Método para Automatizar Clasificación de Malware PE32 mediante Técnicas de Procesamiento de Lenguaje Natural

Bruce W. Percílio Azevedo[12], Luis Alberto Martínez Hernández[1], Robson de Oliveira Albuquerque[13], Luis Javier García Villalba[1]

[1]Grupo de Análisis, Seguridad y Sistemas (GASS)
Departamento de Ingeniería del Software e Inteligencia Artificial (DISIA)
Facultad de Informática, Despacho 431, Universidad Complutense de Madrid (UCM)
Calle Profesor José García Santesmases, 9, Ciudad Universitaria, 28040 Madrid
Email: {luisal11, robsonde, javierv}@ucm.es

[2] Universidad Politécnica de Madrid (UPM), Madrid, España
Email: bruce.percilio@alumnos.upm.es

[3] Universidad de Brasilia (UnB), Brasil
Email: robson@redes.unb.br

***Resumen*—El malware se define como un programa informático que causa daños a los activos de información una vez que se ejecuta. Existen varios tipos de malware, siendo los más comunes los descargadores, keyloggers, mineros, ransomware, rootkits y spyware, entre otros. Cada uno de ellos tiene su propio comportamiento y características y, en algunos casos, pueden clasificarse en más de un tipo debido a sus características. El malware se suele escribir en estructuras con un mayor número de funcionaldades, con el objetivo de afectar al mayor número de activos de información posible. Una de las estructuras más utilizadas es la PE32, que pertenece al estándar de objetos de la familia de sistemas operativos Microsoft Windows, donde han estado disponibles como ejecutables o binarios. Existen dos metodologías para analizar malware, estática y dinámica, cada una compuesta por técnicas como sandbox, que emula el binario y mapea puntos clave, como la memoria y el sistema de archivos, para identificar comportamientos maliciosos. Aunque estas técnicas son muy utilizadas por investigadores de malware, por defecto, debido a su complejidad, suelen consumir mucho tiempo. Con el fin de optimizar el análisis de los binarios, esta investigación propone identificar de manea oportuna un posible malware antes de enviarlo a técnicas que realizan una inspección de la muestra de manera más profunda. La metodología se elabora mediante el uso de técnicas de desensamblado para extraer funciones y archivos de sistemas operativos Microsoft Windows. Después, se estructura un conjunto de datos con los datos extraídos, y se realiza una interpretación léxica basada en word2vec, con el objetivo de identificar funciones y tipos de malware. A continuación, la idea es utilizar los resultados como entrada para un modelo de aprendizaje profundo con técnicas de memoria a largo plazo, que se utilizan ampliamente como sustitutas de las redes neuronales convolucionales. La memoria a largo plazo mapeará los patrones y creará un modelo para identificar nuevas entradas (con la misma estructura) que los tipos de malware del entrenamiento original.**

***Index Terms*—Deep Learning, Lexical Analysis, Malware, PE32.**

Tipo de contribución: *Investigación en desarrollo*

I. Introducción

Los ordenadores se han convertido en parte importante de nuestras vidas, utilizados por particulares, empresas e incluso naciones. Aunque mejoran significativamente la eficiencia y la velocidad, las personas también se han vuelto dependientes de ellos. El coste de no tener acceso a sistemas de infraestructuras críticas o de no poder realizar transacciones financieras durante periodos prolongados es incalculable. Podría incluso provocar la pérdida de vidas.

El software es un elemento crucial de los productos informáticos y los sistemas de información. Sin embargo, el software malicioso o malware se ha convertido en una amenaza importante para la seguridad de estos sistemas. En los últimos años, los atacantes han recurrido cada vez más a este tipo de software para llevar a cabo sus ataques. En el desarrollo de malware se utilizan varias estructuras, uno de los mejores ejemplos es la estructura Ejecutable Portable (PE, del inglés *Portable Executable*), que ocupa la primera posición en el ranking de software analizado por virustotal [1].

En la actualidad existen dos categorías de analizadores de malware en archivos binarios, estático y dinámico [2]. En el método estático se realiza sin la ejecución del archivo, las técnicas más utilizadas son por análisis de hashing [3] y el análisis manual [4] mediante métodos de desensamblaje. Mientras que el análisis dinámico se realiza ejecutando el archivo, mediante el uso de sandbox [5] y el monitoreo de paquetes IP para identificar anomalías en el comportamiento [6]. Desafortunadamente, a la complejidad dada la naturaleza del análisis, las técnicas actuales, a pesar de ser efectivas, son consideradas lentas, consumiendo un tiempo considerable para concluir con lo que hace el ejecutable o el binario.

El objetivo de esta investigación es proponer una estrategia de análisis para binarios que estén construidos sobre la estructura PE32. Este análisis no reemplaza las técnicas existentes, es una forma de verificar qué binario vale la pena pasar por un proceso de análisis más detallado, complementando el análisis de un perfil de devsecops que se espera sea más rápido en la validación. Para ello se utilizarán técnicas de desensamblado, análisis léxico con Word2vec y modelos de aprendizaje

profundo como redes neuronales de memoria larga a corto plazo (LSTM, del inglés *Long Short-Term Memory*). Esto se implementará dentro de una estructura automatizada y fácil de usar construida sobre Docker. Los resultados obtenidos de esta estrategia se compararán con los hallazgos de investigaciones anteriores y se discutirán para obtener una comprensión más profunda de la evolución de la clasificación e identificación de malware.

Este documento está organizado en las siguientes en secciones. La Sección II presenta los antecedentes teóricos, que abarcan la estructura de archivo PE32, los tipos de malware, el análisis léxico y el aprendizaje profundo. La Sección III describe trabajos relacionados con la investigación. La Sección III presenta el método propuesto, que abarca la construcción de conjuntos de datos, el análisis léxico y la memoria a largo y corto plazo. En la Sección V se presentan y discuten los resultados, y finalmente, la Sección VI presenta con la conclusión y los trabajos futuros.

II. Estado del arte

PE32 es una estructura muy utilizada por los ciberdelincuentes para la creación de malware. Sin embargo, las técnicas actuales para analizar este tipo de estructura, dado la complejidad requerida, consumen mucho tiempo o requieren intervención manual. Este artículo presenta una propuesta de análisis que utiliza word2vec y LSTM. Debido a la profundidad de la propuesta y a las tecnologías actuales que considera, es necesario comprender con más detalle los componentes que la rodean.

II-A. Ejecutable Portátil

En 1993, Microsoft lanzó Windows NT e introdujo el concepto de ejecutable portable, que formaba parte de las especificaciones originales de las aplicaciones Win32. El término se acuñó para disponer de una estructura única que pudiera ejecutarse en todas las versiones de Windows y fuera compatible con todos los modelos de CPU. Este objetivo se cumplió en general, y la misma estructura se convirtió en el estándar para Windows en formato servidor y versiones posteriores, y Windows 95 y sus sucesores [7].

PE32 es una estructura de archivo que un sistema operativo utiliza para ejecutar programas. Estos programas vienen en varias extensiones de archivo, siendo .exe y .dll las más comunes. Todos siguen la misma estructura, con algunas diferencias que indican cómo debe tratarse el archivo.

La mayoría del software en la estructura PE depende en gran medida de funciones de otras DLL, que es necesario importar. Durante el proceso de inicio, Windows Loader localiza e importa estas funciones, haciéndolas accesibles en memoria [7].

Cuando Windows Loader crea un enlace a las funciones de otra DLL, también crea un enlace a la propia DLL. Estas funciones se cargan en memoria y se vinculan al programa PE32 mediante llamadas a la API.

Con la introducción de Windows de 64 bits, fue necesario hacer algunos cambios en la estructura PE. Esto dio lugar al estructura actualizado denominado PE32+. No se añadieron nuevos campos, pero se eliminó uno. Los campos restantes se ampliaron de la estructura de 32 bits al de 64 bits. En la mayoría de los casos, los archivos escritos en la estructura de 32 bits se reconocen automáticamente en los sistemas operativos con una estructura construida sobre la de 64 bits.

II-B. Tipos de Malware

El malware es un software diseñado para dañar los ordenadores o a sus usuarios de diversas formas, como robar información, corromper archivos, conceder acceso no autorizado a recursos o simplemente molestar a los usuarios [8].

Los creadores de malware buscan constantemente nuevos métodos para lograr sus objetivos. Mejoran sus técnicas utilizando la ofuscación y la exfiltración e intentando infectar al mayor número posible de víctimas. Últimamente, el número de incidentes de seguridad causados por programas maliciosos ha aumentado considerablemente.

El malware suele dividirse en tipos según su objetivo. La comunidad de investigadores de malware tiende a etiquetar el malware en categorías específicas según su comportamiento, funciones y otros criterios. Como no se puede calcular con precisión la cantidad de malware, estas categorías ayudan a identificar y diferenciar entre los distintos tipos de programas maliciosos que se encuentran en la estructura PE32. A continuación se listan los principales tipos de malware que existen en la actualidad.

Downloader: Un downloader es un tipo de malware diseñado para descargar e instalar otros programas maliciosos en los ordenadores infectados [9]. Los downloaders utilizan técnicas avanzadas para ocultar su verdadero propósito y camuflar el tráfico de fuentes externas, lo que dificulta su detección y eliminación.

- Keylogger: Los keyloggers son programas diseñados para robar información confidencial [10] mediante el control de las pulsaciones de teclado en los ordenadores infectados. Buscan PIN, números de tarjetas de crédito, textos confidenciales y otros datos valiosos.
- Miner: Miner es un malware diseñado específicamente para apoderarse de los recursos informáticos de la víctima y utilizarlos para minar criptomonedas para los atacantes [11]. El objetivo principal de esta herramienta es generar ganancias financieras para los atacantes.
- Ransomware: El ransomware es un tipo de software malicioso que, cuando se ejecuta, desactiva de alguna manera las funciones del ordenador [8]. El ransomware funciona normalmente cifrando los archivos del ordenador y exigiendo después el cumplimiento de unas demandas específicas para restaurar el acceso a los archivos. Se trata de una forma de chantaje, y puede ser muy perjudicial para particulares y organizaciones.
- RootKit: El rootkit es un tipo de software diseñado para ocultar la existencia de otros programas en dispositivos que llevan a cabo análisis de software conservando un acceso privilegiado [12]. Es una práctica habitual que el malware utilice este tipo de programas para permitir su propia instalación y activación en entornos ya infectados. Una vez instalados, estos programas pueden modificar el sistema operativo y el sistema que detecta el malware.
- Spyware: El spyware es un tipo de software malicioso que se instala secretamente en el ordenador de la víctima sin su conocimiento o consentimiento [13]. Una vez activado, vigila el comportamiento, los archivos y las

actividades del usuario y puede llegar a robar las credenciales de inicio de sesión y otra información confidencial. Los datos recopilados por el spyware son enviados a los creadores del malware.

Los desarrolladores de software utilizan ampliamente la estructura PE32. Sin embargo, su uso extensivo se ha convertido en la estructura preferido por los ciberdelincuentes que tienen como objetivo el mayor número posible de víctimas. Está clasificado como la estructura más analizada por Virustotal [1].

II-C. Análisis léxico

La similitud semántica es un tema crucial en lingüística, sobre todo a la hora de entender el significado de las palabras. En términos lingüísticos, dos palabras diferentes pueden compartir el mismo significado. El proceso de cálculo del grado de similitud entre ellas se ha basado tradicionalmente en el pensamiento humano. Sin embargo, este cálculo ha servido de base para desarrollar técnicas dentro del campo de la informática, como el Procesamiento del Lenguaje Natural (NLP, del inglés *Natural Language Processing*) y la Minería de Textos.

Una de las ramas de la inteligencia artificial es la NLP, que se ocupa de la interacción entre los sistemas informáticos y el lenguaje humano [14]. Para identificar el contexto de la frase, los sistemas informáticos deben procesar primero el lenguaje humano entrante y hacer algún razonamiento sobre los datos de entrada y lo que se espera de ellos.

Word2vec se desarrolló en 2013 [15] y actualmente es una de las técnicas más populares para calcular la similitud y relevancia de las palabras. Proporciona una representación vectorial única para cada palabra, teniendo en cuenta la semántica mediante la asignación de tokens similares con el objeto de ofrecer representaciones similares.

Word2vec no se considera el mejor ni el único modelo de NLP basado en la incrustación de palabras, analogías y similitudes [16]. Sin embargo, es sencillo y accesible; cualquiera puede descargar y utilizar su código.

II-D. Deep Learning

Aprendizaje Automático (ML, del inglés *Machine Learning*) es un término utilizado para describir algoritmos computacionales que permiten a los sistemas resolver problemas sin estar explícitamente programados para ello. Introduce técnicas que permiten aprender de la experiencia, desarrollar conceptos a partir de ejemplos y extraer patrones de datos brutos [17].

Han surgido nuevos enfoques para problemas que requieren más complejidad. Estos enfoques no pretenden sustituir al ML, sino complementarlo, especialmente cuando se trata de funciones o dominios específicos. Uno de estos enfoques es el Aprendizaje Profundo (DL, del inglés *Deep Learning*). Permite a los sistemas informáticos aprender conceptos complejos descomponiéndolos en otros más simples y creando una jerarquía de capas. Esto se consigue creando una red neuronal para lograr este objetivo.

Una red neuronal es un sistema complejo compuesto por muchas partes interconectadas llamadas neuronas, similar en estructura al cerebro humano. Estas neuronas están dispuestas en capas y el DL se refiere al aprendizaje logrado a través de redes neuronales con múltiples capas de neuronas. Como se observa en la Figura 1, las capas de una red neuronal incluyen la capa de entrada, las capas ocultas y la capa de salida.

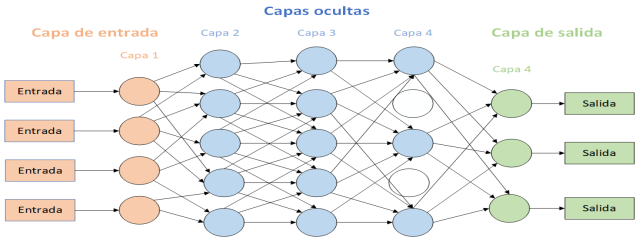

Figura 1. Red Neural

Cada neurona de una red neuronal recibe varios números como entrada y produce un único valor de salida denominado activación neuronal. Este proceso ocurre una sola vez y los datos nunca vuelven a pasar por la neurona. El procesamiento de datos en una neurona depende únicamente de la entrada que recibe y no tiene relación con ningún dato procesado anteriormente.

Las redes neuronales pueden estructurarse como Redes Neuronales Recurrentes (RNN, del inglés *Recurrent Neural Network*) para tratar el flujo de datos que pasan a las neuronas, formando un ciclo dentro de la red. Las RNN utilizan un estado interno que puede guardarse durante largos periodos y al que se puede acceder desde otras neuronas.

Durante el entrenamiento, la red neuronal utiliza la retropropagación para calcular los gradientes, que parten de las capas más profundas y fluyen de vuelta a la inicial. Sin embargo, al entrenar una RNN, los gradientes procedentes de las capas más profundas deben pasar por operaciones continuas de este tipo. El algoritmo de retropropagación utiliza la regla de la cadena para calcular los gradientes, y cuanto más alejados están de las capas anteriores, más multiplicaciones con números pequeños se producen. Esto conduce a un gradiente exponencialmente decreciente, lo que significa que la información se pierde, y el modelo ya no puede aprender de las entradas iniciales.

II-D1. Long-Short Term Network: Sepp Hochreiter y Jürgen Schmidthuber introdujeron el concepto LSTM en 1997 [18]. La LSTM es una arquitectura de red neuronal que aborda el problema del gradiente descendiente imponiendo un flujo de error constante a través de los estados internos. Esto se consigue mediante un algoritmo basado en gradientes que evita tanto la explosión como la fuga.

La LSTM es una red neuronal que utiliza una célula de memoria para almacenar información relevante. El contenido de la memoria puede modificarse mediante tres capas sigmoidales de la red neuronal denominadas “puertas”: la puerta de entrada, la puerta de olvido y la puerta de salida. Cada puerta devuelve un valor entre 0 y 1, donde 0 significa que no pasa ninguna información y 1 que pasa toda la información [19]. Las tres puertas se entrenan para influir en el contenido de la celda de memoria y en la salida final. Así es como las LSTM pueden evitar el problema de la desaparición o explosión de gradientes aprendiendo qué información es necesario conservar y cuál debe olvidarse.

III. Trabajos relacionados

Dai et al. [20] propusieron utilizar redes neuronales para clasificar el malware analizando los volcados de memoria tomados de los programas durante su ejecución. Ejecutaron programas maliciosos en un sandbox y capturaron la memoria mientras los programas se ejecutaban para crear un conjunto de datos. Utilizaron este conjunto de datos para entrenar un modelo basado en redes neuronales. Inicialmente, utilizaron una Red Neuronal Convolucional (CNN, del inglés *Convolutional Neural Network*) para construir el primer modelo, que más tarde actualizaron a una red LSTM. El modelo actualizado dio mejores resultados, principalmente gracias a la mejora del manejo del gradiente.

Young Jun Lee et al. [21] han propuesto una estructura propia de análisis léxico para analizar código ensamblador con el fin de encontrar vulnerabilidades en formas similares de código expuesto. Para mapear similitudes en el código, emplearon CNN. El procedimiento demostró su eficacia, con una tasa de validación del 94 %, lo que indica el éxito del uso combinado de análisis léxico y aprendizaje profundo. El uso del análisis léxico fue fundamental para el éxito de la comparación y validó la eficacia del uso conjunto de estas tecnologías.

Wartschinski [22] creó una herramienta de análisis de vulnerabilidades para su tesis de maestría utilizando Python y recursos de código abierto. Comenzó mapeando el repositorio primario de Python en GitHub y descargando los datos comprometidos. A continuación, utilizó Word2Vec para mapear los patrones de tokens dentro del lenguaje y creó una estructura numérica para alimentar un modelo de red neuronal basado en LSTM. Este modelo utiliza la clasificación binaria para comprobar si el código es vulnerable. Al convertir los scripts generados en una herramienta de código abierto, pudo reproducir todo el proceso con los modelos Word2Vec y LSTM.

Visweswaran *et al.* [23] analizaron archivos en la estructura PE32. Extrajeron funciones de llamada a la API de un conjunto de datos de malware utilizando técnicas de desmontaje y tokenización para convertir su base en una numérica. A continuación, introdujeron esta nueva base numérica en un algoritmo de red neuronal CNN, cuyo objetivo era mapear los patrones de los tipos de malware. Sin embargo, la solución no puede recrearse porque la técnica utilizada para crear la estructura que convierte los datos en números no puede guardarse. Cada vez que se utiliza el modelo, hay que volver a crearlo. Incluso con el mismo conjunto de datos, diferirá del original, aunque sólo sea por unos pocos detalles.

IV. Método propuesto

Se pueden utilizar tres métodos para analizar archivos en la estructura PE32: técnicas de desensamblado, análisis heurístico y sandboxing. Herramientas como PE Explorer han sido desarrolladas específicamente para manipular las características de la estructura PE32, utilizando técnicas de desensamblaje para obtener una comprensión más profunda de cómo se comportan los programas. El análisis heurístico es un método que utiliza técnicas de descompilación para extraer tokens específicos del código de programas sospechosos, que luego se comparan con programas maliciosos conocidos para identificar similitudes que puedan utilizarse para encontrar amenazas potenciales. El sandboxing consiste en emular el entorno especificado para la aplicación sospechosa y mapear puntos clave de su comportamiento, como el uso de RAM, disco y CPU. Esto se hace para identificar las características que se asemejan al comportamiento de los programas maliciosos.

Los procesos de análisis de archivos en la estructura PE32 pueden llevar mucho tiempo y a menudo requieren la intervención de un especialista. Para solucionarlo, este trabajo propone utilizar técnicas de desensamblaje en un conjunto de datos de malware para extraer funciones de los binarios y crear un conjunto de datos en formato CSV. Este conjunto de datos entrenará un modelo de análisis léxico que utiliza el algoritmo word2vec, que luego se alimentará como entrada a un modelo de aprendizaje profundo que utiliza LSTM. En la Figura 2 se muestra la representación visual del método propuesto.

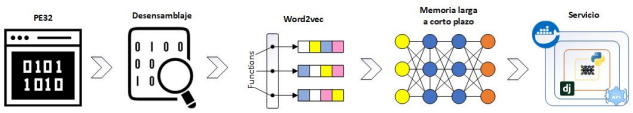

Figura 2. Propuesta de método

Utilizando modelos word2vec y LSTM, esta propuesta crea una estructura docker capaz de recibir binarios de malware e identificar su tipo. Para completar cada tarea planificada es necesario seguir una serie de pasos.

IV-A. Construcción de conjuntos de datos

El primer paso para construir la solución propuesta fue reunir un conjunto de datos compuesto por distintos tipos de malware. Con el objetivo de desarrollar una propuesta lo más parecida posible a un entorno real, en este trabajo se utilizaron tipos reales de malware a partir de información, fechas y detalles ampliamente difundidos. Para construir el conjunto de datos, se hace uso del repositorio de intercambio de malware Virus Share [24], un framework de intercambio abierto y gratuito destinado exclusivamente al almacenamiento de malware, en diferentes formatos y estructuras. Fue necesario filtrar las muestras para obtener únicamente binarios en la estructura PE32.

Se eligieron seis tipos de malware, cuyas muestras se descargaron y almacenaron localmente para componer el conjunto de datos de origen. La Tabla I especifica la cantidad y el nombre de cada tipo de malware adquirido de Virus Share y su valor total.

Tabla I
Cantidad de Malware por tipos

Tipo	Cantidad
Malware	200
Downloader	200
Miner	200
Ransomware	200
RootKit	200
Spyware	200
Total	1200

IV-B. Desensamblaje de binarios para extraer funciones

Tras crear el conjunto de datos de malware, se desarrolló un programa en C++ utilizando la biblioteca pibpe [25]. Esta

biblioteca contiene técnicas de desensamblaje [26] que extraen puntos clave de muestras binarias en PE32 para analizarlas.

El objetivo principal del programa C++ es extraer las funciones que utilizan los binarios PE32 de las DLL externas necesarias para la ejecución de los binarios.

El programa C++ analiza un directorio completo pasado como argumento, extrae las funciones de todos los binarios dentro del directorio y exporta un archivo CSV con tres columnas: las funciones, el hash del binario y el tipo.

IV-C. Análisis léxico

Se ha desarrollado un algoritmo en Python para realizar el análisis léxico. El primer paso del algoritmo consiste en importar el archivo CSV generado en la sección anterior. Filtra las funciones y los tipos y divide las funciones en una lista de cadenas.

Para el análisis léxico se utilizó el algoritmo word2vec, cuya implementación se importó de la biblioteca gensim [27], escrita en Python. Se pasó una lista de cadenas para crear una instancia del modelo con los argumentos enumerados en la Tabla II.

Tabla II
DESCRIPCIÓN DE LOS ARGUMENTOS DE WORD2VEC

Parámetro	Descripción	Valor
vector_size	Dimensionalidad de los vectores de palabras	400
window	Distancia máxima entre la palabra actual y la prevista dentro de una frase.	100
min_count	Ignora todas las palabras cuya frecuencia total sea inferior a este valor.	10
workers	Utiliza estos hilos de trabajo para entrenar el modelo.	8

El resultado del modelo word2vec es similar a un diccionario, formado por claves y valores. Además se agrega un método que devuelve el valor numérico correspondiente para cualquier parámetro de la cadena dada.

Para convertir cadenas en valores numéricos, se creó la siguiente función para iterar a través de las cadenas y llamar a una función que realiza la conversión utilizando el modelo word2vec.

```
def word2token(word)
    return model.wv.key_to_indx[word]
for wd in data:
    wd_tokken = np.array([word2token(w)
        for w in wd if w != " "])
```

Tras convertir la matriz de cadenas en una matriz de números, el modelo word2vec se guardó localmente.

IV-D. Creación y utilización de modelos LSTM

El programa LSTM fue desarrollado utilizando el lenguaje Python, con la ayuda de la librería Keras creada en 2015 y adoptada como interfaz por el equipo Tensorflow de Google [28].

Keras necesita un conjunto estructurado de datos en un array de 2 dimensiones. La primera dimensión de la matriz contiene una lista de números de la sección IV-C, denominada X. La segunda dimensión representa el tipo de malware con valores que van de 0 a 5, como se define en la Tabla III. Esta dimensión se denomina Y.

Para el entrenamiento y la validación, el conjunto de datos se dividió en matrices bidimensionales X e Y con un 90 % utilizado para el entrenamiento y un 10 % para la prueba, insertados en dos nuevas variables, *X_prueba* y *Y_prueba*.

Tabla III
LISTA DE TIPOS DE MALWARE EN FORMA DE NÚMEROS

Id	Malware
0	Downloader
1	keylogger
2	Miner
3	Ransomware
4	RootKit
5	Spyware

Para utilizar el modelo LSTM en Keras, se recomienda crear previamente una capa Embedding. Para ello, necesitamos extraer información específica del modelo word2vec, como el tamaño del vocabulario (que se refiere al número de funciones diferentes extraídas del binario), la dimensión de la capa densa (que se calcula multiplicando el número de binarios diferentes analizados por el número máximo de funciones por binario) y la lista de pesos de todos los números (que se genera en función de la frecuencia de cada función).

Las entradas de Word2vec extraídas se pasaron a la capa Embedding, seguida de una capa LSTM con un dropout de 0,7 y un dropout recurrente de 0,2.

Después de la capa LSTM, se añade una capa oculta con la función de penalización "ReLU" seguida de otra capa oculta con función de penalización regularizadora para ajustar la capa de salida.

Se ha añadido al modelo una capa de salida que utiliza la función softmax. Los modelos de clasificación multiclase suelen utilizar esta función. La estructura completa se muestra en la Figura 3.

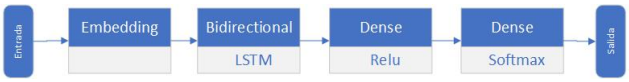

Figura 3. Modelo LSTM

Se utilizó el análisis de entropía cruzada binaria como función de pérdida, ya que se suele emplear en problemas de multiclasificación. El optimizador seleccionado fue Adam, un método de descenso de gradiente estocástico basado en la estimación adaptativa, muy utilizado en este tipo de análisis. Para medir la tasa de éxito del modelo se utilizó la precisión como métrica.

El modelo se entrenó proporcionando datos de entrada como matrices X e Y, épocas (interacción sobre la totalidad de X e Y proporcionadas) como 100, tamaño de lote (número de muestras por actualización de gradiente) como 200, y X_test e Y_test para la prueba y para calcular la precisión del modelo.

V. RESULTADOS

Esta sección se divide en tres partes para explicar con más precisión los resultados de la investigación: la lista de las herramientas utilizadas y sus versiones, los resultados del modelo entrenado, la estructura de automatización del método propuesto, una prueba práctica con muestras reales como entrada al modelo y una comparación con trabajos relacionados.

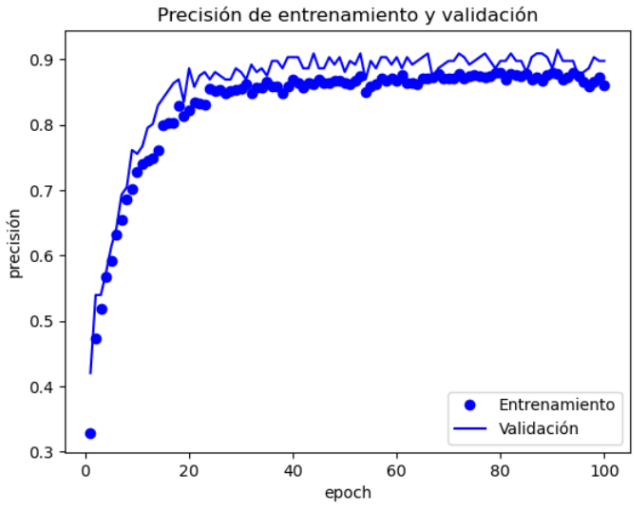

Figura 4. Precisión de Entrenamiento y Validación

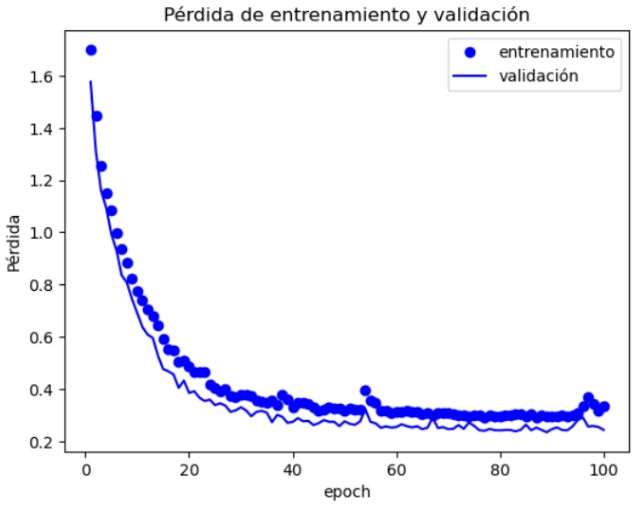

Figura 5. Perdida de Entrenamiento y Validación

V-A. *Herramientas y versiones*

Las herramientas y versiones utilizadas para construir el modelo LSTM, la estructura de automatización y la herramienta de desmontaje se enumeran en la Tabla IV.

Tabla IV
HERRAMIENTAS Y VERSIONES UTILIZADAS EN ESTE TRABAJO

Modelo LSTM	**Estructura de automatización**	**Herramienta binaria de Desensamblaje**
Python 3.10+	Docker 20.10.21	VS 2019
TensorFlow 2.12.0+	Python 3.10	MSVC 14.38.33130
Keras 2.9.0+	Tensorflow 2.15	
Gensim 4.1.2+	Keras 2.15.0	
Jupyter notebook 6.4.12+	Django 5.0.3	
	Objdump 2.4	
	Gensim 4.3.2	

V-B. *Resultados del modelo entrenado*

La principal métrica utilizada para medir el modelo entrenado fue la precisión. El proceso de entrenamiento alcanzó una precisión del 91 %, y la tasa de pérdida, obtenida a partir del cálculo de la entropía cruzada binaria, de alrededor de 0,3. La Figura 4 muestra la tasa de precisión por época, mientras que la Figura 5 muestra la tasa de pérdida por época.

Los resultados obtenidos mediante el entrenamiento de los modelos se aproximaron mucho a los resultados esperados por los validadores como se puede observar en la Figura 4 y la comparación de pérdidas del modelo entrenado y su validación en la Figura 5.

Para comprender mejor los resultados del modelo construido, la figura presenta la matriz de confusión del modelo entrenado. Las clases se basan en la información de la Tabla III, y muestra el número de aciertos y errores teniendo en cuenta el número de clases en que se basó el conjunto de datos.

El resultado que presenta la matriz de confusión es una escala de 0 a 5 en los ejes X e Y, que representa los valores de malware especificados en la Tabla III. El valor de los bloques varía de 0 a 30, lo que significa el número de veces que se predijo que un malware pertenecía a la clase corregida o a otra. Por ejemplo, con X=0 e Y=0, se predijo que el malware

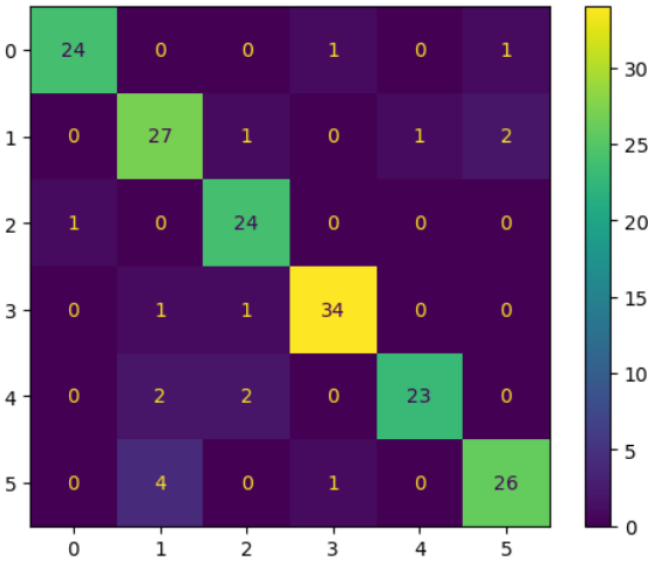

Figura 6. Matriz de confusión

era un Downloader, y de hecho, lo era. Según esta métrica, cuanto mayor sea el valor, mayor será la posibilidad de una predicción correcta utilizando un conjunto de datos similar.

Tras analizar los datos, se puede concluir que los miners y los spyware tienen más probabilidades de ser detectados en comparación con un conjunto de datos similar. Por el contrario, el ransomware y los rootkits tienen la tasa de acierto más baja.

V-C. *Estructura de automatización*

Para automatizar el análisis proporcionado por los modelos creados, incluyendo word2vec y LSTM, se creó una estructura de contenedores en Docker. Se utilizó el servidor web Django para crear un endpoint de API REST. La estructura se presenta en la Figura 7.

Esta automatización considera que la API desarrollada puede tomar un archivo binario en la estructura PE32 y someterlo a un proceso de desensamblado, utilizando objdump, para

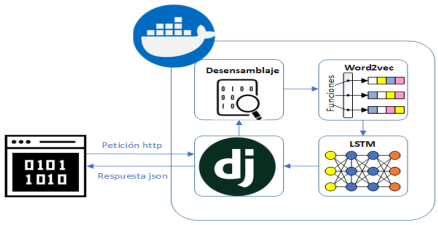

Figura 7. Estructura de automatización

extraer las funciones utilizadas para comunicarse con las DLL. A continuación, estas funciones se estructuran en un formato de matriz de cadenas y se traducen utilizando el mismo modelo word2vec utilizado para crear el modelo primario.

A continuación, el framework carga el modelo LSTM creado, junto con su función de predicción, que recibe la matriz numérica para comprobar con qué estructura de malware se clasifica la matriz. Tras el análisis, la aplicación responderá a la llamada a la API REST en formato JSON con el tipo de malware resultante.

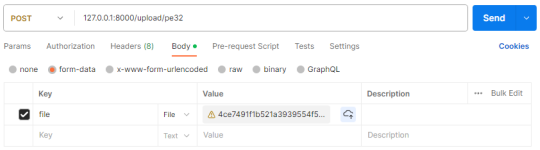

Figura 8. Ejemplo de solicitud de Postman a la API

La estructura se puede levantar fácilmente con un comando Docker, y es posible categorizar los binarios al instante. La figura 8 muestra el uso de Postman para enviar un binario para su análisis adjuntándolo al cuerpo de la petición.

Una vez validado el binario, la aplicación responde con la categoría más adecuada. La Figura 9 muestra un ejemplo de la respuesta, que fue Ransomware.

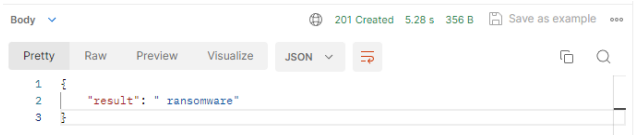

Figura 9. Ejemplo de respuesta a solicitud de postman desde la API

V-D. *Prueba práctica con muestras reales*

Se envió un conjunto de muestras de malware no utilizadas en el entrenamiento ni en las pruebas a través de la API REST para probar el proceso de categorización automatizado. Los detalles del malware como hash, resultado esperado y resultado obtenido se pueden encontrar en la Tabla V.

La herramienta acertó 3 de 6, con ocho opciones posibles. El porcentaje medio de aciertos fue del 50 % en esta pequeña prueba con datos precisos. Este resultado se considera bajo en términos de precisión, pero es esencial tener en cuenta que la herramienta está en fase de desarrollo y aún no está lista para la producción. Pero también es importante que la herramienta se someta a una prueba práctica en escenarios reales.

Esta estructura fácil de usar viene con categorización automatizada y se puede integrar perfectamente con la estructura de verificación de pipelines a través de APIs bien conocidas. Puede ser un componente crucial de una solución de problemas del mundo real mediante el análisis de archivos en estructura PE32 antes de ponerlos a disposición [29] de los usuarios y para hacer pruebas dinámicas de seguridad de aplicaciones (DAST) [30] análisis.

Tabla V
PRUEBAS REALIZADAS CON MALWARE REAL

Hash	Esperado	Obtenido
fe93d59439b3ac1df36215872d6daf16c68a38eba8bb05b892108d0a13ac4d7b	downloader	miner
c199782b8aa80fa9f4bed0d2fa0b5befdf8df070feb2acf2b967ae8752e04cab	keylogger	spyware
ff6b9d9786c4505bd80d5156a083648e16ec4a821d89d0fe1687080f3bed60be	miner	miner
ffc338c0c7be278245747c53266b86517c3fd3ccff8a243d9537ffc0afeb9d2c	ransomware	ransomware
000b2d3b52831632a410ec905de2e97499f4f0fa867efdcab31194f3952b1d6f	rootkit	rootkit
f43a19ef1e0e2d0b727f34629140e50f35522269e8e094729a65a15c6188965f	spyware	downloader

V-E. *Comparación de trabajos relacionados*

Visweswaran *et al.* [23] analizaron archivos PE32 mediante aprendizaje profundo, que es el trabajo que más se acerca a los resultados de esta investigación. Su modelo alcanzó una tasa de precisión del 96 %, mientras que nuestro modelo alcanzó el 91 % en las primeras pruebas. Ellos utilizaron CNN, mientras que este trabajo utilizó LSTM (una versión mejorada), y su conjunto de datos era más significativo que el incorporado en este trabajo, lo que repercutió en el mapeo de patrones. Sin embargo, este trabajo utilizó word2vec, lo que significa que tiene el mismo diccionario que el utilizado para crear el modelo. Esto permite a cualquier interesado utilizar el mismo proceso para verificar patrones de un nuevo binario analizado, mientras que el trabajo de Visweswaran et al. necesita reconstruir todo el proceso para verificar un patrón binario. Por lo tanto, es difícil hacer posible un proceso espontáneo y automatizado. Consulte la Tabla VI para obtener una visión general.

Tabla VI
PROPUESTA ACTUAL VS. TRABAJO DE VISWESWARAN

	Propuesta actual	Trabajo de Visweswaran
Algoritmo	LSTM	CNN
Clasificación	Multi-class	Multi-class
Precisión	91 %	96 %
Recuento de datos	1200	10000
Epoch	100	100
String para número	Word2vec	Tokenizer
Reproducible	Yes	No

VI. CONCLUSIONES Y TRABAJO FUTURO

Basándose en las características del método propuesto, que tiene como objetivo acelerar la detección de malware en estructura PE32, es posible: a) automatizar las técnicas de desensamblado y utilizar su salida como entrada para automatizar el análisis de malware; b) considerar el análisis léxico con word2vec como una técnica prometedora con sus parámetros para realizar análisis de malware ML; c) entender que el análisis de aprendizaje profundo con LSTM ha logrado resultados significativos en la categorización de malware.

Como los resultados de la aplicación consideran pipelines seguras, la estructura de análisis automatizado recién construida funciona como un framework. Permite la combinación e integración con soluciones de categorización de malware. Dispone de un formato de API bien conocido que permite una comunicación e integración accesibles con otras soluciones.

La infraestructura desarrollada también tiene el potencial de apoyar el análisis en tiempo real de archivos en estructura PE32 en escenarios del mundo real, ayudar en pipelines complejos destinados a automatizar el proceso de despliegue, y puede ser muy bien utilizada en DevSecOps. Los resultados primarios tuvieron un bajo índice de éxito. Sin embargo, la propuesta de análisis es prometedora.

Como trabajo futuro, el siguiente paso sería ampliar el conjunto de datos existente para mejorar la precisión del modelo. Además, es necesario crear una nueva rama que primero categorice los binarios PE32 como maliciosos o no maliciosos utilizando un clasificador binario y, a continuación, se utilice el modelo actual para categorizar su tipo. Otra posible vía es construir una estructura de análisis automatizado a partir del nuevo modelo de categorización y aplicarlo en situaciones del mundo real, como pipelines de verificación, para integrarlo con el análisis DAST y sandbox.

VII. Agradecimientos

Este trabajo se ha realizado con los fondos del Plan de Recuperación, Transformación y Resiliencia, financiados por la Unión Europea (Next Generation), a través de la Cátedra "Ciberseguridad para la Innovación y la Protección Digital" INCIBE-UCM.

This work was also supported by Horizon Europe Programme, as part of the project LAZARUS (https://lazarus-he.eu/) (Grant Agreement no. 101070303). The content of this article does not reflect the official opinion of the European Union. Responsibility for the information and views expressed therein lies entirely with the authors.

Referencias

[1] Virustotal, "Virus total statistics. available on: https://www.virustotal.com/gui/stats," 2024.

[2] M. Egele, T. Scholte, E. Kirda, and C. Kruegel, "A survey on automated dynamic malware-analysis techniques and tools," *ACM computing surveys (CSUR)*, vol. 44, no. 2, pp. 1–42, 2008.

[3] F. Pagani, M. Dell'Amico, and D. Balzarotti, "Beyond precision and recall: understanding uses (and misuses) of similarity hashes in binary analysis," in *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy*, 2018, pp. 354–365.

[4] K. Yakdan, S. Dechand, E. Gerhards-Padilla, and M. Smith, "Helping johnny to analyze malware: A usability-optimized decompiler and malware analysis user study," in *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2016, pp. 158–177.

[5] S. Iqbal, A. Ullah, S. Adlan, and A. R. Soobhany, "Malware prediction using lstm networks," in *Proceedings of International Conference on Information Technology and Applications: ICITA 2021*. Springer, 2022, pp. 583–604.

[6] J. A. Morales, A. Al-Bataineh, S. Xu, and R. Sandhu, "Analyzing and exploiting network behaviors of malware," in *Security and Privacy in Communication Networks: 6th Iternational ICST Conference, SecureComm 2010, Singapore, September 7-9, 2010. Proceedings 6*. Springer, 2010, pp. 20–34.

[7] M. Pietrek, "An in-depth look into the win32 portable executable file format, part 2," *MSDN Magazine, March*, 2002.

[8] R. Tahir, "A study on malware and malware detection techniques," *International Journal of Education and Management Engineering*, vol. 8, no. 2, p. 20, 2018.

[9] C. Rossow, C. Dietrich, and H. Bos, "Large-scale analysis of malware downloaders," in *Detection of Intrusions and Malware, and Vulnerability Assessment: 9th International Conference, DIMVA 2012, Heraklion, Crete, Greece, July 26-27, 2012, Revised Selected Papers 9*. Springer, 2013, pp. 42–61.

[10] A. Singh, P. Choudhary *et al.*, "Keylogger detection and prevention," in *Journal of Physics: Conference Series*, vol. 2007, no. 1. IOP Publishing, 2021, p. 012005.

[11] Z. Salehi, M. Ghiasi, and A. Sami, "A miner for malware detection based on api function calls and their arguments," in *The 16th CSI international symposium on artificial intelligence and signal processing (AISP 2012)*. IEEE, 2012, pp. 563–568.

[12] S. Kim, J. Park, K. Lee, I. You, and K. Yim, "A brief survey on rootkit techniques in malicious codes." *J. Internet Serv. Inf. Secur.*, vol. 2, no. 3/4, pp. 134–147, 2012.

[13] M. Egele, C. Kruegel, E. Kirda, H. Yin, and D. Song, "Dynamic spyware analysis," in *Proceedings of the 2007 USENIX Annual Technical Conference*. Carnegie Mellon University, 2007.

[14] A. Handler, *An empirical study of semantic similarity in WordNet and Word2Vec*. Columbia University, 2014.

[15] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," *Advances in neural information processing systems*, vol. 26, 2013.

[16] X. Rong, "word2vec parameter learning explained," *arXiv preprint arXiv:1411.2738*, 2014.

[17] J. Heaton, "Ian goodfellow, yoshua bengio, and aaron courville: Deep learning: The mit press, 2016, 800 pp, isbn: 0262035618," *Genetic programming and evolvable machines*, vol. 19, no. 1-2, pp. 305–307, 2018.

[18] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[19] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with lstm," *Neural computation*, vol. 12, no. 10, pp. 2451–2471, 2000.

[20] Y. Dai, H. Li, Y. Qian, and X. Lu, "A malware classification method based on memory dump grayscale image," *Digital Investigation*, vol. 27, pp. 30–37, 2018.

[21] Y. J. Lee, S.-H. Choi, C. Kim, S.-H. Lim, and K.-W. Park, "Learning binary code with deep learning to detect software weakness," in *KSII the 9th international conference on internet (ICONI) 2017 symposium*, 2017.

[22] L. Wartschinski, "Detecting software vulnerabilities with deep learning," Ph.D. dissertation, Master's thesis, Humboldt University, Berlin, 2014.

[23] N. Visweswaran, M. Jeevanantham, C. T. Kani, P. Deepalakshmi, and S. Sathiyandrakumar, "Automated pe32 threat classification using import table and deep neural networks," in *2019 IEEE International Conference on Clean Energy and Energy Efficient Electronics Circuit for Sustainable Development (INCCES)*. IEEE, 2019, pp. 1–5.

[24] Virushare, "virushare: opened malware share repository. available on: https://virusshare.com," 2024.

[25] Jovibor, "libpe: github library created by jovibor/libpe. available on: https://github.com/jovibor/libpe," 2024.

[26] K. Hahn and I. Register, "Robust static analysis of portable executable malware," *HTWK Leipzig*, vol. 134, 2014.

[27] R. Rehurek and P. Sojka, "Gensim – statistical semantics in python," in *NLP Centre, Faculty of Informatics, Masaryk University, Brno, Czech Republic*, 2011. [Online]. Available: https://www.fi.muni.cz/usr/sojka/posters/rehurek-sojka-scipy2011.pdf

[28] F. Chollet, *Deep learning with Python*. Simon and Schuster, 2021.

[29] N. Loi, C. Borile, and D. Ucci, "Towards an automated pipeline for detecting and classifying malware through machine learning," *arXiv preprint arXiv:2106.05625*, 2021.

[30] Y. Pan, "Interactive application security testing," in *2019 International Conference on Smart Grid and Electrical Automation (ICSGEA)*. IEEE, 2019, pp. 558–561.