UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Inteligencia de archivos binarios: caracterización binaria automatizada y modelado de inteligencia artificial para archivos ejecutables

Trabajo Fin de Máster

Máster Universitario en Ciberseguridad y Ciberinteligencia

AUTOR/A: Benlloch López, Sergio

Tutor/a: Such Aparicio, José Miguel

CURSO ACADÉMICO: 2022/2023

Resumen

La toma de decisiones en el ámbito de la ciberseguridad es un proceso complejo debido a la falta de información completa que pueda respaldar una elección óptima en cada situación. El propósito de la ciberinteligencia es proporcionar datos que permitan tomar decisiones informadas en materia de ciberseguridad.

El problema actual, es la transformación y procesado de estos datos para crear inteligencia. A cada segundo que pasa, en el mundo conectado de hoy en día, se producen ingentes cantidades de información que tiene que ser tratada para ser útil.

Este trabajo, enfoca el procesado y transformación de información proveniente de un tipo concreto de datos, los archivos ejecutables. Estos archivos binarios ejecutables forman parte fundamental de los sistemas informáticos actuales y, como cualquier componente del sistema, en ocasiones son utilizados por actores maliciosos.

En este documento se narra el proceso seguido para extraer información de estos archivos y hacer uso de ella en una aplicación de aprendizaje automático. Esta información tiene muchos usos dentro y fuera del área de la ciberseguridad. Pero se ha decidido solventar un problema con un nuevo enfoque, clasificación de archivos maliciosos.

Palabras clave:

Ciberseguridad, Archivos binarios, Aprendizaje automático, Inteligencia artificial, Análisis, Ingeniería inversa

Resum

La presa de decisions en l'àmbit de la ciberseguretat és un procés complex a causa de la manca d'informació completa que puga recolzar una elecció òptima en cada situació. El propòsit de la ciberintel·ligència és proporcionar dades que permeten prendre decisions informades en matèria de ciberseguretat.

El problema actual, és la transformació i processament d'estes dades per a crear intel·ligència. A cada segon que passa, en el món connectat d'avui en dia, es produeixen ingents quantitats d'informació que ha de ser tractada per a ser útil.

Este treball, enfoca el processament i transformació d'informació provinent d'un tipus concret de dades, els fitxers executables. Estos fitxers binaris executables formen part fonamental dels sistemes informàtics actuals i, com qualsevol component del sistema, en ocasions són utilitzats per actors maliciosos.

En aquest document es narra el procés seguit per a extreure informació d'estos fitxers i fer ús d'ella en una aplicació d'aprenentatge automàtic. Esta informació té molts usos dins i fora de l'àrea de la ciberseguretat. Però s'ha decidit solucionar un problema amb un nou enfocament, classificació de fitxers maliciosos.

Paraules clau:

Ciberseguretat, Arxius binaris, Aprenentatge automatitzat, Intel·ligència artificial, Anàlisi, Enginyeria inversa

Abstract

The decision-making process in the field of cybersecurity is a complex one due to the lack of complete information that can support an optimal choice in each situation. The purpose of cyberintelligence is to provide data that enables informed decision-making in cybersecurity matters.

A prevailing challenge, however, lies in the transformation and processing of this data into actionable intelligence. In our interconnected world, every second brings about the generation of vast amounts of information, which require meticulous processing to be rendered useful.

The focus of this work is on processing and transforming a specific type of data: executable files. These binary executable files are a fundamental element of modern computer systems. Yet, they can occasionally be exploited by malicious actors, like any other system component.

This document provides a detailed account of the process employed to extract information from these files for use in a machine learning application. This extracted information holds vast potential for a myriad of applications, within and beyond the scope of cybersecurity. In this instance, we've adopted a novel approach to tackle the challenge of classifying malicious files.

Key words:

Cybersecurity, Binary files, Machine Learning, Artificial intelligence, Analysis, Reverse engineering

Índice general

Índice de figuras

Índice de tablas

Siglas

BERT Bidirectional Encoder Representations from Transformers

ELF Executable and Linkable Format

EMBER Endgame Malware BEnchmark for Research

IA Inteligencia Artificial

JSON JavaScript Object Notation

ML Aprendizaje automático

NSA Agencia de Seguridad Nacional

PE Portable Executable

Agradecimientos

Este trabajo no habría existido sin la ayuda de mi tutor, José Such. Gracias a él, me he adentrado en el apasionante mundo de la inteligencia artificial aplicada a la ciberseguridad.

También es preciso agradecer a los mantenedores y creadores de las herramientas de código abierto utilizadas en este trabajo. Al crear y compartir su trabajo, fomentan la creación de nuevas soluciones y permiten que la tecnología avance a hombros de gigantes.

Gracias a todos ellos.

CAPÍTULO 1

Introducción

En la era contemporánea, marcada por una gran hiperconectividad, la ciberseguridad se ha convertido en una preocupación fundamental, no sólo para las empresas y los gobiernos, sino también para los usuarios individuales. A medida que nuestra dependencia de los sistemas informáticos aumenta, también lo hace la necesidad de protegernos contra posibles ciberataques.

Un sistema informático moderno está compuesto por múltiples capas de abstracción, lo que complica enormemente su análisis y mantenimiento. En estos sistemas, encontramos diversas fuentes de datos que pueden ser utilizadas para la generación de inteligencia.

La ciberinteligencia es una disciplina que trabaja con la enorme cantidad de datos generados en el entorno digital. Su principal objetivo es convertir estos datos, que a primera vista pueden parecer irrelevantes o carentes de sentido, en información valiosa y aplicable. Este proceso de transformación nos permite tomar decisiones más fundamentadas, proteger nuestros recursos digitales y contrarrestar de manera efectiva las amenazas cibernéticas.

El papel de la ciberinteligencia es esencial en el campo de la ciberseguridad. Mediante el análisis de patrones y correlaciones en los datos, podemos identificar comportamientos sospechosos y detectar intrusiones, permitiéndonos así proteger nuestros sistemas y datos de manera proactiva.

En este trabajo, se aborda el proceso de extracción de información de una sección específica de estos sistemas: los binarios, centrándonos específicamente en dos tipos de archivos binarios ejecutables, Executable and Linkable Format (ELF) y Portable Executable (PE). Un archivo binario ejecutable es un tipo de archivo binario que incluye un programa que puede ser ejecutado o iniciado por un sistema operativo, conteniendo instrucciones codificadas que el ordenador puede leer y ejecutar de forma directa.

Estos archivos son ricos en información, pero también pueden ser empleados de forma no deseada.

A lo largo de este estudio, expondremos cómo hemos extraído información de estos archivos, creando una estructura organizada preparada para ser utilizada por modelos de Aprendizaje automático (ML). A continuación, centraremos nuestra investigación en la detección de archivos maliciosos, concluyendo con el desarrollo de un prototipo operativo.

1.1 Objetivos

El principal objetivo de este trabajo es llevar a cabo una investigación completa para la creación de una herramienta basada en aprendizaje automático a partir de los datos extraídos de archivos binarios.

A continuación, se presenta una lista con los objetivos a cumplir durante este trabajo.

1. Explorar la estructura y características de los archivos binarios ejecutables.
2. Desarrollar un proceso efectivo para la extracción de información a partir de archivos binarios ejecutables.
3. Generar una estructura organizada de los datos extraídos que sea adecuada para su utilización en modelos de aprendizaje automático.
4. Aplicar técnicas de aprendizaje automático para identificar patrones y correlaciones en los datos.
5. Orientar la investigación hacia la detección de archivos maliciosos, con el objetivo de mejorar la protección contra ciberataques.
6. Diseñar y desarrollar un prototipo operativo que aplique los hallazgos de la investigación para la detección de amenazas y la protección proactiva de los sistemas y datos.
7. Realizar pruebas y evaluaciones con un prototipo funcional para recoger datos de uso auténticos.
8. Contribuir al campo de la ciberseguridad mediante la presentación de un enfoque práctico y efectivo para la gestión de la ciberinteligencia y la protección contra amenazas.

1.2 Estructura

Este trabajo está dividido en 4 capítulos principales que detallan el proceso de construcción de las herramientas necesarias para la obtención y transformación de los datos, así como la investigación realizada y la creación del prototipo final.

En el segundo y cuarto capítulo, también se presentan los conocimientos adquiridos para realizar el trabajo y se analizan los trabajos previos relevantes.

1.3 Disponibilidad del material

Este trabajo se ha realizado con el objetivo de contribuir a la comunidad académica y de ciberseguridad en general. Todos los materiales producidos durante este trabajo, incluyendo los códigos y los resultados, están disponibles de manera abierta y se han liberado bajo una licencia de software libre. Además, todos los datos que se han podido liberar de acuerdo con sus respectivas licencias también están disponibles.

El objetivo de esta apertura no es sólo fomentar la transparencia, sino también permitir la replicabilidad del trabajo y fomentar futuras investigaciones en este campo.

Todos estos materiales pueden ser accedidos, utilizados, modificados y distribuidos por cualquier persona interesada, siempre y cuando se respeten los términos de las licencias de software libre y de los datos utilizados.

Creemos firmemente en la importancia de la colaboración y el intercambio de conocimientos para el avance de la ciberseguridad y la ciberinteligencia.

CAPÍTULO 2
Analizador

En este capítulo, vamos a exponer la investigación, diseño e implementación necesarios para desarrollar un analizador de binarios.

Tal como se menciona en la introducción, este estudio abordará la utilización de los datos disponibles en archivos binarios que emplean los formatos ELF y PE.

En la primera sección, realizaremos un repaso al estado del arte de estos archivos. Identificaremos los campos que se encuentran en su estructura interna y revisaremos algunas de las soluciones existentes para el análisis de estos tipos de archivos.

Posteriormente, presentaremos el diseño de la solución propuesta. Finalmente, ofreceremos una breve descripción de la implementación final de este proyecto.

2.1 Contexto tecnológico

En esta primera sección, nos centraremos en los archivos binarios, específicamente en los formatos ELF y PE, que son fundamentales. Para establecer el contexto, comenzaremos analizando la estructura de ambos formatos. Tras ello, analizaremos algunas de las soluciones existentes.

2.1.1. Formato ELF

El formato *Executable and Linkable Format* (ELF) [1] [2] [3] es una pieza esencial en sistemas Unix para binarios ejecutables, archivos objeto y bibliotecas compartidas. Este formato es conocido por su flexibilidad, eficiencia y compatibilidad, lo que lo convierte en una elección predominante en el mundo Unix. Su estructura, ilustrada en la Figura 2.1, tiene diferentes partes estandarizadas que todo sistema tiene que seguir. Las describiremos a continuación:

1. **Encabezado ELF**: El Encabezado ELF actúa como la "tabla de contenidos"del archivo. Suministra metadatos críticos sobre el binario, distinguiendo si el archivo es un ejecutable, una biblioteca compartida o un archivo objeto. También especifica para qué arquitectura de sistema, por ejemplo x86 o ARM, se compiló el archivo. El punto de entrada del programa, que establece la posición inicial de la ejecución del

programa, también se detalla en este encabezado. Adicionalmente, presenta información sobre la localización y tamaño de los encabezados de sección y segmento en el archivo.

2. **Segmentos y Secciones de datos**: Los segmentos y secciones del archivo ELF contienen la mayoría de los datos útiles que se necesitan para ejecutar o vincular el programa. Estos segmentos incluyen código de máquina, datos de programa, información de reubicación, que ayuda al enlazador [4] a resolver referencias cuando se vincula un archivo, y símbolos para vinculación, que también son utilizados por el enlazador.

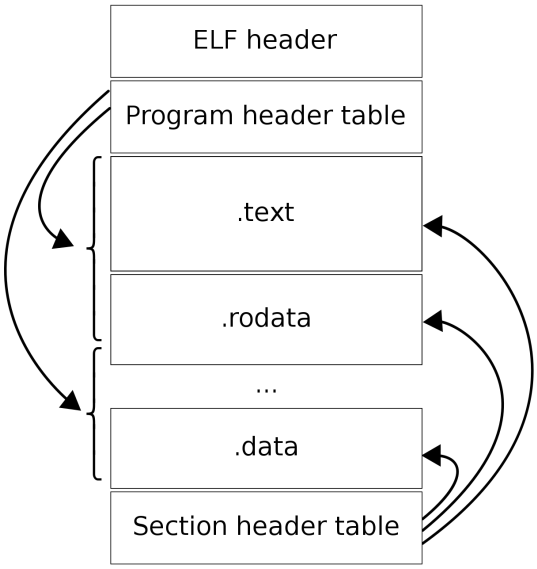

Figura 2.1: Estructura formato ELF.

Las secciones más comunes en un archivo ELF incluyen:

- **.text**: Esta sección contiene el código de la máquina que la CPU ejecutará. A menudo, es la sección más grande de un archivo ELF.
- **.data y .rodata**: Estas secciones contienen los datos inicializados que el programa utilizará. La sección .rodata contiene datos de solo lectura que no se modificarán durante la ejecución del programa, mientras que la sección .data contiene datos que el programa puede modificar.
- **.bss**: Esta sección es un poco especial, ya que no ocupa ningún espacio en el archivo ELF en sí. En cambio, indica al sistema operativo que reserve una cierta cantidad de memoria durante la ejecución del programa, que se inicializa a cero.
- **.symtab y .dynsym**: Estas secciones contienen las tablas de símbolos, que son utilizadas por el enlazador para resolver referencias a funciones y variables cuando se vincula un programa. .symtab se utiliza para la vinculación estática, mientras que .dynsym se utiliza para la vinculación dinámica (es decir, para bibliotecas compartidas).

Cabe mencionar que el formato ELF no se utiliza únicamente en sistemas Unix, sino que también es empleado en sistemas operativos basados en BSD, como FreeBSD [5].

Aunque en esencia el formato ELF en FreeBSD es similar al utilizado en Unix, existen algunas diferencias importantes.

FreeBSD ha introducido modificaciones y características específicas que optimizan el rendimiento y la funcionalidad en su entorno específico. Por ejemplo, FreeBSD utiliza un conjunto diferente de banderas en los encabezados ELF y una disposición ligeramente distinta en algunas secciones del archivo.

Estas variaciones, aunque pequeñas, pueden tener implicaciones significativas en la forma en que se ejecutan y analizan los binarios ELF en estos sistemas, subrayando la importancia de un análisis detallado y adaptado a cada sistema concreto en el estudio de los archivos binarios.

2.1.2. Formato PE

El formato *Portable Executable* (PE) [6] es esencial para los binarios ejecutables, archivos de objeto y bibliotecas dinámicas en los sistemas operativos de Microsoft, incluyendo Windows y algunas versiones de MS-DOS.

Este formato es reconocido por su versatilidad y compatibilidad, convirtiéndolo en la elección predominante en el mundo Microsoft. Su estructura consta de varios componentes estandarizados que todos los sistemas deben seguir. Los describiremos a continuación:

1. **Encabezado MZ**: Este es el primer encabezado que aparece en los archivos PE y proporciona información sobre el tamaño y la ubicación del programa en el archivo. El encabezado MZ originalmente proviene del formato de archivo MS-DOS.

2. **Encabezado PE**: El Encabezado PE actúa como la "tabla de contenidos"del archivo, proporcionando metadatos críticos sobre el binario. Distingue si el archivo es un ejecutable, una biblioteca dinámica o un archivo objeto y especifica la arquitectura del sistema para el que se compiló el archivo, como x86 o ARM. También se detalla el punto de entrada del programa, estableciendo la ubicación inicial de la ejecución del programa, y adicionalmente ofrece información sobre la ubicación y tamaño de las secciones en el archivo.

3. **Encabezado COFF**: Este encabezado sigue al encabezado PE y contiene información más detallada sobre el archivo, como el tamaño de las secciones y los recursos que utiliza el archivo.

4. **Secciones de datos**: Las secciones del archivo PE contienen la mayoría de los datos útiles necesarios para ejecutar o vincular el programa. Estas secciones incluyen código de máquina, datos del programa, información de reubicación que asiste al enlazador en la resolución de referencias durante la vinculación de un archivo, y símbolos para vinculación que también son utilizados por el enlazador.

Las secciones más comunes en un archivo PE incluyen:

- **.text**: Esta sección contiene el código de máquina que la CPU ejecutará. A menudo es la sección más grande en un archivo PE.

- **.data**: Contiene los datos inicializados que el programa utilizará, y que el programa puede modificar durante su ejecución.
- **.rdata**: Similar a .data, pero estos son datos de solo lectura que no se modificarán durante la ejecución del programa.
- **.bss**: Al igual que en ELF, no ocupa ningún espacio en el archivo PE en sí, pero indica al sistema operativo que reserve una cierta cantidad de memoria durante la ejecución del programa, que se inicializa a cero.
- **.idata**: Contiene las tablas de importación de funciones y datos desde otras bibliotecas.
- **.edata**: Contiene las tablas de exportación de funciones y datos a otras bibliotecas o módulos.

2.1.3. Herramientas de análisis

El análisis de archivos binarios es un aspecto crucial en el ámbito de la ingeniería inversa, la ciberseguridad y el desarrollo de software. Un archivo binario puede contener una riqueza de información, desde el código de la máquina que ejecuta hasta los metadatos que describen su estructura y propósito. El análisis de binarios permite a los investigadores entender el comportamiento de un programa sin tener acceso a su código fuente.

Para realizar este tipo de análisis, se utilizan diversas herramientas especializadas. Estas herramientas pueden variar en términos de funcionalidad, complejidad, y el nivel de detalle proporcionado. Algunas herramientas son más adecuadas para el análisis a gran escala o automatizado, mientras que otras se diseñan para el análisis interactivo y profundo.

Herramientas genéricas de análisis de binarios

A continuación, se presentan varias de las herramientas más utilizadas para el análisis de binarios y sus respectivos metadatos:

- **Ghidra:** Ghidra [7] es una *suite* de ingeniería inversa desarrollada por la Agencia de Seguridad Nacional (NSA) de los Estados Unidos. Es de código abierto y está diseñada para funcionar en múltiples plataformas. Ghidra puede desensamblar y decompilar binarios en código legible. Tiene una arquitectura de entrada de plugins que permite a los usuarios agregar nuevas funcionalidades.
- **IDA Pro:** IDA Pro [8] es una de las herramientas más utilizadas para el análisis de binarios. Tiene un desensamblador y un depurador incorporado. También proporciona un lenguaje de scripting llamado IDC para automatizar tareas de análisis. Funciona en Windows, Linux, y macOS.
- **Radare2:** Radare2 [9] es una herramienta de código abierto que permite el análisis, la depuración, y la manipulación de binarios. Radare2 soporta scripting en varios lenguajes, incluyendo Python y JavaScript, y es compatible con diversas arquitecturas y sistemas operativos.

- **Angr:** Angr [10] es un marco de análisis binario de código abierto y multiplataforma desarrollado por la Universidad de California, Santa Bárbara. Su objetivo es proporcionar un entorno unificado para analizar ejecutables en busca de vulnerabilidades de seguridad, y para ese fin, tiene la capacidad de realizar análisis simbólicos y concretos, lo que lo hace particularmente poderoso para el análisis automatizado de binarios. Angr puede interactuar con Python, permitiendo a los usuarios escribir scripts o usarlo como una biblioteca.

- **LIEF:** LIEF [11] es una biblioteca para analizar y editar archivos binarios en formato ELF, PE, MachO. Es multiplataforma y proporciona bindings para varios lenguajes de programación, incluyendo Python y C++, lo que permite su uso en scripts o como una librería.

- **Readelf:** Readelf [12] es una herramienta de Unix/Linux que se usa para mostrar información sobre archivos ELF. Proporciona una visión detallada de las diferentes secciones y segmentos de los archivos ELF.

- **pefile:** pefile [13] es una biblioteca de Python para leer y trabajar con archivos PE. Permite analizar los metadatos y el contenido de los archivos PE, incluyendo el encabezado, las secciones, las importaciones y las exportaciones.

- **pyelftools:** pyelftools [14] es una biblioteca de Python para analizar archivos ELF y DWARF, los formatos utilizados por muchos sistemas basados en Unix. Proporciona tanto *parsers* de bajo nivel que permiten un acceso detallado a la estructura interna de los archivos, como clases de alto nivel que permiten una interpretación más abstracta de la información contenida en ellos. Dado que es una biblioteca de Python, puede integrarse fácilmente en scripts o utilizarse como parte de otros programas Python.

- **PEview:** PEview [15] es una herramienta de Windows que proporciona una vista rápida y fácil de los archivos PE. Muestra los encabezados y las secciones, así como la tabla de importaciones y exportaciones.

- **objdump:** objdump [16] es una herramienta versátil para manipular y analizar archivos objeto en Unix/Linux. Puede mostrar información sobre secciones y encabezados, desensamblar código de máquina y mucho más.

- **elfutils:** elfutils [17] es una colección de utilidades para trabajar con archivos ELF en Unix/Linux. Incluye libelf (una biblioteca para leer y escribir archivos ELF), readelf (para mostrar información detallada sobre archivos ELF), y otros.

Concluyendo, cada una de estas herramientas tiene sus fortalezas y debilidades, y la elección de la herramienta correcta a menudo depende del tipo de análisis que se está realizando y de las necesidades específicas del usuario. La flexibilidad, la facilidad de uso, el soporte para diferentes formatos de archivos y la capacidad de interactuar con otras herramientas son factores clave a tener en cuenta al seleccionar una herramienta de análisis de binarios.

Frameworks de trabajo

Además de las herramientas de análisis de binarios individuales, existen varios *frameworks* que proporcionan una *suite* más completa de herramientas y capacidades. Estas

herramientas suelen incluir funcionalidades para la recolección de datos, el preprocesamiento, el análisis y la visualización, y están diseñados para facilitar el análisis en gran escala y automatizado de archivos binarios. A continuación, se presentan algunos de los *frameworks* más populares:

- **EMBER:** Endgame Malware BEnchmark for Research (EMBER) [18] es un conjunto de datos y una herramienta de código abierto para la evaluación de modelos de aprendizaje automático en la detección de malware. Proporciona características extraídas de archivos PE, como metadatos de encabezado, histogramas de bytes, secciones de datos y más. Su uso principal es en la investigación de aprendizaje automático y detección de malware.
- **YARA:** YARA [19] es un *framework* de trabajo utilizado en la detección de malware. Permite a los usuarios escribir descripciones de las características del malware basadas en reglas lógicas. Estas reglas pueden utilizarse luego para identificar y clasificar muestras de malware. YARA es altamente flexible y se utiliza en una amplia variedad de contextos, desde la investigación de seguridad hasta las operaciones de respuesta a incidentes.

Estas herramientas de trabajo ofrecen capacidades avanzadas para el análisis automatizado de binarios y la detección de malware. Al integrar diversas utilidades y técnicas, proporcionan una solución más completa y flexible para los desafíos del análisis de binarios. Sin embargo, su utilización requiere un conocimiento más profundo y a menudo una mayor inversión de tiempo y recursos en comparación con las herramientas individuales.

2.2 Características

Una vez comprendida la estructura de un archivo binario y los formatos en cuestión, podemos proceder a describir las características fundamentales que buscamos extraer. Estas se detallan a continuación:

- **Cabeceras**: Corresponde a la información extraída de las cabeceras (*headers*) de los archivos binarios.
- **Cadenas**: Representa el texto legible encontrado dentro de los archivos.
- **Entropía**: Esta característica mide la entropía (una medida de la aleatoriedad o incertidumbre del archivo) en las diferentes secciones contenidas en el archivo. Esta información es útil para evaluar la compresibilidad, seguridad o contenido del archivo.
- **Tamaño y hash**: Contiene información relacionada con el tamaño del archivo, además de los resultados de las funciones hash MD5 y SHA256 aplicadas al archivo.
- **Desensamblado y estadísticas**: Implica el proceso de desensamblar el binario para cada función en el mismo, lo que permite inferir estadísticas relacionadas con la frecuencia de cada instrucción de ensamblador.

Las características mencionadas incluyen elementos que pueden extraerse directamente del archivo, así como otros que requieren un procesamiento adicional. El objetivo es representar toda la información relevante del archivo binario para su posterior análisis.

2.3 Diseño de la solución

El diseño de un analizador capaz de extraer información de múltiples formatos de entrada plantea desafíos particulares. Debe considerar la extracción de características comunes y únicas. Aunque los formatos ELF y PE comparten similitudes, requieren interpretaciones distintas, obligándonos a considerar la especificidad de cada formato.

Además, la elección entre utilizar herramientas existentes o construir una lectura de formato desde cero es crucial en la definición de la estructura de trabajo. En este caso, tras probar diferentes dependencias y soluciones desde cero, hemos optado por el uso de herramientas ya existentes. En particular, utilizamos *pyelftools* [14] para la extracción de información del formato ELF, y su contrapartida, *pefile* [13], para el formato PE.

Estas herramientas nos permiten obtener información relevante de las cabeceras de los archivos binarios, que constituye la fuente principal de los datos que buscamos.

Para la extracción de características que requieren un procesamiento adicional del archivo, hemos dividido el flujo de trabajo en dos etapas. La primera etapa abarca tareas como la obtención de información relativa al tamaño del archivo, hashes y entropía, implementadas directamente sin el uso de ninguna dependencia externa.

En la etapa final, realizamos manualmente la extracción de cadenas de texto. Sin embargo, para la extracción de información relevante al desensamblado, recurrimos a una dependencia de análisis estático de binarios: *Angr* [10], que nos permite obtener información sobre las funciones.

Es importante señalar que tras la extracción de las cadenas de texto y las funciones de desensamblado, decidimos añadir dos campos más a las características extraídas. Por un lado, introducimos un análisis de inteligencia dentro de las cadenas, buscando elementos como URLs, paths, emails, enlaces, años e IPs. Por otro lado, al utilizar *Angr*, decidimos extraer el código VEX (una representación de más alto nivel) de cada una de las funciones.

Con esto obtenemos el flujo representado en la Figura 2.2 donde vemos el funcionamiento básico del analizador.

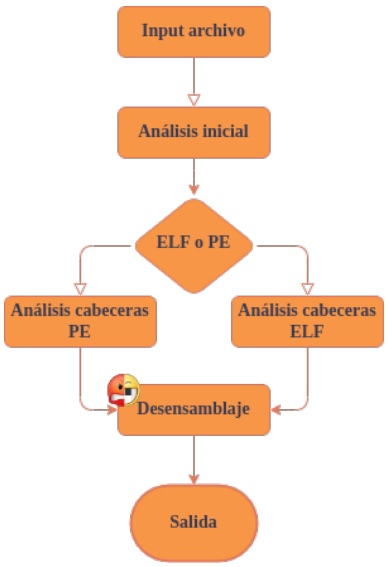

Figura 2.2: Diseño analizador de binarios.

2.4 Implementación

La implementación de la solución de análisis de binarios implicó varias etapas y consideraciones técnicas importantes.

2.4.1. Integración de Herramientas

El primer paso fue la integración de las herramientas seleccionadas. Esto incluyó el desarrollo de *wrappers* o interfaces de programación para *pyelftools* y *pefile* para estandarizar la forma en que nuestra aplicación interactúa con estas herramientas.

2.4.2. Desarrollo de Funciones Propias

Para las tareas que no requerían el uso de herramientas externas, como la obtención del tamaño del archivo y el cálculo de hashes y entropía, desarrollamos nuestras propias funciones utilizando Python.

2.4.3. Extracción y Procesamiento de Cadenas

Para la extracción manual de cadenas de texto, desarrollamos un algoritmo que lee el archivo binario y busca secuencias de caracteres legibles. Luego, las cadenas extraídas son procesadas para identificar elementos como URLs, paths, emails, enlaces, años e IPs.

2.4.4. Desensamblado y Análisis

Para el desensamblado y análisis de los binarios, desarrollamos un módulo que utiliza *Angr* para extraer información sobre las funciones en los archivos binarios y obtener su código VEX.

2.4.5. Etiquetado

En muchas ocasiones, para poder entrenar un modelo de clasificación, necesitamos una etiqueta que nos sirva como test o validación.

Para ello, incluimos una opción para poder incluir cualquier campo en el archivo final de características. Con ello podemos usar este campo como etiqueta de comprobación.

2.4.6. Salida

Por último, una vez creado toda la lógica necesaria para extraer las diferentes características, realizamos las funciones necesarias para volcar la información obtenida en un archivo JavaScript Object Notation (JSON). Elegimos este tipo de archivo por el balance existente entre lectura para un usuario y la velocidad de carga de un ordenador.

Uno de los problemas es que a pesar de que los archivos JSON son ligeros, el tamaño se vuelve considerable cuando se generan miles de ellos. En respuesta a esto, implementamos un algoritmo de compresión antes de escribir los datos y, correspondientemente, un proceso de descompresión tras la lectura de los mismos.

Al elegir el algoritmo de compresión a utilizar fuimos cautelosos con el tiempo de compresión y descompresión, ya que no queríamos ralentizar el proceso de análisis de un binario. Finalmente, tras una pequeña comparación, decidimos hacer uso del algoritmo LZ4, disponible como paquete en Python [20].

En el artículo [21], los autores realizan un análisis de varios algoritmos de compresión y proporcionan una tabla comparativa (Tabla 2.1), que evidencia que el algoritmo que elegimos es uno de los más rápidos.

Algorithm	Compression Time (s)	Decompression Time (s)	Compressed File Size (GB)	Compression Ratio
ROOT(ZLIB-6)	228.67	18.45	1.54	4.16
ZLIB-1	86.47	21.51	1.79	3.58
ZLIB-5	159.84	19.20	1.58	4.05
ZLIB-9	1715.25	18.28	1.49	4.30
LZ4	11.26	2.97	2.17	2.95
LZ4HC-5	95.13	2.81	1.75	3.66
LZ4HC-9	275.31	2.54	1.66	3.86
LZMA-1	823.84	230.64	1.35	4.74
LZMA-5	3318.35	211.71	1.23	5.20
LZMA-9	4969.20	212.47	1.21	5.29

Tabla 2.1: Comparación entre algoritmos de compresión

2.4.7. Conclusión

La construcción de una herramienta para trabajar con archivos binarios nos ha permitido llegar a algunas conclusiones.

El manejo de archivos binarios inevitablemente conlleva errores y desafíos. La gran variedad de variaciones presentes en los archivos binarios dificulta la realización de pruebas exhaustivas y la cobertura total de todas las posibilidades. Este desafío se amplifica cuando se analizan binarios que se han modificado de manera malintencionada u ofuscado para ocultar información.

Este entendimiento resalta la necesidad de ir con pies de plomo al realizar y tomar decisiones automáticas basadas en el software que analiza archivos binarios de esta naturaleza.

2.4.8. Licencia y Acceso al Proyecto

Esta parte del trabajo, al ser independiente, se encuentra publicada en un único repositorio de software por separado [22].

El software se ha publicado bajo una licencia de software libre que permite el acceso al código a todo usuario.

Adicionalmente, en el Apéndice A se puede encontrar un manual de uso.

CAPÍTULO 3

Ingeniería de Características

Hasta ahora hemos creado una herramienta capaz de obtener información de archivos binarios y almacenarla en un archivo con una estructura ordenada. Aunque esta información resulta útil para los humanos, no puede ser utilizada directamente como entrada en un modelo de clasificación.

Los modelos de ML requieren que la entrada sea representada por números. En la actualidad, estos modelos emplean vectores de características, que son conjuntos de números, para definir la entrada del modelo y generar la salida correspondiente a través de inferencias.

En este capítulo narramos el trabajo realizado para elegir, ordenar y convertir los datos de la salida de *BinSniff*, la herramienta de análisis, a estos vectores.

Es importante destacar que las características o atributos seleccionados han sido pensados considerando su utilidad final. Como se explica en la introducción de este documento, se ha optado por enfocarse en la detección de malware como objetivo principal, con el fin de llevar a cabo el ciclo completo de desarrollo de una herramienta de ciberseguridad que haga uso de Inteligencia Artificial (IA).

3.1 Obtención de datos

Para crear un conjunto de datos de entrenamiento, o *dataset*, es necesario obtener unos datos iniciales para obtener las características finales. A partir de estos datos se puede empezar el proceso de creación y limpieza, acabando con el entrenamiento de los modelos.

En nuestra situación, nuestro objetivo es desarrollar modelos para identificar binarios maliciosos que afectan múltiples plataformas. Estamos tomando en cuenta binarios PE que se originan en Windows, así como binarios ELF System V. Por lo tanto, necesitamos conseguir PE y ELF tanto maliciosos como benignos.

A continuación, se proporcionará una breve descripción de cómo se han obtenido los dos tipos de binarios benignos.

Portable Executable

Los binarios con formato PE son naturales de Windows. Existen varias formas de obtener estos binarios.

En el caso de los binarios benignos, la forma más básica sería compilarlos nosotros mismos y crear un conjunto de datos de binarios a partir de un conjunto de datos de código benigno (C, C++, Golang, Rust). Sin embargo, en este caso hemos optado por una solución más sencilla y probablemente menos propensa a problemas futuros.

Hemos decidido buscar este tipo de archivos en todo el sistema, utilizando una versión limpia de Windows (cualquier versión moderna es válida) y la tienda oficial de Microsoft[1].

El sistema inicial sin ninguna aplicación instalada ya contiene miles de binarios PE, pero, si queremos obtener más, una buena fuente confiable es instalar aplicaciones de la tienda oficial del sistema. Esta tienda sigue una norma estricta para que una aplicación sea publicada en ella [23].

Executable and Linkable Format

Para los binarios benignos con formato ELF la solución es muy similar a la usada anteriormente. Hemos decidido, en este caso, usar un sistema Ubuntu actual, y utilizar los repositorios oficiales[2] para obtener más binarios.

Estos repositorios también siguen una certificación similar a la usada por Microsoft [24], pero, en este caso es muy importante que sea en un sistema limpio ya que el uso de PPA no oficiales maliciosas esta muy extendido.

El caso de los binarios maliciosos es más común. Para ello, podemos hacer uso de múltiples repositorios de malware que almacenan muestras de malware actual e histórico. Existen una gran cantidad de repositorios, pero los utilizados en este trabajo han sido: VirusShare [25], VX-underground [26] y MalwareBazaar [27].

Como se vera en los siguientes capítulos, con estos métodos se pueden obtener miles de binarios, suficientes para el alcance de este trabajo.

3.2 Extracción de las *features*

Hasta ahora, hemos desarrollado un método eficaz para obtener binarios que se ajustan a nuestras necesidades. En el capítulo anterior, detallamos el proceso de creación de una herramienta diseñada para la extracción de información relevante de dichos binarios.

Ahora, simplemente se requiere ejecutar esta herramienta sobre los binarios de interés. En el repositorio de la herramienta mencionada previamente [22], se encuentra una

[1] En la tienda de Microsoft no se encuentran binarios directamente, pero muchas de estas aplicaciones están creadas, total o parcialmente, mediante el uso de binarios de este tipo.

[2] En estos repositorios, de forma muy similar a la tienda de Microsoft, los paquetes no contienen siempre binarios, pero es muy frecuente su uso total o parcial.

utilidad adicional, *miner*, que simplifica este proceso y nos permite realizar la extracción de binarios de manera sistemática y organizada.

Durante el análisis de los binarios, hemos aprovechado la oportunidad para añadir las etiquetas, utilizando la opción correspondiente proporcionada por *BinSniff*.

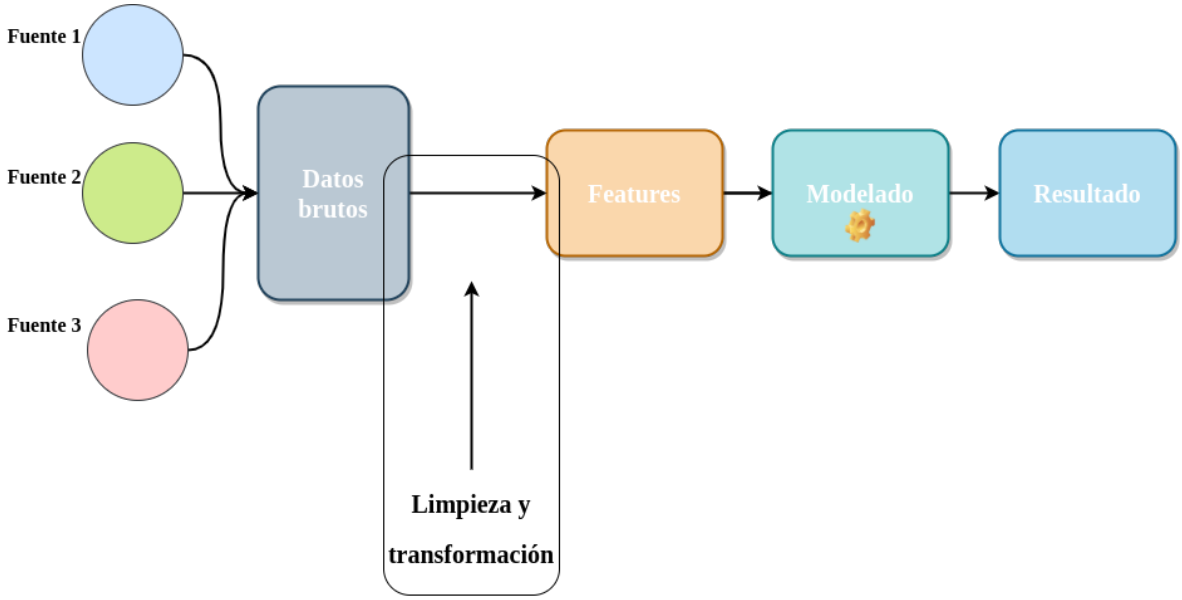

Figura 3.1: Esquema de funcionamiento.

En la Figura 3.1, se presenta el flujo de trabajo que seguimos. Hasta ahora, hemos obtenido los binarios, que corresponden a los datos brutos. Posteriormente, hemos iniciado el proceso de limpieza y transformación, sin embargo, aún no contamos con datos válidos que puedan ser empleados como características de entrenamiento.

El primer paso para llegar a unas *features* válidas consiste en la selección de datos [28] comunes entre los tipos de binarios para su transformación y organización

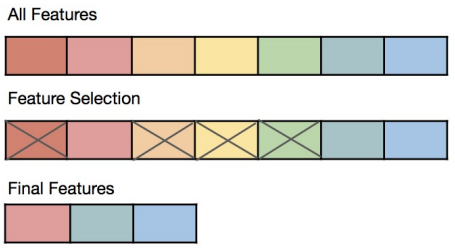

Figura 3.2: Selección de características.

Tras seleccionar esta información, debemos convertirla a un formato adecuado para entrenar y utilizar los modelos. Dichos modelos requieren listas simples de números, generalmente entre 0 y 1, que representan de alguna forma las características.

Existen diferentes técnicas para convertir los datos al formato deseado, algunas de ellas automáticas y otras manuales. A continuación, procederemos a detallar las *features* seleccionadas y a describir el proceso que hemos seguido para su transformación.

3.2.1. Estadísticas mnemónicos

Una de las características más simples de transformar son las estadísticas de los mnemónicos. Como se muestra en la Figura 3.3, los archivos de información creados por *BinSniff* ya contienen la información en el formato deseado, numérico y normalizado[3] en un rango de 0 a 1.


```
"INSTS_STATS": {
  "num_insts": 21766,
  "num_inst_types": 174,
  "inst_type_freq": {
    "mov": 0.1926,
    "push": 0.0094,
    "sub": 0.0232,
    "ldmdb": 0.0003,
    "ldr": 0.1222,
    "ldrb": 0.0172,
    "cmp": 0.0678,
    "bne": 0.0233,
    "pop": 0.0101,
    "bx": 0.0138,
    "ldrne": 0.0016,
    ...
```


Figura 3.3: Frecuencias de mnemónicos.

Para convertir al formato final, hemos decidido agrupar las instrucciones en "grupos lógicos". Los grupos lógicos son conjuntos de instrucciones que realizan acciones similares. A continuación, se muestra una lista con una breve descripción de cada grupo:

1. ***Arithmetic***: en este grupo se recogen instrucciones que realizan operaciones aritméticas (sub, add).
2. ***Memory***: en este grupo encontramos instrucciones que interactúan con la memoria (mov, push, pop).
3. ***Floating Point***: instrucciones que realizan acciones de coma flotante (fcom, fistp, fisub).
4. ***System***: instrucciones relacionadas con el sistema (in, out, cli).
5. ***Type Conversion***: instrucciones que realizan conversiones de tipos (cdq, cbw, cwde).
6. ***Atomic Operations***: instrucciones que realizan operaciones atómicas (lock cmpxchg, lock sbb, lock xadd).
7. ***Unconditional Jump***: instrucciones de salto incondicional (jmp, call, ret).
8. ***Conditional Jump***: instrucciones de salto condicional (jno, js, jae, je).

[3] La normalización es el proceso de ajustar los valores de una característica dentro de un rango específico [29].

9. ***Privileged***: instrucciones que requieren privilegios elevados (int, endbr64, bnd jmp).

10. ***Crypto***: instrucciones relacionadas con operaciones criptográficas (aesenc, aesdec, aesimc).

11. ***Logic***: instrucciones lógicas (xor, test, and).

12. ***Addressing***: instrucciones para la manipulación y cálculo de direcciones de memoria (lea, leave, enter).

13. ***Comparison***: instrucciones de comparación (cmp, setg, setb).

En las listas no se recogen todas las instrucciones posibles. Sólo se han agrupado 200 de ellas, que se han obtenido usando las estadísticas de las instrucciones más usadas a partir de los *datasets* creados.

Con esto, tenemos las instrucciones convertidas a características simples que se pueden usar en un modelo de IA. También hemos decidido añadir el número total de instrucciones y el número de instrucciones diferentes en el binario.

Todo esto se guarda, siguiendo la jerarquía que crea la herramienta **miner**, en un archivo JSON.

3.2.2. Cadenas de caracteres

La siguiente característica planteada es el uso de las cadenas de texto disponibles en el binario. Las cadenas de texto son secuencias de caracteres que se encuentran dentro del binario y pueden contener información relevante sobre su funcionalidad o características.

Para ello, utilizaremos un modelo de ML ya entrenado, específicamente un transformer llamado Bidirectional Encoder Representations from Transformers (BERT). BERT es un modelo de lenguaje basado en transformers, desarrollado originalmente por Google. La versión que utilizaremos ha sido implementada y puesta a disposición por Hugging Face [30].

BERT destaca por su capacidad para comprender el contexto y las relaciones entre palabras en textos. Es ampliamente utilizado en tareas de procesamiento de lenguaje natural, como clasificación de texto y generación de lenguaje natural.

En nuestro caso, nos centraremos en utilizar el tokenizer de BERT. Un tokenizer es una herramienta que se encarga de dividir el texto en unidades más pequeñas, como palabras o subpalabras y las traduce a un entero representativo.

En la Figura 3.4, podemos ver una ilustración que muestra el funcionamiento de un tokenizer básico. El conjunto de caracteres que el modelo puede describir está influenciado por la diversidad del dataset de entrenamiento. Así, un dataset de entrenamiento más diverso conducirá a un conjunto de caracteres más amplio.

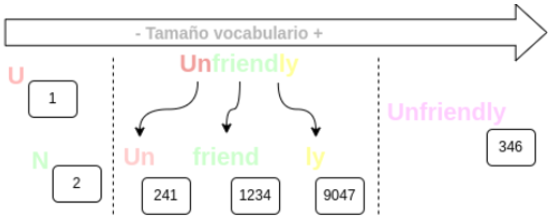

Figura 3.4: Tokenizer.

Por lo tanto, para conseguir un vector numérico con el formato deseado, usamos un transformer concatenando todas las cadenas que aparecen en el binario en una sola separada por espacios. Esto devuelve una lista de enteros, que luego normalizamos para conseguir un vector numérico con el formato deseado.

La normalización se hace utilizando 0 como el valor mínimo y el tamaño máximo utilizado por el transformer como el valor máximo. Esta normalización es esencial para garantizar que los valores alimentados al modelo estén en un rango que sea manejable y útil para el aprendizaje del modelo.

3.2.3. Entropía de las secciones

Otra característica relevante es la entropía de cada una de las secciones del archivo binario. Como se ha mencionado anteriormente, la entropía puede indicar la presencia de información comprimida u ofuscada, entre otras cosas. Dado que estos comportamientos pueden considerarse potencialmente maliciosos, resulta relevante para nuestra tarea de análisis.

En el conjunto de entrenamiento, existen diversos tipos de secciones, que suman miles en total. Para evitar contar con un número excesivo de características que dificulten el entrenamiento, y también para evitar ignorar secciones que no estén presentes en nuestro conjunto de datos, hemos decidido utilizar las 1000 secciones más frecuentes en nuestro conjunto de entrenamiento.

Con esta selección, obtendremos 1000 características directamente relacionadas con las secciones. Además, hemos considerado agregar otra característica para capturar la sección con la entropía más alta que no forme parte de las más comunes, así como una característica adicional que represente la entropía global del archivo binario.

Finalmente, para asegurar el formato deseado, solo será necesario normalizar los datos de entropía, donde 0 es el mínimo y el máximo, al ser una entropía basada en bytes, es 8.

3.2.4. Funciones importadas

Al igual que las secciones, las funciones importadas pueden indicar un uso potencialmente malicioso del archivo binario.

De manera similar, existen miles de funciones diferentes. Siguiendo el enfoque utilizado para las secciones, hemos seleccionado las 2000 funciones más frecuentes en nuestros datos y hemos agregado dos características adicionales: un contador de las funciones no consideradas y otro para el total de funciones.

Para esta característica, la diferencia principal radica en el valor utilizado en la lista. Hemos optado por utilizar una entrada binaria para indicar si se utiliza o no la función correspondiente al índice actual, junto con el recuento total de funciones en formato entero.

3.2.5. Características genéricas

Para completar, se incluyen características no consideradas en las secciones anteriores. Esta lista de características adicionales fue creada con el propósito de generar una nueva perspectiva de análisis.

En este conjunto de características, se agrupan distintos aspectos que incluyen el número mágico, tipo de archivo, arquitectura del binario, características específicas del binario, secciones relevantes y librerías relevantes.

Además, se toman en cuenta el tamaño en bytes del archivo, el número de secciones, la cantidad de cadenas de caracteres, las mitigaciones de seguridad detectadas y el hash MD5.

Con esto obtenemos una lista final que conforma el último vector de características.

CAPÍTULO 4

Investigación de la solución

En este capítulo, presentamos los pasos seguidos para obtener un resultado experimental para la detección de binarios maliciosos usando Machine Learning.

La estructura de este capítulo esta compuesta por una descripción teórica inicial, la presentación del diseño usado para la búsqueda de modelo y parámetros óptimos y la presentación y discusión de los resultados.

4.1 Contexto tecnológico

El campo de la tecnología ha experimentado un crecimiento exponencial en las últimas décadas, y un área que ha evolucionado particularmente rápido es la del ML.

El ML utiliza algoritmos que mejoran automáticamente a través de la experiencia y el uso de datos, proporcionando a las computadoras la capacidad de aprender[1] y realizar tareas sin ser explícitamente programadas, como la detección de correo no deseado en los correos electrónicos [31].

Dentro del ámbito del ML, podemos distinguir tres tipos principales de aprendizaje: supervisado, no supervisado y por refuerzo.

4.1.1. Tipos de Aprendizaje

Aprendizaje Supervisado

El aprendizaje supervisado se produce cuando un algoritmo aprende de datos de entrada y salida ya etiquetados para predecir futuras salidas. Los algoritmos de aprendizaje supervisado son capaces de realizar tareas de clasificación o regresión, como la predicción de enfermedades o el análisis de riesgos de crédito [32]. Aunque son poderosos, pueden ser propensos al sobreajuste si no se manejan correctamente.

[1] En el contexto de Machine Learning, «aprender» se refiere a la capacidad de un algoritmo para mejorar su rendimiento o precisión a través de la experiencia. Es decir, a medida que el algoritmo es expuesto a más datos de entrenamiento, es capaz de «aprender» patrones o estructuras subyacentes en esos datos que luego puede usar para hacer predicciones o tomar decisiones informadas. Este es un proceso iterativo y dinámico, y es por ello que se emplea la terminología de «aprendizaje».

Aprendizaje No Supervisado

El aprendizaje no supervisado se centra en encontrar patrones y relaciones en un conjunto de datos sin etiquetar, como en la segmentación de clientes [33]. No obstante, los resultados pueden ser difíciles de interpretar y requerir conocimientos expertos.

Aprendizaje por Refuerzo

El aprendizaje por refuerzo es un tipo de aprendizaje en el que un agente aprende a comportarse en un entorno realizando ciertas acciones y observando los resultados. Se utiliza en áreas como el control de robótica y el juego automático de videojuegos [34], aunque puede requerir una gran cantidad de tiempo y datos para entrenar correctamente.

4.1.2. Modelos Seleccionados para el Estudio

Dentro de estas categorías, existen diversas familias de modelos que presentan características y ventajas únicas. Para este estudio, nos centraremos en los modelos implementados en la popular biblioteca de Machine Learning, Scikit-Learn [35]. Aunque poderosos, no todos los modelos se desempeñarán igualmente bien en todas las situaciones, y es importante seleccionar el modelo adecuado para el problema en cuestión.

Modelos Basados en Árboles

Los modelos basados en árboles son técnicas de aprendizaje supervisado poderosas para problemas de regresión y clasificación. Los modelos notables en esta familia incluyen:

- **Árboles de Decisión**: Se construyen al dividir repetidamente el espacio de las características en dos, de acuerdo con una regla simple basada en una sola característica [36].
- **Bosques Aleatorios**: Combina las predicciones de varios árboles de decisión para producir una salida más robusta y menos susceptible al sobreajuste [37].
- **Gradient Boosting**: Mejora un modelo débil añadiendo nuevas versiones que corrigen los errores de los árboles existentes [38].
- **LightGBM**: Es una implementación de Gradient Boosting que ha sido optimizada para ser más eficiente y flexible [39].

Modelos Lineales

Esta familia de modelos, también aplicables en aprendizaje supervisado por su simplicidad y eficiencia, es popular en una amplia gama de contextos. Los modelos relevantes en esta familia incluyen:

- **Regresión Lineal**: Asume una relación lineal entre las características y la variable objetivo [40].

- **Regresión Logística**: Es un algoritmo de clasificación que modela la probabilidad logarítmica de una clase binaria [41].

Con estos modelos seleccionados, procederemos a examinar en profundidad sus fortalezas y debilidades en diferentes contextos, proporcionando una visión integral de su aplicabilidad y rendimiento en el entorno tecnológico actual.

Al mismo tiempo, es importante tener en cuenta que la tecnología de ML sigue avanzando, y nuevos modelos y técnicas, como las redes neuronales profundas, están emergiendo constantemente [42].

4.1.3. Métricas

Al desarrollar y evaluar modelos de Machine Learning, es fundamental contar con medidas cuantitativas que nos permitan comparar y evaluar su rendimiento. Estas medidas nos proporcionan una forma objetiva de determinar qué modelo es más efectivo en función de los datos disponibles y los objetivos del problema.

Antes de profundizar en las métricas, es importante comprender la terminología utilizada en problemas de clasificación binaria. Las siguientes definiciones son relevantes:

- **Verdaderos Positivos (TP)**: Instancias positivas correctamente clasificadas como positivas.
- **Falsos Positivos (FP)**: Instancias negativas incorrectamente clasificadas como positivas.
- **Verdaderos Negativos (TN)**: Instancias negativas correctamente clasificadas como negativas.
- **Falsos Negativos (FN)**: Instancias positivas incorrectamente clasificadas como negativas.

A continuación, se presentarán algunas métricas comunes utilizadas en problemas de clasificación binaria, junto con sus fórmulas y explicaciones:

- **Accuracy (Exactitud)**: Mide la proporción de predicciones correctas realizadas por el modelo en comparación con el total de predicciones realizadas.

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

- **Precision (Precisión)**: Mide la proporción de instancias positivas correctamente clasificadas con respecto a todas las instancias clasificadas como positivas.

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

- **Recall (Recuperación)**: Mide la proporción de instancias positivas correctamente clasificadas con respecto a todas las instancias que son realmente positivas.

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

- **F1-Score**: Es una medida que combina la precisión y el recall en un solo valor. Se calcula como la media armónica de la precisión y el recall.

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Estas métricas proporcionan una manera sistemática y objetiva de evaluar y comparar modelos, brindando una base sólida para la toma de decisiones informadas sobre qué modelo utilizar en diferentes situaciones.

4.2 Diseño

Para las búsqueda de un modelo óptimo existen muchas vías. Esta sección está dedicada a la exploración y selección de un modelo óptimo para nuestro propósito. Hay numerosas metodologías disponibles para este fin, y en las siguientes líneas, vamos a explicar la que hemos escogido y los motivos por los cuales se han descartado otras.

Para cualquier modelo, hay un conjunto de parámetros que regulan su comportamiento. Estos se conocen como hiper-parámetros y, al buscar el modelo ideal, también es crucial encontrar el conjunto de hiper-parámetros óptimos.

La búsqueda del modelo óptimo puede cubrir todo el espacio de diseño, lo que se conoce como diseño factorial completo, o solo una parte de él. Para facilitar esta tarea, existen librerías como *Scikit-learn* [35] que proporcionan varios métodos de búsqueda.

Scikit-learn ofrece una herramienta denominada *GridSearch*, que permite llevar a cabo una búsqueda exhaustiva en los modelos y hiper-parámetros especificados. Esta herramienta toma una matriz de configuración y entrena y valida cualquier modelo definido.

En contraposición a la búsqueda exhaustiva, Scikit-learn también proporciona una herramienta de búsqueda heurística llamada *RandomSearch*. Aunque es similar a *GridSearch*, *RandomSearch* selecciona un número limitado de modelos para entrenar, en lugar de explorar todas las combinaciones posibles. Los modelos y hiper-parámetros que se prueban se seleccionan aleatoriamente.

Para calcular el número total de combinaciones posibles dada una matriz de configuración, podemos usar la siguiente fórmula:

$$M \times N^O \tag{4.1}$$

donde:

M es el número total de modelos,

N es el número total de hiper-parámetros,

O es el número de opciones posibles para cada hiper-parámetro.

Los métodos que encontramos en Scikit-learn tienen un problema, no se puede controlar el tiempo que tardan. Para un usuario como nosotros, que dispone de tiempo limitado para el entrenamiento de los modelos esto es un problema. Lo mas similar que encontramos es la selección de intentos en *RandomSearch*, pero esto no se asemeja a tener un tiempo máximo, porque cada intento puede variar considerablemente en tiempo.

Método de búsqueda

Con la finalidad de poder determinar un tiempo máximo de entrenamiento, hemos decidido crear un método similar a *RandomSearch*, pero limitando el tiempo de ejecución máximo. Con ello, seguimos el proceso presentado en la Figura 4.1.

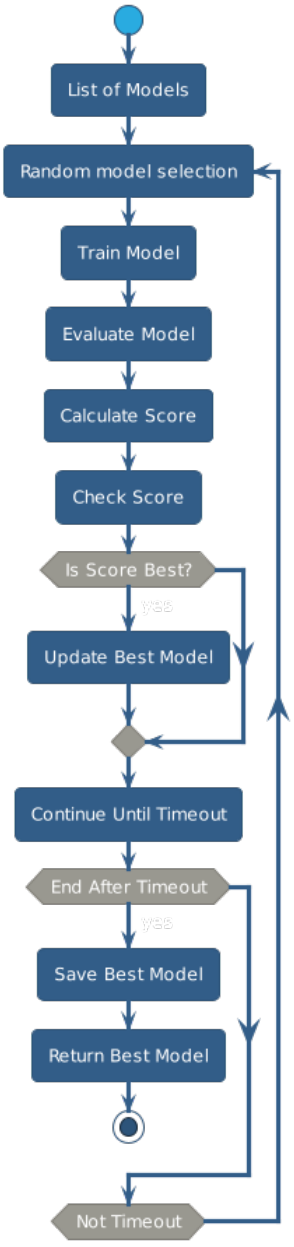

Figura 4.1: Búsqueda de modelo.

De esta forma, podemos seleccionar varios modelos y parámetros, y buscar la combinación óptima dentro de esta lista, teniendo en cuenta un límite de tiempo máximo para la búsqueda.

4.3 Búsqueda de modelos

Como se relata en la primera sección de este capítulo, existen muchas formas de crear y entrenar modelos de ML. Para este trabajo hemos considerado modelos reconocidos que se proporcionan en la librería Scikit-learn [35] y en la librería LightGBM [43]. En la tablas 4.1, 4.2 y 4.3 encontramos los modelos escogidos para nuestro caso. Estas tablas están separadas por familias de modelos y muestran los parámetros seleccionados.

Estos modelos y parámetros no son una representación completa pero son modelos reconocidos y parámetros que completan una buena búsqueda.

Modelo	**Parámetros**
LogisticRegression	C: [0.1, 1.0, 10.0, 100.0] penalty: [l1, l2, elasticnet, none]
SGDClassifier	alpha: [0.0001, 0.001, 0.1] penalty: [l1, l2, elasticnet]
LinearSVC	C: [0.1, 1.0, 10.0, 100.0] penalty: [l1, l2]
RidgeClassifier	alpha: [0.1, 1.0, 10.0, 100.0]

Tabla 4.1: Modelos Lineales.

Modelo	**Parámetros**
RandomForestClassifier	n_estimators: [12000, 15000] max_depth: [100, 500, 1000]
DecisionTreeClassifier	max_depth: [None, 500, 1000] criterion: [gini, entropy]
ExtraTreesClassifier	n_estimators: [5000, 10000, 12000] max_depth: [100, 500]
XGBClassifier	n_estimators: [5000, 7000] max_depth: [100, 200, 500]
LGBMClassifier	n_estimators: [5000, 10000, 12000] max_depth: [500, 1000]

Tabla 4.2: Modelos basados en árboles.

Modelo	**Parámetros**
SVC	C: [0.1, 1.0, 10.0, 100.0] kernel: [linear, poly, rbf, sigmoid]
AdaBoostClassifier	n_estimators: [1000, 5000, 10000] learning_rate: [0.1, 1.0, 10.0]
GradientBoostingClassifier	n_estimators: [100, 12000] learning_rate: [0.1, 1.0, 10.0]
BaggingClassifier	n_estimators: [1000, 5000, 10000]
KNeighborsClassifier	n_neighbors: [5, 50, 100] weights: [uniform, distance]
GaussianNB	-

Tabla 4.3: Otros Modelos.

Esta búsqueda se debe realizar para las cinco características seleccionadas. Hemos decidido separar y crear cinco modelos diferentes, uno para cada una de las características.

Las razones para separar los modelos y no unificar las características se deben a la limitación en el uso de la memoria; a más características, mayor uso de memoria. También es importante señalar el uso de características que no están relacionadas y la posibilidad de entrenarlas con diferentes tamaños de dataset.

El dataset inicial comprende 40,000 binarios, divididos en cuatro grupos de 10.000, benignos y maliciosos, ELF y PE. Estos se distribuyen en tres conjuntos: validación (16 %), prueba (20 %) y entrenamiento (64 %). El conjunto de prueba se utiliza exclusivamente para las estadísticas finales, no exponiéndose al modelo en ningún otro momento. El conjunto de validación sirve para recopilar métricas útiles en la selección de parámetros. Las gráficas presentadas emplean el conjunto de prueba para medir dichas métricas.

A continuación, se detallan los resultados y las decisiones tomadas durante la búsqueda del modelo para cada una de las características.

Mnemónicos

Las características que corresponden a los mnemónicos son obtenidas a partir del análisis que realiza Angr, que es una librería para análisis estático de binarios. Con estos datos de los binarios se han obtenido los resultados de la Figura 4.2.

La Figura 4.2, muestra el máximo resultado obtenido por cada tipo de modelo de la métrica de exactitud. Como vemos, estos resultados han sido muy prometedores, hay varios modelos, sobre todo los basados en arboles de decisión, que han obtenido unas muy buenas métricas.

Para esta característica no se ha encontrado ningún problema y se han usado 18 horas de entrenamiento.

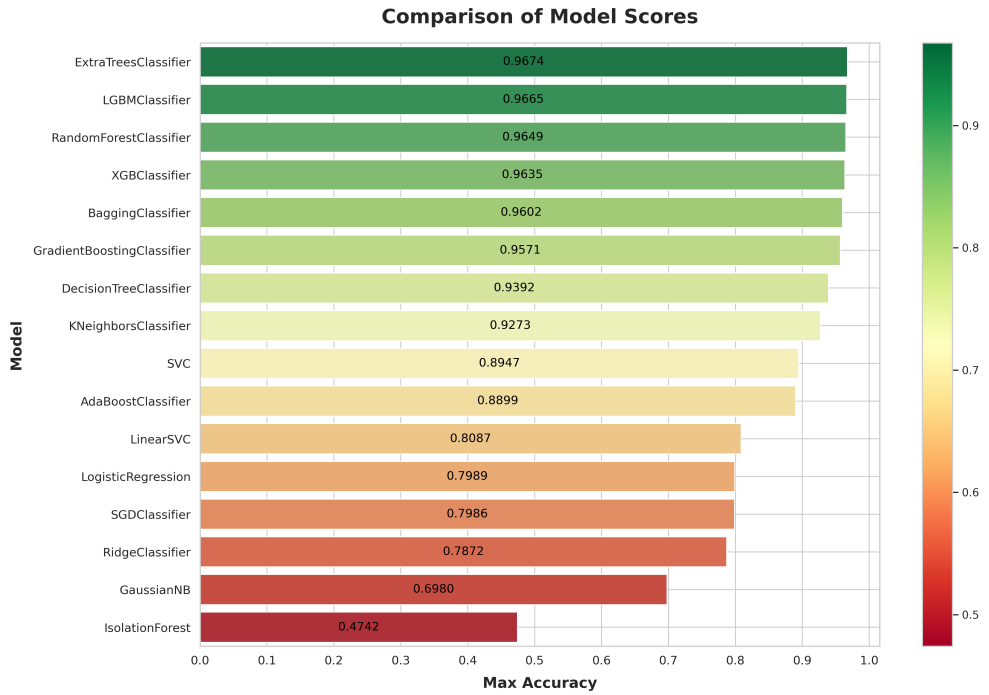

Figura 4.2: Resultados mnemónicos.

Cadenas de caracteres

En el estudio de la característica de cadenas de caracteres, y en las características siguientes, hemos utilizado 200.000 binarios. Esta decisión se debe a que, previamente, al hacer uso de solo 40.000 binarios, obteníamos una exactitud inferior a 0.90.

La principal razón para no hacer uso de 200000 binarios en todas las características es por el funcionamiento de las dependencias para extraer los datos de mnemónicos. Esta dependencia hace un uso algo más lento, relevante para esta cantidad de binarios, y no se usa esa fuente de datos en cualquier otra característica.

Por lo tanto, decidimos aumentar el tamaño del *dataset* a 200.000 binarios en el resto de las características, manteniendo las proporciones para cada uno de los tipos de binarios.

Figura 4.3: Resultados de cadenas de caracteres.

Como se puede observar en la Figura 4.3, ahora sí que se obtienen muy buenos resultados. Estos buenos resultados también han sido obtenidos con modelos basados en árboles, pero esta vez encontramos un modelo diferente que entrega el mejor resultado.

Por lo tanto, podemos concluir que al aumentar el tamaño de los datos se han logrado mejorar los resultados.

Funciones importadas

En la Figura 4.4, vemos los resultados obtenidos con la característica referente a las funciones importadas. Otra vez, los modelos basados en arboles, en este caso los más actuales, son los que mejores métricas obtienen.

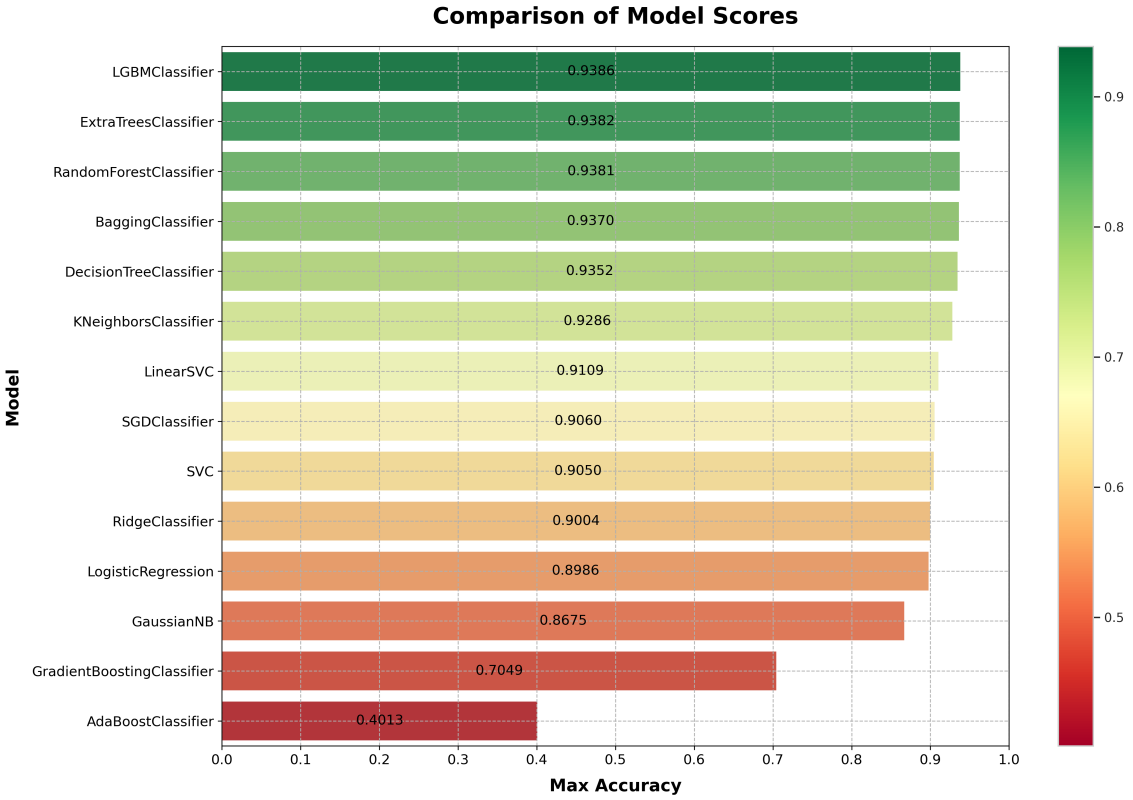

Figura 4.4: Resultados funciones importadas.

Entropía de las secciones

La entropía de las secciones es otra característica que en algunas ocasiones puede indicar que se trata de un programa malicioso.

Para esta característica también se han obtenido unos muy buenos resultados. Esta vez, y como veremos en todas las características, los modelos basados en arboles han destacado sobre los demás.

Los modelos basados en árboles pueden destacar sobre otros tipos de modelos debido a su capacidad para capturar relaciones no lineales, manejar características mixtas, ser robustos frente a datos ruidosos y tener una mayor interpretabilidad. Esto puede estar relacionado con el tipo de datos y las características específicas del problema.

En la Figura 4.5, vemos estos buenos resultados.

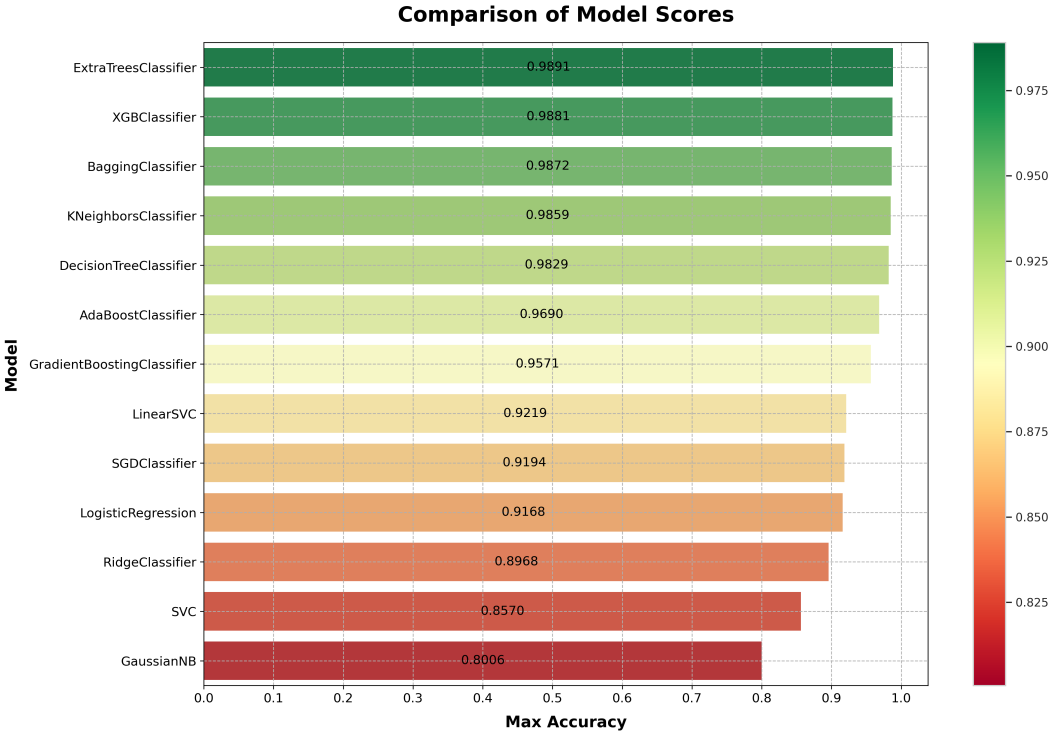

Figura 4.5: Resultados entropía secciones.

Características genéricas

Terminando con las características genéricas, vemos que, por un valor despreciable, es la que ha obtenido mejores resultados. Como se evidencia en la Figura 4.6, los modelos habituales son los que han destacado.

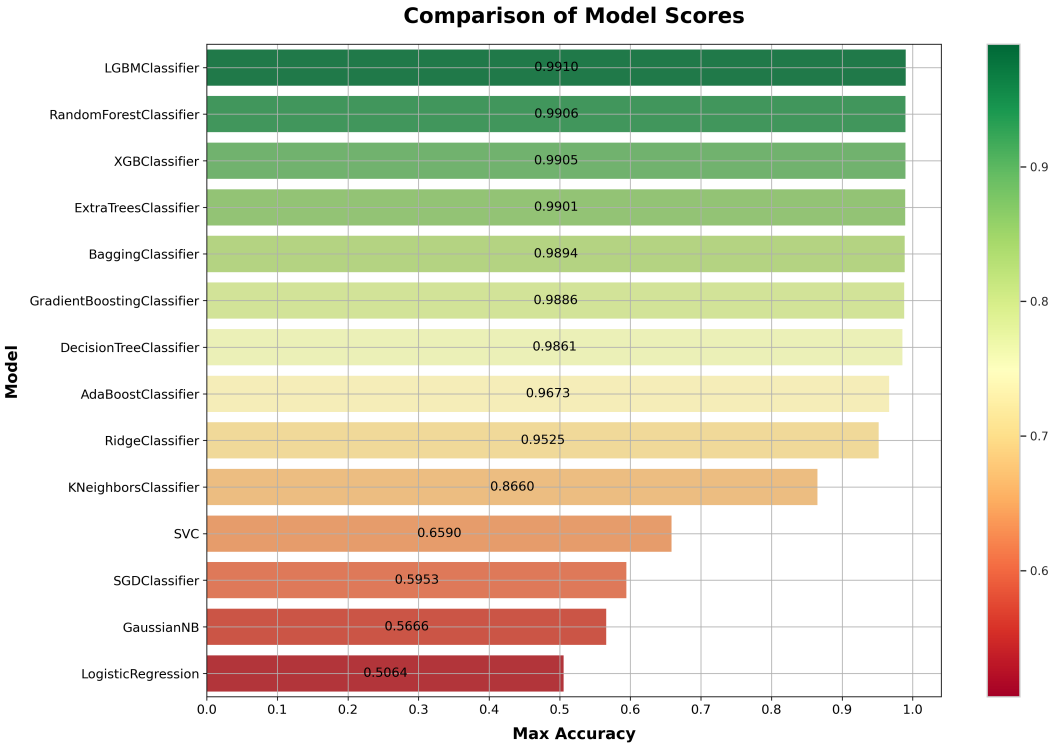

Figura 4.6: Resultados características genéricas.

Conclusiones de la búsqueda

Como hemos podido ver en los resultados, en todos los modelos de cada una de las características, los modelos que han destacado han sido los basados en árboles.

En concreto, dos modelos han destacado sobre los demás generalmente, ***ExtraTrees Classifier*** y ***Light GBM Classifier***. Ambos modelos basados en arboles y que innovan en el campo de esta familia.

Una vez hemos entrenado estos modelos, podemos realizar una pequeña comparación de las métricas más comunes. Podemos encontrar los resultados de esta comparación en la Tabla 4.3.

	Precision	**Recall**	**F1-score**	**Accuracy**
Functions	0.923	0.990	0.955	0.938
Entropy	0.990	0.988	0.989	0.989
Generic	0.991	0.99	0.991	0.991
Strings	0.987	0.955	0.971	0.977
Mnemonics	0.972	0.960	0.966	0.967

Tabla 4.4: Comparación resultados investigación

Como vemos, podemos concluir que las métricas son muy buenas y que los modelos han sobresalido en todas ellas.

La implementación de un sistema de votación para combinar las fortalezas del ***ExtraTrees Classifier*** y ***Light GBM Classifier*** podría impulsar aún más la eficacia y robustez de nuestras soluciones de machine learning.

En el próximo capítulo, se traslada la investigación a un prototipo usable y se implementa el sistema de votación para ello.

CAPÍTULO 5

De la Investigación al prototipo

La transición de la investigación de Machine Learning a una aplicación funcional implica múltiples desafíos. El primero radica en la brecha entre la investigación y la implementación; los modelos exitosos en entornos controlados pueden no rendir igual con datos del mundo real. A su vez, las características específicas utilizadas en la investigación pueden ser inaccesibles o complicadas de calcular en una aplicación en producción.

En la creación de una aplicación local específicamente diseñada para Machine Learning, la gestión del rendimiento y la interoperabilidad son desafíos clave. Los modelos de ML requieren recursos computacionales significativos, y garantizar que operen de manera eficiente en un dispositivo local puede ser un desafío.

Por último, la experiencia del usuario y la validación son aspectos críticos. Es esencial diseñar una aplicación de ML que sea intuitiva y útil para los usuarios, al mismo tiempo que se gestionan correctamente los datos de los usuarios.

En este capítulo, se afronta el problema de crear un prototipo usable para poner en práctica la creación de herramientas en producción que hacen uso de aprendizaje automático.

5.1 Problemática y solución propuesta

En esta sección se afronta el principal problema encontrado a la hora de llevar a cabo el prototipo y el diseño del mismo.

Como vemos en la Figura 5.1, el problema es evidente. Los modelos cargados en memoria usarán una cantidad de memoria mayor al del objeto serializado y eso haría muy costoso el conservar los modelos cargados durante la ejecución.

Si decidiéramos cargar cada vez un modelo haríamos la ejecución muy lenta, por lo tanto tenemos que encontrar una solución efectiva para mantener los modelos cargados y no usar excesiva memoria.

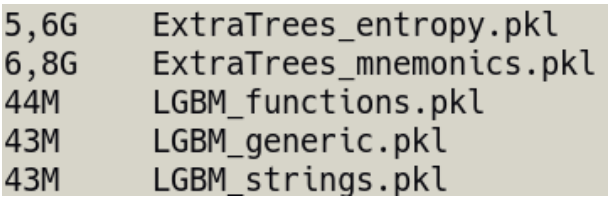

Figura 5.1: Tamaño modelos

Este problema tiene una solución sencilla. Se puede ver en la misma imagen. Los modelos basados en LightGBM son drásticamente más ligeros, por estar basados en la expansión de las hojas y no en la profundidad de los arboles de decisión. Y si nos fijamos en los resultados obtenidos en la sección anterior, en la Tabla 4.3, la pérdida por pasar a usar siempre este tipo de modelo no sería significativa.

En la Tabla 5.1, vemos los resultados de todas las métricas aplicadas a estos modelos. En concreto, cabe fijarse en los modelos basados en características de entropía y en mnemónicos. En definitiva, el coste es asumible y tenemos un gran ahorro de memoria principal y secundaria.

	Precision	**Recall**	**F1-score**	**Accuracy**
Functions	0,923	0,990	0,955	0,954
Entropy	0,989	0,990	0,990	0,990
Generic	0,991	0,990	0,991	0,991
Strings	0,987	0,955	0,971	0,971
Mnemonics	0,965	0,965	0,965	0,965

Tabla 5.1: Evaluación del rendimiento.

El siguiente problema a afrontar es la combinación de los modelos. Para ello hemos decidido hacer uso de un sistema de votación. En contraposición al uso de *stacking*, un sistema que crea un modelo nuevo que toma como entrada la salida de los anteriores. Un sistema de votación podría ser más óptimo, y sería más simple de desarrollar.

Votos

Existen muchos sistemas de votación, un voto positivo marca el resultado positivo, todos los votos deben de ser positivos para que el resultado sea positivo o una mayoría marca el resultado.

Para este caso, hemos decidido hacer uso de una mayoría. Con esta decisión podemos aumentar las métricas y podemos optimizar la herramienta.

La optimización realizada consiste en usar solo la característica más lenta como desempate. Esto deja a la característica basada en las estadísticas de los mnemónicos como el voto de desempate. Si no es necesario no se hará uso de su voto.

Probabilidades

Hasta el momento no se había explorado el uso de las probabilidades. Algunos modelos facilitan la obtención de probabilidades para cada clase y nuestro tipo de modelo lo facilita.

Se ha decidido dar esta opción a la herramienta final, devolviendo la probabilidad de cada modelo para las dos posibles clases y se ha realizado una media entre los modelos que después es mostrada. Cabe destacar que se podría ponderar la media a partir del resultado de una métrica y devolver el resultado ponderado.

5.2 Resultados

Para verificar el funcionamiento del prototipo creado se ha creado una pequeña utilidad que haciendo uso de las funcionalidades de la herramienta escanea una carpeta y guarda los resultados.

Para una claridad y énfasis sin ambigüedades, es crucial mencionar que se han incorporado aproximadamente 20000 binarios inéditos en este proceso. Es decir, estos binarios, en ninguna circunstancia, han sido empleados anteriormente para ningún tipo de entrenamiento o procedimiento, lo cual subraya su singularidad y relevancia. La distribución de los archivos empleados se puede observar en la Tabla 5.2

	Malware	**Beningware**
Number of ELFs	5606	5297
Number of PEs	5133	5054
Total	10739	10351

Tabla 5.2: Distribución binarios prototipo.

Con estos binarios se ha ejecutado el prototipo y se han obtenido los resultados mostrados en la Tabla 5.3. En esta Tabla, vemos los resultados por modelo y los resultados de la votación.

Model	**Accuracy**	**Precission**	**Recall**	F1 **Score**
Functions	0,925	0,881	0,985	0,930
Entropy	0,961	0,949	0,976	0,962
Strings	0,955	0,977	0,934	0,955
Generic	0,770	0,713	0,917	0,802
Mnemonics	0,783	0,557	0,666	0,607
Votes	0,966	0,951	0,984	0,967

Tabla 5.3: Métricas devAV.

Los resultados son generalmente positivos, a partir de un 0.95 de cada una de las métricas se considera un buen resultado y en este caso las métricas finales, la de los votos, está por encima de ese umbral. También cabe destacar que por lo general el uso de los votos ha mejorado los resultados finales.

Otro caso a destacar, aunque en este caso se refiere a un resultado negativo, es la bajada de métricas en los modelos Generic y Mnemonics. Esto puede deberse a diferentes razones, entre ellas sobreentrenamiento o por el uso de características que dificultan su generalización como los hashes en las características genéricas, pero sería necesario un estudio más amplio al abarcado en este trabajo.

Algo que si que podemos afirmar es la generalización positiva de los modelos Functions, Entropy y Strings. Han disminuido algo sus métricas pero no han obtenido métricas negativas.

Como cierre a la presentación de resultados, es esencial subrayar que se han llevado a cabo diversos tipos de pruebas.

Inicialmente, realizamos pruebas donde únicamente un voto se consideraba como positivo. Los detalles específicos de estos resultados pueden ser consultados en la Tabla 5.4. Vemos que no hay mejora por el uso de un único voto, y podemos descartar esta alternativa.

Además, efectuamos una serie de pruebas donde se excluyeron los modelos basados en mnemónicos y en características genéricas. Los datos obtenidos en estas pruebas están detallados en la Tabla 5.5.

Cabe destacar que los resultados extraídos de esta última serie de pruebas muestran una notable similitud con los ya adquiridos previamente. Por lo tanto, si se encontraran restricciones de tiempo o espacio, sería plausible contemplar el uso de únicamente estos tres modelos.

	Accuracy	**Precission**	**Recall**	F1 **Score**
Sistema monovoto	0,776	0,695	0,998	0,819

Tabla 5.4: Métricas prototipo considerando un voto como positivo.

Model Type	**Accuracy**	**Precission**	**Recall**	**F1 Score**
Functions	0,925	0,881	0,985	0,930
Entropy	0,961	0,949	0,976	0,962
Strings	0,955	0,977	0,934	0,955
Voting	0,966	0,950	0,985	0,967

Tabla 5.5: Métricas prototipo con 3 modelos.

5.3 Comparación Experimental: ClamScan y la Solución Propuesta

ClamScan [44] de ClamAV es un antivirus de código abierto ampliamente utilizado en el campo de la seguridad informática. Diseñado para detectar y eliminar malware, se ha convertido en una herramienta estándar en la protección contra amenazas cibernéticas.

En esta sección, se realizará una comparación experimental entre ClamScan y la solución propuesta, con el objetivo de evaluar su rendimiento y capacidad para detectar y neutralizar malware. Esta comparación permitirá determinar la eficacia de la solución propuesta y su potencial para mejorar o complementar las funcionalidades de ClamScan en la detección de amenazas informáticas.

Los resultados obtenidos en la comparación se muestran en la Tabla 5.6, donde se puede observar que el prototipo creado obtiene resultados ligeramente mejores, aunque la solución de ClamAV mantiene una mejor precisión. Se utilizaron los mismos datos para el proceso de evaluación y fue la primera vez que ambas soluciones los enfrentaron.

Solución	Accuracy	Precisión	Recall	F1-Score
Prototipo	0.966	0.951	0.984	0.967
ClamScan	0.8679	1	0.7359	0.8478

Tabla 5.6: Comparación de resultados entre el Prototipo y ClamScan.

5.4 Prototipo final

La aplicación final se ha puesto a disposición de cualquier usuario en Github [45], está disponible bajo licencia de código abierto y cualquier usuario puede acceder a ella. Se han añadido junto a la herramienta, todo el código y datos ya procesados usados para el desarrollo de este proyecto.

Adicionalmente, en el Apéndice B podemos encontrar la guía de uso y en la Figura 5.2, una ilustración con el aspecto actual del prototipo.

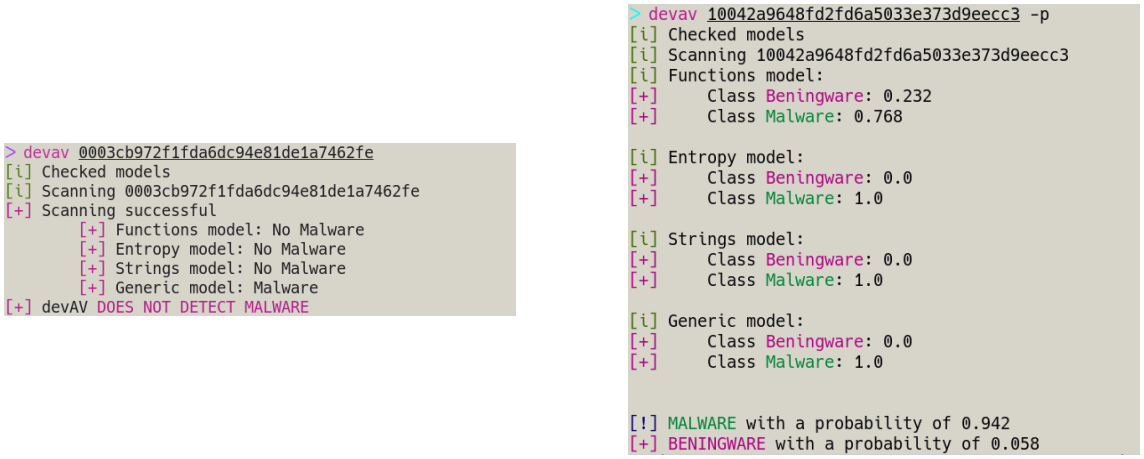

Figura 5.2: Aspecto prototipo.

CAPÍTULO 6

Conclusión

Durante este trabajo se ha explorado la estructura de los archivos binarios ejecutables. Se ha observado su complejidad, pero se ha logrado extraer de manera automática información relevante de estos archivos de manera exitosa.

Como se ha visto, se han recopilado binarios y se han creado procedimientos para la conversión y transformación de esta información, a fin de obtener los formatos deseados y necesarios para continuar una investigación relacionada con el aprendizaje automático.

Se ha completado de forma muy positiva la aplicación de nuevas técnicas de inteligencia artificial a la ciberseguridad, logrando crear un prototipo exitoso. Los experimentos realizados respaldan nuestro trabajo y muestran resultados muy buenos.

El camino para alcanzar una aplicación utilizable ha sido tortuoso pero exitoso. Hemos considerado aspectos relacionados con las necesidades espaciales y temporales para lograr una aplicación funcional y escalable. Hemos evaluado y comprobado el funcionamiento de esta aplicación en un entorno real.

Y para finalizar, hemos puesto a disposición de la comunidad los resultados obtenidos y el código desarrollado, para que la comunidad pueda beneficiarse de la misma forma en que nos ha permitido avanzar en este proyecto.

En futuro, cabría y sería necesario realizar algunas mejoras. Como usar sistemas pensados para el entrenamiento de estos modelos, aumentar el tamaño del dataset con todos los posibles archivos y usar un lenguaje óptimo para la herramienta en producción.

El trabajo realizado no tiene la intención de reemplazar ninguna solución existente, pero sí busca contribuir a nuevas formas que pueden ser parte, y de hecho ya lo son, de las herramientas que nos protegen.

Por último, instamos a todos aquellos que lean este documento a sumergirse en el ámbito de la seguridad, sin importar el campo, desde una perspectiva de justicia, con el objetivo de lograr un mundo mejor y aprovechar los beneficios que la tecnología, de la cual dependemos tanto, nos brinda.

"The computer is incredibly fast, accurate, and stupid.
Man is unbelievably slow, inaccurate, and brilliant.
The marriage of the two is a challenge and opportunity beyond imagination."

— Stuart G. Walesh

Bibliografía

[1] The Santa Cruz Operation, Inc. and Caldera International and The SCO Group and Xinuos, Inc., "System V Application Binary Interface - DRAFT - 10 June 2013," 2013. [Online]. Disponible en: https://www.sco.com/developers/gabi/latest/contents.html (Accedido por última vez: 2023-06-11).

[2] Hewlett Packard. (1997) ELF-64 Object File Format, Version 1.4. [Online]. Disponible en: http://www.staroceans.org/e-book/elf-64-hp.pdf (Accedido por última vez: 2023-06-11).

[3] TIS Committee. (1995) Tool Interface Standard (TIS) Executable and Linking Format (ELF) Specification, Version 1.2. [Online]. Disponible en: https://refspecs.linuxfoundation.org/elf/elf.pdf (Accedido por última vez: 2023-06-11).

[4] J. R. Levine, *Linkers & Loaders*. Morgan Kaufmann, 1999, chapter 1, Section: Linking vs. loading, Page 10.

[5] The FreeBSD Project, "elf(5) - FreeBSD 13.0," 2022. [Online]. Disponible en: https://man.freebsd.org/cgi/man.cgi?query=elf&sektion=5&manpath=FreeBSD+13.2-RELEASE+and+Ports (Accedido por última vez: 2023-06-11).

[6] Microsoft. (2023) Pe format. [Online]. Disponible en: https://learn.microsoft.com/en-us/windows/win32/debug/pe-format (Accedido por última vez: 2023-06-11).

[7] National Security Agency, "Ghidra." [Online]. Disponible en: https://ghidra-sre.org/ (Accedido por última vez: 2023-06-22).

[8] Hex-Rays, "Ida pro." [Online]. Disponible en: https://www.hex-rays.com/products/ida/

[9] radare, "Radare2." [Online]. Disponible en: https://www.radare.org/ (Accedido por última vez: 2023-06-22).

[10] "Angr," https://angr.io, (Accedido por última vez: 2023-06-14).

[11] Quarkslab, "Lief." [Online]. Disponible en: https://lief.quarkslab.com/ (Accedido por última vez: 2023-06-22).

[12] Linux man-pages project, "readelf(1) - linux man page." [Online]. Disponible en: https://linux.die.net/man/1/readelf (Accedido por última vez: 2023-06-22).

[13] "pefile," https://pefile.readthedocs.io, (Accedido por última vez: 2023-06-14).

[14] "pyelftools," http://pyelftools.readthedocs.io, (Accedido por última vez: 2023-06-14).

[15] Wayne Radburn, "Peview." [Online]. Disponible en: http://wjradburn.com/software/ (Accedido por última vez: 2023-06-22).

[16] Linux man-pages project, "objdump(1) - linux man page." [Online]. Disponible en: https://linux.die.net/man/1/objdump (Accedido por última vez: 2023-06-22).

[17] Red Hat Inc., "Elfutils." [Online]. Disponible en: https://sourceware.org/elfutils/ (Accedido por última vez: 2023-06-22).

[18] Endgame, "Ember." [Online]. Disponible en: https://github.com/endgameinc/ember (Accedido por última vez: 2023-06-22).

[19] VirusTotal, "Yara." [Online]. Disponible en: https://virustotal.github.io/yara/ (Accedido por última vez: 2023-06-22).

[20] python lz4, "Lz4 bindings for python," https://github.com/python-lz4/python-lz4, 2023, github repository. (Accedido por última vez: 2023-06-16).

[21] J. Doe, "Exploring compression techniques for root io," https://www.researchgate.net/figure/Comparison-among-compression-algorithms_tbl1_316451176, 2022, (Accedido por última vez: 2023-06-14).

[22] S. Benlloch, "Binsniff," https://github.com/sg1o/BinSniff, 2023, (Accedido por última vez: 2023-06-16).

[23] Microsoft, "The app certification process - windows apps," 2023. [Online]. Disponible en: https://learn.microsoft.com/en-us/windows/apps/publish/publish-your-app/app-certification-process?pivots=store-installer-msix#in-the-store (Accedido por última vez: 2023-06-16).

[24] Ubuntu, "Securityteam/policies - ubuntu wiki," 2021. [Online]. Disponible en: https://wiki.ubuntu.com/SecurityTeam/Policies (Accedido por última vez: 2023-06-16).

[25] VirusShare. (2023) Virusshare.com. [Online]. Disponible en: https://virusshare.com/ (Accedido por última vez: 2023-06-16).

[26] vx underground. (2023) vx-underground.org. [Online]. Disponible en: https://vx-underground.org/ (Accedido por última vez: 2023-06-16).

[27] "Malwarebazaar." [Online]. Disponible en: https://bazaar.abuse.ch/ (Accedido por última vez: 2023-06-16).

[28] A. Kumar. (2023) Feature selection vs feature extraction: Machine learning. Vitalflux.com. [Online]. Disponible en: https://vitalflux.com/machine-learning-feature-selection-feature-extraction/ (Accedido por última vez: 2023-06-16).

[29] Google. Data preparation and feature engineering for machine learning, normalization. [Online]. Disponible en: https://developers.google.com/machine-learning/data-prep/transform/normalization (Accedido por última vez: 2023-06-20).

[30] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Brew, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, and A. Rush, "Transformers: State-of-the-art natural language processing," 2020.

[31] E. G. Dada, J. S. Bassi, H. Chiroma, S. M. Abdulhamid, A. O. Adetunmbi, and O. E. Ajibuwa, "Machine learning for email spam filtering: review, approaches and open research problems," *Heliyon*, vol. 5, no. 6, p. e01802, 2019. [Online]. Disponible en: https://www.sciencedirect.com/science/article/pii/S2405844018353404 (Accedido por última vez: 2023-06-22).

[32] S. Shi, R. Tse, W. Luo *et al.*, "Machine learning-driven credit risk: a systemic review," *Neural Computing and Applications*, vol. 34, pp. 14 327–14 339, 2022, (Accedido por última vez: 2023-06-22).

[33] A. Ullah, M. I. Mohmand, H. Hussain, S. Johar, I. Khan, S. Ahmad, H. A. Mahmoud, and S. Huda, "Customer analysis using machine learning-based classification algorithms for effective segmentation using recency, frequency, monetary, and time," *Sensors*, vol. 23, no. 6, 2023. [Online]. Disponible en: https://www.mdpi.com/1424-8220/23/6/3180 (Accedido por última vez: 2023-06-22).

[34] J. Fürnkranz, "Machine learning and game playing," in *Encyclopedia of Machine Learning*, C. Sammut and G. Webb, Eds. Boston, MA: Springer, 2011, (Accedido por última vez: 2023-06-22).

[35] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux, "API design for machine learning software: experiences from the scikit-learn project," in *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, 2013, pp. 108–122, (Accedido por última vez: 2023-06-20).

[36] Y. Song and Y. Lu, "Decision tree methods: applications for classification and prediction," *Shanghai Archives of Psychiatry*, vol. 27, no. 2, pp. 130–135, 2015, pMID: 26120265. (Accedido por última vez: 2023-06-22).

[37] L. Breiman, "Random forests," *Machine Learning*, vol. 45, pp. 5–32, 2001, (Accedido por última vez: 2023-06-22).

[38] A. Natekin and A. Knoll, "Gradient boosting machines, a tutorial," *Frontiers in Neurorobotics*, vol. 7, p. 21, 2013, pMID: 24409142. (Accedido por última vez: 2023-06-22).

[39] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," ser. NIPS'17. Red Hook, NY, USA: Curran Associates Inc., 2017, p. 3149–3157, (Accedido por última vez: 2023-06-22).

[40] J. Lederer, "Linear regression," 2021, (Accedido por última vez: 2023-06-22).

[41] M. D. Fiuza Pérez and J. C. Rodríguez Pérez, "La regresión logística: una herramienta versátil," *Nefrología*, vol. 20, no. 6, pp. 495–500, 2000. [Online]. Disponible en: https://www.revistanefrologia.com/es-la-regresion-logistica-una-herramienta-articulo-X0211699500035664 (Accedido por última vez: 2023-06-22).

[42] I. H. Sarker, "Deep learning: A comprehensive overview on techniques, taxonomy, applications and research directions," *SN Computer Science*, vol. 2, no. 6, p. 420, 2021, pMID: 34426802. (Accedido por última vez: 2023-06-22).

[43] Microsoft, "Lightgbm: A fast, distributed, high performance gradient boosting (gbt, gbdt, gbrt, gbm or mart) framework based on decision tree algorithms, used for

ranking, classification and many other machine learning tasks," 2023. [Online]. Disponible en: https://github.com/microsoft/LightGBM (Accedido por última vez: 2023-06-16).

[44] "Clamscan," https://www.clamav.net/clamav-clamscan, (Accedido por última vez: 2023-07-08).

[45] S. Benlloch, "devav," https://github.com/sg1o/devAV, 2023, (Accedido por última vez: 2023-06-22).

APÉNDICE A

Uso de BinSniff

BinSniff[1] es una herramienta diseñada específicamente para extraer características de los archivos binarios.

Este software fue desarrollado y probado principalmente en el sistema operativo Ubuntu 22.04 LTS. Fue concebido como parte de una iniciativa más amplia dirigida a simplificar la aplicación de técnicas de aprendizaje automático en el campo de la seguridad de bajo nivel, particularmente a través de la automatización del proceso de creación de conjuntos de datos.

A.1 ¿Qué es BinSniff?

BinSniff es una herramienta compacta pero potente que facilita la extracción de características informativas de los archivos binarios. Su función principal es analizar archivos binarios e identificar características clave. Estas incluyen el formato del archivo, los encabezados, las funciones importadas, el ensamblado de funciones, el código VEX de las funciones y las estadísticas de los mnemónicos.

A.2 Instalación

El software BinSniff se puede instalar siguiendo los pasos a continuación:

1. Clonar el repositorio desde su fuente, normalmente se realiza a través de un sistema de control de versiones como Git.
2. Ejecutar el script de instalación `setup.py` usando el comando `pip install -e .`.

A.3 Uso

BinSniff opera desde cualquier directorio. Los usuarios invocan el comando `binsniff` con las opciones necesarias para iniciar el proceso de análisis. Por ejemplo:

[1] https://github.com/sg1o/BinSniff

```
$ binsniff -b /ruta/al/archivo/binario.exe -o /ruta/al/salida
```

El comando anterior incluye *flags* que indican la ruta al archivo binario que se va a analizar (`-b`) y la ubicación donde se almacenará el archivo JSON de salida (`-o`). Las *flags* adicionales proporcionan más opciones de personalización, como:

- `-j` o `-json-name`: Establece un nombre personalizado para el archivo JSON de salida. Por defecto es `features.json`.
- `-harcode`: Pasa un archivo JSON para convertirlo en un diccionario e incrusta valores específicos en el archivo JSON de salida.

También se puede hacer uso de BinSniff como módulo de Python, pudiendo con ello crear herramientas que lo utilicen.

A.4 Funcionalidades Adicionales

El proyecto BinSniff también incluye una utilidad llamada `miner.py` ubicada dentro del directorio `tools`. Esta utilidad permite a los usuarios extraer características de múltiples binarios dentro de un directorio simultáneamente.

A.5 Características extraídas por BinSniff

BinSniff es capaz de extraer una amplia gama de características, convirtiéndolo en una herramienta completa para el análisis de archivos binarios:

- **Cadenas**: Determinar las cadenas presentes en el archivo binario.
- **Secciones**: Proporcionar información detallada sobre las secciones dentro del archivo binario.
- **Importaciones**: Identificar funciones importadas por el archivo binario.
- **Encabezados**: Enumerar los encabezados presentes dentro del archivo binario.
- **Ensamblado**: Detallar el ensamblado de funciones en el archivo binario.
- **VEX**: Especificar el código VEX de las funciones dentro del archivo binario.
- **Mnemónicos**: Proporcionar estadísticas de mnemónicos del archivo binario.

BinSniff admite los formatos de archivo ELF y PE para la extracción de características.

A.6 Licencia

BinSniff está licenciado bajo la Licencia GPLv3.

APÉNDICE B

Uso de devAV

devAV[1] es una herramienta diseñada para construir un detector de malware basado en técnicas de aprendizaje automático. Cubre todos los aspectos del desarrollo de aplicaciones, desde la minería de datos y la extracción de características, hasta la selección de modelos y la implementación de prototipos. Aunque devAV es un prototipo funcional, no pretende reemplazar el software antivirus profesional.

B.1 Descripción de los Modelos

devAV aplica varias técnicas para clasificar archivos como malware o benignos. Aquí se ofrece una breve descripción de los modelos utilizados:

- **Funciones**: Utiliza funciones importadas para la clasificación.
- **Cadenas**: Emplea un modelo BERT para extraer características de las cadenas para la clasificación.
- **Mnemónicos**: Clasifica según la frecuencia de los mnemónicos por grupos de tipos de instrucciones.
- **Entropía**: Utiliza la entropía de las secciones dentro de un archivo binario para la caracterización y clasificación.
- **Genérico**: Realiza la clasificación basándose en características básicas de los archivos PE y ELF.

Para garantizar el resultado más confiable, devAV aplica un **Sistema de Votación** para la toma de decisiones finales. Esto implica utilizar los resultados de los modelos mencionados, cada uno emitiendo un "voto" sobre la clasificación del archivo. La mayoría de los votos decide el resultado final.

[1] https://github.com/sg1o/devAV

B.2 Datos y Resultados

Nuestras pruebas involucraron un conjunto de datos de 21,090 archivos, compuesto por 10,739 archivos de malware y 10,351 archivos benignos. Las métricas de rendimiento resultantes fueron impresionantes:

	Functions	**Entropy**	**Strings**	**Generic**	**Mnemonics**	**Voting**
Accuracy	0,925	0,961	0,955	0,770	0,783	0,966
Precision	0,881	0,949	0,977	0,713	0,557	0,951
Recall	0,986	0,976	0,934	0,917	0,667	0,985
F1 Score	0,931	0,963	0,955	0,803	0,607	0,968

Tabla B.1: Métricas de devAV

Las estadísticas anteriores subrayan el potencial y la eficacia del aprendizaje automático en la ciberseguridad.

B.3 Instalación y Configuración

Clonar el repositorio de forma recursiva para incluir el submódulo:

```
git clone --recursive https://github.com/sg1o/devAV.git
```

Navegar al directorio del proyecto e instalar el proyecto:

```
cd devAV
pip install -e .
pip install -e binsniff/requirements.txt
```

Descomprima los modelos disponibles en ‘devav/models/compressed-files‘ usando un descompresor 7z.

Una vez instalado, puede utilizar el comando ‘devav‘ para escanear archivos:

```
devav --help
```

B.4 Licencia

Este proyecto se encuentra bajo la licencia GPL v3.

ANEXO

OBJETIVOS DE DESARROLLO SOSTENIBLE

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenible	**Alto**	**Medio**	**Bajo**	**No procede**
ODS 1. **Fin de la pobreza.**				x
ODS 2. **Hambre cero.**				x
ODS 3. **Salud y bienestar.**			x	
ODS 4. **Educación de calidad.**			x	
ODS 5. **Igualdad de género.**				x
ODS 6. **Agua limpia y saneamiento.**				x
ODS 7. **Energía asequible y no contaminante.**				x
ODS 8. **Trabajo decente y crecimiento económico.**				x
ODS 9. **Industria, innovación e infraestructuras.**		x		
ODS 10. **Reducción de las desigualdades.**				x
ODS 11. **Ciudades y comunidades sostenibles.**				x
ODS 12. **Producción y consumo responsables.**				x
ODS 13. **Acción por el clima.**				x
ODS 14. **Vida submarina.**				x
ODS 15. **Vida de ecosistemas terrestres.**				x
ODS 16. **Paz, justicia e instituciones sólidas.**				x
ODS 17. **Alianzas para lograr objetivos.**				x

ETS Enginyeria Informàtica
Camí de Vera, s/n, 46022, València
T +34 963 877 210
F +34 963 877 219
etsinf@upvnet.upv.es - www.inf.upv.es

Reflexión sobre la relación del TFG/TFM con los ODS y con el/los ODS más relacionados.

En general, el documento no cumple con la mayoría de ODS, pero si podemos decir que cumple alguno de ellos.

Por un lado, podemos afirmar que el documento trata y trabaja sobre detección de archivos maliciosos, que en parte y por desgracia han afectado de diferentes formas a infraestructuras médicas y de otro tipo relacionadas con la salud.

También podemos decir, que, al tratarse de un documento con un tono divulgativo participa en una educación de calidad.

Por último, el documento es un trabajo de investigación, aporta nuevas ideas y nuevas aproximaciones, por lo tanto va en línea del ODS 9, Industria, innovación e infraestructura.

Escola Tècnica Superior d'Enginyeria Informàtica
ETS Enginyeria Informàtica
Camí de Vera, s/n, 46022, València
T +34 963 877 210
F +34 963 877 219
etsinf@upvnet.upv.es - www.inf.upv.es