.csv` se divide en los *dataset* transformados `arpFloodingSW.csv`, `arpFloodingLSTM.csv`...

- `/machineTrain` – Directorio donde se ubicarán los archivos `.py` destinados al entrenamiento de un modelo de inteligencia artificial. En general, los archivos de este directorio acceden a un *dataset*, lo transforman para el entrenamiento y guardan y, efectivamente, entrenan el modelo y también lo guardan para evitar el entrenamiento cada vez que se inicie el programa. Por legibilidad se recomienda usar nombres iguales o parecidos a los *datasets* que usa, y nombre exactamente igual al algoritmo de defensa que usará el modelo ubicado en `/defenseAlgorithms`.
- `/models` – Lugar donde se almacenarán los modelos que usarán los algoritmos de defensa, por ejemplo en formatos `.h5` o `.pkl`. Por legibilidad se recomienda usar nombres iguales o parecidos a los *datasets* que usa y nombre exactamente igual al algoritmo de defensa que usará el modelo ubicado en `/defenseAlgorithms`, y al archivo de `/machineTrain` usado para entrenarlo.
- `/modelTests` – Directorio de uso opcional donde se ubicarán posibles tests de rendimiento que se quieran hacer a los modelos. Por legibilidad se recomienda crear un directorio dentro de este con el nombre del modelo a probar, y albergar ahí los archivos `.py` necesarios para la evaluación o los *datasets* transformados que el desarrollador necesite.
- `/encoders` – Directorio de uso opcional donde se ubicarán, si fuera necesario, diccionarios en formato `.pkl` necesarios para la transformación y entrenamiento de modelos. Por legibilidad, se recomienda que el nombre del archivo alojado en esta ruta sea exactamente igual al modelo y algoritmo de defensa que lo usará.

4.1.1.9 DIRECTORIO ROUTES

Esta carpeta está destinada al enrutamiento del *BackEnd* construido con Flask. Todas las peticiones http que recibe del *FrontEnd* son procesadas aquí, de manera que cada archivo `.py` representa un Blueprint para organizar mejor las rutas.

Por ejemplo, cuando el *FrontEnd* carga, solicita al *BackEnd* el nombre de todos los módulos cargados, para ello envía una petición a `/loadDefenseAlgorithms/loadedNames`. La primera parte de la ruta representa el *Blueprint* y la segunda la acción a realizar dentro de la carga de algoritmos de defensa. Cuando el archivo destinado a la carga de algoritmos de defensa recibe esta petición, llama a la función correspondiente del *BackEnd*. De esta manera, centralizamos el flujo de comunicación con el *FrontEnd* mediante este directorio.

4.1.2 FRONTEND

La interfaz de usuario de este proyecto está programada en JavaScript usando el framework Next.js. En principio ni el usuario ni el desarrollador tienen que modificarla en ningún caso, ya que la gestión de módulos de defensa se hace automáticamente cada vez que se inicia el programa.

Sobre el uso de la interfaz, este podrá consultarse en el *Manual de Usuario de SecurAI*, orientado al consumidor final del programa.

Tanto los componentes importados de fuentes externas como los propios del programa, se ubican en la carpeta `/compontents`.

La navegación se basa en el App Router de Next.js introducido en Next.js 13, de manera que cada pestaña de la interfaz se ubica en una carpeta independiente, y dentro de ella tenemos un archivo `page.js` que es el código de la página en sí.

4.1.2.1 COMUNICACIÓN CON EL BACKEND

Para lanzar una petición estándar puntual al cerebro de SecurAI, al *BackEnd*, se usa la función nativa `fetch` de JavaScript, no necesitamos ninguna librería adicional.

Es importante notar que el *FrontEnd* muestra información en tiempo real del estado del *BackEnd*, ya que SecurAI no contempla que el usuario tenga que observar la terminal del *BackEnd*. Un ejemplo de esto es el estado de la cola o buffer de mensajes, o de las estadísticas que proporciona el programa. Para suplir este problema hacemos uso de la librería Socket.IO para establecer una conexión WebSocket. Los archivos `.js` necesarios para el uso de cada hook personalizado se encuentran en la carpeta `/useSocket`. La nomenclatura utilizada sigue un patrón que hace referencia al propósito del hook, compuesto por el prefijo *use* seguido del nombre del servicio que proporciona. Por ejemplo, `useBufferSocket.js` maneja la comunicación WebSocket relacionada con el tamaño del buffer en tiempo real, mientras que `useNumberStatsSocket.js` gestiona la recepción de estadísticas numéricas enviadas por el *BackEnd*.

4.1.3 EMPAQUETAMIENTO MULTIPLATAFORMA

El enfoque multiplataforma ha sido uno de los principales retos durante el desarrollo de SecurAI, ya que es importante que la aplicación funcione tanto en macOS (Intel y ARM) como en Windows, garantizando una experiencia fluida y homogénea.

El *BackEnd*, desarrollado en Python, requiere un proceso de empaquetado específico porque las dependencias de Python, especialmente aquellas relacionadas con machine learning y procesamiento de red, ocupan un espacio considerable y no son nativamente ejecutables en todos los sistemas. Por ello, usamos herramientas como PyInstaller para crear un ejecutable autónomo que incluya todas las librerías necesarias.

En paralelo, el *FrontEnd*, construido con React, se debe compilar por separado para generar un paquete estático optimizado. Finalmente, Electron actúa como puente, integrando ambos componentes en una única aplicación de escritorio, permitiendo que el *BackEnd* y *FrontEnd* se lancen paralelamente.

La construcción de los ejecutables no habría sido posible sin herramientas de CI/CD (Integración continua, despliegue continuo), GitHub Actions en nuestro caso.

El proyecto tiene automatizado el lanzamiento de nuevas *releases* (versiones) mediante un *workflow* (flujo de trabajo) que se detallará más adelante, en el Apartado 6.4 CI/CD. Un *workflow* no es más que un conjunto automatizado de procesos definidos para ejecutar acciones específicas en respuesta a determinados eventos, en nuestro caso un cambio dentro del repositorio de GitHub

Cuando se modifica el código y se hace un commit a la rama main, estos procesos automatizados generan nuevos ejecutables para la nueva *release*, usando las herramientas de empaquetado que describimos a continuación:

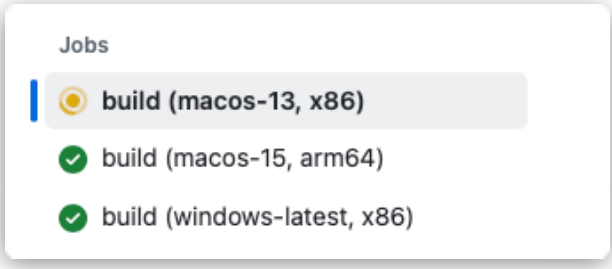


Fig 12: Workflow generando una nueva release

4.1.3.1 EMPAQUETAMIENTO DEL *BACKEND*

El primer reto a resolver a la hora de usar PyInstaller para el empaquetamiento del *BackEnd* es que los ejecutables que genera son propietarios de la plataforma que los compila. Como para el

desarrollo SecurAI se ha utilizado el sistema operativo MacOS y no teníamos disponible un equipo Windows, no se podía probar el *BackEnd* en dicha plataforma. Otro problema es el tiempo que se tarda en compilar el programa, y el espacio que ocupa en el proceso.

Estos problemas tienen solución usando las técnicas de CI/CD que mencionamos, permitiendo usar un *runner* Windows de GitHub.

Aún así, aparecen fallos causados por que el nuevo ejecutable no ha cogido todas las librerías, han cambiado las rutas de los archivos, problemas a la hora de firmar el ejecutable...

Gran parte de la culpa la tiene TensorFlow, ya que utiliza modelos, configuraciones, recursos internos y bibliotecas dinámicas, que a veces PyInstaller no detecta automáticamente. Esto causa errores de importación y fallos en tiempo de ejecución debido a archivos faltantes.

Todos estos problemas han sido solucionados y el ejecutable del *BackEnd* de SecurAI se construye gracias al archivo `BackEnd/run.spec` personalizado.

Aparte del problema de la compatibilidad de los ejecutables entre sistemas operativos, hemos sufrido también errores de compatibilidad entre MacOS x86 y MacOS arm64: las librerías de TensorFlow son incompatibles entre sí. Al principio nuestra idea sería compilar el sistema para MacOS x86 y que los usuarios de arm64 ejecutaran SecurAI usando Rosetta 2, pero debido a este contratiempo esta aproximación ya no sería posible (véase Fig. 13).

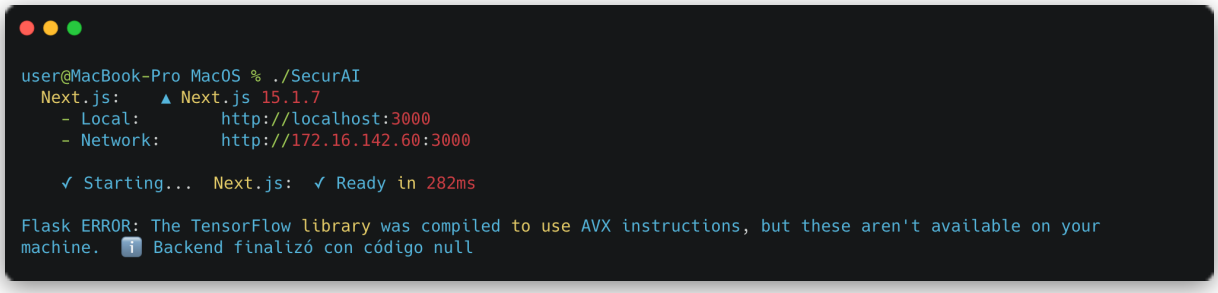


Fig 13: Problema de compatibilidad de TensorFlow al usar Rosetta 2

Compilar SecurAI específicamente para MacOS arm64 ha supuesto un gran reto, pero era importante hacerlo para dotar al proyecto de mayor longevidad. Contaremos su proceso en el Apartado 4.1.3.3 Empaquetado final.

De esta manera, finalmente cada vez que se genera una nueva versión se ejecuta el workflow en tres *runners* independientes, uno por cada plataforma (véase Fig. 12). Podemos tomar un *runner* como una máquina virtual independiente, en la que podemos instalar dependencias y ejecutar programas.

4.1.3.2 CONSTRUCCIÓN DEL FRONTEND

A diferencia del *BackEnd* no hay que empaquetar independientemente en *FrontEnd*, porque será lanzado por Electron, sino que hay que construir la build de producción de la app de React (véase Fig. 14). Para ello en el *runner* de GitHub Actions ha de preparar las dependencias de Node.js necesarias y ejecutar `npm run build`.

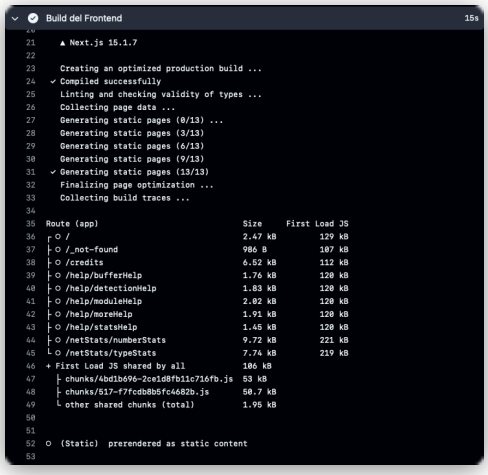


Fig 14: Construcción del FrontEnd en el runner

Lo que sí fue necesario fue la inclusión de una distribución de Node.js para que el *FrontEnd* de que el equipo del usuario tenga instalado esta herramienta. En concreto instalamos la versión 20.18.3, y es importante que cada *runner* instale la versión adecuada al sistema operativo que ejecuta.

Una vez acaba la construcción, se genera el proyecto en `.next/`, y Electron lo usará para lanzar el *FrontEnd* a la hora de arrancar SecurAI.

4.1.3.3 EMPAQUETADO FINAL

El empaquetado final de SecurAI se realiza gracias a la herramienta Electron, que agrupa el ejecutable de Python con el proyecto de Next.js.

Para realizar el empaquetado de las diferentes arquitecturas de SecurAI, es necesario instalar las dependencias de Node.js en los respectivos *runners*, y utilizamos los comandos npx electron-builder `npx electron-builder --mac –x64`, `npx electron-builder --mac –arm64` y `npx electron-builder --win –x64` respectivamente.

La configuración de las dependencias de Electron, y de su archivo principal `main.js` ha tenido sus dificultades.

Uno de ellas por ejemplo ha sido que las versiones recientes de Electron no funcionan en los *runners* de GitHub arm64, ha sido necesario usar exactamente la versión 23.6 de electron-builder. Otro problema que hubo que solucionar fueron las rutas dentro del paquete, que no eran válidas al usar ASAR. Fue necesario que cambiar el flujo entero de acceso a la aplicación por parte de Electron para desactivarlo. Además ello conllevó a otro problema copiando la carpeta de Node.js node_modules del *FrontEnd* dentro del paquete. Esta carpeta es necesaria para ejecutar el servidor

de Node.js independientemente de que el usuario lo tenga instalado, cumpliendo el objetivo de una aplicación accesible y fácil de instalar para todos.

Otro escenario complejo que fue necesario enfrentar fue la adecuación a la forma de trabajar a los diferentes sistemas. Por ejemplo *Scapy* no puede ejecutarse nativamente en Windows, por lo que tuvimos que agregar un instalador de *Ncap*. Otro ejemplo leve pero curioso fue que, a la hora de implementar logs en el proyecto, estos tenían un problema al guardase en Windows ya que no podíamos usar `print()` con emojis.

Por último, otro gran problema que ralentizó el desarrollo fue el tiempo que tardaba el *runner* de GitHub Actions generar el ejecutable (véase Fig. 15), aunque el rendimiento era más que aceptable teniendo en cuenta que hablamos de un servicio gratuito que fue crucial para poder hacer SecurAI multiplataforma.

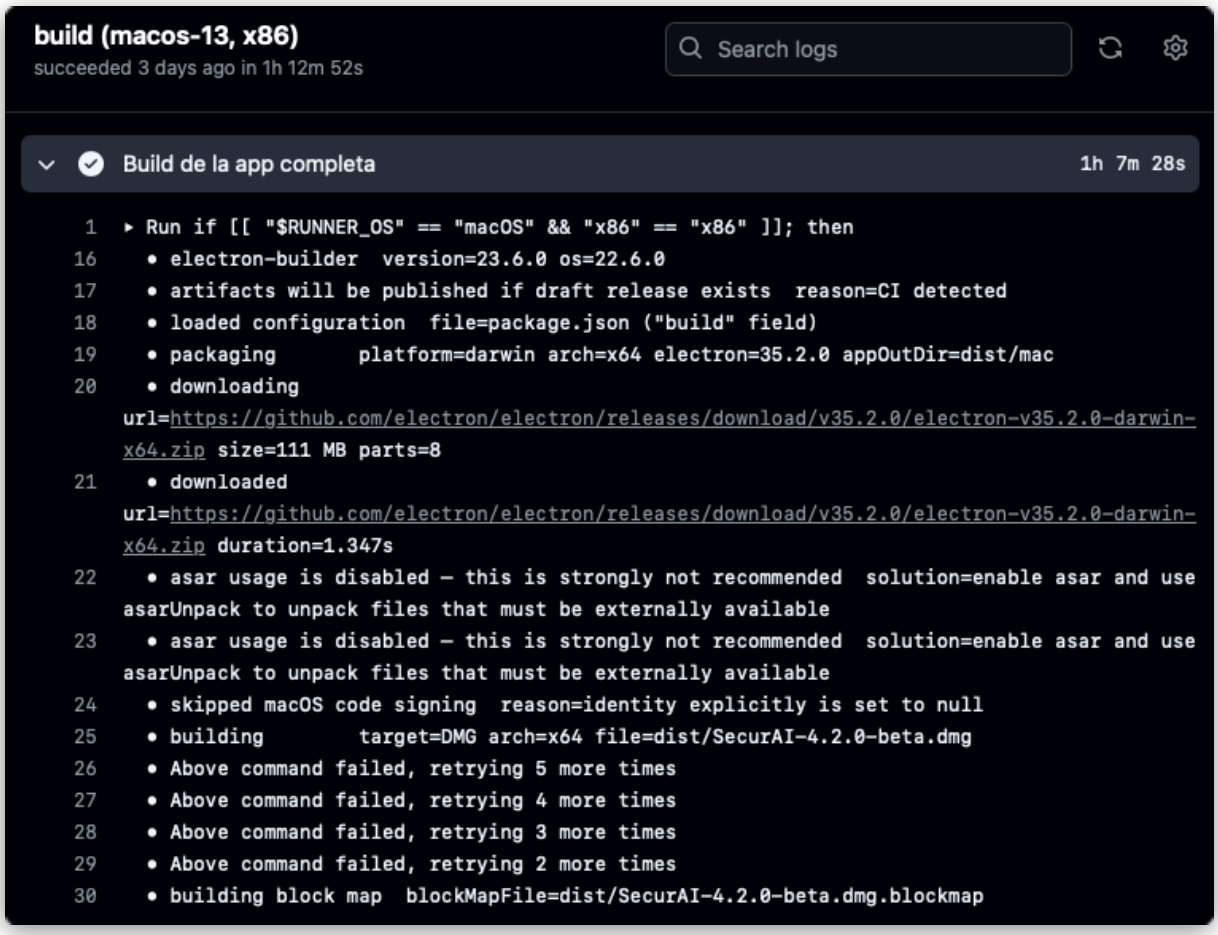


Fig 15: Ejemplo de generación notablemente lenta del ejecutable final. Nótese el tiempo de ejecución del paso "Build de la app completa" en un runner MacOS x86.

4.2 ENFOQUE MODULAR

Una de las principales fortalezas de SecurAI es la posibilidad de ampliarlo para adaptarse a las necesidades del usuario. Este proyecto está pensando desde los cimientos para poder ser ampliado fácilmente, alojando la lógica de detección de amenazas en módulos desacoplados del núcleo que, una vez cargados, pueden desactivarse sin necesidad de reiniciar el programa.

4.2.1 POR QUÉ IMPLEMENTAR UN MÓDULO

Es normal que los módulos implementados en SecurAI parezcan en principio algo escasos. Con objeto de pensar en este proyecto como uno muy escalable y que perdure en el tiempo hicimos de SecurAI un sistema completamente modular.

Con el paso del tiempo surgen nuevas amenazas que podrían ser detectadas por SecurAI, o quizás una empresa quiera detectar un tipo muy específico de ataque. Para estos casos el desarrollador puede entrenar nuevos modelos de detección completamente personalizados para sus necesidades concretas y añadirlos a SecurAI de forma sencilla.

4.2.2 CÓMO IMPLEMENTAR UN MÓDULO

Realmente es muy sencillo implementar un nuevo módulo, pero en este documentos se explicará con detalle el procedimiento recomendado, con el fin de implementar buenas prácticas de programación, que el desarrollador puede seguir o no libremente.

Se recomienda encarecidamente observar (y estudiar) cómo han sido implementados los módulos existentes ya con la instalación base del programa.

Destacar que el nuevo módulo no tiene por qué necesariamente implementar algún tipo de red neuronal, a fin de cuentas el uso de SecurAI es libre.

Si el desarrollador quiere desplegar manualmente el *BackEnd* y el *FrontEnd*, con fines de desarrollo o para la implementación o uso de un módulo personalizado, se recomienda que se desplieguen en este orden para que la comunicación se haga correctamente entre ellos.

Para trabajar sobre ellos en un editor como VSCode, con objeto de que las rutas relativas funcionen correctamente es aconsejable abrir en el editor la carpeta `/BackEnd` para trabajar con el *BackEnd*, y abrir la carpeta `/FrontEnd/app` para trabajar con el *FrontEnd*.

4.2.2.1 REQUISITOS

- El módulo a implementar estará escrito en un único archivo `.py`, que se almacenará en la carpeta `defenseAlgorihtms` del *BackEnd*.
- El módulo puede obtener los paquetes del buffer de mensajes de forma libre (individualmente, por lotes...) siempre que al final del análisis de esos paquetes, se marquen individualmente como completados usando la función `mark_processed(ALGORITHM_NAME)`. Para ello es imprescindible que la línea `from app.packetCapture import packetBuffer, packetBufferLock` esté presente en el módulo. El nombre del algoritmo será automáticamente detectado por SecurAI como el nombre del archivo.
- Para notificar al *FrontEnd* de la existencia de un ataque, es necesario usar la función `attackNotifier.notifyAttack(ALGORITHM_NAME)`. Para ello es imprescindible que la línea `from app import attackNotifier` esté presente en el módulo.

4.2.2.2 FLUJO DE TRABAJO

Suponiendo que un desarrollador va a implementar un nuevo módulo que va a usar redes neuronales, el flujo de trabajo recomendado para su desarrollo sería el siguiente:

1. Crear o descargar el *dataset* que usaremos para el entrenamiento de la red neuronal, sin procesar. Se guardará en `/app/machineModels/dataSetsOriginals` siguiendo la convención de nombres de la carpeta. Usualmente se llamará como el nuevo módulo a implementar.
2. Probablemente sea necesario aplicar algún tipo de procesamiento al *dataset*, y seguro que será imperativo entrenar la red neuronal a partir del mismo. Para ello se creará un archivo `.py` en `/app/machineModels/machineTrain` que se llamará como el nuevo módulo a implementar en el que situaremos las operaciones necesarias de modificación al *dataset* y entrenamiento de la red neuronal con el mismo. El *dataset* final que se ha usado para el entrenamiento se guardará

en `/app/machineModels/dataSetsTransformed` y el modelo en `/app/machineModels/models`, ambos como se llamaran como el nuevo módulo a implementar.

3. Si se quiere probar el rendimiento del modelo, crearemos una carpeta en `/app/machineModels/modelTests` con el nombre del nuevo módulo a implementar con los archivos necesarios para la prueba, tanto `.py` como `.csv`.
4. Quizás sea necesario guardar *encoders* para nuestro modelo y usarlos en el algoritmo de detección, es posible guardarlos con el nombre del nuevo módulo a implementar en `/app/machineModels/encoders`.
5. Ahora hay que desarrollar el módulo de detección en sí mismo. Se seguirán los requisitos indicados previamente y se alojará en `/defenseAlgorithms` con el mismo nombre que se ha usado en el resto de archivos.
6. ¡Listo! Para probar finalmente el nuevo módulo, simplemente hay que iniciar SecurAI en modo desarrollo es decir, sin generar los ejecutables. Para ello se ejecutarán los siguientes comandos (desde la carpeta base del repositorio):
    - 6.1. El desarrollador se sitúa en el directorio del *BackEnd* ejecutando desde la carpeta inicial del repositorio `cd /BackEnd`.
    - 6.2. El desarrollador instalará las dependencias ejecutando `pip install -r requirements.txt`, obviamente es necesario tener Python instalado (preferiblemente 3.9).
    - 6.3. El desarrollador ejecutará el *BackEnd* ejecutando `python3 run.py` desde la carpeta `/BackEnd`.
    - 6.4. El desarrollador entrará al directorio del *FrontEnd* ejecutando dede la desde la carpeta inicial del repositorio `cd /FrontEnd/app`.
    - 6.5. El desarrollador instalará las dependencias ejecutando `npm install -i`, de nuevo tendremos que tener Node.js instalado.
    - 6.6. El desarrollador ejecutará el *FrontEnd* ejecutando `npm run dev` desde la carpeta /FrontEnd, o podemos construir una build de producción ejecutando `npm run build`, y luego iniciarla con `npm run start`.

Como podemos observar en el flujo de trabajo aquí descrito, las carpetas `/machineTrain`, `/dataSetsOriginals`, `/dataSetsTransformed` y `/modelTests` son de uso exclusivo del desarrollo de

módulos y el entrenamiento de modelos y no se usa en la ejecución normal de SecurAI, donde se usan modelos ya entrenados, alojados en `/models`, y en caso de tenerlos, encoders en `/encoders`.

4.2.2.3 RECOMENDACIONES

Siguiendo el estilo de los módulos incluidos con la instalación base del programa, se recomienda tener cuatro secciones dentro del propio módulo con funciones claramente diferenciadas, con ánimo de mejorar la legibilidad y uniformidad del programa:

1. Importación del modelo a utilizar.
2. Creación de estructuras necesarias para el procesamiento de los paquetes.
3. Obtención y procesamiento del paquete.
4. Marcado del paquete como analizado, y obtención del siguiente paquete.

Es importante prestar especial atención al uso de *locks* para la obtención de mensajes de la cola de mensajes, para no generar problemas de concurrencia. También es imperativo tener cuidado con la obtención del siguiente mensaje de la cola de mensajes, ya que el proceso de limpiado de la misma es independiente al resto de procesos, por lo que el índice de paquetes puede cambiar.

El desarrollador puede encontrar más información y referencias en el Apartado Capítulo 5: Módulos ya implementados.

4.3 CONCURRENCIA DE PROCESOS

Debido a la filosofía modular y escalable de SecurAI, una cola de mensajes robusta es muy importante para que cada módulo y proceso sea independiente uno de otro, y no se generen incoherencias en el procesamiento de los paquetes.

4.3.1 EL PROBLEMA DE CONCURRENCIA

Cada módulo es un proceso independiente de los demás, y puede obtener los paquetes de forma y en un tiempo diferente. Además el proceso de limpieza de la cola de mensajes también es independiente, y tiene que respetar que cada módulo de detección procese los paquetes a su ritmo.

La hebra limpiadora no puede eliminar un paquete sin estar completamente procesado por todos los módulos independientes, y los módulos no saben cuando la hebra limpiadora actuará, así que no pueden confiar en la posición absoluta de los mensajes en la cola.

4.3.2 CÓMO INTERACTÚAN LOS PROCESOS

SecurAI en su funcionamiento normal está compuesto por multitud de procesos: módulos de detección, hebra limpiadora, módulos de ataque, notificación del estado del buffer... Todos esos módulos se sincronizan gracias el *lock* de la cola de mensajes, `packetBufferLock`.

Todos los procesos son lanzados automáticamente por SecurAI dependiendo de sus necesidades. Por ejemplo, detecta cuántos módulos de detección hay y lanza un proceso (realmente es una hebra) por cada uno de ellos.

4.3.3 LA COLA DE MENSAJES

La cola de mensajes es la estructura principal de SecurAI. La hebra capturadora deposita aquí los mensajes, los módulos de detección los obtienen también de aquí, y la hebra limpiadora la limpia para que no crezca demasiado.

En SecurAI existe una única estructura para todos los procesos independientes, y está definida en `packetCapture.py`. La cola de mensajes se llama realmente `PacketBuffer`, y es una lista de objetos `PacketIndexed`, que no es mas que un envoltorio que añade al propio paquete "metadatos", como un diccionario de los módulos que ya han procesado el paquete.

4.3.3.1 LA HEBRA LIMPIADORA

La hebra limpiadora se encarga de que la cola de mensajes no crezca en exceso. Se ejecuta periódicamente para eliminar los paquetes que ya han sido procesados por todos los módulos de detección.

La hebra limpiadora está definida en `bufferCleaner.py` y se ejecuta cada diez segundos. Mientras se ejecuta usa `packetBufferLock` para evitar la modificación de la cola de mensajes. También escribe información de depuración en la consola.

4.3.3.2 QUÉ HACER SI LA COLA DE MENSAJES CRECE EN EXCESO

Este comportamiento suele deberse a varios motivos, principalmente por errores en la implementación de los módulos de defensa:

- Un módulo de defensa ha marcado un paquete como completado, por lo que la hebra deja de limpiar al llegar a él.
- Un módulo no ha usado `packetBufferLock` y ha generado una excepción en la hebra limpiadora, deteniendo su ejecución.
- Hay muchos módulos activos y tardan en procesarse. Este comportamiento es esperable, no un error, y es por eso por lo que SecurAI permite desactivar módulos de defensa.

CAPÍTULO 5
MÓDULOS YA IMPLEMENTADOS

5.1 ESTRUCTURA GENERAL DE UN MÓDULO

Como se ha explicado en el Apartado 4.2.2.3: Recomendaciones para implementar un módulo, los módulos incluidos con la instalación base de SecurAI tienen cuatro secciones dentro del propio módulo con funciones claramente diferenciadas, con ánimo de mejorar la legibilidad y uniformidad del programa:

1. Importación del modelo a utilizar.
2. Creación de estructuras necesarias para el procesamiento de los paquetes.
3. Obtención y procesamiento del paquete.
4. Marcado del paquete como analizado, y obtención del siguiente paquete.

Aunque es muy sencillo implementar un módulo en SecurAI, es importante conocer cómo se relacionará este con el núcleo del sistema para evitar problemas de seguridad o de rendimiento.

Debido a la independencia entre módulos y procesos de SecurAI, no podemos confiar en la posición absoluta de paquetes en la cola de mensajes.

Es visible en los módulos ya incluidos en la instalación base de SecurAI que, cada vez que es necesario acceder a un paquete, este se busca activamente (véase Fig. 16): Si el paquete que se

analiza era el paquete número X, no podemos confiar en que el siguiente sea el paquete X + 1, ya que los índices pueden haber cambiando por ejemplo debido a la hebra limpiadora.

```
with packetBufferLock:
    current_index = packetBuffer.index(current_packet)
    remaining_packets = len(packetBuffer) - (current_index + 1)
    next_packet = packetBuffer[current_index + 1]
```

Fig 16: Búsqueda activa del siguiente paquete, cálculo del nuevo índice

Otro motivo para conocer cómo se relaciona el módulo con el núcleo del sistema de SecurAI es que el nombre del módulo será el nombre visible que aparecerá en el *FrontEnd*, así que se recomienda encarecidamente seguir la nomenclatura establecida para los archivos a crear.

5.2 ARP FLOODING

Estos módulos de defensa se centran en el estudio del protocolo ARP y su vulnerabilidad ARP Flooding o MAC Flooding, observando qué métodos son más eficaces para su detección en tiempo real usando técnicas de inteligencia artificial.

El protocolo ARP (Address Resolution Protocol) se usa en redes IPv4 para asociar una dirección IP con una dirección física (MAC) en una red local. Cuando un equipo quiere enviar datos a una IP concreta dentro de la misma red, ARP le permite descubrir qué dirección MAC tiene esa IP.

5.2.1 ¿QUÉ ES?

Consistente en consumir la memoria de la tabla de direcciones MAC de la víctima llenándola de contenido ilegítimo, ARP Flooding es un tipo de ataque de denegación de servicio (DoS) que impide el mantenimiento y uso de las entradas legítimas de la tabla de direcciones MAC del dispositivo y agota los recursos de procesamiento de estas solicitudes, impidiendo la correcta conexión del equipo atacado con la red a nivel de la capa de enlace (véase Fig. 17).

Durante un ataque ARP Flooding, los dispositivos de red deben procesar una gran cantidad de solicitudes ARP, lo que consume recursos significativos de CPU y memoria. Por ejemplo, se ha observado que en un entorno de red Ethernet rápida, un tráfico de ataque ARP que consume el

40% del ancho de banda puede agotar hasta el 85% de la capacidad de procesamiento de un procesador a 3.06 GHz [19].

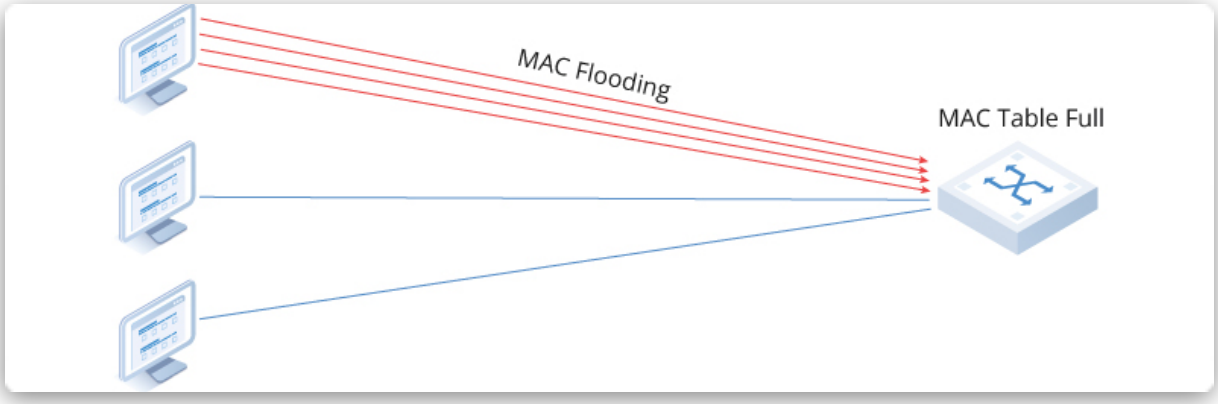


Fig 17: Esquema del ataque ARP Flooding. FS Technology. (2023). En Basic switch security concepts explained. FS. Recuperado el 9 de junio de 2025, de https://www.fs.com/es/blog/basic-switch-security-concepts-explained-6191.html

Este tipo de ataque es conocido por hacer notables daños en equipos personales [20] (como los susceptibles usuarios de SecurAI), aparte de equipos destinados a la gestión y funcionamiento de la red.

5.2.2 CASOS DE ÉXITO

Este tipo de ataque fue el primero que se desarrolló, y se estudiaron diversos enfoques para ver qué aproximación daba mejor rendimiento en su detección.

5.2.2.1 ARPFLOODING

`arpFlooding` fue el primer módulo que fue desarrollado. Una vez probamos su rendimiento en pruebas experimentales, fue la referencia para ver si el resto de módulos funcionaban correctamente, y marcó el modelo de lo que sería un módulo de detección.

Este módulo es muy rápido tanto en detectar ataques como en marcar que el entorno ya es seguro nuevamente. Quizás por eso, una vez que su eficiencia fue contrastada, se utilizó tanto como métrica para comparar el rendimiento con el resto de módulos de ataque ARP Flooding.

`arpFlooding` es un modelo de machine learning entrenado con un datasets que incluyen las siguientes características:

- mac_diferente_eth_arp: Indica si la dirección MAC en la capa Ethernet es diferente de la dirección MAC en el paquete ARP (1 si son distintas, 0 si son iguales).

- arp_packets_por_mac: Es el conteo acumulado de paquetes ARP emitidos por cada dirección MAC.
- arp_request_count: Número de solicitudes ARP (op_code = 1) emitidas por una dirección MAC.
- arp_reply_count: Número de respuestas ARP (op_code = 2) emitidas por una dirección MAC.
- ratio_request_reply: Relación entre el número de solicitudes y respuestas ARP para una dirección MAC, calculada como (arp_request_count) / (arp_reply_count + 1).
- unique_dst_ip_count: Para cada dirección MAC de origen en paquetes ARP, cuenta cuántas direcciones IP de destino distintas ha contactado esa MAC.

Estos contadores se generan a partir del dataset original [21], `dataSetsOriginals/arpFlooding.csv` y se guardan en el nuevo *dataset* `dataSetsTransformed/arpFlooding.csv`, que es el que al final se utiliza para el entrenamiento del modelo. A parte de generar estos contadores, hemos realizado procesamiento adicional al *dataset* original, como realizar un balanceo de clases (véase Fig. 18).

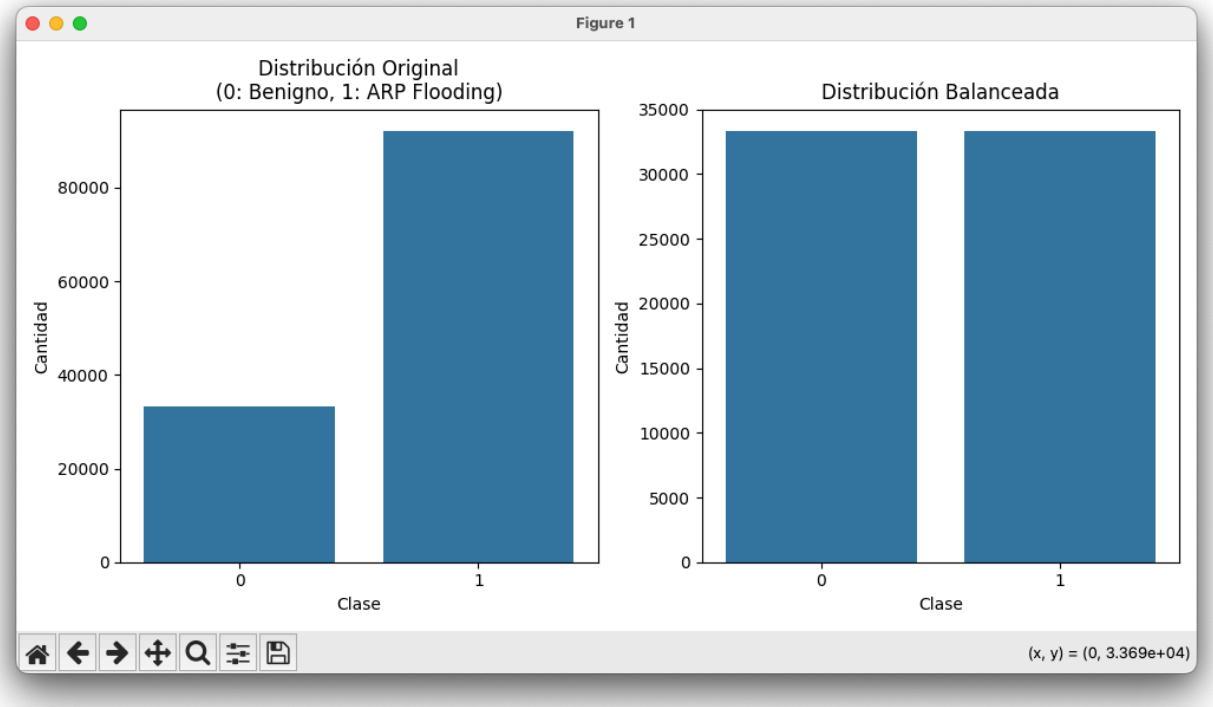


Fig 18: Resultado del balanceo de clases

En concreto, `arpFlooding` detecta los paquetes maliciosos mejor si son de tipo ARP Reply, al menos durante las pruebas experimentales realizadas. Esto suele ser suficiente para alertar al usuario. Las pruebas realizadas no son especialmente agresivas y pueden repetirse, ya que los módulos de

ataque se proporcionan con la instalación base de SecurAI, y este en concreto se explica en el Apartado 6.2.1 ARP Flooding.

Como punto negativo, los contadores que utiliza en este módulo no se reinician hasta que se reinicia SecurAI. Esto significa que con el paso de un considerable lapso de tiempo, que dependiendo de la afluencia de paquetes ARP del entorno puede variar entre horas y días, es posible que los paquetes ARP Reply se marquen como maliciosos sin estar bajo un ataque (falso positivo).

Para probar el modelo no solo se ha utilizado la simulación de ataque antes mencionada, sino que hemos usado otro *dataset* [22] diferente al que se usó para entrenarlo, `arpFlooding+.csv`. Así se podía comprobar cómo se comportaba en una situación que no era generaba por nosotros y podía asemejarse mucho a un uso real.

Para realizar el test se implementó un script que evalúa el modelo usando este dataset (`arpFlooding+.csv`) transformado para tener las mismas características que se usaron durante el entrenamiento. Además, para forzar el modelo lo máximo posible ampliamos el dataset artificialmente copiando datos ya existentes en el mismo hasta alcanzar nueve veces su tamaño.

Los resultados fueron muy buenos (ver Fig. 19) y tanto el script utilizado como los *datasets* modificados generados están disponibles en `/machineModels/modelTests/arpFlooding`.

	precision	recall	f1-score	support
0	0.81	1.00	0.89	9486
1	1.00	0.92	0.96	28962
accuracy			0.94	38448
macro avg	0.91	0.96	0.93	38448
weighted avg	0.95	0.94	0.94	38448

Fig 19: Rendimiento del modelo arpFlooding usando el dataset arpFlooding+ aumentado nueve veces su tamaño

5.2.2.2 ARPFLOODINGSW

`arpFloodingSW` es la evolución natural de `arpFlooding`, ya que soluciona el problema del falso positivo antes descrito. Como contrapartida, tarda más tiempo en detectar el ataque que su predecesor.

`arpFloodingSW` sigue la misma filosofía que que `arpFlooding`, añadiendo el concepto de ventana deslizante (*sliding window*). La ventana deslizante es una lista que almacena los paquetes ARP recibidos durante los últimos 90 segundos. Cada vez que llega un nuevo paquete, el sistema elimina los paquetes antiguos de la lista (más de 90 segundos de antigüedad), añade el nuevo paquete y calcula estadísticas solo con los paquetes dentro de esa ventana.

Para implementar esta característica se utilizó otro *dataset* [22], que contenía marcas de tiempo, disponible en `dataSetsOriginals/arpFlooding+.csv`.

Para el cálculo de las métricas en base al tiempo en el *dataset* de entrenamiento, es necesario que ordenarlo, ya que las tramas vienen desordenadas. Luego, tras el cálculo de las métricas es importante volver a desordenar las tramas del *dataset* antes de predecir. Esto mejora el rendimiento del modelo ya que los ataques en este conjunto de datos pueden concentrarse en zonas concreta del mismo, y el modelo saca de ese patrón conclusiones erróneas.

El cálculo del tamaño de la ventana no ha sido aleatorio, sino que ha sido sujeto de un concienzudo estudio. Se probaron iterativamente diferentes tamaños y comparado su rendimiento (véase Fig. 20). Los resultados están disponibles en `/machineModels/modelTests/arpFloodingSW`.

En concreto, `arpFloodingSW` detecta los paquetes maliciosos sólo si son de tipo ARP Request, al contrario que el `arpFlooding`, al menos durante nuestras pruebas experimentales. Esto suele ser suficiente para alertar al usuario.

Las pruebas realizadas no son especialmente agresivas y pueden repetirse, ya que los módulos de ataque se proporcionan con la instalación base de SecurAI, y este en concreto se explica en el Apartado 6.2.1 ARP Flooding.

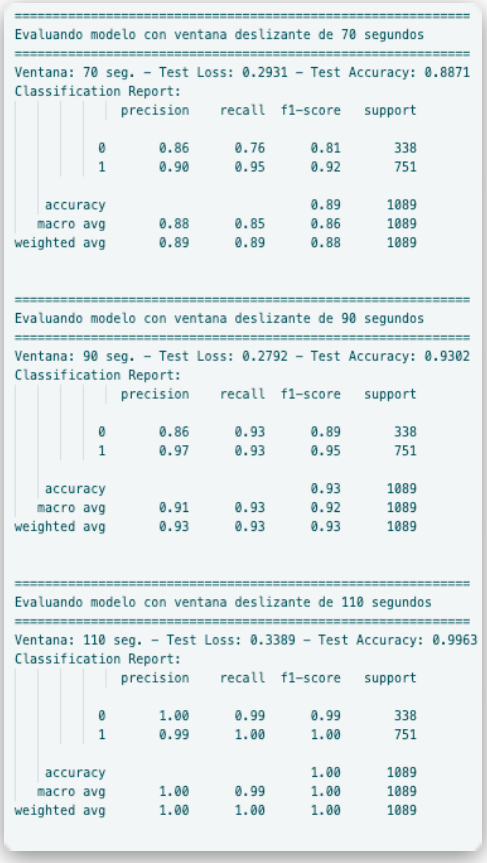


Fig 20: Resultados de la evaluación iterativa del tamaño de la ventana

Consideramos una opción muy interesante la ejecución de estos dos módulos `arpFlooding` y `arpFloodingSW` simultáneamente, ya que potencias sus fortalezas y mitiga sus debilidades.

5.2.2.3 ARPFLOODINGSVM

`arpFloodingSVM` supuso un cambio de paradigma, ya que en vez de usar redes neuronales como los anteriores usa máquinas de vectores de soporte (SVM), algoritmos muy para clasificación. Hablamos de esto anteriormente en el Apartado 2.1.3 SVM.

Este modelo también usa como base `arpFlooding`, y buscaba originalmente comparar el rendimiento de redes neuronales y SVM, el preprocesamiento de los datos es muy parecido a este. Para comprender la lógica del preprocesamiento de los datos recomendamos la lectura del Apartado 5.2.2.1 arpFlooding.

Al igual que `arpFlooding`, `arpFloodingSVM` en los experimentos realizados tiene un rendimiento muy bueno tanto para detectar un ataque como para notificar de que el entorno ya es seguro (véase Fig. 21).

Al final se consiguió un modelo muy robusto sin apenas cambios con el modelo original, en nuestra experiencia mucho más resistente al problema de `arpFlooding` de detectar los benignos como malignos (falso positivo) con el paso del tiempo sin tener que aplicar la lógica de la ventana deslizante.

Al igual que con el módulo `arpFlooding`, `arpFloodingSVM` detecta los paquetes maliciosos mejor si son de tipo ARP Reply, al menos durante nuestras pruebas experimentales. Esto suele ser suficiente para alertar al usuario. Las pruebas realizadas son relativamente leves y pueden repetirse para probar el rendimiento. Los módulos de ataque se proporcionan con la instalación base de SecurAI, y este en concreto se explica en el Apartado 6.2.1 ARP Flooding.

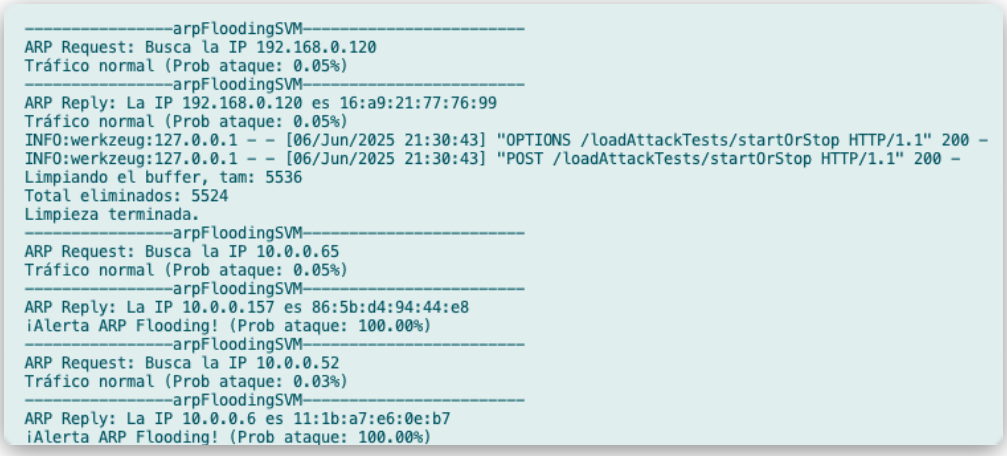


Fig 21: Rendimiento de arpFloodingSVM antes y tras activar la simulación del ataque experimental

5.2.3 INTENTOS FALLIDOS

En el desarrollo de SecurAI ha sido necesario lidiar con múltiples retos y se intentaron hacer funcionar módulos y tecnologías que al final no han dado buenos resultados. Aun así, el aprendizaje de ellos fueron notables.

5.2.3.1 ARPFLOODINGLSTM

En el proceso de desarrollo de los módulos para detectar ARP Flooding, la componente temporal de un ataque DoS nos hacía pensar que sería útil utilizar una red LSTM (Long Short-Term Memory).

Ya se profundizó en su funcionamiento en el Apartado 2.1.2.2 LSTM pero en definitiva, se observó que su capacidad de "memoria" podría tener un impacto decisivo en la detección de los paquetes en el tiempo. Ataques como ARP Flooding tienen una firma temporal: muchos paquetes similares, seguidos a intervalos regulares...

Sin embargo, los resultados, pese a múltiples intentos, fueron muy deficientes. Los modelos LSTM están diseñados para predecir secuencias temporales completas, y el enfoque secuencial usado, basado en ventanas de 50 paquetes generaba etiquetas para cada uno de ellos. Esta estructura, aunque coherente desde el punto de vista teórico, dificultaba la detección precisa en tiempo real en nuestras pruebas experimentales.

Las pruebas a las que este módulo ha sido sometido no son especialmente agresivas y pueden repetirse para probar su rendimiento, ya que los módulos de ataque se proporcionan con la instalación base de SecurAI, y este en concreto se explica en el apartado 6.2.1 ARP Flooding.

Además la intermitencia del ataque experimental, menos constante que en la teoría, reducía la efectividad del aprendizaje de patrones temporales consistentes.

En resumen, `arpFloodingLSTM` solo se presenta en la instalación base de SecurAI con fines de investigación.

5.3 TCP SYN

Tras la realización de todos los módulos para predecir el ataque ARP Flooding, se consideró oportuno estudiar otro tipo de ataque diferente. En este caso se cambió de capa y probó con un protocolo de la capa de transporte, como lo es TCP.

TCP (Transmission Control Protocol) es un protocolo que proporciona una comunicación fiable entre dispositivos a través de la red. Se basa en un proceso conocido como *three-way handshake* para iniciar una conexión, consistente en establecerla antes del intercambio de datos [23].

Durante esta negociación para el inicio de la conexión y en el resto de la comunicación, TCP utiliza una serie de indicadores o *flags* (como SYN, ACK, FIN o RST) que indican el propósito de cada paquete dentro del flujo de conexión.

En concreto, el proceso de negociación de inicio de la conexión (*three-way handshake*) consta de tres pasos: el cliente envía un paquete SYN, el servidor responde con un SYN-ACK, y el cliente finaliza con un ACK (véase Fig. 22).

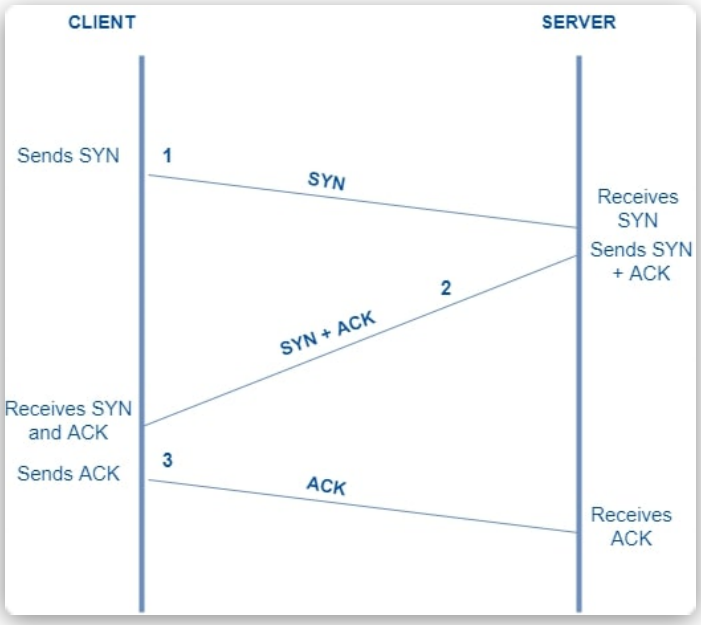


Fig 22: Ilustración del three-way handshake de TCP. Adaptado de TCP Three-Way Handshake in Computer Networks, por Workat.tech, s.f. https://workat.tech/core-cs/tutorial/tcp-three-way-handshake-in-computer-networks-yoo7331910lh

5.3.1 ¿QUÉ ES?

El ataque TCP SYN es un tipo de ataque de denegación de servicio (DoS) que explota el proceso de establecimiento de conexiones (*handshake*) del protocolo TCP.

El ataque TCP SYN consiste en enviar una gran cantidad de solicitudes de conexión (paquetes con el *flag* SYN activo) a un servidor, sin completar el proceso de enlace de tres vías (*three-way handshake*). Esto hace que el servidor mantenga abiertas múltiples conexiones incompletas,

consumiendo recursos y eventualmente impidiendo que usuarios legítimos establezcan nuevas conexiones (véase Fig. 23) [24].

Este ataque se aprovecha de la forma en que el protocolo TCP maneja las solicitudes de conexión entrantes, manteniéndolas en una cola de espera hasta que se complete el *handshake*. Al inundar esta cola con solicitudes falsas, el atacante puede saturar el sistema.

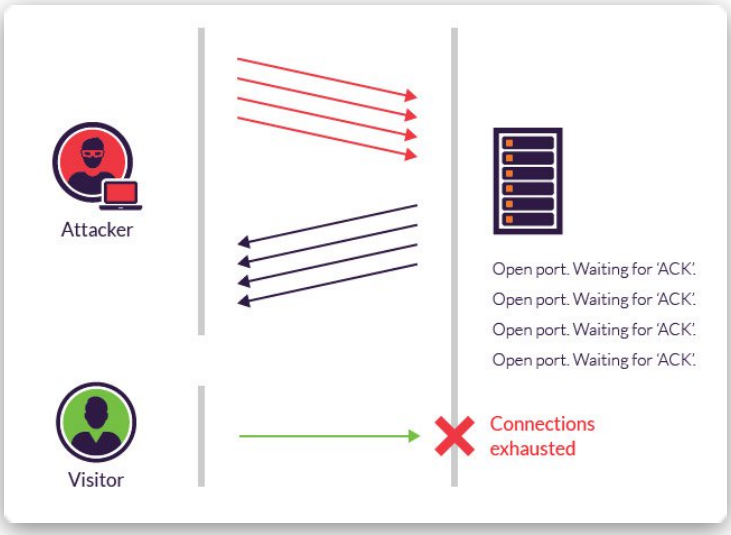


Fig 23: Diagrama del ataque TCP SYN Flood. Adaptado de What is a SYN Flood Attack?, por Imperva, s.f. https://www.imperva.com/learn/ddos/syn-flood/

5.3.2 EL MÓDULO

A la hora de desarrollar un módulo, era muy importante ir evaluando su rendimiento con los script de ataque de prueba. La primera dificultad fue averiguar cómo se podían enviar paquetes al mismo equipo que detectaba el ataque. Esta pregunta no surgió con los módulos anteriores, ya que ARP usa paquetes que se difunden por toda la red.

Las pruebas que hemos realizado son relativamente leves y pueden repetirse para comprobar el funcionamiento de este módulo de defensa. Los módulos de ataque utilizados se proporcionan con la instalación base de SecurAI, y este en concreto se explica en el Apartado 6.2.2. TCP SYN, donde se explicará en detalle cómo se solucionó ese problema.

El *dataset* [25] que utiliza el módulo `tcpSYN` ha sido creado a partir de archivos `.pcap` y dentro del proyecto está disponible en el archivo `dataSetsOriginals/tcpSYN.csv`. El procesamiento se realizó con un script de Python que identificaba los flujos de comunicación TCP/IP únicos. Por cada paquete se extraen características como *flags*, tiempo entre paquetes, o número de conexiones incompletas (en una ventana deslizante de tiempo de 3 minutos). El dataset resultante balancea la cantidad de paquetes de las clases ataque y benigno.

Para el entrenamiento se eliminó la columna *Flow ID* antes de entrenar, porque no aporta valor predictivo y podría sesgar el modelo, el *dataset* final se puede encontrar en `dataSetsTransformed/tcpSYN.csv`.

5.4 DNS AMPLIFICATION

Con este módulo de ataque se cubren distintas capas del modelo OSI relevantes para el ámbito de la ciberseguridad en redes: ARP afecta a la capa de enlace/red, TCP SYN flood ataca a la capa de transporte, y DNS amplification se dirige a la capa de aplicación. Este fue, de hecho, el último módulo que se desarrolló,

Para la capa de aplicación se eligió un tipo de ataque basado en el protocolo DNS (*Domain Name System*), fundamental para la resolución de nombres de dominio en direcciones IP.

Un nombre de dominio es una etiqueta legible por humanos, como www.ejemplo.com, que identifica un recurso en la red, proporcionando una forma fácil de recordar y acceder a sitios web y servicios en línea [26].

DNS es un sistema jerárquico y distribuido que traduce los nombres de dominio a direcciones IP numéricas que los ordenadores utilizan para comunicarse entre sí. Cuando un usuario quiere acceder a una web y lo escribe en el navegador, el ordenador mediante el protocolo DNS sabe a qué IP tiene que conectarse (ver Fig. 24).

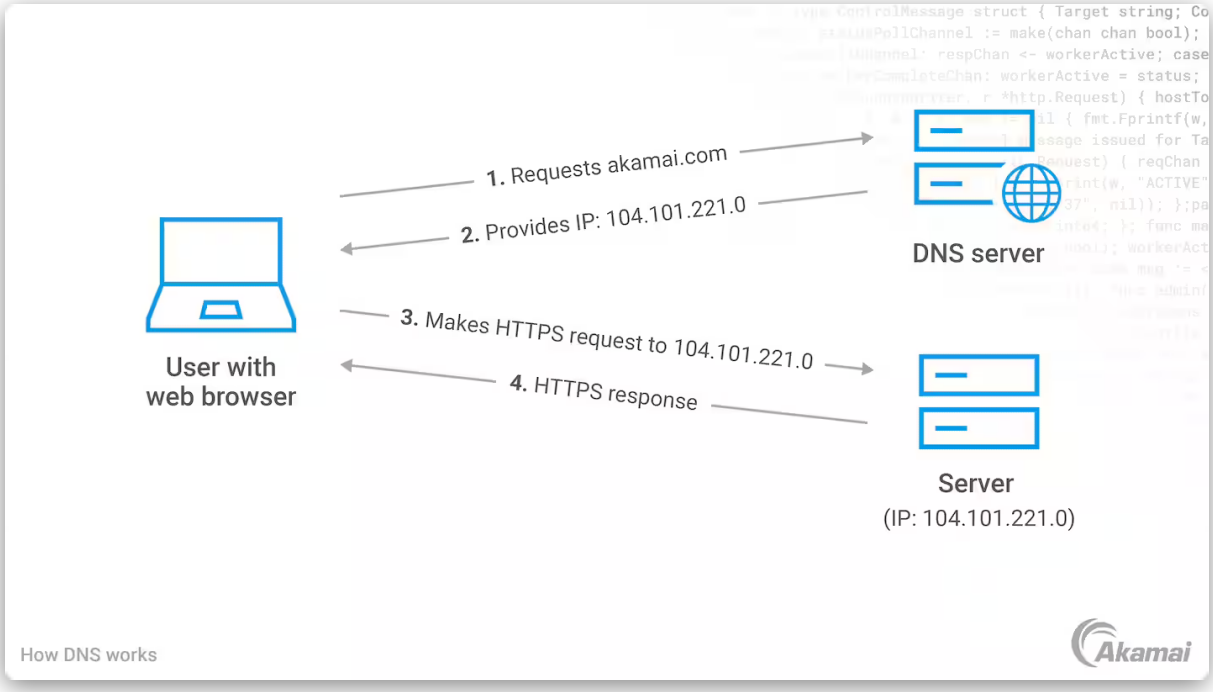


Fig 24: Esquema del funcionamiento de DNS. What are DNS servers? Akamai. https://www.akamai.com/es/glossary/what-are-dns-servers

5.3.1 ¿QUÉ ES?

El ataque DNS Amplification es un tipo de ataque de denegación de servicio distribuido (DDoS) que aprovecha la arquitectura de servidores DNS abiertos para amplificar el volumen de tráfico dirigido a una víctima [27].

El término denegación de servicio distribuido se refiere a un ataque que usa múltiples sistemas (distribuidos) para saturar un objetivo y dejarlo inaccesible.

El atacante envía de esta manera solicitudes DNS con la dirección IP origen falsificada. Colocan como origen de la solicitud DNS la dirección IP del objetivo, y lo envían a servidores DNS públicos, que responden con mensajes mucho más grandes. Esta amplificación puede multiplicar el tráfico enviado, saturando la red y recursos del objetivo.

5.3.2 EL MÓDULO

El módulo `dnsAmplification` sigue la misma filosofía de diseño que los módulos anteriores basados en redes neuronales. El *dataset* original [28-32] fue procesado para quedarse con un conjunto de características reducido, manteniendo únicamente aquellas que consideramos más relevantes para la detección del ataque:

- dbytes: Número de bytes transferidos desde el destino al origen.
- ct_dst_ltm: Número de conexiones que tienen la misma dirección de destino en las últimas 100 conexiones, según la marca de tiempo. Ayuda a detectar si una IP está siendo objetivo de muchas conexiones.
- ct_src_dport_ltm: Número de conexiones que tienen la misma dirección de origen y el mismo puerto de destino en las últimas 100 conexiones. Puede indicar escaneos o repeticiones sospechosas hacia un mismo servicio.
- ct_dst_src_ltm: Número de conexiones entre un mismo par origen-destino en las últimas 100 conexiones.
- Label: Variable objetivo, identifica si el paquete pertenece a tráfico benigno o no.

El conjunto resultante se balanceó mediante *undersampling* exacto (reducción de la clase mayoritaria hasta igualar el número de muestras de la minoritaria), técnica que ya había sido

probada con éxito en otros módulos, para evitar que el modelo aprendiera un sesgo hacia la clase mayoritaria.

El *dataset* resultante del procesamiento se puede encontrar en `dataSetsTransformed/dnsAmplification` y en las pruebas experimentales ha muy buen resultado, similar a `arpFlooding` a nivel de rapidez de detección y de volver a considerar un entorno como seguro tras un ataque.

Las pruebas realizadas aún así no han sido demasiado largas para evitar problemas con los servidores DNS. Las pruebas no son especialmente agresivas y pueden repetirse en cualquier momento para comprobar su rendimiento, ya que los módulos de ataque se proporcionan con la instalación base de SecurAI, y este en concreto se explica en el Apartado 6.2.3. DNS Amplification.

CAPÍTULO 6
FUNCIONALIDADES EXTRA

En este capítulo se detallan funcionalidades adicionales a los objetivos definidos en el Apartado 1.2. Objetivos. Se han considerado que estas características eran de utilidad y encajaban en el alcance del proyecto, ya que contribuyen a mejorar la experiencia de uso, la capacidad de análisis y el valor práctico de SecurAI como herramienta de ciberseguridad.

6.1 RECOPILACIÓN DE ESTADÍSTICAS

Durante el desarrollo de SecurAI se notó la necesidad de conocer el estado de la red que estábamos analizando.

Por ejemplo, a la hora de comprender las predicciones en las fases beta de los módulos de defensa, era útil saber si la red estaba congestionada en ese momento. Para comprobar la robustez de la hebra limpiadora y en general de la gestión de la concurrencia era muy útil comprobar la afluencia de paquetes en cada momento. También era muy útil saber el estado de la red a la hora de simular ataques desde SecurAI, para comprobar que se generaban adecuadamente.

Observando la utilidad de esta herramienta para el equipo de desarrollo, se supuso que también lo sería para los usuarios de SecurAI, así que se estableció como objetivo crear una herramienta ligera y bien integrada en la interfaz de la aplicación (véase Fig. 25), utilizable por todos.

Fig 25: Acceso a la sección de estadísticas desde el FrontEnd

La lógica de la creación de los gráficos está en el *FrontEnd* de SecurAI, y no depende del *BackEnd* en ningún momento, `packetCapture.py` simplemente emite, mediante un socket información de cada paquete que captura.

Siguiendo la filosofía de SecurAI, el desarrollador que quiera modificar la aplicación debe de poder hacerlo fácilmente, y por lo general no debe de tener que modificar el *FrontEnd*. Al desacoplar la visualización de estadísticas del *BackEnd* permitimos que su funcionamiento sea completamente transparente tanto para el usuario como para el desarrollador.

6.1.1 NÚMERO DE PAQUETES

Esta sección genera un gráfico temporal (véase Fig. 26) sobre el número de paquetes que recibe el equipo que ejecuta SecurAI. El gráfico comienza el conteo en cuanto se accede a la página (no corre continuamente en segundo plano) y se actualiza cada cinco segundos, actualizándose en tiempo real. Para evitar un crecimiento indefinido del historial, solo se guardan los últimos 15 puntos de muestreo, simulando así una "ventana deslizante" temporal. Esta ventana puede cambiarse fácilmente por el desarrollador modificando el `netStats/numerStats/page.js` en el *FrontEnd* si lo considera necesario.

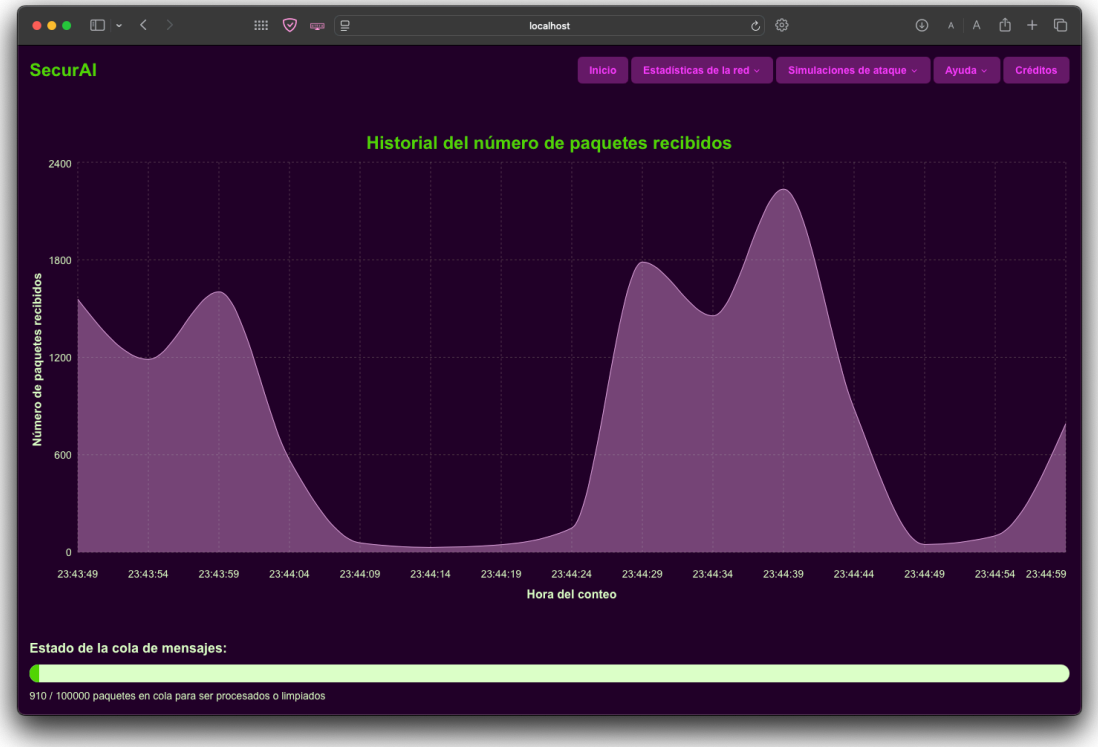


Fig 26: Página del FrontEnd para la visualización del historial de número de paquetes recibidos

Esta funcionalidad permite al usuario observar visualmente patrones como picos de tráfico, congestiones o comportamientos anómalos. Para observar claramente esto, proponemos al lector activar o desactivar alguna simulación de ataque mientras observa el gráfico.

Para la realización de esta funcionalidad se utilizaron *hooks* de React, que son una herramienta que permite a los componentes funcionales mantener estado interno, reaccionar a eventos, realizar operaciones periódicas... de forma ordenada y eficiente.

En concreto, se creó un *custom hook* llamado `useStatsSocket` para encapsular la lógica de escucha del socket, recuento de paquetes y almacenamiento del historial. Esto permite separar responsabilidades: el *hook* se ocupa de gestionar los datos y el componente `numberStats` se dedica únicamente a representarlos.

6.1.2 TIPOS DE PAQUETES

Esta sección genera un gráfico circular (véase Fig. 27) sobre el tipo de paquetes que recibe el equipo que ejecuta SecurAI. A diferencia del histórico del apartado anterior, que usa un intervalo fijo de cinco segundos, este gráfico se actualiza inmediatamente cada vez que el *BackEnd* captura un nuevo paquete.

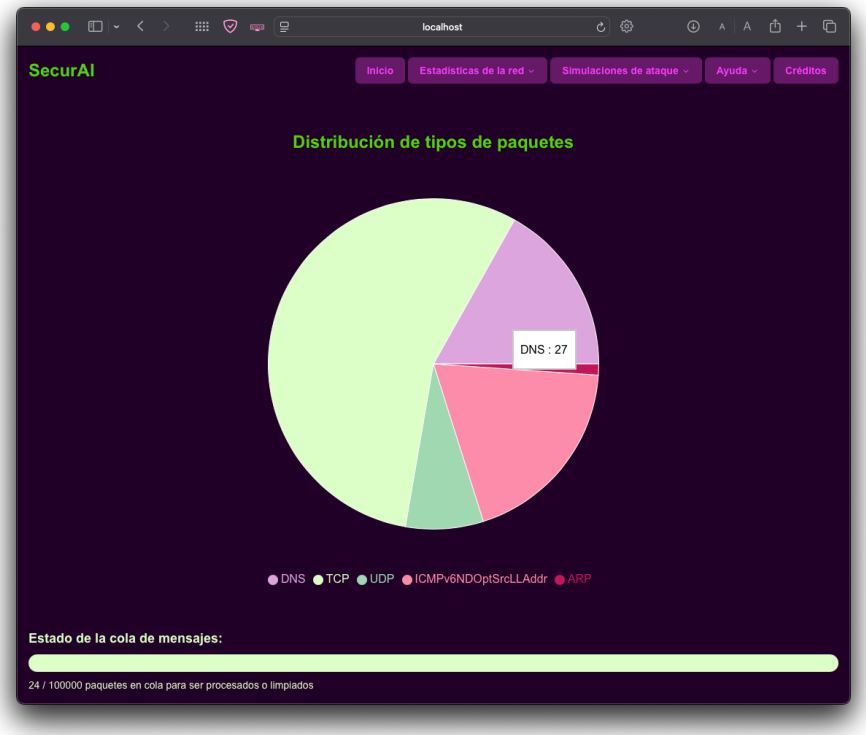


Fig 27: Página del FrontEnd para la visualización de los tipos de paquetes recibidos

Para la compresión de esta características es importante saber que lo que se denomina el "tipo de un paquete" se refiere al protocolo que usa en su capa más alta. Esta capa se interpreta como la que define el propósito más específico del paquete, por lo que es útil para evaluar la naturaleza del tráfico que circula por la red.

Esta identificación se hace cuando se captura el paquete y se indexa, en `packetCapture.py`, y es independiente a la carga del gráfico. En se realiza en el método `get_last_layer()` de la clase `PacketIndexed`, que recorre todas las capas del paquete desde la más baja hasta la más alta, y devuelve la más alta exceptuando capas genéricas como *Raw* o *Padding*, que no aportan información relevante para el análisis.

Al igual que en la funcionalidad anterior, para la realización de esta se utilizan hooks de React. En concreto, creamos un *custom hook* llamado `useTypeStatsSocket` para encapsular la lógica de escucha del socket y acumulación del recuento por protocolo. Esto permite separar, de nuevo, responsabilidades: el *hook* se encarga de procesar y mantener los datos actualizados, mientras que el componente `typeStats` se dedica únicamente a representarlos en un gráfico circular de forma clara y reactiva.

6.2 SIMULACIONES DE ATAQUES

Las simulaciones de ataque han sido clave para el desarrollo de SecurAI. Consideramos oportuno ir más allá del marco teórico y probar si los módulos de defensa desarrollados realmente detectaban un ataque, y para ello creamos los módulos de ataque.

Al igual que con el visionado de estadísticas, esta característica evolucionó de un script de desarrollo a una herramienta ligera y bien integrada en la interfaz de la aplicación (véase Fig. 28), utilizable por todos.

Aunque los ataques en general son internos, el ataque DNS Amplification si usa equipos externos. Recordar al lector la importancia de un uso responsable de la red, y utilizar los módulos de ataque en redes propias.

Fig 28: Activación o desactivación de módulos desde la interfaz

6.2.1 ARP FLOODING

El ataque ARP Flooding implementado se ejecuta mediante el envío continuo de tramas ARP falsificadas, tanto de tipo request como reply, utilizando direcciones IP generadas aleatoriamente dentro de la red (10.0.0.X) y direcciones MAC también aleatorias. Ambas tramas se encapsulan en tramas Ethernet con dirección de destino tipo *broadcast* (difusión), asegurando su propagación por toda la red local.

Se ha incorporado una pausa cada 200 paquetes para simular comportamientos más realistas y evitar bloqueos de red o detección inmediata. Además así se comprueba que el módulo es sensible a ataques no demasiado voluminosos.

6.2.2 TCP SYN

Para simular el ataque TCP SYN Flooding de forma realista, el script genera paquetes TCP con flag SYN activado y con IPs y puertos de origen aleatorios. En lugar de enviar los paquetes a la dirección *localhost* (127.0.0.1), que Scapy no puede interceptar correctamente debido a que los paquetes nunca llegarían a la interfaz física del sistema, se resuelve dinámicamente la IP local real del equipo (por ejemplo, 192.168.X.X) utilizando un socket UDP apuntando a 8.8.8.8.

Para la correcta realización del ataque y para evitar enviar un paquete TCP a un puerto cerrado, se abre temporalmente el puerto de destino con un socket TCP antes de enviar el paquete, emulando un servidor esperando conexiones.

El uso de IPs públicas y privadas aleatorias como IPs de origen es fundamental para simular tráfico malicioso sin necesidad de control de múltiples dispositivos. La velocidad de envío se regula con `sleep(0.01)` para balancear carga y detección, además así se comprueba que el módulo es sensible a ataques no demasiado voluminosos.

6.2.3 DNS AMPLIFICATION

En este ataque se construyen solicitudes DNS con el tipo de consulta ANY, el cual le indica al servidor DNS que devuelva toda la información conocida para un dominio dado. Este tipo de consulta no suele usarse en tráfico legítimo y es conocido por generar respuestas muy grandes incluso con solicitudes pequeñas, lo que permite un alto grado de amplificación.

El script selecciona aleatoriamente entre una lista de servidores DNS públicos (Google, Cloudflare, Quad9, entre otros) y dominios válidos.

El tipo ANY fue elegido estratégicamente por su potencial de amplificación, en contraste con tipos comunes como A o MX, que devuelven respuestas mucho más acotadas. Aunque algunos servidores modernos limitan su uso, muchos aún lo aceptan sin restricciones, lo cual se refleja en la selección diversa de servidores. Las peticiones se envían a intervalos controlados (`sleep(0.3)`) para evitar ser bloqueados durante las pruebas, además así se comprueba que el módulo es sensible a ataques no demasiado voluminosos.

Este tipo de ataque **sí** que envía contenido del equipo de lo ejecuta hacia internet. Se recomienda un uso muy contado y limitado de esta herramienta.

6.3 GUARDADO DE LOGS

Esta funcionalidad fue desarrollada en las últimas etapas del desarrollo de SecurAI, siguiendo la filosofía de la metodología iterativa que se adoptó durante el proyecto.

Tras experimentar algunos problemas a la hora de desplegar la aplicación en Windows, donde no siempre quedaba claro si el fallo residía en el *FrontEnd*, el *BackEnd* o en algún proceso intermedio como la instalación de Npcap, fue imperativo observar y estudiar qué ocurría exactamente, así que se implementó la funcionalidad de guardado de logs (véase Fig. 29).

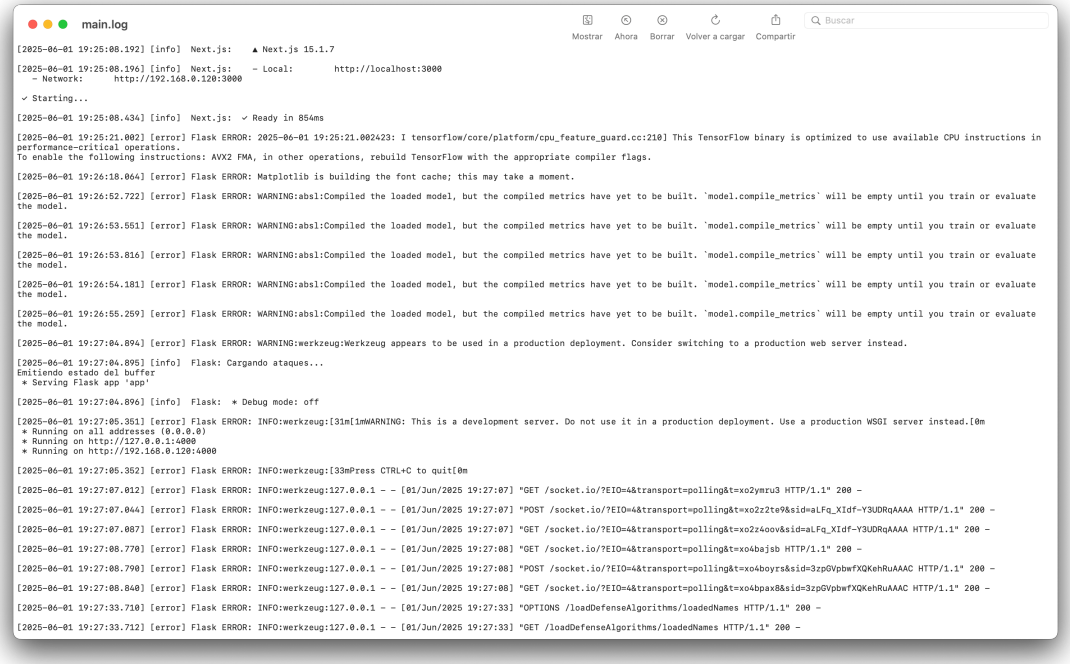


Fig 29: Logs generados en un sistema MacOS

Para ello, se utilizó el paquete electron-log, que permite registrar las salidas estándar que se generan en el propio proceso principal de Electron para hacer puntos de control sobre el proceso del arranque de la aplicación.

Los logs se almacenan en la ruta determinada por `app.getPath('logs')`, que es compatible tanto con Windows como con macOS:

- En Windows: La ruta típicamente es `C:\Users\<Usuario>\AppData\Roaming\SecurAI\logs`.
- En MacOS: `La ruta típicamente es /Users/<Usuario>/Library/Logs/SecurAI/`.

6.4 CI/CD

Un objetivo que se complicó considerablemente más de lo esperado fueron las capacidades multiplataforma de SecurAI. Para generar un ejecutable final es necesario generar primero un ejecutable del *BackEnd*, y luego tener dos proyectos de Node.js separados: uno es el *FrontEnd* (con una build de producción realizada por Next.js), y el otro es el proyecto de Electron (la herramienta que utilizamos para generar el ejecutable final).

El proceso de generar una nueva versión del programa puede implicar sin técnicas de CI/CD a fin de cuentas unos veinte minutos de tiempo y unos veinte gigabytes en disco, magnitudes completamente inasumibles si es necesario hacer pequeños cambios para probar el funcionamiento de la versión final.

Otro gran problema es que todo el programa fue desarrollado usando MacOS, y los ejecutables del *BackEnd* eran compilados para ese sistema operativo. Sin acceso a una máquina Windows, era imposible la creación de una versión para este sistema.

Estos problemas se resolvieron al adoptar técnicas de integración continua y entrega continua (CI/CD) mediante un único *workflow* de la plataforma GitHub Actions que agrupa todos los pasos de compilación y publicación.

Se explica en detalle cómo solucionamos el problema de la compatibilidad multiplataforma el Apartado 4.1.3. Empaquetado multiplataforma. También detallamos conceptos sobre tecnologías relacionadas en los apartados 2.2.1.4. PyInstaller y 2.2.2.3. Electron.

Entrando en detalle sobre la funcionalidad de CI/CD, esta permite generar una nueva *release* con nuevos ejecutables automáticamente cuando el repositorio de GitHub detecta un cambio en los archivos, y no es necesario hacerlo manualmente (véase Fig. 30). GitHub Actions se encarga de

detectar los cambios, construir la aplicación y generar automáticamente una nueva release lista para distribuir.

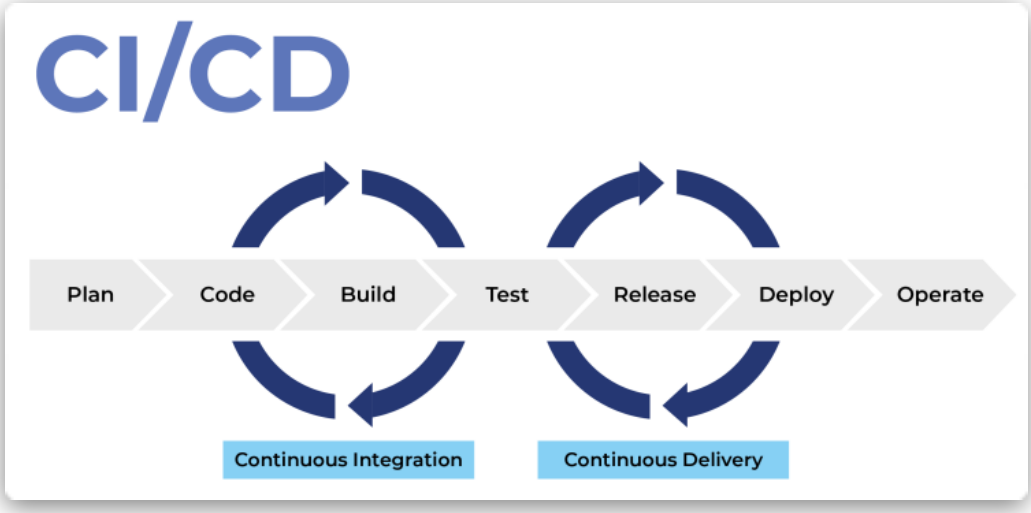


Fig 30: Esquema básico de Entrega Continua, Despliegue Continuo. Imagen reutilizada por Pankaj Gaikar, en Velotio (2023), Disponible en: https://www.velotio.com/engineering-blog/github-ci-cd-vs-xcode-cloud-a-comprehensive-comparison-for-ios-platform

Además, al tratarse de un repositorio público, es posible aprovechar los *runners* gratuitos que ofrece GitHub, lo que nos ha permitido implementar todo el flujo de CI/CD sin coste, ideal para proyectos académicos y de código abierto como SecurAI.

Para lograr nuestra automatización se han utilizado elementos clave de GitHub Actions, como el `job` y la funcionalidad de `strategy.matrix`, que permiten definir de forma estructurada y flexible los pasos a ejecutar en distintos entornos.

Un `job` (trabajo) en GitHub Actions representa un conjunto de pasos (`steps`) que se ejecutan en un mismo entorno de ejecución (por ejemplo, una máquina virtual con Windows). Por otro lado, la propiedad `strategy.matrix` permite definir una matriz de combinaciones de valores (por ejemplo, sistemas operativos) que se utilizarán para ejecutar automáticamente múltiples versiones del mismo `job` en paralelo sin tener que duplicar el código.

Otro aspecto importante en GitHub Actions es entender el uso de la caché, que permite almacenar archivos o dependencias entre ejecuciones de *workflows*, para evitar tener que descargarlas o reconstruirlas desde cero cada vez, ahorrando tiempo y recursos. Esto es especialmente útil cuando se trabaja con herramientas que mucho trabajo de cómputo o descarga de archivos, como gestores de paquetes (npm o pip en nuestro caso) o compilaciones intermedias.

En el caso concreto del workflow utilizado en este proyecto, disponible en `.github/workflows`, se ha definido un `job` que, gracias a la funcionalidad de `strategy.matrix`, se ejecuta de forma paralela en tres entornos distintos (`macOS-13`, `macOS-15` y `Windows-latest`), lo que permite generar los

ejecutables finales para ambos sistemas operativos y arquitecturas sin necesidad de disponer físicamente de una máquina Windows.

El *workflow* se dispara (comienza a ejecutarse) en dos situaciones concretas:

- `workflow_dispatch`: ejecución manual desde la interfaz de GitHub.
- `push`: cuando se realizan cambios en cualquier archivo del *BackEnd*, del *FrontEnd*, en archivos de Electron como `main.js` o `package.json` o en los propios flujos de trabajo (`.github/workflows/*.yaml`).

A continuación, se describe brevemente cada uno de los `steps` (pasos de ejecución):

1. Checkout del código: Se descarga el repositorio en el *runner* para disponer de la última versión de todos los ficheros.
2. Inicialización de Python: Se instala Python 3.9 y habilitamos el cacheo de dependencias de pip, lo que reduce el tiempo de instalación en ejecuciones sucesivas.
3. Instalación de dependencias del *BackEnd*: Se ejecuta `pip install -r BackEnd/requirements.txt` para traer todas las librerías necesarias a la máquina de GitHub. La caché de las dependencias son gestionadas automáticamente por `actions/setup-python@v5`.
4. Uso de la versión específica de TensorFlow: Este paso está diseñado para forzar al *runner* de MacOS arm64 a instalar su versión específica de TensorFlow para evitar problemas de compatibilidad.
5. Caché y compilación del *BackEnd* con PyInstaller: Se comprueba si ya hay un caché válido del *BackEnd* construido. En caso negativo se guarda en caché las carpetas `BackEnd/build` y `BackEnd/dist`, utilizando como clave la combinación del sistema operativo, arquitectura y el hash de los ficheros fuente y del `.spec` de PyInstaller. Gracias a `strategy.matrix` generamos los tres ejecutables del *BackEnd* para los dos sistemas operativos en este paso.
6. Inicialización de Node.js: Configuramos Node.js 20.18.3 y habilitamos cacheo de npm.
7. Instalación de dependencias del *FrontEnd*: Se instalan las dependencias necesarias para la ejecución del *FrontEnd* en caso de que no estén ya presentes en la caché, la cual está gestionada automáticamente por `actions/setup-node@v4`.
8. Caché y compilación del *FrontEnd* con Next.js: Se cachea la carpeta `.next`, que alberga la build del *FrontEnd*. En caso de que no exista, se genera en el *runner* ejecutando `npm run build`.

9. Descarga de Node.js embebido: Para evitar que el usuario final tenga que tener Node.js instado en su equipo, se proporciona una versión en el instalador que servirá al *FrontEnd*.
10. Instalación de dependencias de Electron: Se instalan las dependencias necesarias para la ejecución de Electron para que empaquete la aplicación. La caché de dependencias sigue estando gestionada automáticamente por `actions/setup-node@v4`.
11. Caché y compilación del proyecto raíz: Se cachea la carpeta `/dist`, que alberga la release y otros archivos importantes generados por Electron. Si no existe caché, se invoca `npx electron-builder` con el parámetro adecuado según el sistema operativo (`--mac --x64` , `--mac –arm64` o `--win –x64`). Es muy importante el filtro del sistema operativo en este paso, ya que gracias a `strategy.matrix` este paso se ejecutará tanto en una máquina Windows como en dos máquinas MacOS. Si los tres *runners* generaran las versiones para Windows y Mac, como ocurre por defecto, existirían redundancias en el número de ejecutables.
12. Publicamos la release: Obteniendo la versión del archivo `package.json`, se renombran los instaladores para que sean más accesibles y, si el cambio se ha hecho a la rama main del repositorio, se publica el instalador en GitHub Releases usando el token `GH_TOKEN`.

CAPÍTULO 7
CONCLUSIONES Y TRABAJOS FUTUROS

En este apartado se le da punto y final al proyecto, hablando sobre las conclusiones que se han sacado de su desarrollo. Adicionalmente, se comentan algunos puntos que podrían mejorarse en el futuro, con el fin de ampliar y mejorar SecurAI.

7.1 CONCLUSIONES

SecurAI ha sido un proyecto apasionante, gracias al cual hemos podido aprender muchísimo sobre el uso de técnicas de inteligencia artificial, la ciberseguridad de las redes y el despliegue de aplicaciones multiplataforma.

El mundo de la ciberseguridad es increíblemente complejo, y hemos aprendido a valorar a la comunidad investigadora que proporciona *datasets* de libre uso académico para el estudio de diferentes ataques.

Cada módulo desarrollado fue un reto diferente al anterior. Aunque íbamos aprendiendo poco a poco y agilizábamos el flujo de trabajo, debido a las particularidades de cada ataque el proceso era lento y "personalizado" para cada módulo. `arpFlooding` fue el primer módulo que desarrollamos, y quizás por eso le tenemos especial cariño, fue un evento cuando conseguimos hacerlo funcionar.

Otro elemento muy importante (si no el que más) de SecurAI es el núcleo y su arquitectura modular. Pensamos mucho tiempo cómo podríamos implementarlo de la forma más ordenada y escalable posible. Para ello fue fundamental la aplicación de los conocimientos de concurrencia adquiridos durante la carrera.

El hecho de integrar las funcionalidades que teníamos en mente en una aplicación accesible y fácil de usar ha sido un verdadero quebradero de cabeza. Por ejemplo el hecho de no poder usar el modo monitor nos limitaba mucho los ataques a detectar, por no hablar de las dificultades que hemos encontrado a la hora de implementar las funcionalidades de CI/CD.

Sobre esto último, fue duro de asimilar cuando descubrimos que, aunque la aplicación en sí ya era perfectamente funcional, nos quedaba aún bastante trabajo para poder compilarlo en los diferentes sistemas propuestos.

Aún así, estamos muy contentos de poder lanzar SecurAI como una aplicación fácilmente instalable sin necesidad de, por ejemplo, instalar Node.js o Python por separado. ¡Nos hace ilusión que todo el mundo pueda probar la plataforma que hemos creado!

En definitiva, estamos muy contentos de que SecurAI haya salido a la luz y estamos muy orgullosos de nuestro proyecto. Creemos firmemente en él y en que puede convertirse en una herramienta de utilidad para muchas personas.

7.2 TRABAJOS FUTUROS

SecurAI no es ni mucho menos perfecto. Para su desarrollo hemos tenido tiempo y recursos limitado, por lo que no se han podido implementar todas las funcionalidades deseadas. Se deja por escrito aquí algunas de ellas:

- Más módulos: Sin duda la mejora más obvia para SecurAI es que pudiera detectar más tipos de ataque. Debido a la naturaleza cambiante de paradigma de las redes y la ciberseguridad el proyecto está pensado para ser fácilmente ampliable y escalable, ya que nunca vamos a poder estar a la vanguardia con nuestros recursos actuales, y confiamos en que la comunidad pueda adaptar SecurAI a sus necesidades. Sin embargo, una ampliación de los módulos distribuidos con la instalación base de SecurAI podría ser un impulso para él mismo, que aparte de cubrir más escenarios de ataque puede servir como fuente de inspiración a los desarrolladores para crear más módulos aún.
- Más ataques: En principio los módulos de ataque no están pensados para ampliarse por un desarrollador típico de SecurAI, por lo que no es una tarea tan sumamente sencilla como es ampliar un nuevo módulo. Aunque la simulación del ataque puede hacerse desde fuera de la propia aplicación y desde fuera del propio equipo, lo que además crearía un entorno de simulación

mas cercano a la realidad, añadir más tipos de ataque genéricos preestablecidos a SecurAI puede ser una gran ayuda para el desarrollo de nuevos módulos, además de servir como aprendizaje a los usuarios de a pie.

- Capacidad de guardar tráfico: Esta funcionalidad sería mas compleja de implementar, ya que implicaría algunos cambios en el núcleo de SecurAI, como añadir condiciones de limpieza para la hebra limpiadora o la creación de otra hebra que recupere mensajes de la cola de mensajes antes de borrarlos. El hecho de tener un botón *rec* para empezar a grabar tráfico e incluso poder analizarlo luego sería de mucha utilidad en condiciones en la que se quiere estudiar un caso concreto de ataque sufrido, o para analizar mejor futuros módulos de ataque incluidos.
- Notificaciones de escritorio: Actualmente es necesario la vigilancia activa de la interfaz de SecurAI para observar si estamos siendo atacados. Sería de mucha utilidad implementar una notificación de escritorio para avisar al usuario de un ataque. En caso de implementación tendría especial importancia la integración con los diferentes sistemas operativos en los que se distribuye SecurAI.
- Uso de multiproceso: El aprovechamiento de varios cores del equipo que ejecuta SecurAI sería un elemento crucial para maximizar la escalabilidad y el rendimiento de los módulos de defensa. Ahora mismo para la creación de procesos se usan hebras/hilos y no un proceso diferente. La creación de un proceso nuevo sería lo óptimo debido a que cada proceso sería por ejemplo un módulo, y permitiría el análisis de paquetes concurrentemente a nivel de cómputo. Aunque hemos estudiado concienzudamente esta mejora de SecurAI, no hemos podido implementarla en una *release* final de la aplicación debido a escasez de tiempo y recursos. El problema viene con aspectos que van más allá de cambiar la estructura compartida de tipo `threading` a tipo `multiprocessing`, como que para crear procesos y estructuras compartidas en Python, el código que los crea debe de estar dentro de un bloque `if __name__== '__main__':`, lo cual implica la necesidad de refactorizar muchas partes del código del proyecto. Para realizar este cambio es muy importante estar muy familiarizado tanto con el funcionamiento del proyecto como con la gestión del multiproceso en Python, ya que estos cambios tienen que hacerse sin poder probar el programa hasta que la actualización esté completa.
- Release para sistemas Linux: Esta ampliación es el siguiente paso lógico a implementar tras la implementación de las funcionalidades de integración continua y despliegue continuo del proyecto (CI/CD). En principio sería suficiente con añadir un nuevo *runner* en el *workflow* de GitHub Actions, pero por experiencia sabemos que es muy posible que aparecieran nuevas

complicaciones. También destacamos en este punto que posiblemente sean necesarias labores de mantenimiento en el *workflow* que se usa actualmente, debido a la naturaleza cambiante de los sistemas operativos y necesidades de compilación.

- Tutorial: La realización de un tutorial interactivo en el primer inicio de SecurAI sería un aliciente para que personas sin experiencia pudieran adentrarse en el mundo de la ciberseguridad de redes, ayudando a suavizar la curva de aprendizaje de la herramienta.

REFERENCIAS BIBLIOGRÁFICAS

(1) IT Digital Security. (2024, diciembre). Crece la complejidad de los ciberataques con estructuras más complejas. IT Digital Security. https://www.itdigitalsecurity.es/actualidad/2024/12/crece-la-complejidad-de-los-ciberataques-con-estructuras-mas-complejas

(2) CyberSecurity News. (2024, diciembre). La creciente profesionalización y complejidad de los ataques plantean desafíos para la seguridad digital. CyberSecurity News. https://cybersecuritynews.es/la-creciente-profesionalizacion-y-complejidad-de-los-ataques-plantean-desafios-para-la-seguridad-digital/

(3) Suri-Oculus. (2025). Using AI in Suricata: Enhancing Intrusion Detection System Capabilities. Recuperado el 16 de febrero de 2025, de https://suri-oculus.com/using-ai-in-suricata-enhancing-intrusion-detection-system-capabilities/

(4) Ndong, M. (2023, abril 5). Autonomous Systems: The Power of IDS + AI. LinkedIn. Recuperado el 16 de febrero de 2025, de https://www.linkedin.com/pulse/autonomous-systems-power-ids-ia-madjiguene-ndong/

(5) Redes Plus. (2021, noviembre 4). 📢IPS IDS Linux🐧 Instalar SURICATA y configurar las REGLAS y Alertas 🌐[Video]. YouTube. https://www.youtube.com/watch?v=vQNB7nenT2E

(6) Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. Psychological Review, 65(6), 386–408. https://doi.org/10.1037/h0042519

(7) Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press. https://www.deeplearningbook.org/

(8) INTEF. (2021). Aprendizaje profundo: ¿Cómo se entrena una red neuronal? Aula en Abierto. https://formacion.intef.es/aulaenabierto/mod/book/view.php?id=5077&chapterid=6492

(9) Gil Ferrer, Alejandro (2021). Análisis de rendimiento de redes neuronales con tres estructuras diferentes = A neural network performance analysis with three different model structures. Trabajo Fin de Grado / Proyecto Fin de Carrera, E.T.S. de Ingenieros Informáticos (UPM), Madrid, España.

(10) González Palmero, M. (2024). Entre palabras y algoritmos: Aprendizaje profundo contra el ciberacoso en redes sociales [Trabajo Fin de Grado, Universidad de Málaga]. Departamento de Lenguajes y Ciencias de la Computación.

(11) Almseidin, M., Alzubi, J. A., & Alkasassbeh, M. (2024). Enhanced intrusion detection with LSTM-based model, feature selection, and data imbalance handling for IoT networks. Applied Sciences, 14(2), 479. https://doi.org/10.3390/app14020479.

(12) Cervantes, J., García-Lamont, F., Rodríguez-Mazahua, L., & López-Chau, A. (2020). A comprehensive survey on support vector machine classification: Applications, challenges and trends. Neurocomputing, 408, 189–215. https://doi.org/10.1016/j.neucom.2019.10.118.

(13) Khoshgoftaar, T. M., & Van Hulse, J. (2021). A systematic review on overfitting control in shallow and deep neural networks. Artificial Intelligence Review, 54(3), 1-38. https://doi.org/10.1007/s10462-021-09975-1

(14) Valeaal. (2025, abril 30). Comentario en "Issue #7772" [Comentario en GitHub]. GitHub. https://github.com/electron-userland/electron-builder/issues/7772#issuecomment-2857677789

(15) Valeaal. (2025, mayo 16). Comentario en "Issue #7768" [Comentario en GitHub]. GitHub. https://github.com/electron-userland/electron-builder/issues/7768#issuecomment-2879768615

(16) Valeaal. (2025, mayo 16). Comentario en "Issue #6290" [Comentario en GitHub]. GitHub. https://github.com/electron-userland/electron-builder/issues/6290#issuecomment-2879168615

(17) Valeaal. (2025, mayo 16). Comentario en "Issue #3104" [Comentario en GitHub]. GitHub. https://github.com/electron-userland/electron-builder/issues/3104#issuecomment-2879768615

(18) Biondi, P. (2015). Scapy Documentation. https://scapy.readthedocs.io/en/latest/

(19) Krishnamurthy, N., & Subashini, R. (2011). Denial of Service Due to Direct and Indirect ARP Storm Attacks in LAN Environment. Communications and Network, 3(1), 16–22. https://doi.org/10.4236/cn.2011.31003

(20) S. Surisetty y S. Kumar, "¿Es el sistema operativo Leopard iMac de Apple seguro bajo los ataques de inundación basados en ARP?", *2010 Quinta Conferencia Internacional sobre Monitoreo y Protección de Internet*, Barcelona, España, 2010, pp. 60-64, doi: 10.1109/ICIMP.2010.30.

(21) huja, R., Singal, A., & Mukhopadhyay, S. (2022). Smart Intrusion Detection System Using Hybrid Deep Learning Model. Mendeley Data, V2. https://doi.org/10.17632/yxzh9fbvbj.2

(22) Raki, A. (2023). Cyber Attack Dataset: ARP, SYN, Ping Flood [Conjunto de datos]. Kaggle. https://www.kaggle.com/datasets/aleksandarraki/cyber-attack-dataset-arp-syn-ping-flood

(23) Forouzan, B. A. (2013). Data communications and networking (5th ed.). McGraw-Hill Education. https://archive.org/details/datacommunicatio0000foro_i8a0_5ed

(24) Eddy, W. M. (2007). TCP SYN Flooding Attacks and Common Mitigations (RFC 4987). Internet Engineering Task Force. Recuperado de https://datatracker.ietf.org/doc/html/rfc4987

(25) Kumar, A., & Gupta, B. B. (2025). Cyber-attack detection in software-defined networking using machine learning techniques. Data in Brief, 52, 109857. https://www.sciencedirect.com/science/article/pii/S2352340925000460

(26) Mockapetris, P. (1987). Domain names - concepts and facilities (RFC 1034). Internet Engineering Task Force. https://doi.org/10.17487/RFC1034

(27) Mirkovic, J., & Reiher, P. (2004). A taxonomy of DDoS attack and DDoS defense mechanisms. ACM SIGCOMM Computer Communication Review, 34(2), 39-53. https://doi.org/10.1145/997150.997156

(28) Moustafa, Nour, and Jill Slay. "UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)." Military Communications and Information Systems Conference (MilCIS), 2015. IEEE, 2015.

(29) Moustafa, Nour, and Jill Slay. "The evaluation of Network Anomaly Detection Systems: Statistical analysis of the UNSW-NB15 dataset and the comparison with the KDD99 dataset." Information Security Journal: A Global Perspective (2016): 1-14.

(30) Moustafa, Nour, et al. "Novel geometric area analysis technique for anomaly detection using trapezoidal area estimation on large-scale networks." IEEE Transactions on Big Data (2017).

(31) Moustafa, Nour, et al. "Big data analytics for intrusion detection system: statistical decision-making using finite dirichlet mixture models." Data Analytics and Decision Support for Cybersecurity. Springer, Cham, 2017. 127-156.

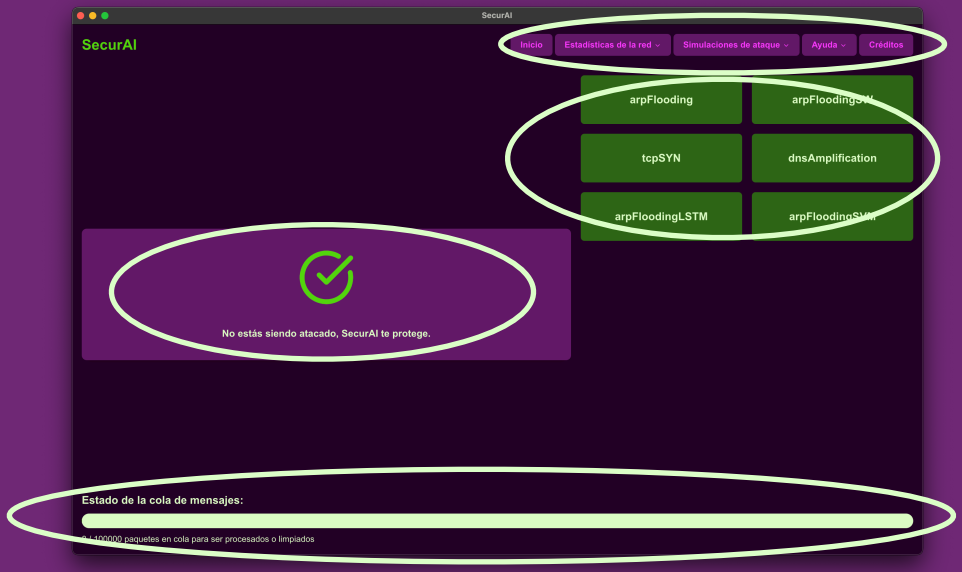
(32) Sarhan, Mohanad, Siamak Layeghy, Nour Moustafa, and Marius Portmann. NetFlow Datasets for Machine Learning-Based Network Intrusion Detection Systems. In Big Data Technologies and Applications: 10th EAI International Conference, BDTA 2020, and 13th EAI International Conference on Wireless Internet, WiCON 2020, Virtual Event, December 11, 2020, Proceedings (p. 117). Springer Nature.

APÉNDICE I

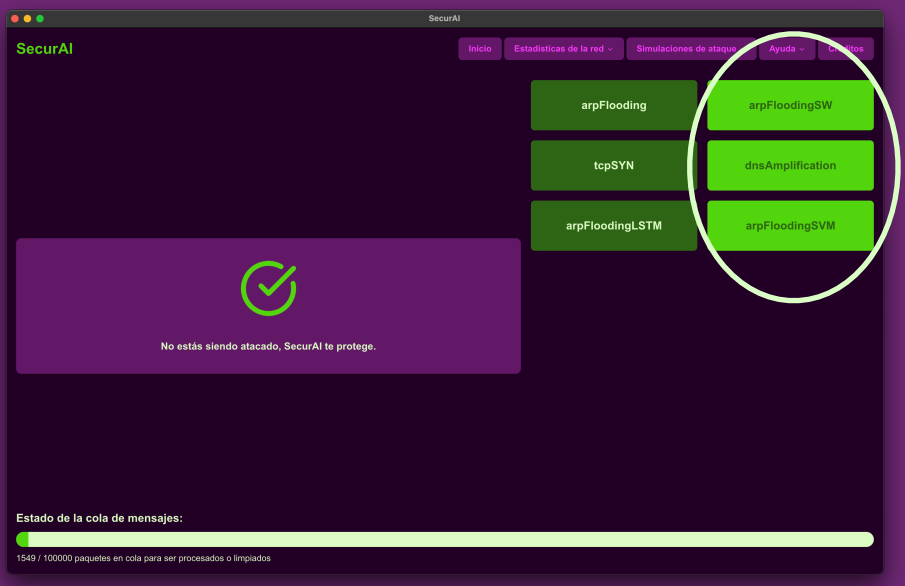
MANUAL DE USUARIO EN IDIOMA ESPAÑOL

Manual de Usuario de SecurAI (Español)

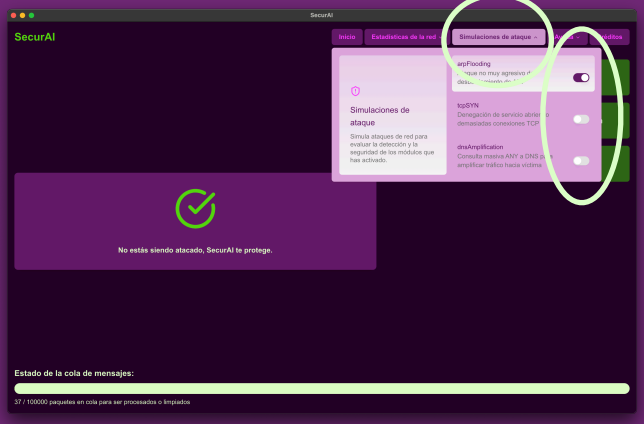
1- Aprenda qué elementos se encuentran en la página inicial de SecurAI:

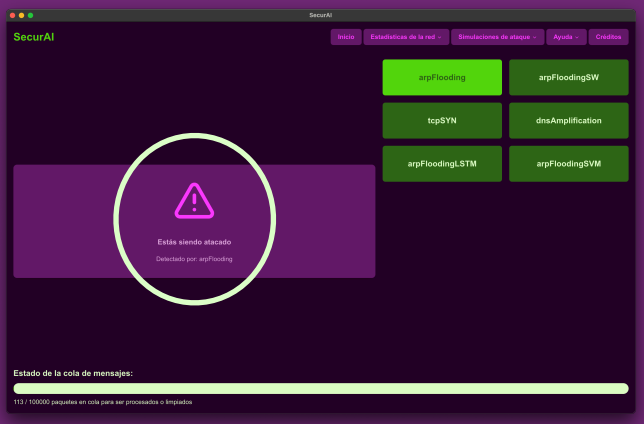


2- Aprenda a activar un módulo: Simplemente haga click en uno de los elementos de la sección de módulos:

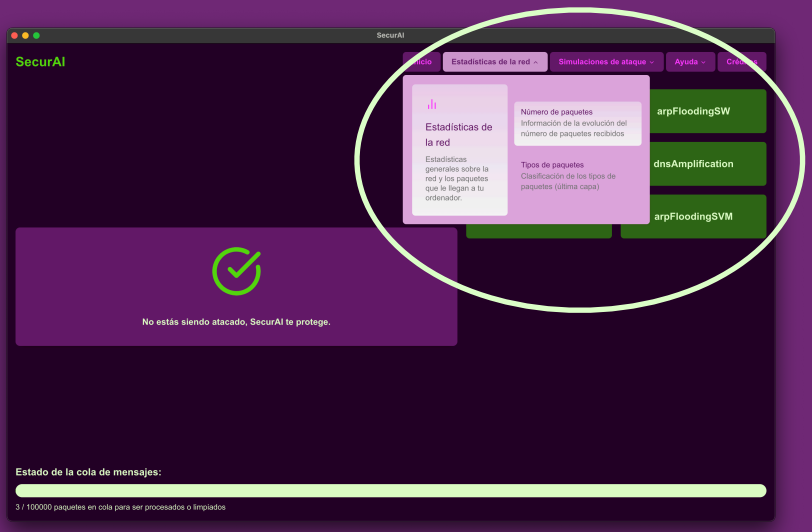


3- Pruebe el módulo: Con un módulo activo, puede iniciar la simulación de un ataque relacionado con el mismo para probar su efectividad::





4- Si quiere ver las estadísticas de su red, puede acceder al menú de estadísticas y seleccionar la que desee:



5- Consulte más información y reciba ayuda desde el menú de ayuda de la barra de menús:

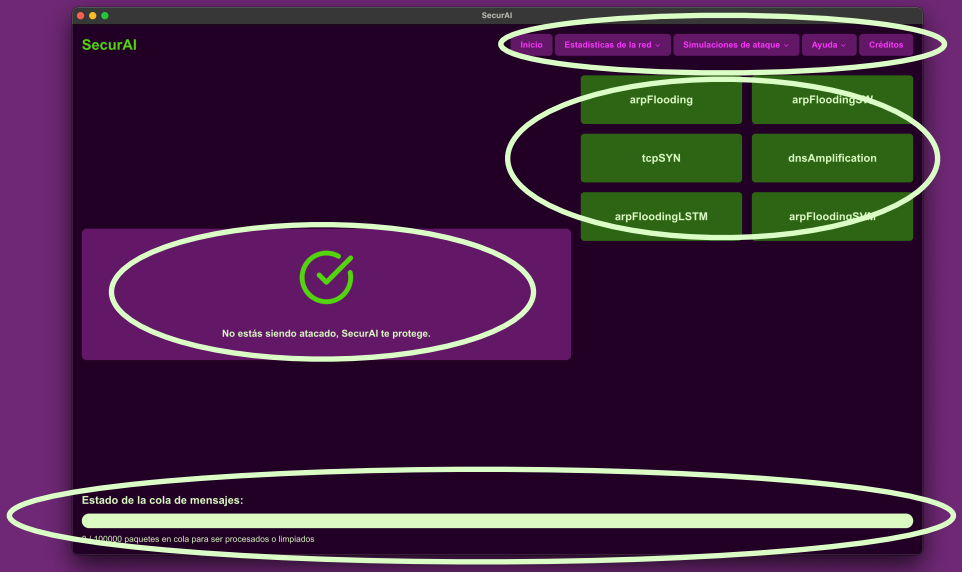
APÉNDICE II

MANUAL DE USUARIO EN IDIOMA INGLÉS

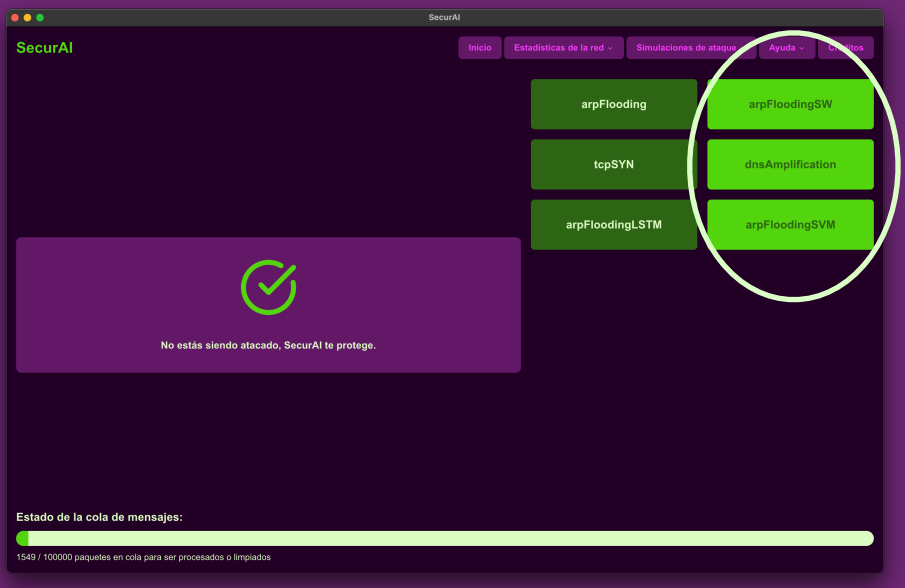
SecurAI User's Manual (English)

SecurAI

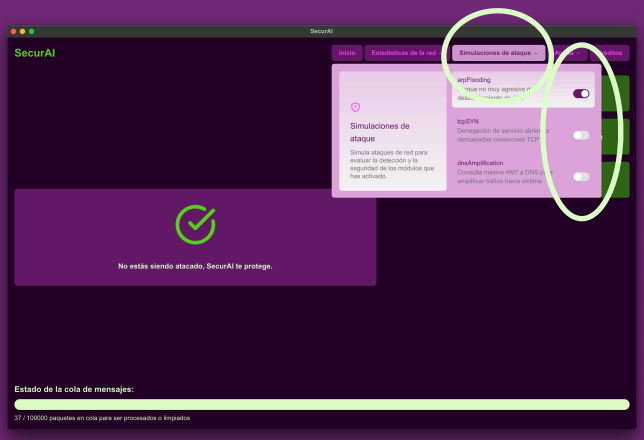
1- Learn what elements are on the SecurAI home page:

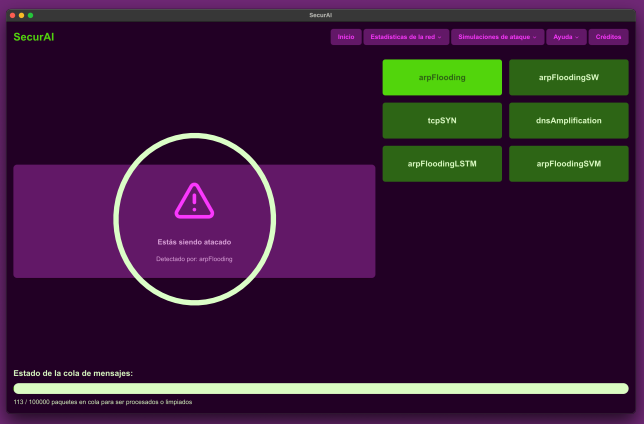


2- Learn how to activate a module: Simply click on one of the elements in the module section:

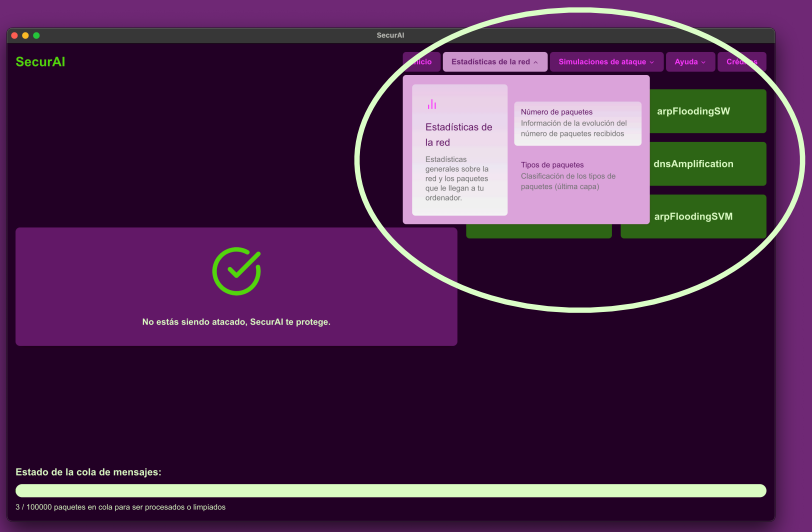


3- Test the module: With an active module, you can start the simulation of an attack related to it to test its effectiveness:





4- If you want to see the statistics of your network, you can access the statistics menu and select the one you want:



5- See more information and get help from the help menu of the menu bar:

UNIVERSIDAD DE MÁLAGA | uma.es

E.T.S de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga

E.T.S. DE INGENIERÍA INFORMÁTICA