



UNIVERSIDAD DE CASTILLA-LA MANCHA

ESCUELA SUPERIOR DE INGENIERÍA INFORMÁTICA

GRADO EN INGENIERÍA INFORMÁTICA

TECNOLOGÍA ESPECÍFICA DE COMPUTACIÓN

TRABAJO FIN DE GRADO

**Detección y Clasificación de Malware usando técnicas
inteligentes**

Rubén Donate Serrano

Junio de 2018





UNIVERSIDAD DE CASTILLA-LA MANCHA

ESCUELA SUPERIOR DE INGENIERÍA INFORMÁTICA

GRADO EN INGENIERÍA INFORMÁTICA

TECNOLOGÍA ESPECÍFICA DE COMPUTACIÓN

TRABAJO FIN DE GRADO

Detección y Clasificación de Malware usando técnicas
inteligentes

Autor: Rubén Donate Serrano

Directores: José Luis Martínez Martínez
José Miguel Puerta Callejón

Junio de 2018

La dedicatoria que quiera hacer.

Declaración de Autoría

Yo, Rubén Donate Serrano con DNI 70519349S, declaro que soy el único autor del trabajo fin de grado titulado Detección y Clasificación de Malware usando técnicas inteligentes y que el citado trabajo no infringe las leyes en vigor sobre propiedad intelectual y que todo el material no original contenido en dicho trabajo está apropiadamente atribuido a sus legítimos autores.

Albacete, a 23 de junio de 2018

Fdo.: Rubén Donate Serrano

Resumen

Este sería el resumen del TFG.

Agradecimientos

Agradezco el apoyo de mi familia amigos y de mis tutores.

Índice general

ÍNDICE DE FIGURAS	XI
Lista de Figuras	XIII
ÍNDICE DE TABLAS	XIII
Lista de Tablas	1
1. INTRODUCCIÓN	1
1.1. Motivación	1
1.2. Objetivos	1
1.3. Estructura de la memoria	1
2. TÉCNICAS UTILIZADAS	3
3. ANTECEDENTES Y ESTADO DE LA CUESTIÓN	7
3.1. Solución 1	7
3.2. Solución 2	9
4. METODOLOGÍA Y DESARROLLO	23
5. EXPERIMENTOS Y RESULTADOS	35
6. CONCLUSIONES Y PROPUESTAS	37
6.1. Conclusiones	37
6.2. Trabajo futuro	37
BIBLIOGRAFIA	41

ÍNDICE DE FIGURAS

3.1. Lista de instrucciones ASM manejadas.	10
3.2. Lista de secciones ASM manejadas.	12

ÍNDICE DE TABLAS

4.1. Clase de malware en la base de datos train.	26
--	----

Capítulo 1

INTRODUCCIÓN

1.1. Motivación

1.2. Objetivos

1.3. Estructura de la memoria

Capítulo 2

TÉCNICAS UTILIZADAS

1. ngram
2. sklearn
 - a)* randomForrestClassifier
 - b)* DictVectorizer
 - c)* TF-IDF [\[1\]](#)
 - d)* NMF [\[2\]](#) [\[3\]](#)
 - e)* Linear Support Vector Classification [13](#)
 - f)* KFold
3. numpy
 - a)* Concatenate
4. pickle
5. cPickle
6. glob
7. subprocesss
8. funciones linux:
 - a)* echo
 - b)* cat
 - c)* wc
 - d)* grep
9. pandas DataFrame

a) read_csv

b) concat

c) to_csv

10. Matriz Dispersa

a) csrMatrix

11. os

a) path

12. joblib

a) parallel

b) delayed

13. LibLinear: Una librería para completa clasificación lineal [4]

14. hickle

15. XGBoost [5]

16. Gradient Boost Tree

17. logloss

18. Dual Optimization Problem [6]

19. Random

a) Sample

b) Choice

20. Bagging

21. Scipy

a) Stats

1) rv_discrete

22. MongoDB

23. NoSQL

24. JSON

25. BSON

26. Docker

27. re

Capítulo 3

ANTECEDENTES Y ESTADO DE LA CUESTIÓN

Para el proyecto de Kaggle que se ha seleccionado para este Trabajo de Final de Grado, se he realizado una búsqueda para intentar encontrar soluciones presentadas para este reto. De esta búsqueda, se ha conseguido encontrar varias soluciones presentadas para este mismo reto. A continuación, se explicaran las soluciones presentadas que se han encontrado para este reto de Kaggle.

3.1. Solución 1

Esta primera solución que se va a comentar, fue realizada por Vishnu Chevli [7] y en la clasificación del Reto de Kaggle obtuvo un resultado en el ranking publico de 0,023121984 en la posición 90 y en el ranking privado 0,018856579 en la posición 72.

Esta solución se desarrollo para que pudiera ser ejecutada de manera indistinta en Python 2 y Python 3, es muy sencilla y consiste en:

En primero lugar, se descomprimir las dos bases de datos que se proporcionan. Estas bases de datos contienen dos tipos de ficheros que contienen la información obtenida de IDA al desensamblar la muestras en dos formatos, siendo estos formatos ASM para los ficheros con extensión .asm y binario en formato hexadecimal para los ficheros con extensión .bytes. Para esta solución solo se utilizaran los ficheros con extensión .bytes. Después, estos ficheros son nuevamente comprimidos en formato gzip de manera separada y separando las bases de datos en dos carpetas distintas. El motivo de realizar esto es para ahorrar espacio en disco, ya que el espacio de la base de datos completas con todos los tipos de ficheros sin comprimir es de aproximadamente 1 Tb.

El siguiente paso, consiste en extraer las características de los ficheros que se han generado en el paso anterior. Esta extracción consiste en generar un array donde cada posición corresponde a cada uno de los posibles uno ngrama 1, formado por dos caracteres en formato hexadecimal, incluyendo el ngram '??' que indica que el valor correspondiente no esta mapeado, y contar las veces que aparece cada uno de ellos en el fichero, para concluir, guardando los resultados en un fichero csv de manera separada para cada una de las bases de datos. Este proceso ser realizara de manera conjunta para las dos bases de datos y de manera paralela para 2 ficheros.

El siguiente paso, es construir el modelo que para este caso es un RandomForrest-Classifer 2a. Para ello, lo primero se tiene que realizar es obtener la clase para cada uno de los fichero, para lo cual se tiene que abrir y recorrer el fichero 'trainLabels.csv' que se proporciona y almacenar la información del mismo en un diccionario que posteriormente se utilizará. A continuar, se crea una matriz con numpy 3 de tamaño el numero de ficheros de la base de datos de train por el número de uno ngramas 1 mas uno adicional para la clase, en este caso $10868 * 258$. Estos datos son obtenidos del fichero csv correspondiente a la base de datos de train para los valores de los uno ngramas 1 y del diccionario creado anteriormente para la clase. Después, se crea el modelo con los valores por defecto, excepto la semilla que utiliza 123, el número de hilos que utiliza 5 y el nivel de información que muestra que utiliza la mas detallada 2. Por ultimo, solo queda entrenar el modelo pasando le la matriz de numpy 3 separando las características y la clase, o lo que es lo mismo la matriz sin la ultima columna y la ultima columna por separado, siendo esta ultima columna la clase.

Para terminar, solo nos queda realizar la previsión para la base de datos de test. Para ello, ahí que repetir el proceso de recuperar las características creando una matriz con numpy 3 pero en este caso sin añadir un columna para la clase, también se crea un array donde se almacena el nombre del fichero en el mismo indice que la fila de la matriz, esto es así porque la predicción ha de ser identificada con es valor. Lo siguiente, es realizar la predicción para la base de datos de test. Para concluir, escribiendo los resultados en un fichero comprimido en el que guarda la información con el formato que se nos proporciona en el ejemplo, siendo este el id del fichero seguido de la probabilidad que sea cada una de las clases separado todo por comas.

3.2. Solución 2

Esta segunda solución que se va a comentar, fue realizada de manera conjunta por Mikhail Trofimov, Dmitry Ulyanov y Stanislav Semenov [8] y en la clasificación del Reto de Kaggle obtuvo un resultado en el ranking público de 0,005984430 en la posición 14 y en el ranking privado 0,003969846 en la posición 3.

Esta solución es mucho más compleja como se puede suponer de la posición que obtuvo en el ranking, de hecho según sus creadores utilizaron una máquina con 16 cores y 120 Gb de RAM para su ejecución. Ahora, se analizará en qué consiste esta solución:

Lo primero que realizar es crear los directorios, para ello existe un script en bash llamado `'create_dirs.sh'` para generar los directorios necesarios. A continuación, toca ejecutar otro script en bash llamado `'main.sh'` para ejecutar esta solución, siendo el primer paso, la extracción de las características de los ficheros con extensión `'.asm'` que posteriormente se van a utilizar para construir el modelo. Pero antes de eso, ahí que establecer en el fichero llamado `'set_up.py'` los parámetros fijos para esta solución en variables, como pueden ser el número de hilos, y las distintas direcciones de las carpetas con las que trabaja la solución. A continuación, se comentará este proceso de extracción que consisten en:

El primer paso en el proceso de extracción de características de los ficheros con extensión `'.asm'`, consiste en contar las veces que aparece cada una de las secciones en cada uno de las muestras de las bases de datos y la suma total del número de ocurrencias de cada sección para cada muestra. Para llevar a cabo esto, se recorre cada fichero con extensión `'.asm'`, siendo pasado el nombre de éste por parámetro a la función, y de cada línea, se tiene que coger la primera palabra, estando esta delimitada con dos puntos, y se mantiene en un diccionario el conteo de ocurrencias que aparece cada una de las secciones y el número total de ocurrencias de cada sección. Para proporcionar persistencia y no tener que almacenar toda la información en memoria, se guarda en un archivo serializado con el paquete `pickle` [4](#) el diccionario obtenido para ese fichero. Para obtener el nombre del fichero que se le pasa para hacer la extracción, se consigue la dirección completa de los ficheros con extensión `'.asm'` en cada una de las carpetas donde se almacenan las bases de datos, usando para esto la función `glob` del paquete `glob` [6](#). Posteriormente, a los elementos de esta lista se les aplica una función `map` para dejar solo el nombre del fichero que es el identificador de la muestra. Para terminar, se crea una lista de procesos cuya longitud es el número de hilos establecidos. Esto es así para realizar el procesamiento de las muestras de manera paralela. A estos procesos, se les pasa una sublista de los ficheros y ejecuta una función `worker` que se encarga de

procesar cada uno de los elementos de esa sublista y realizar la extracción comentada anteriormente. La manera de formar estas sublista es mediante el resto del índice de la posición en la lista original entre el número de hilos que se ejecutan, siendo este el índice del proceso en el cual la muestra tiene que formar parte de su sublista.

El segundo paso en el proceso de extracción de características en los ficheros con extensión '.asm', consiste en llevar un conteo de las líneas que posee el fichero de la muestra y en cuantas de estas, aparece los elementos de un conjunto establecido de instrucciones asm. Este conjunto de instrucciones que se utiliza en la solución, se puede observar en la figura 3.1, está formado por 23 instrucciones ASM, siendo estas las mas habituales, almacenadas en una lista de strings.

```
specter = ['jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'nop', 'sub', 'inc', 'dec', 'add',  
          'imul', 'xchg', 'or', 'shr', 'cmp', 'call', 'shl', 'ror', 'or', 'rol', 'jnb']
```

Figura 3.1: Lista de instrucciones ASM manejadas.

Se comienza por almacenar en el fichero llamado 'fnames' el nombre de la muestra, usando la función echo 8a de linux, que se le pasa a la función, siendo este nombre generado de la misma manera que en el paso anterior. Realizarlo de esta manera implica que no sea posible la paralelización de este proceso, ya que la línea dentro de cada fichero que se va a utilizar establece la muestra a la que se hace mención. Se continúa realizando el conteo de las líneas del fichero con extensión '.asm' de la muestra. Para llevar a cabo el proceso de conteo, se utiliza la función call del paquete subprocess 7 para de esa manera, utilizar las utilidades de linux. En este caso, para contar el número total de líneas del fichero de muestra se utiliza la función cat 8b para recorrer el y se le pasa esa información mediante una tubería a la utilidad wc 8c con el parámetro '-l', la cual cuenta las líneas que se le han pasado y añade esa información en un fichero llamado 'line_count' en la posición correspondiente para esa muestra. A continuación, se realizará el conteo de las líneas en las que aparece cada una de las instrucciones en esa muestra. Para conseguir este objetivo, hace uso de la función grep 8d para seleccionar las líneas dentro de la muestra que contiene la instrucción correspondiente, para posteriormente pasárselo mediante una tubería a la función wc 8c, igual que en el paso anterior, y añadir la información en el fichero correspondiente a la instrucción en la posición que le corresponde. Este paso se repite, para cada una de las instrucciones establecidas. En este proceso, se realiza de manera serializada para todos los ficheros de la base de datos y en el caso de ser el primer fichero y no existir los ficheros todos ellos son generados cuando se añade el primer elemento.

El siguiente paso en el proceso de extracción de características en los ficheros con

extensión '.asm', consiste en seleccionar de la muestra las líneas en las que aparece la cadena de caracteres '__stdcall' y almacenarlas en un fichero por cada muestra. Para llevar esto a cabo, hace uso de la función `call` del paquete `subprocess` 7 y de la función `grep` 8d, igual que en el paso anterior, pero en esta ocasión se almacena por nombre de la muestra, la cual se le pasa a la función del mismo modo que en los pasos anteriores. Esto implica que en este caso si sea posible paralelizar el proceso de extracción de características, por lo cual se realiza de la misma manera que se llevo a cabo en el primer paso, creando una sublista que procesara un proceso para todas las muestras de las bases de datos.

Por ultimo en el proceso de extracción de características en los ficheros con extensión '.asm', consiste en seleccionar de las muestras las líneas en las que aparece la cadena de caracteres 'FUNCTION'. Como se puede observar, consiste en repetir el proceso anterior una nueva cadena de caracteres.

Una vez terminado el proceso de extracción de características de los ficheros con extensión '.asm', toca preprocesar la información obtenida de los ficheros con extensión '.asm'. Para ello, lo primero que se realiza es obtener dos dataframes a partir de los ficheros 'trainLabel.csv' y 'sampleSubmission.csv', los cuales son proporcionados en el reto, con `pandas` 9. Éstos contienen los identificadores junto con otra información de las muestras de las dos bases de datos `train` y `test` respectivamente, y se usarán para que durante el proceso de preprocesado todas las características obtenidas de las muestras guarden el mismo orden, hecho por el cual esto implica que este procedimiento no puede ser paralelizado. Además, los procesos que se comentara a continuación se repite para estos dos dataframes. Este procedimiento consiste en:

En primer lugar, se tratan las secciones obtenidas, para ello se recorre el dataframe fila a fila, y para el campo `Id` se abre el fichero de las secciones calculado correspondiente con el paquete `cPickle` 5, el cual se trata de un diccionario en python, por lo cual se recorre y si pertenece a un conjunto establecido, el cual se puede observar en la figura 3.2, se incluye en otro diccionario, para terminar guardándolo en la posición del array que viene establecida por el dataframe. En este primer paso, cuando se trata del dataframe de la base de datos de `train` también se crea un array, en el cual se almacena la etiqueta correspondiente de la muestra en el índice establecido por el dataframe. Para continuar, transformando el array de diccionarios en una matriz, para esto usa el modelo `DictVectorizer` 2b del paquete `sklearn` 2 con el parámetro `sparse` a `false` para que no almacene los datos en una matriz dispersa 10. Este modelo se entra con el array generado con el dataframe de la base de datos de `train` y posteriormente se transforman los dos arrays en su correspondiente matriz. Con estas matrices se procede a calcular

la entropía correspondiente a cada una de ellas. Terminando así el tratamiento de las secciones.

```
section_whitelist = set(['.bss', '.data', '.edata', '.idata', '.rdata', '.reloc',
                        '.rsrc', '.text', '.tls', 'bss', 'code', 'data', 'header'])
```

Figura 3.2: Lista de secciones ASM manejadas.

En segundo lugar, consiste en obtener el tamaño de los archivos con extension '.asm' y '.bytes' para después obtener también el ratio entre estos. De este modo, se crea una matriz con la longitud del dataframe y dos columnas una para cada fichero, donde se almacena con la ayuda de la función `getsize` del paquete `os.path` [11a](#) el tamaño estos correspondientes a la muestra en la posición establecida por el dataframe. A partir de esta matriz, se genera otra nueva con una sola columna, la cual es el resultado de hacer la división de float, ya que para garantizarlo se multiplica en numerador por 1,0, del tamaño obtenido del fichero con extensión '.bytes' entre el tamaño obtenido del fichero con extensión '.asm', concluyendo así este paso.

En tercer lugar, consisten en almacenar la información obtenida de las instrucciones asm en una matriz de numpy y el calculo de la entropía de la misma. Para ello, primero se tiene que abrir y recorrer el fichero 'fnames' que se genero en el proceso de extracción de características para almacenar en un diccionario donde se almacena en el valor de indice la muestra se almacena un diccionario con los valores obtenidos para las instrucciones manejadas, vease la figura [3.1](#), para esa muestra. Una vez se recuperado la información, se procede a almacenar la en una matriz de numpy conforme se va haciendo, donde el indice que el dataframe establece la fila y en indice del array de figura [3.1](#) indica la columna. Una vez realizado esto se procede se calcula la entropía de cada característica. Este proceso se repite para los dos dataframe y con esto se termina este paso.

El siguiente paso, consiste recuperar la información del número de lineas de los ficheros con extensión '.asm' y almacenarlo en un diccionario. Esto se realiza de este modo debido a que se tiene que recorrer el fichero 'fnames' para obtener el Id de valor almacenado en el fichero 'line_count'. Una vez hemos recuperado la información se procede a almacenarla en una matriz con una sola columna en el orden establecido y por el dataframe y se calcula la entropía para todos los valores de la misma. Este proceso se repite de igual manera para los dos dataframe, terminando lo así.

A continuación, toca obtener la información de las llamadas a las funciones, estando estas en las lineas seleccionadas en las que aparecía la cadena '__stdcall' almacenadas

en el proceso anterior. Para esto, se procede a recorrer los ficheros obtenidos anteriormente en el orden establecido por el dataframe y en un primer paso, partir esa línea para obtener el nombre de la función para a continuación concatenar los y almacenarlos en un string separadas por espacios para cada muestra, el cual es almacenada en un array en la posición establecida por el dataframe. Una vez obtenido este array para los dos dataframe, se procede a crear un `TfidfVectorizer` [2c](#) mediante el cual se obtienen como máximo las 10000 características mas significativas según este modelo. Para ello, se entrena el modelo con la unión de los dos array para posteriormente, transformar estos una matriz dispersa [10](#) con la probabilidad de como máximo las 10000 características mas significativas para cada una de las muestras. Una vez obtenida esta matriz, se le realiza una factorización de matriz no negativa [2d](#), para de ese modo concluir obteniendo las 10 características mas representativas representativas. Por lo cual, se construye en modelo NMF para 10 componentes y se entrena con una matriz dispersa [10](#), consistente en la unión de las dos matriz obtenidas en el paso anterior, para concluir transformado cada una de de las matrices y así obtener una matriz dispersa con la probabilidad factorizada de las características de las dos bases de datos.

Para terminar con el proceso de preprocesado de los fichero con extensión `'.asm'`, solo nos queda tratar las llamadas al sistema, siendo las líneas seleccionados en las que aparecen la cadena `'FUNCTION'`. Este proceso es idéntico al llevado a cabo en el paso anterior salvo que se cogen los nombres de las funciones en la que en la línea aparece la palabra `'PRESS'`.

Para concluir, se almacena en una matriz de numpy donde se incluye en columnas la información preprocesada de la secciones, el ratio del tamaño, las instrucciones, el tamaño de los ficheros, la entropía de las secciones, las características obtenidas de las llamadas a las funciones y por ultimo las características de las llamadas la sistema. Este proceso se repite de igual manera para las dos bases de datos. Por ultimo se almacena en un fichero serializado con joblib [12](#) llamado `'X_basepack'` guardando las dos matrices en una tupla. Con esto se termina el proceso de preprocesado de las características de los ficheros con extensión `'.asm'`.

Llegados a este punto, el siguiente paso consiste en extraer las características de los ficheros con extensión `'.bytes'`. Este paso comienza por, crear un diccionario mediante el cual traducir un n-grama de dos caracteres en hexadecimal a decimal. A continuación, se abre el fichero de la muestra y cada línea se transforma en un array teniendo como delimitador el carácter espacio en blanco `' '`. De los arrays generados, se selecciona todos los elementos salvo en primero, que corresponde con la dirección en memoria, para a continuación transformar a decimal los n-gramas con el diccionario generado en

le primer paso, y concatenar todos en un nuevo array donde se encuentras todos los n-gramas en formato decimal del fichero ordenados. Una vez tenemos este ultimo array, lo recorremos en grupos de 4 n-gramas, para continuar por transformar el 4 n-grama en un índice número y guardar en un diccionario las ocurrencias de estos índices en el fichero. Para concluir, transformando la información almacenada en diccionario en dos array, uno para los índices y otro para las ocurrencias del mismo. Para ello, recorre las keys del diccionario y almacena en la misma posición en ambos array el índice y el valor de este, respectivamente. Para terminar, almacena en un archivo serializado con pickle [4](#) un tupla con estos dos array. Este proceso es llevado acabo para las dos bases de datos. Para ello, se obtiene los ficheros que los ficheros en la ubicación de la base de datos con la función glob de glob [6](#). Un vez tenemos la lista de ficheros de la base de datos, se procesan de manera paralela todos ellos haciendo uso de todos las CPU disponibles utilizando la función parallel [12a](#) del paquete joblib [12](#).

En este momento, se comienza con el preprocesado de la información de los ficheros con extensión '.bytes', siendo este primer paso contar las veces que aparece cada n-grama representado por un índice en formato decimal en los primeros 4000 ficheros. Para ello, se obtienen los ficheros de características obtenidas en el paso anterior con la función glob del paquete glob [6](#). A continuación, se genera un array de numpy de ceros con tamaño el número de 4 n-gramas posibles, o lo que es lo mismo 257^4 . Para continuar, obtener la información obtenida para a continuación sumar uno en el array de numpy generado en los índices que posee la muestra. Una vez, procesados 4000 de estos ficheros se termina guardando los resultados en un archivo serializado con la función pickle del paquete cPickle [5](#).

El segundo paso que se realizar en este preprocesado, de la información conseguida del paso anterior selecciona la posición de los que hayan obtenido un valor superior a 10 y se crea un diccionario donde se almacena el índice original y su posición en el nuevo posición. Con esta información, el siguiente paso consiste en modificar las características obtenidas seleccionando solo las que forman parte del diccionario que se acaba de obtener. Para ello, se recuperan las características obtenidas, recorriéndolas y si pertenecen al diccionario de características frecuentes se incluye en los nuevos array, ya que se almacena mediante dos array uno primero para almacenarlas características y otro segundo para almacenar el valor de esa característica en la misma posición, del mismo modo que se almaceno en un primero momento. Este proceso se realiza para todos los archivos que se han generado en la extracción de características. Además, se paraleliza con la función parallel [12a](#) del paquete joblib [12](#) para 15 de los 16 CPUs disponibles.

Por ultimo, se procede a recuperar las características y almacenarlas en una matriz

dispersa 10, para a continuación pasársela al modelo Linear Support Vector Classification 2e para reducir los outliers que hayan detectado. Para ello, en primer lugar se recuperan los datos en el formato correcto, por lo cual se abre el archivo 'trainLabels.csv' correspondiente a la base de datos de train, el cual se proporciona. Llevando a cabo esto con la función `read_csv` del paquete pandas 9, obteniendo un dataframe, el cual recorreremos y obtenemos los valores que se han generado en el paso anterior. Estos datos están formados por los valores los cuales se incluyen en un array llamado 'data', los índices de los n-gramas que se incluyen en un array llamado 'índices' y por último en un array llamado 'ptr' donde se almacena el índice donde empieza cada una de los valores de cada una de las muestra, para conseguir esto, se acumula la longitud de los valores recuperados en una variable llamada 'cur_bound' y se añade el valor de esta variable en cada momento en el susodicho array. Una vez formados estos arrays se guarda en una tupla formada por arrays de numpy de los tres array obtenidos en un archivo serializado con el paquete hickle 14 y a continuación, se crea una matriz dispersa con la función `csr_matrix` 10a del paquete 10, la cual también es guardada en un archivo serializado con el paquete joblib 12.

A continuación, se repite el proceso anterior para la base de datos de test, por lo cual se genera un dataframe a partir del fichero 'sampleSubmission.csv' proporcionado con la función `read_csv` de paquete pandas 9 y repitiendo con este el proceso de recuperación y almacenaje en una matriz dispersa de las características comentado en el paso anterior.

Llegado a este punto, ahí que construir y entrenar el modelo Linear Support Vector Classification 2e, por lo cual, se recupera la matriz dispersa de las características generada en el paso anterior para la base de datos de train, mediante el paquete joblib 12. También se obtiene de nuevo el dataframe de la base de datos de train, mediante la función `read_csv` del paquete pandas 9 leyendo el fichero 'trainLabels.csv' proporcionado. A continuación se crea el modelo Linear Support Vector Classification 2e, con los siguientes parámetros `penalty=l1` con la cual se establece que la penalización utiliza en el modelo sea la 'l1' del módulo liblinear 13, `max_iter=20` con el que se establece el número máximo de iteraciones que va a realizar el modelo, `dual=False` con la cual se establece que se utiliza el problema de optimización dual 18 y `verbose=1` con el que se establece que el nivel de detalle de información que se muestra mientras se crea el modelo siendo que no muestre información. Para después, entrenar el modelo con la matriz dispersa y las etiquetas correspondientes a las muestras, obtenidas del dataframe y guardando el modelo una vez entrenado en un archivo serializado con el paquete 12. Para terminar, transformando la matriz a partir del modelo aprendido eliminar los outliers, en caso de que la matriz no se haya recuperado, en el caso de la matriz

dispersa de las características de test, primero se obtiene la misma mediante el paquete `joblib` 12. Además, las dos nuevas matrices son guardadas en un archivo serializado con el paquete `joblib` 12, terminando de este modo el preprocesado de las características.

El siguiente paso en esta solución, es reducir la dimensionalidad de las características de los ficheros con extensión `'.bytes'`. Para ello, se recuperan los datos preprocesados en el paso anterior para las base de datos de train y de test y transformando la matriz dispersa en una matriz, también se obtiene mediante la función `read_csv` del paquete `pandas` 9 el dataframe de la base de datos de train del archivo `'trainLabels.csv'` proporcionado. A continuación, se parte la base de datos de train almacenada en la matriz y la etiquetas correspondientes en dos grupos, uno de train y otro de test de un 70 % y 30 % respectivamente. Estos dos grupos se subdividen en características representada con un `X` al principio y etiquetas representada con un `y` al principio, quedando almacenadas en las siguientes variables respectivamente, `X_train`, `X_test`, `y_train`, `y_test`. El siguiente paso, consiste en construir el modelo `Random Forest Classifier` 2a con los parámetros `n_jobs=-1` que indica que se usen todas la CPUs y `n_estimators=1000` que establece el número de árboles de decisión que formaran el modelo y entrenándolo con las variables `X_train` y `y_train`. A continuación, de la característica `features_importances_` del modelo se seleccionan las que tengan un valor superior a 0,0014. Para terminar, construyendo una tupla formada por las dos matrices de los datos preprocesado y que formen parte de las características seleccionadas, terminando por guardar esta tupla en un archivo serializado con el paquete `cPickle` 5. Terminando así el proceso de selección de variables de los archivos con extensión `'.bytes'`.

En este punto, es cuando se comienza a la construcción de los modelos necesarios para realizar la predicción en el reto. Este proceso se puede dividir en 4, siendo estos, la creación un primer modelo base, seguido de otros dos modelos que serán la base de la predicción final y por ultimo, una ponderación de los resultados obtenidos por estos dos últimos modelos. A continuación se explicará cada uno de pasos.

El primero paso, conforme se ha indicado es construir un modelo inicial que servirá de apoyo en para la predicción posterior de otro modelo. Para ello, se comienza por obtener los datos generados tras sus correspondientes preprocesados y selección de variables de los ficheros con extensión `'.asm'` y `'.bytes'`, para a continuación, juntar los en una matriz llamada `'Xtrain'` y `'Xtest'` respectivamente para la base de datos de train y test, haciendo uso de la función `hstack` del paquete `numpy`, esto es posible ya que todas las informaciones han sido almacenadas según las posiciones establecidas por el dataframe correspondiente a cada base de datos. También se obtiene la etiqueta de clase de las muestras pertenecientes a la base de datos de train, para ello se obtiene

el valor de las clase del dataframe del fichero 'trainLabels.csv' proporcionado con la función `read_csv` 9a del paquete `pandas` 9 para almacenarlo en una array de `numpy` 3 restándole 1 para las etiquetas comiencen en el valor 0 y guardándolas en una variable llamada 'ytrain'.

Una vez contamos con los datos necesarios el siguiente paso consiste en establecer los parámetros necesarios y construir el modelo, siendo este un `XGBoost` 15. Los parámetros son guardados en un diccionario transformar los item que la forman en una lista almacenándolo en una variable llamada `plst`. Los parámetros que se establecen son `booster=gbtree` con el que se establece que el modelo boost que se creará sea un `Gradient Boost Tree` 16, `objective=multi:softprob` con el que se establece que el resultado sera una matriz donde para cada elemento se mostrara la probabilidad de cada una de las posibles etiquetas de clase, `num_class=9` que establece que el número de etiquetas para clasificar la clase ese de 9 según se establece en el reto, `eval_metric=logloss` con el cual se establece que la métrica a utilizar es `logloss` 17, `scale_pos_weight=1.0` con el que se establece que el control del balance de los pesos positivos y negativos que se utiliza habitualmente para desequilibrar la clase ??, `bst:eta=0.3` con el cual se establece un peso en tanto por 1, en este caso 0,3, con el que se ponderan los pesos que va generan durante la construcción del modelo para establecer la importancia del modelo construido según su acierto a la hora de predecir las muestras proporcionadas al modelo, con el valor que se le ha proporcionado hace que el comportamiento del modelo sea conservador teniendo mas importancia las características ya aprendidas que las nuevas características, `bst:max_depth=6` con el cual se establece la profundidad máxima de los árboles que construirá el modelo, `bst:colsample_bytree=0.5` con el que se establece el porcentaje en tanto por 1 el número de características que van a ser seleccionadas para construir el árbol de decisión en cada iteración, `silent=1` con el cual se establece que no se muestre información durante la generación del modelo y `nthread=16` con el que se establece el número de hilos que se usarán para generar el modelo. Además, aparte de los parámetros establecidos en el diccionario también se establecen una variable llamada `num_round=100` que establece el número de rondas que realiza el modelo y una lista vacía llamado `watchlist` que establece que no se revisa los resultados del modelo con un conjunto de muestras que no es utilizado en el entrenamiento.

Llegado a este punto, se genera un array con los índices de las muestras y una matriz de `numpy` de ceros con tamaño el número de muestras a clasificar por el número de clases posibles, en este caso 9. Para continuar creando 20 modelos, comenzando por asignar al diccionario de parámetros el valor de la semilla 'seed', siendo este el número del modelo añadiéndole uno más, y transformando los elementos en una lista para pasárselo al modelo. A continuación, se genera un nuevo array que corresponderá

a los índices de las muestras de la base de datos de training que se van a utilizar en el modelo. Para lo cual, lo primero se genera una permutación del array de índices de las muestras que se ha generación al inicio de este punto, haciendo uso de la función `sample` 19a del paquete `random` 19, para a continuación, añadir de manera aleatoria índices del array permutado hasta dejar un array de 8 veces el tamaño de la base de datos.

El siguiente paso, es crear dos `DMatrix`, una para cada base de datos, que es utilizada por pasar al modelo `XGBoost` 15 los datos. La `DMatrix` de la base de datos de Train se construye seleccionando los valores de las características de la matriz `'Xtrain'` y de la etiquetas de `'ytrain'` de esa base de datos con los índices creados en el paso anterior y en el caso de la base de datos de test, la matriz `DMatrix` se construye únicamente pasándole los valores de `'Xtest'`. A continuación, se construye el modelo `XGBoost` 15 propiamente dicho, para lo cual se le pasan los parámetros en forma de lista la Matriz `DMatrix` correspondiente a la base de datos de train, las etiquetas de los índices seleccionados, el número de rondas establecido en la variable `num_round`, y los valores con los que se comprobará establecido en la variable `watchlist`.

A continuación, toca realizar la predicción de este modelo para los valores de la base de datos de test almacenados en la `DMatrix` y reordenarlos para que se ajusten a la matrix de resultados que se genero, donde se acumularan los resultados de todas la predicciones, para a continuación, obtener la media de todos los modelos generados en este primer proceso.

Los resultados obtenidos serán guardados en un fichero llamado `'release0.csv'`, para lo cual se abre el fichero `'sampleSubmission.csv'` proporcionado para este reto, con la función `read_csv` 9a del paquete `pandas` 9. Para a continuación, asignar los resultados obtenidos y terminar guardándolo en el fichero ya indicado con la ayuda de la función `to_csv` 9c del paquete `pandas` 9 terminando así este primer modelo, siendo este una implementación propia de un algoritmo de Bagging 20 donde el modelo base es un `XGBoost` 15.

El siguiente paso, consiste en construir el primero de los modelos que se utilizarán para obtener la predicción. Para lo cual, se obtienen los datos obtenidos de los ficheros `'bytes'` de los ngramas 1 preprocesados pero sin reducir, los cuales son dos matrices dispersas por lo cual son transformadas en dos arrays, los cuales se llaman `Xtrain` y `Xtest` respectivamente con la bases de datos de train y test. También se recupera la etiqueta de la clase del mismo modo que se realizó en el modelo anterior. También los parámetros son los mismos que en el modelo anterior, con la salvedad que el número de rondas ahora es 150 en lugar de los 100 que se utilizo en el modelo anterior. La

construcción del modelo también es muy parecida siendo todo igual salvo que no se amplía la base de datos de train para construir el modelo. Para concluir, se guardan los resultados de la predicción en un fichero 'release0.5.csv' de la misma manera que en el modelo anterior, terminando así este modelo.

Ahora, solo queda construir el último modelo, para lo cual, se recuperan los mismos datos y de la misma manera que en el primero de los modelos. Además, también se recupera la predicción realizada por dicho modelo, para realizar esto se hace uso de la función `read_csv` 9a del paquete `pandas` 9. El siguiente paso, consiste en array de `numpy` según corresponda respectivamente a la etiquetas de la clase y el resto de los datos, quitando si aparece el nombre de la muestra. Se continua, creando un array donde se almacena distribuciones de probabilidad para cada una de la muestras de la base de datos de test, usando para ello la función `rv_discrete` 21a1 del paquete `stats` 21a dentro de `scipy` 21, al cual se le pasa las posibles etiquetas y la predicción para cada una de estas etiquetas de cada muestra. Después, se generan 10 particiones de tamaño $\frac{1}{10}$ de la base de datos de test donde en cada partición una de las partes es seleccionada para test sin ser repetidas y las restantes para train, donde el contenido son los índices de correspondientes a las muestras, haciendo uso de `KFold` 2f y se realiza una copia del array de predicción generado al principio de este paso. Seguidamente para cada una de las partes generadas, se procede a añadir a los datos de la base de datos de train la parte seleccionada para cada caso de train de la base de datos de test, utilizando la función `concatenate` 3a del paquete `numpy` 3, seleccionando los datos correspondientes a la parte de test de esta partición y inicializando un array llamado `pr_vals` con un array vacío donde se irán añadiendo la predicción de cada partición. A continuación, se construyen 20 modelos para cada conjunto de datos, para lo cual lo primero que se realiza, es obtener a partir de las distribuciones de probabilidad generadas anteriormente un valor aleatorio para la etiqueta de la clase para las muestras de la base de datos de test, debido a que es necesaria esa etiqueta, ya que se ha aplicado la base de datos de train con muestras de la base de datos de test y les faltan a estas últimas ese valor, y guardan en una variable llamada `y_train` en un array de `numpy` 3 los valores de la etiqueta clase de la base de datos train y los valores obtenidos para las muestras seleccionadas de la base de datos de test. El siguiente paso consiste en generar dos arrays de índices uno de el tamaño de los datos ampliados y otro con el tamaño de los datos seleccionados de la base de datos de test. A continuación, se procede a realizar una permutación del array de índices completo, para posteriormente ampliarlo con siete veces del tamaño de los datos seleccionados de la base de datos de test, añadiendo de manera aleatoria muestras de este subconjunto. Se continua, generando las `DMatrix` que se utilizarán en el modelo, siendo una con los datos completos para train con su correspondiente valores para la etiqueta y la correspondientes a los valores que se han dejado para el

test. También se establecen los parámetros para correspondiente modelo utilizando un diccionario, siendo para este caso `num_round` con el que se establece que el número de rondas, `seed` con el que se establece la semilla para la generación de números aleatorios, siendo en este caso $123 + 13141 * i$ donde i es el índice del modelo, `max_depth` por el cual se establece que la profundidad de los árboles que construirá el modelo sea como máximo de 3, `gamma` con el que se establece el nivel de pérdida mínima necesaria para que se sigue expandiendo un nodo según una función de pérdida, `eta` por el cual se establece un peso en tanto por 1, en este caso 0,22, con el que se ponderan los pesos que va generando durante la construcción del modelo para establecer la importancia del modelo construido según su acierto a la hora de predecir las muestras proporcionadas al modelo, con el valor que se le ha proporcionado hace que el comportamiento del modelo sea conservador teniendo mas importancia las características ya aprendidas que las nuevas características, `silent` con el que se establece que no se muestre información durante la generación del modelo, `objective` con `multi:softprob` por el cual se establece que el resultado sera una matriz donde para cada elemento se mostrara la probabilidad de cada una de las posibles etiquetas de clase, `num_class` con un 9 con el que establece que el número de etiquetas para clasificar la clase ese de 9 según se establece en el reto, `subsample` por el cual se establece el porcentaje en tanto por 1 de muestras que son seleccionadas para la construcción del modelo, `colsample_bytree` con el que se establece el porcentaje en tanto por 1, el número de características que van a ser seleccionadas para construir el árbol de decisión en cada iteración y `nthread` por el cual se establece que se generen 16 hilos para construcción de los modelos. También, se crea una array vacío que se llama `watchlist` que establece que no se revisa los resultados del modelo con un conjunto de muestras que no es utilizado en el entrenamiento. A continuación, se entrena el modelo XGBoost 15 con los datos y parámetros generados y después se realiza la predicción con los datos de test seleccionados, para a continuación añadirlos al array `pr_vals`. Una vez terminados los 20 modelos se obtiene la media de los resultados almacenados en el array `pr_vals` y se concluye por guardar este resultados en el la copia del array de predicciones. Para concluir, una vez completado todo este proceso para todas las particiones se abre el fichero 'sampleSubmission.csv' con la función `read_csv` 9a del paquete `pandas` 9, se le asignan los resultados guardado en la copia del array de predicciones y se guardan los resultados en un fichero llamado 'release1.csv'. Habiendo terminado así la construcción de todos los modelos necesarios para esta solución.

En este punto, solo queda abrir dos los archivos de predicción y el fichero 'sampleSubmission.csv' con la función `read_csv` 9a del paquete `pandas` 9 para guardar en el dataframe de este ultimo fichero una media pondera de los dos ficheros de predicción obtenidos, siendo un 95 % el ultimo modelo y el resto el otro modelo utilizado para la predicción y se concluye guardando los resultados en un fichero llamado 'release2.csv',

el cual es el que obtuvo la calificación indica al principio.

Capítulo 4

METODOLOGÍA Y DESARROLLO

Llegados a este punto, procederé a explicar el procedimiento llevado a cabo para la obtención de la solución del reto propuesto.

Conforme se estableció en la planificación del anteproyecto, el primer paso era la familiarización con la seguridad informática y concretamente con los conceptos necesarios para el abordaje de este reto. Por esto, además de realizar una búsqueda de soluciones propuestas por otros equipos, las cuales se ha explicado en el apartado de antecedentes y estado de la cuestión, también se buscó artículos en los que se mostrarán métodos de análisis que pudieran ser aplicados a este reto. De la cual, se obtuvo información de las familias de malware utilizados en este reto y sus correspondientes características identificativas, las cuales ayudarán a elegir cuáles serán las características más adecuadas a utilizar para el reto, siendo los tipos de malware utilizados los siguientes, según la página de Glosarios de Microsoft Windows Defender Security Intelligence [9]:

1. Worm o Gusano es un tipo de malware que se propaga a otros ordenadores. Este puede usar para propagarse uno o más de los siguientes métodos: Archivo adjunto o link en un email, envío a través de un sistema de mensajería instantánea como puede ser Skype, en los ficheros descargados o subidos en redes de intercambio de ficheros p2p, enviando un mensaje a todos tus contactos de una red social, en el auto-arranque de un disco extraíble o explotando una vulnerabilidad de un software.
2. Adware es un tipo de malware que muestra publicidad adicional que no puedes controlar estas usando el ordenador.
3. Trojan es un tipo de malware que a diferencia del worm [1](#) no se propaga por sí solo. Esto hace que tenga que aparentar ser un software inofensivo y que la víctima lo descargue e instale para de ese modo infectar su ordenador. Un vez, el

sistema esta comprometido puede robar tu información personal, descargar otro malware y pudiendo llegar a permitir al atacante tener acceso y/o control sobre tu ordenador.

4. Backdoor este tipo de malware es un clase de trojan 3 que permite al atacante tener acceso y control a tu ordenador.
5. TrojanDownloader, al igual que en el caso del backdoor 4, este tipo de malware es una clase de trojan 3, la cual instala otros ficheros maliciosos, incluido malware, en tu ordenador. El puede descargar el fichero desde un ordenador remoto o instalarlo directamente desde un copia incluida dentro de él.
6. Any kind of obfuscated malware no se trata de un tipo de malware, sino que son técnicas usadas para ocultar o hacer parecer limpio un malware, aunque en la página de glosarios de términos [9] que estamos utilizando si lo identifican como un tipo de malware. Lo que realizar es ocultar su código para hace mas difícil que los programas de seguridad lo detecten o eliminen.

De estos tipos de malware, se han seleccionado las familias que utilizaremos en el reto, siendo estas:

En primer lugar la familia Ramnit que según la información del reto es de tipo Worm 1 aunque en la propia página de Microsoft Defender Security [10] no se llega a clasificar, aunque se puede deducir ya que si se menciona que el vector de entrada más habitual es mediante la utilización de un disco extraíble, generalmente pendrive, ya infectado. Este malware roba información sensible del usuario, como pueden ser usuario y contraseña de acceso a su banco. Además, puede proporcionar acceso y control de nuestro ordenador al atacante.

En segundo lugar nos encontramos la familia Lollipop que según la información del reto y la propia página de Microsoft Windows Defender Security [11] es de tipo Adware 2. Esta familia de malware lo que realiza es mostrar publicidad emergente mientras se esta utilizando el navegador y redirige los resultados de tu motor de búsqueda. Debido a que redirige los resultados del motor de búsqueda, le permite mostrar la publicidad basándose en las palabras claves utilizadas en esto además de su ubicación geográfica.

En tercer lugar es el turno de Kelihos_ver3 que según la información del reto es de tipo Backdoor 4 y según la propia página de Microsoft Windows Defender Security [12] lo clasifican como un trojan 3 que puede dar acceso y control de nuestro ordenador al atacante, que conforme se puede observar es la definición de backdoor 4. Es la tercera versión de este malware y se propaga mediante el envío de correos electrónicos

no deseados que contienen enlaces a otros malware. Además, puede comunicarse con otros ordenadores para intercambiar información sobre el envío de correos electrónicos no deseados, robar tu información sensible o descargar y ejecutar ficheros maliciosos.

El siguiente es Vundo que según la información del reto es de tipo Trojan 3 aunque en la propia página de Microsoft Windows Defender Security [13] lo clasifica como una familia de malware formada por múltiples familias, el cual muestra ventanas emergentes con publicidad fuera de contexto. Además, algunas de sus variantes puede descargar y ejecutar otros ficheros, incluyendo otros tipos de malware. El vector de entrada mas habitual es mediante un complemento para el navegador. También, utiliza técnicas avanzadas para defenderse y mantenerse a salvo de las detecciones y ocultarse cuando se eliminan.

A continuación, nos encontramos con Simda que según la información del reto y la propia página de Microsoft Defender Security [14] es de tipo Backdoor 4. Lo que realizar es robar la contraseñas del usuario del sistema infectado.

El siguiente en la lista es Tracur que según la información del reto es de tipo TrojanDownloader 5 pero en la propia de página de Microsoft Defender Security [15] lo clasifica como una familia de Trojans 3 que puede redirigir tus buscadores web. Realizan esto para ganar dinero con la publicidad fraudulenta. Este malware también puede descargar y ejecutar otros ficheros, esto incluye otros posibles malware, y de esta manera proporcionar al atacante el control de tu ordenador. Los vectores de entrada mas habituales por este malware son que sea instalado por otro malware, cuando haces click en enlaces sospechosos o mediante un archivo adjunto en un email.

A continuación, es el turno de Kelihos_ver1 este tipo de virus es la primera versión del ya mencionado Kelihos_ver3 4, el cual en el reto es clasificado como de tipo Backdoor, mientras en la propia pagina de Microsoft Windows Defender Security [16] lo clasifican como un trojan 3, igual que ocurría con Kelihos_ver3.

Siguiendo con la lista nos encontramos con Obfuscator.ACY que según la información del reto y la propia página de Microsoft Windows Defender Security [17] se trata de una técnica mediante la cual se intenta ocultar 6 el malware y así evitar que los sistemas de seguridad la detecten.

Y por ultimo, nos encontramos con Gatak que según la información del reto es de tipo Backdoor 4 y la propia página de Microsoft Windows Defender Security [18] es de tipo Trojan 3. Según la página mencionada, esta familia de malware recoge información

del equipo infectado y enviarla posteriormente al atacante y el punto de entrada suele ser a través de un generador de llaves para una aplicación o mediante una aparente actualización legítima de una aplicación. Una de las acciones más significativas que realiza que además permite identificar que estamos infectados con este malware es modificar el registro de Microsoft Windows para que la aplicación de mensajería y videollamada Google Talk se arranque de manera automática y Skype se inicie de manera automática de manera minimizada y sin mostrar la pantalla durante esta arrancando. Esto es llamativo porque estas aplicaciones pueden ser utilizadas para exfiltrar la información recabada del usuario.

Según se nos explica en el siguiente artículo [19], para este reto se nos proporcionan dos bases de datos, *train* y *test*, las cuales contienen muestras de malware de las clases mencionadas anteriormente. Estas están formadas por dos tipos de fichero, por un lado un fichero en binario, al cual se le ha eliminado la cabecera para hacerlo inofensivo, representado en formato hexadecimal y otro que contiene metadatos obtenidos mediante el desensamblado de las muestras con la utilidad IDA dissassembler tool. Donde la base de datos de *train* está compuesta por 10868 muestras de malware las cuales se reparten por las clases de malware según muestra en la tabla 4.1, en cuanto a la base de datos de *test* como esta formada por 10873 muestras de malware, las cuales ahí que clasificar en la clase que le corresponde.

Id	Clase	Muestras	Tipo
1	Rammit	1541	Worm
2	Lollipop	2478	Adware
3	Kelihos_ver3	2942	Backdoor
4	Vundo	475	Trojan
5	Simda	42	Backdoor
6	Tracur	751	TrojanDownloader
7	Kelihos_ver1	398	Backdoor
8	Obfuscator.ACY	1228	Any kind of obfuscated malware
9	Gatak	1013	Backdoor

Tabla 4.1: Clase de malware en la base de datos *train*.

Además, se consiguió el artículo ‘A scalable multi-level feature extraction technique to detect malicious executables’ de Mohammad M. Masud, Latifur Khan y Bhavani Thuraisingham de 2007 [20], en el cual se explica la investigación que han realizado con respecto a la detección de malware. Esta investigación demuestra que una extracción de características mixta mejora los resultados de los modelos. Esta perspectiva híbrida supone extraer y manejar características de los ficheros binarios en función

del número de ngrams [1](#), instrucciones en ensamblador, llamadas a funciones tanto del sistema como propias, etc. Esto supone incrementar y mucho el número de características, haciendo necesario realizar un proceso de selección de las mas significativas para que el modelo a construir sea abordable. Resulta interesante que se trata de la misma perspectiva utiliza en la solución 2 [3.2](#). Para mi solución, he decidido utilizarla también.

Llegado a este punto, comenzaré a explicar el proceso llevado para la construcción de mi solución.

La primera decisión, después de tener claro que se va a utilizar un sistema de características híbridas, es el número de ngrams [1](#) que se se van a utilizar. Una vez estudiado tanto el artículo anterior como la solución 2 [3.2](#), tomé la decisión de hacer un esfuerzo para utilizar en mi solución los mismos 4 ngrams que se utilizaba en la solución 2 [3.2](#). Esta decisión es más importante de lo que parece, ya que va repercutir en todas las decisión posteriores, debido a los recursos de los que voy a hacer uso son solamente mi portátil personal (Toshiba L850-150) y mi disco duro externo, siendo las características las siguientes:

- Procesador Intel Core i7-3610QM a 2,30 GHz.
- 8 Gb RAM DDR3.
- Tarjeta Gráfica Radeon 7670M.
- Disco Duro SSD Samsung EVO 850 de 250 Gb.
- Disco Duro externo Seagate Expansion de 4 Tb.

Estos recursos implican no poder tener todas las características extraídas en memoria ni tan siquiera de una sola muestra. Esto hace que sea necesario utilizar alguna tecnología que permita almacenar de manera persistente esta información. Para solucionar este problema, la idea fue utilizar una base de datos, concretamente MongoDB [22](#). El motivo por el que me decante por está me decidí por utilizar esta base de datos fue que se trataba de una base de datos NoSQL [23](#) con lo facilidad que esto implica ya que te permite almacenar en los atributos estructuras como pueden ser arrays y objetos BSON [25](#), además tiene una buenas compatibilidad con el lenguaje de programación elegido Python 3 y es una opción muy estable, consolidada, eficaz y open source.

Otra cuestión a tratar antes de entrar a explicar la configuración de la base de datos MongoDB, es explicar el motivo por el cual se ha tomado la decisión de utilizar docker y la configuración que se ha llevado a acabo. Para empezar el motivo, ha sido para fomentar la portabilidad de la solución y facilitar su ejecución y prueba. También, se ha buscado de alguna manera controlar los recursos que dispongo y aislar la ejecución del

sistema, ya que se manejan muestras de malware y aunque se hayan tomado medidas para convertirlos en inofensivos es mejor aislarlos. Por estos motivos y por tratarse de una sistema de virtualización ligero se ha optado por la utilización de Docker [26](#).

En primer lugar, se tiene que instalar Docker y Docker Compose conforme se puede en los siguientes enlaces [\[21\]](#) y [\[22\]](#) respectivamente. En este punto, comenzaremos con la configuración para lo cual se utilizar un fichero de tipo yaml donde se establecerán la configuración que posteriormente utilizará Docker Compose. En este fichero, se crearán dos servicios, db y tfg, y una red, cuyo nombre se tfg, para posibilitar la comunicación entre los servicios.

La red se llama tfg y se le ha indicado que el driver es bridge, mediante el cual se establece que los contenedores se mantienen aislados y utilizan esta red virtual para comunicarse entre ellos. También se le configura que el sistema administrador de direcciones IP utiliza como driver el que tiene establecido por defecto y que el segmento de red que utilizarán los contenedores es 172.16.0.0/24.

El servicio db se establece la configuración de la base de datos de MongoDB [22](#), para lo cual se establece que la imagen que se va a utilizar es la última disponible de mongo mediante el parámetro `image:mongo:3.6.5`. A continuación los parámetros `hostname` y `container_name` establece el nombre de la maquina y el nombre del contenedor que se creará respectivamente, en mi caso son `mongo` y `mongo_tfg`. El siguiente parámetro es `command` mediante el cual se establece el comando que se ejecutará al lanzar el servicio, en mi caso los que se ejecutará el servidor de MongoDB [22](#) con su correspondiente configuración. La configuración que se ha decidido establece para el servidor de base de datos MongoDB [22](#) es utilicé como motor de base de datos `wiredTiger` con el parámetro `storageEngine`, que cree un directorio por cada base de datos con el parámetro `directoryperdb`, que guarde en un directorio los índices y en otro las colecciones con el parámetro `wiredTigerDirectoryForIndexes`, que comprima las colecciones con el algoritmo que se establece con el parámetro `wiredTigerCollectionBlockCompressor`, en mi caso se ha establecido `zlib`, ya que es el mas eficaz a la hora de reducir espacio, que comprima también los índices con el parámetro `wiredTigerIndexPrefixCompression` y por ultimo se le establece la cantidad de RAM que utilizará con el parámetro `wiredTigerCacheSizeGB`, en caso le he establecido 4 Gb. El siguiente parámetro que se establece en el servicio es `volume` mediante el cual se establece un almacenamiento permanente al contenedor y en este caso también se le establece la dirección dentro del propio ordenador donde se almacenará, en este caso se ha optado por almacenarlo en el directorio `home` de mi ordenador en la carpeta `TFGDataBase` que esta ubicada en el disco solido.

Por ultimo, ya solo nos queda establecer la configuración de red que tendrá el contenedor para la red que se ha establecido para los contenedores, siendo la configuración una dirección IP dentro del rango establecido en la red tfg con el parámetro `ipv4_address` y alias con el parámetro `aliases`, siendo los valores para estos 172.16.0.2 y mongo respectivamente.

Por ultimo, el servicio tfg al igual que en el comando db se le ha establecido el nombre a la maquina, el nombre al contenedor, los volúmenes que establecen la ubicación del almacenamiento permanente y la configuración de red del contenedor. Además de esta configuración, se utiliza el parámetro `depends_on` para indicar que este servicio necesita de manera obligatoria el servicio db, con lo cual tiene que ejecutarse del servicio db antes de que comience la ejecución de éste, y el parámetro `enviroment` donde se establecen una listas de variables de entorno con las que se ajustará la ejecución de la solución. Estas variables de entorno son `SAMPLE` con la cual se indica que seleccione una muestra de de los fichero de train, `PERCENTAGE_SAMPLE` con el que se le indica el porcentaje concreto de la muestra, solo es utilizado conjuntamente con la variable `SAMPLE`, `PROJECT` con la cual se le asigna un nombre al proyecto, `IP_SERVER_MONGO` con la que se le dice la dirección IP que se le ha asignado al servicio db, `DB_NAME` con el cual se le indica el nombre de la base datos que va a manejar el proyecto, `ORDER_EXTRACT_FEUTURES_ASM` con la que se establece el número de ngrams [1](#) que se van a utilizar en la solución, `DIR_DATA_RAW` con la cual se le indica la dirección donde se encuentran los datos que maneja la solución, `SEED` con el que se le establece una semilla, `THREAD` con el cual se le indica el número de hilos que se utilizará en los procesos de paralización, `PERFORM_METHOD` con el que se le indica que parte de la solución se ejecutara y `NUM_FEATURES` con la cual se se le indica el número de características que se utilizara en el modelo que se construya. También se le establece que el nombre del contenedor sea `code_tfg`, el nombre de la maquina sea `code` y la imagen que utilice para crear el contenedor sea tfg, aunque en este caso esta no es una imagen oficial, como en el caso de MongoDB [22](#), esto implica que se tiene que construir está, por lo cual también se tiene el parámetro `build`. Este parámetro hace que tengamos que crear un fichero llamado `Dockerfile` donde estableceremos la configuración especifica para generar esta imagen. En este fichero se le configura el parámetro `FROM` con el cual se le indica que utilice como base la imagen oficial `python:3.6.5`, el parámetro `MAINTAINER` mediante el cual se le establece el autor de la imagen, el parámetro `LABEL` con el que se le establece una etiqueta y se le establece un nombre al proyecto, siendo este `detectorMalware`, el parámetro `ADD` con el que se copia los ficheros de la maquina huésped a la maquina virtual dentro del contenedor, el parámetro `RUN` con el que se ejecutan comandos dentro de la maquina

del contenedor, en mi caso va al directorio que se ha copiado y instala pip3 install -r requirements.txt los paquetes necesarios para ese proyecto, el comando WORKDIR con el que se establece como directorio inicial el directorio donde hemos copiado los ficheros y por ultimo el parámetro CMD con el cual se establece el comando que va a ejecutar cuando se inicie el contenedor, en mi caso python3 detectorMalware.py. Con esto se termina la configuración de Docker [26](#).

El directorio, que se copia en el servicio tfg [4](#) que contiene la implementación de mi solución, se estructura de la siguiente manera. En este directorio tenemos el fichero requirements.txt en el cual establecemos los paquetes de python necesarios para ejecutar mi solución, el fichero __init__.py en el que se establece metainformación del proyecto, como puede ser nombre del paquete, versión y autor, el fichero detectorMalware.py el cual es el encargado de ejecutar y controlar la ejecución de mi solución y por ultimo el directorio src el cual se subdivide en 4 directorios, siendo los siguientes extract_features el cual contiene los ficheros necesarios para llevar a cabo la fase de extracción de características, misc en el que se almacena los ficheros varios que se utilizarán en la solución, por ejemplo sample, split entre otros, model el cual contiene los ficheros que manejan desde el preprocesado de los datos hasta la predicción del modelo posando por la construcción del mismo y mongo donde se guarda el fichero que establece la conexión con la base de datos MongoDB [22](#).

Comenzaré la explicación de mi solución siguiente el proceso de ejecución del fichero detectorMalware.py, siendo el primer paso recuperar los valores de las variables de entorno establecidos en el fichero docker-compose.yml que se comentaron anteriormente. El siguiente paso, es comprobar si se desea ejecutar la solución sobre una muestra, esto se realiza así debido a que el volumen de información a manejar es excesivamente grande y durante el proceso de implementación sería muy tedioso tener que esperar el tiempo que supondría ejecutarlo sobre el conjunto completo de datos, por ese motivo se ha optado por esta opción para reducir el tiempo que supondría. Para seleccionar la muestra se ha creado un fichero sample.py dentro del directorio misc donde se ha implementado la función selectSample, el cual selecciona una muestra estratificada del fichero trainLabels.csv, para lo cual hace uso de pandas [9](#) creando un DataFrame [9](#) y seleccionando de manera aleatoria una muestra de los valores de la clase 1 y repitiendo el proceso para el resto de valores de la clase y concatenándolos en el dataframe [9](#) del primer valor con el comando de pandas [9](#) concat [9b](#) para terminar guardando el dataframe [9](#) en un nuevo fichero csv y terminado por devolver la dirección con esta almacenado el mismo.

A continuación, se abre el origen ya sea el proporcionado por en el reto o en el gene-

rado durante el proceso de muestra, para lo cual se hace uso de la función `open_sources` la cual abre el el fichero de donde se van a leer los datos que se pasa a la función por parámetro y leyendo la información del fichero. Se continua separando esta información por líneas y desechando la primera porque son las cabeceras del fichero y la ultima porque en el fichero ahí una linea la final del documento sin información. Después, se obtiene el número de líneas que tiene el documento y se parte cada una de las líneas con una función aplicándole una función que se le pasa por parámetro a la tupla obtenida de aplicar la función `enumerate` a los datos. Por ultimo, se cierra el fichero y se devuelve la longitud y los datos ya particionados. Las funciones para realizar el particionado de los ficheros origen se encuentran en fichero `split` en `misc`. En este fichero tenemos la función para particionar los datos de origen de `train`, de `test` y una función genérica común para ambas funciones. La función `split_Source` lo que hace generar un array en el que la primera posición es el indice que tiene ese elemento en el fichero origen seguido de todos los elementos que posee el elemento después de dividirlos por la coma y quitarle las comillas dobles. En el caso de `split_Source_Test` se devuelve una tupla de los dos primeros elementos del array que general la función general. En cambio, en `split_Source_Train` se hace igual pero el tercer elemento del array se transforma de `string` a un entero.

El siguiente paso, es comprobar el valor que tiene la variable de entorno `PERFORM_METHOD`. Los posibles valores que se le puede asignar son `EXTRACT_TRAIN`, `DISCRETE`, `SELECT`, `MODEL`, `EXTRACT_TEST`, `MODEL_PREDICT`, `PREDIT`, `EXTRACT_ALL` Y `ALL` que establece si se extraen las características de la base de datos de `train`, si se discretizan las características de la base de datos de `train`, si se seleccionan las características que mayor información proporcionan de la base de datos de `train`, si se crea el modelo con las características seleccionadas de la base de datos de `train`, si se extraen las características de la base de datos de `test`, si se crea el modelo y se predice el resultado de la base de datos de `test`, si se predice el resultado de la base de datos de `test`, si se extrae las características de las dos bases de datos o todo lo anterior respectivamente. Esto se ha realizado así para durante el proceso de implementación poder excluir los pasos ya concluidos y ahorrar todo el tiempo que fuese posible.

El primero de estos pasos que ahí que ejecutar, por orden es `EXTRACT` el cual, lo primero que realizar establecer la conexión con la base de datos para los cual se usa la clase creada en el fichero `database.py` dentro de la carpeta `mongo`. En este fichero, contiene una clase llamada `DataBaseMongo` la cual se ha creado para manejar y controlar la conexión con la base de datos, para lo cual cuando se crea una instancia de la misma ahí que indicarle tanto la dirección del servidor de base de datos MongoDB [22](#) y el nombre de la base de datos a la que nos queremos conectar. Esta clase también posee

un método para seleccionar una colección pasándole como parámetro el nombre de la misma. Esto se ha realizado así para garantizar de que no se le pase ningún parámetro erróneo y para manejar los posibles excepciones tanto de la conexión con el servidor como la selección de la colección.

Una vez, establecida la conexión con la base de datos se procede a borrar todas las colecciones que tiene establecidas, para lo cual selecciona cada colección y la elimina con el método drop proporcionado para la API de MongoDB [22](#) para python.

A continuación, se procede a la extracción de las características de la base de datos train, proceso que se realiza de manera paralela utilizando el número de hilos que se ha establecido en la configuración, y realizando así la extracción de las características de manera simultánea. Este proceso de extracción consiste en llamar en la función `extract_features` dentro del fichero `extract_features.py`, al cual ahí que pasarle por parámetro la ip del servidor de base de datos MongoDB [22](#), el nombre de la base de datos a la que nos conectaremos, la ubicación dentro del contenedor donde se puede acceder a los datos proporcionado, el prefijo y el nombre de las colecciones que se utilizarán, el número de ngrams [1](#) a extraer y por último del fichero del que se extraerán las características, siendo todos los parámetros fijos salvo este último que cada hilo tratará uno de estos ficheros. Los datos proporcionados se les pasan al sistema en un fichero comprimido en zip de donde se seleccionan el elemento a procesar. Esto se ha realizado de esta manera por ahora espacio en el disco duro y ha sido sacada la idea de la solución [1](#) [3.1](#), ya que de esta manera el peso de las dos bases de datos es de algo más de 56 Gb mientras si se trata de manera descomprimida ocuparía más de 400 Gb complicando donde almacenar tanta información, aunque como no se podía manejar los archivos comprimidos que se suministran a tenido que descomprimirse el fichero 7z y posteriormente comprimirlo de nuevo en formato zip, el cual si es posible manejarlo con python. Este proceso se realiza por separado tanto a la base de datos de train como para test. Llegados a este punto, el primer paso es establecer la conexión con la base de datos y seleccionar la colección `train_total` donde se inserta un registro con el id, nombre y etiqueta del elemento que se va a procesar en dicha colección. El siguiente paso consiste en realizar las modalidades de extracción que se van a realizar, extracción de binario y de ensamblador, para lo cual se utiliza la función `extract_Features_Byte` dentro del fichero `extract_features_bytes.py` y la función `extract_Features_Asm` dentro del fichero `extract_features_asm.py` respectivamente. Empezaré explicando la función `extract_Features_Byte`, la cual se ha adaptado la implementación de la solución [2](#) [3.2](#) que se esta parte de la solución a mi solución, siendo lo primero se conecta con la base de datos de MongoDB [22](#) y seleccionar tanto la colección total como la features donde se almacenará la información extraída. Después, se abre de dentro del fichero comprimido

en zip para esa base de datos el fichero establecido en el origen y se acumulan todos los ngrams 1 en un array para posteriormente recorrer este array en bloque establecidos por el parámetro order que establece el número de ngrams 1 que se extraerán llevando el conteo en un diccionario. Se continua recorriendo este diccionario para obtener el número total de ngrams 1 y actualizar el valor del registro insertando anteriormente en la colección total con el número total de ngrams 1 del fichero, esto es realizado para realizar una normalización de los conteos respecto del fichero. Para concluir, insertando todos los elementos del diccionario en la colección bytes donde se almacena para cada bloque de ngrams 1 del fichero en id del fichero, el nombre del fichero, la característica que esta formada por "b_" seguido del bloque de ngrams 1 que corresponde, el número de veces que aparece en el fichero tanto normalizado respecto del fichero como sin normalizar y el tipo de característica de que se trata, siendo en este caso bytes.

En cambio en la función `extract_Features_Asm`, donde se realiza la extracción de la información de los ficheros en ensamblador, el proceso que realiza empieza igual que en el caso de bytes donde se establece la conexión con la base de datos, se seleccionan las colecciones que se utilizarán que son total y features y se obtiene de dentro del fichero comprimido en zip el fichero establecido como origen para extraer los datos, aunque en este punto se le establece que se encuentra codificado en iso8859 debido a que en el paso anterior no era necesario al no manejar caracteres especiales. A continuación, se parte el fichero en líneas haciendo uso del paquete `re` 27 con la función `findall` y la opción `multiline`, esto se realiza así para reconocer varios tipos de salto de línea y evitar así de esta manera posibles problemas de codificación. En este punto, se elimina de la línea los posibles saltos de carro, los comentarios, los espacios en blanco al inicio y los tabuladores, para continuación partiendo la línea una vez por el separador ":" y obteniendo así la sección y la línea que queda pendiente de tratar. El siguiente paso, es partir la línea una vez teniendo en este caso como separador obteniendo en este paso la dirección y almacenando en un diccionario la sección extraída anteriormente un conjunto donde se guardaran las direcciones encontradas, esto es así para solo contemplar una vez cada una de las direcciones.

Capítulo 5

EXPERIMENTOS Y RESULTADOS

Capítulo 6

CONCLUSIONES Y PROPUESTAS

6.1. Conclusiones

6.2. Trabajo futuro

4 Ngrams

Paralelización y Cluster

Microservicios

Orquestación Kubernetes

Optimización

Bibliografía

- [1] R. Montiel Soto, Y. Ledeneva, R. A. García-Hernández, and R. Cruz Reyes, “Comparación de tres modelos de texto para la generación automática de resúmenes,” *Procesamiento del Lenguaje Natural*, no. 43, 2009. 3
- [2] D. D. Lee and H. S. Seung, “Learning the parts of objects by non-negative matrix factorization,” *Nature*, vol. 401, no. 6755, p. 788, 1999. 3
- [3] G. Cobo, X. Sevillano, F. Alías, and J. C. Socoró, “Técnicas de representación de textos para clasificación no supervisada de documentos.” *Procesamiento del lenguaje natural*, vol. 37, 2006. 3
- [4] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin, “LibLinear: A library for large linear classification,” *Journal of machine learning research*, vol. 9, no. Aug, pp. 1871–1874, 2008. 4
- [5] Distributed (Deep) Machine Learning Community. Documentación de parámetros de xgboost. [Online]. Available: <https://xgboost.readthedocs.io/en/latest/parameter.html#parameters-for-tree-boost> 4
- [6] I. Singer, “A general theory of dual optimization problems,” *Journal of Mathematical Analysis and Applications*, vol. 116, no. 1, pp. 77–130, 1986. 4
- [7] V. Chevli. Beating the benchmark for Microsoft Malware Classification Challenge (BIG 2015). [Online]. Available: <https://github.com/vrajs5/Microsoft-Malware-Classification-Challenge> 7
- [8] M. Trofimov, D. Ulyanov, and S. Semenov. Microsoft Malware Classification Challenge third place solution. [Online]. Available: <https://github.com/geffy/kaggle-malware> 9
- [9] Glossary. Microsoft Windows Defender Security Intelligence. [Online]. Available: <https://www.microsoft.com/en-us/wdsi/help/antimalware-security-glossary> 23, 24

- [10] Win32/Ramnit. Microsoft Windows Defender Security Intelligence. [Online]. Available: <https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Win32/Ramnit> 24
- [11] Adware:Win32/Lollipop. Microsoft Windows Defender Security Intelligence. [Online]. Available: <https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Adware:Win32/Lollipop> 24
- [12] Backdoor:Win32/Kelihos.F. Microsoft Windows Defender Security Intelligence. [Online]. Available: <https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Backdoor:Win32/Kelihos.F> 24
- [13] Win32/Vundo. Microsoft Windows Defender Security Intelligence. [Online]. Available: <https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Win32/Vundo> 25
- [14] Win32/Simda. Microsoft Windows Defender Security Intelligence. [Online]. Available: <https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Win32/Simda> 25
- [15] Win32/Tracur. Microsoft Windows Defender Security Intelligence. [Online]. Available: <https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Win32/Tracur> 25
- [16] Win32/Kelihos. Microsoft Windows Defender Security Intelligence. [Online]. Available: <https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Win32/Kelihos> 25
- [17] Virtool:Win32/Obfuscator.ACY. Microsoft Windows Defender Security Intelligence. [Online]. Available: <https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=VirTool:Win32/Obfuscator.ACY> 25
- [18] Win32/Gatak. Microsoft Windows Defender Security Intelligence. [Online]. Available: <https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?name=Win32/Gatak> 25
- [19] R. Ronen, M. Radu, C. Feuerstein, E. Yom-Tov, and M. Ahmadi, “Microsoft malware classification challenge,” *CoRR*, vol. abs/1802.10135, 2018. [Online]. Available: <http://arxiv.org/abs/1802.10135> 26
- [20] M. M. Masud, L. Khan, and B. Thuraisingham, “A scalable multi-level feature extraction technique to detect malicious executables,” *Information Systems Frontiers*, vol. 10, no. 1, pp. 33–45, 2008. 26

- [21] Install docker. Docker Inc. [Online]. Available: <https://docs.docker.com/install/28>
- [22] Install docker compose. Docker Inc. [Online]. Available: <https://docs.docker.com/compose/install/28>