Trabajo Fin de Máster

Seguridad de las Tecnologías de la Información y de las Comunicaciones

Detección de Malware, Métodos Estadísticos y Machine Learning.

Curso académico 2018-2019

Autor: Javier Ruiz Ruiz

Director del máster: Ángel Elbaz Sanz

Resumen

En un mundo parcialmente digitalizado y donde la mayoría de las acciones cotidianas se ven influenciadas por sistemas informáticos, es necesario conocer los riesgos que pueden suponer los ataques informáticos y la distribución de software malicioso o malware.

Este tipo de software, normalmente distribuido por grandes grupos o asociaciones de criminales, trata de obtener el beneficio económico a partir de los daños que este pueda producir en su objetivo final. Se trata de una amenaza constante y presente en el día a día que afecta tanto a los usuarios como a las empresas de todo el mundo. Es importante conocer la presencia de estas actividades y poder llevar a cabo un estudio y análisis de ellas.

Los análisis que se realizan sobre este tipo de software comúnmente están destinados a conocer si el software es realmente mal intencionado o si en cambio se trata de un software legítimo. Actualmente estas clasificaciones y detecciones se realizan en base a "firmas" o reglas presentes en los sistemas de antivirus tratando de reconocer los patrones característicos de la amenaza. El problema que presenta este tipo de detecciones es la corta escalabilidad que se aprecia cuando una muestra es modificada lo suficiente para que estas firmas sean incapaces de reconocerla, esto hace que sea necesario el continuo estudio de las muestras de software por parte de los analistas de seguridad.

Por lo tanto, este documento intenta proponer una solución que facilite el reconocimiento del software malicioso y reduzca el trabajo manual, así como la escalabilidad del sistema utilizando técnicas de machine learning.

Keywords: malware, clustering, machine learning

Abstract

In a partially digitized world and where most everyday actions are influenced by computer systems, it is necessary to know the risks that computer attacks and the distribution of malicious software or malware can suppose.

This type of software, normally distributed by large groups or associations of criminals, tries to obtain the economic benefit from the damages that it can produce in the target. It is a persistent and day-to-day threat that affects both, users and companies around the world. It is important to know the presence of these activities and be able to carry out a study and analysis of them.

The analyses that are perfromed on this type of software are commonly dedicated to know if the software is really bad intentional or if instead it is a legitimate software. Currently, these classifications and detections are made based on "signatures" or rules present in antivirus systems trying to recognize the characteristic patterns of threats. The problem with this type of detections is the short scalability that can be seen when a sample is modified enough for these signatures in order to make them unable to recognize, making the continous study of malware samples necessary by analysts.

Therefore, this document tries to propose a solution that facilitates the recognition of malicious software and reduces manual work, as well as the scalability of the system using machine learning techniques.

Keywords: malware, clustering, machine learning

Contenido

Índice de tablas

Índice de figuras

1. Introducción

1.1. Motivación

Todos los días en Internet se transfieren miles de millones de ficheros, algunos mediante correo electrónico, otros mediante páginas de descargas, a través de memorias USB y otros cientos de medios. Este continuo flujo de ficheros hace que estos sean capaces de llegar a prácticamente cualquier sistema informático involucrado en nuestras vidas.

Es este, el escenario deseado por los criminales para poder explotar sus actividades delictivas a la hora de distribuir ficheros malintencionados o malware cuya finalidad consiste en la obtención de beneficios por medio del daño físico o moral ocasionado en las víctimas.

Fue así como surgieron las primeras empresas de antivirus, compañías con el objetivo de detectar y mitigar este tipo de software malicioso protegiendo así a los usuarios y sistemas. Hasta la fecha, las detecciones realizadas por estas compañías se han basado en la utilización de firmas realizadas a partir de las características extraídas de las muestras de malware tras un análisis exhaustivo de estas. Las firmas se encuentran en forma de base de datos de definiciones y son mantenidas y actualizadas por las compañías, estas firmas suelen estar compuestas por los hashes de muestras conocidas como maliciosas o cadenas de caracteres “strings” encontradas en el interior de los binarios.

Otro tipo de reglas presentes en los antivirus convencionales son las firmas basadas en comportamiento o heurística. Estas reglas son más precisas que las anteriores y pueden seguir detectando las diferentes variantes del malware, aunque estos modifiquen ciertas características. El problema que presenta este tipo de firmas es la necesidad de ejecutar el malware para conocer su comportamiento, por lo tanto, son más costosas en cuanto a recursos se refiere y también cabe destacar que estas reglas son generalmente más complejas de implementar.

La solución propuesta durante los últimos años y que estas compañías de antivirus han empezado a implementar, es la utilización de sistemas basados en machine learning para intentar predecir si un software específico es malicioso o no. Estos sistemas generalmente también se basan en las características extraídas de las muestras de software, usando tanto las conseguidas por medio de análisis estáticos o dinámicos tal y como se realizaba en las firmas y heurística. Mediante estas características se pueden crear inmensas bases de datos que contengan información histórica sobre millones de muestras de software, tanto malicioso como legítimo y a partir de esta información se entrenarán o se realizarán modelos estadísticos para poder agrupar y clasificar las nuevas muestras que se reciben a diario.

No obstante, la detección de malware basado en machine learning es compleja y requiere de una gran recolección de información. Además, aunque estos sistemas presentan un porcentaje muy alto de precisión, no llegan a ser completamente perfectos. Es necesario realizar estudios de los resultados obtenidos y modificar el sistema hasta que se adapte a nuestras especificaciones.

1.2. Objetivos

La solución propuesta en este trabajo busca conseguir un sistema capaz de detectar y agrupar malware de forma automática y con la capacidad de aprender cada día a mejorar el sistema. De esta forma, se reducirán los tiempos empleados por los analistas de seguridad a la hora de descartar muestras ya analizadas y centrarse específicamente en aquellas que sean nuevas o interesantes.

Los objetivos que se pretenden conseguir se enumeran a continuación:

- **Realizar un estudio previo del problema**. Se estudiará el problema y las soluciones propuestas actualmente, de esta forma se contemplarán las diferentes opciones que es posible implementar a la hora del diseño. Debemos tener en cuenta los algoritmos existentes y su aplicación, así como las librerías desarrolladas que se ajusten a nuestras especificaciones.
- **Recolección de información**. Es necesario conseguir una fuente proveedora de la información necesaria referente a las muestras de software, esta contendrá los datos necesarios para su estudio. Esta información deberá estar correctamente organizada para la posterior extracción de la información necesaria.
- **Extracción de las características**. Una vez dispongamos de las correctas fuentes de información, será necesario extraer las características relevantes de cada muestra, de esta forma se podrán reconocer los patrones que identifican cada variante.
- **Implementación del sistema**. Se implementará un sistema que funcione de forma automática y que sea capaz de aplicar los algoritmos necesarios a las muestras recolectadas. Este sistema dispondrá de la capacidad de ser alimentado y de almacenar los resultados obtenidos.
- **Visualización de los resultados**. Al disponer de un sistema en funcionamiento, podremos crear los mecanismos necesarios que faciliten la correcta visualización de los resultados obtenidos y permitan la interacción del analista.

1.3. Contenido del documento

El documento será organizado en diferentes apartados en los cuales se contemplará cada uno de los puntos requeridos para la realización del proyecto. El orden de estos apartados intentará ser acorde a los objetivos definidos y contemplando el cualquier caso el estudio del problema, análisis del sistema y la implementación.

Para comenzar, en el estado del arte, se realizará un estudio del problema abordado y se comentará el estado de alguna de las soluciones actuales, así como de las tecnologías necesarias para el desarrollo de la solución.

A continuación, se realizará un detallado desarrollo de la obtención de fuentes necesarias para la recolección de información, así como de los pasos que se han seguido para finalmente poder disponer de una gran cantidad de información organizada y estructurada para su posterior uso. En este apartado también se describirá el proceso de extracción de las características necesarias para el estudio de las muestras.

Una vez dispongamos de una fuente de información y de las características necesarias para alimentar nuestro sistema, se dedicará una sección del documento al desarrollo y la implementación de nuestro sistema. Finalmente se mostrará la aplicación dada a este sistema y se verá en funcionamiento.

El documento concluirá con el estudio de los resultados obtenidos y las comparaciones realizadas en el apartado de pruebas donde haremos varias las diferentes entradas que serán provistas al sistema. También se resumirá la planificación seguida durante el proyecto y se expondrán las conclusiones y posibles trabajos futuros que han surgido.

2. Estudio del problema

2.1. Estudio del arte

2.1.1. Antivirus

Para conocer en detalle en qué consiste un antivirus o a que se refiere cuando hablamos de este nombre, comenzaremos por su definición: Un antivirus es el software encargado de detectar y neutralizar las amenazas presentes en los sistemas informáticos. Este tipo de software es generalmente desarrollado por empresas de seguridad cuyo objetivo consiste en proteger activamente a los usuarios, detectando cuando alguno de estos ha descargado o está utilizando un software malintencionado.

La historia de los antivirus comienza cuando aparecieron por primera vez en los años 80. Desde aquel entonces, estos han evolucionado a lo largo del tiempo añadiendo nuevas funcionalidades y características hasta convertirse en un software indispensable en los equipos informáticos. La forma en la que estos antivirus realizan las detecciones también ha cambiado con el tiempo y es esta la parte interesante donde encontramos el problema que intentaremos resolver.

Figura 1. Detecciones heurísticas y basadas en firmas. [fuente]

Los antivirus han utilizado principalmente dos formas de realizar las detecciones de malware, las firmas y las detecciones basadas en heurística [1]. Además, en los últimos años hemos visto como algunos han comenzado a implementar soluciones basadas en machine learning.

2.1.1.1. Detección basada en firmas

Esta es la forma más básica y fácil de implementar en la detección de amenazas. Las firmas consisten en una especie de definiciones almacenadas en una base de datos donde el software antivirus contiene información acerca de los virus conocidos y que se pretenden detectar. Cuando una nueva muestra de malware es observada por los analistas de seguridad, estos la analizan y observan las características que la identifican, tales como strings o cadenas de texto que contiene la muestra, bytes inusuales en esta, certificados corruptos y demás. Una vez estudiada, los analistas crean la firma en base a estas características y se distribuye mediante actualización a todos los agentes.

A pesar de las ventajas que presentan las firmas en cuando a su rapidez y su sencillez, también presentan algunas desventajas como la pequeña escalabilidad cuando un malware es lo suficientemente modificado u ofuscado como para que la firma no sea capaz de detectarlo. Otra gran desventaja presente es este tipo de detecciones es que la gran cantidad de malware emergente a diario hace que los analistas no puedan generar firmas para cada una de ellas.

2.1.1.2. Detección basada en heurística

Las detecciones basadas en heurística consisten en unas técnicas de análisis de comportamiento mucho más activo que las observadas mediante firmas. De alguna forma se intenta determinar si el software analizado es malicioso observando el comportamiento de este, como tratando de imitar o emular el trabajo que haría un analista a la hora de analizar la muestra de malware.

Mientras se realiza el análisis del software, una serie de indicadores o patrones reconocidos como maliciosos se van cotejando para verificar que el comportamiento es malicioso. Algunos de estos ejemplos pueden ser la creación de procesos, conexiones de red o escritura en los registros del sistema.

Las detecciones basadas en heurística presentan una serie de ventajas respecto a las basadas en firmas. Gracias a este tipo de detecciones las empresas de antivirus pueden "adelantarse" a las nuevas variantes de malware y detectar estas aun cuando han sido modificadas para evadir las detecciones. No obstante, este tipo de detecciones también presenta una serie de desventajas o inconvenientes que hay que tener en cuenta. Por ejemplo, el rendimiento se ve afectado cuando es necesario realizar un análisis dinámico de la muestra y que puede impactar en la velocidad del antivirus. Otro aspecto importante a tener en cuenta son los falsos positivos, es decir, cuando un software es detectado como malicioso, pero en realidad no lo es. La probabilidad de que esto ocurra es mayor al realizar detecciones basadas en heurística, a menudo encontramos software que puede parecer malicioso pero que resulta no serlo.

2.1.1.3. Machine learning

Es evidente que en los últimos años el machine learning (ML) y la inteligencia artificial (IA) han cobrado gran peso en la sociedad e incluso han conseguido resolver o mejorar algunos problemas. Las soluciones basadas en estas técnicas son cada vez más utilizadas en sistemas de seguridad como los filtros de spam, el phishing o las detecciones de malware. Las empresas de antivirus han comenzado a implementar sistemas capaces de reconocer software malicioso utilizando algoritmos de machine learning y que a su vez son capaces de aprender por ellos mismos con el tiempo. De esta forma, el software antivirus es capaz de trabajar de forma automática sin necesidad de actualizar las firmas.

A continuación, veremos algunos de los ejemplos más conocidos de antivirus que utilizan ML y IA en la detección de amenazas.

Cylance es uno de los antivirus más conocidos cuando nos referimos a estas tecnologías, ya que la propia empresa ha conseguido implementar un antivirus basado únicamente en la aplicación de estas técnicas. Este software es capaz de reconocer amenazas y documentos maliciosos en tiempo real gracias a algoritmos machine learning aplicados a cientos de miles de características extraídas de cada fichero y reducidas hasta ser capaz de reconocer los patrones que diferencian a un documento malicioso de otro legítimo.

Figura 2. Logo Cylance. [fuente]

El agente instalado en los equipos es un software ligero y capaz de poner en cuarentena las amenazas detectadas de forma automática, utilizando IA, sin la interacción del usuario. Soluciones como estas evitan la necesidad de mantener un flujo constante de las actualizaciones de firmas con las definiciones de malware.

Windows Defender Security Center es otra solución implementada por Microsoft y la cual también funciona utilizando este tipo de técnicas. Este software cuenta con modelos de machine learning tanto en el cliente como en la cloud [2].

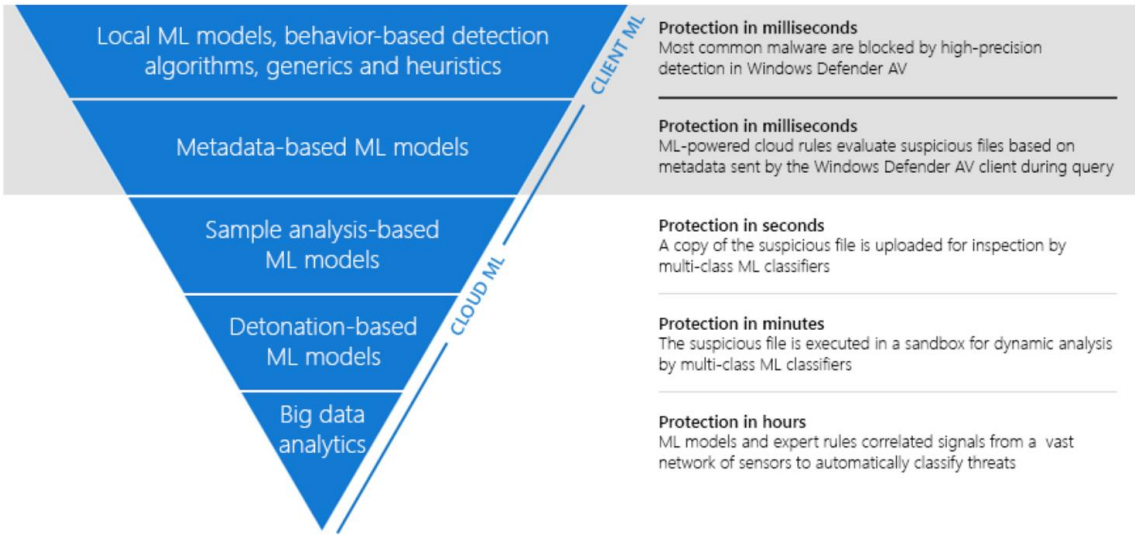

Figura 3. Machine Learning en Windows Defender. [fuente]

La gran mayoría de amenazas son detectadas por el cliente combinando modelos de machine learning con detecciones heurísticos y análisis de comportamiento. El sistema se basa en un modelo compuesto por diferentes capas, tal y como podemos observar en la imagen anterior. A medida que descendemos en la pirámide, los algoritmos y modelos se vuelven más complejos y eso los hace ineficientes en las soluciones basadas en el cliente. Por lo tanto, es necesario utilizar la gran capacidad de cómputo que podemos encontrar en servidores cloud para determinar si un fichero es malicioso o no. Microsoft provee multitud de soluciones basadas en ML dentro de su cloud, una vez el fichero es enviado para su análisis, se le aplican diversos algoritmos para determinar su clasificación, desde modelos lineales hasta deep learning.

2.1.2. Algoritmos de aprendizaje

Cuando hablamos de un sistema basado en machine learning, nos referimos a un sistema capaz de aprender de forma autónoma. Es decir, sistemas que utilizan algoritmos capaces de extraer modelos o utilizar unos datos de entrenamiento para posteriormente poder predecir unos resultados. Entre los diferentes tipos de algoritmos podemos encontrar:

Unsupervised machine learning

Este tipo de algoritmos son utilizados cuando disponemos básicamente de un conjunto de datos de entrada, pero desconocemos el resultado de la salida. El objetivo de estos consiste en encontrar la estructura y distribución de estos datos para poder crear un modelo.

Las aplicaciones más comunes de estos algoritmos se pueden observar a la hora de realizar agrupaciones y en la detección de anomalías. Los datos similares pueden agruparse y formar clusters que contendrán las características comunes de los datos. Un algoritmo de este tipo podría aplicarse sobre un conjunto de datos recolectados sobre muestras de malware y conseguir así agrupar estas muestras por familias.

Supervised machine learning

El algoritmo de aprendizaje supervisado se utiliza cuando conocemos el conjunto de datos de entrada y además conocemos el resultado para estos datos. Podemos utilizar este tipo de algoritmos para predecir los resultados cuando introducimos nuevos datos.

El primer paso consiste en entrenar el algoritmo con los datos conocidos, se construirá el modelo. Después se puede aplicar este modelo a una serie nueva de datos de entrada y predecir el resultado. En la detección de malware, podríamos entrenar un modelo con un conjunto de datos donde conozcamos para cada muestra si es maliciosa o no, posteriormente podríamos intentar predecir este mismo resultado para un nuevo conjunto de muestras en las que no conocemos el resultado de su análisis.

2.1.3. Fuentes de información

Una de las partes más importantes y en la cual se encuentra el potencial de las soluciones basadas en machine learning es en la cantidad y calidad de la información recogida para solucionar el problema. El big data es una nueva tendencia que ha surgido gracias a las grandes recolecciones de información, normalmente suelen ser bases de datos enormes y las cuales contienen millones de entradas. Esta información se puede utilizar para construir los datasets que utilizaremos a la hora de realizar estudios estadísticos y modelos.

En nuestro caso, los datos de interés se centrarán en muestras de malware que contengan información acerca de su comportamiento. De esta forma, podremos identificar los patrones en estos comportamientos y agrupar muestras en base a ello.

Para conocer el comportamiento seguido por un software, es necesario disponer de un entorno en el cual se pueda ejecutar este de forma segura y además se pueda observar o generar un reporte de su actividad. Estos entornos son comúnmente denominados como sandbox y podemos encontrarlos tanto en forma de servicios online o implementados en nuestra propia infraestructura.

A continuación, comentaremos algunas de las soluciones de sandboxing más conocidas donde podremos realizar estos análisis.

Hybrid Analysis es una sandbox online que permite el análisis de muestras y genera un reporte con la información relevante de la actividad. En el reporte podemos observar los antivirus que han detectado la muestra como maliciosa, los indicadores clasificados como maliciosos, sospechosos e informativos, detalles del fichero, actividad de red, procesos, capturas de pantalla, cadenas de texto encontradas en el interior y ficheros escritos en el sistema.

Figura 4. Hybrid Analysis logo. [fuente]

A parte de todo eso, el servicio ofrecido es gratuito y además disponen de un feed público que nos permitirá obtener los reportes de las muestras analizadas por otros usuarios en todo el mundo.

ANY RUN es otro servicio que provee una sandbox online con entornos virtuales en los que se pueden analizar muestras de malware. Al igual que Hybrid Analysis, este servicio es capaz de generar el reporte de la actividad donde podemos encontrar las características de la muestra [3]. Podemos encontrar un plan gratuito y otros más avanzados en los cuales dispondremos de más funcionalidades.

Figura 5. ANY RUN logo. [fuente]

Cuckoo sandbox es probablemente el software open source más conocido en cuanto a análisis dinámico de malware. Con él podemos montar nuestro propio entorno con las máquinas y versiones de sistemas operativos que deseemos. Al ejecutar las muestras obtendremos el reporte de la actividad y podremos extraer las características que nos resulten interesantes.

Figura 6. Cuckoo sandbox logo. [fuente]

2.1.4. Librerías existentes

Una vez conocidos los tipos de algoritmos de machine learning que se pueden aplicar al análisis de malware y las técnicas que deseamos implementar en nuestro sistema, como la visualización gráfica de los resultados, podemos estudiar los desarrollos actuales que nos permitan realizar el trabajo.

Graphviz es un software open source de visualización de gráficos. Este software nos permite obtener una representación gráfica de los resultados obtenidos tras realizar un estudio sobre un gran conjunto de datos. Normalmente es difícil extraer conclusiones claras cuando trabajamos directamente con la base de datos o la salida por consola de un programa. Por lo tanto, un software que nos permita plasmar esa información en un formato visual será de gran ayuda en el análisis del problema.

Scikit-learn es una de las librerías de Python más utilizada en el ámbito del machine learning. Con esta librería podemos aplicar diferentes modelos de machine learning a nuestro problema, desde regresiones lineales hasta arboles de decisión. Su uso es muy sencillo y tan solo nos será necesario preparar los datos de entrada para que la librería pueda trabajar correctamente.

Datasketch es una librería de Python que permite obtener modelos estadísticos y probabilísticos en un conjunto grande datos de forma rápida y eficaz. Esta librería cuenta con importantes implementaciones tales como MinHash, Local Sensitive Hashing LSH, LSH Forest, Weighted MinHash, HyperLogLog, HyperLogLog++ que pueden ser interesantes en la realización del proyecto.

3. Análisis del sistema

Nuestro problema surge cuando nos encontramos con una gran cantidad de muestras de malware y necesitamos clasificarlas dependiendo de sus características, estas agrupaciones serán denominadas familias o clusters. La imposibilidad de realizar estas clasificaciones manualmente aparece cuando disponemos de miles de muestras y un tiempo limitado hasta que aparezcan nuevas, ya que la distribución de malware es constante y necesita de este proceso diariamente.

En este apartado se realizará un análisis detallado del sistema propuesto en cada uno de sus módulos.

3.1. Descripción general

La solución que se propone en este documento consiste en un sistema capaz de agrupar muestras de malware en base a sus características. Debido a que el funcionamiento del sistema necesita de una fuente de información que alimentará continuamente el conjunto de muestras disponibles, el sistema requerirá de ciertos módulos para la recolección y extracción de características.

Así mismo, el siguiente módulo se encargará de procesar e indexar la muestra con sus correspondientes características en una base de datos indicando el cluster correspondiente al que pertenece. El módulo para la visualización será el que utilice el usuario para hacer uso del sistema.

Las funcionalidades de cada módulo se basan en:

- **Módulo de recolección**: La funcionalidad de este módulo será la de conectar con servicios de sandboxing externos en los cuales se puedan obtener reportes sobre la ejecución de muestras de malware.
- **Módulo de extracción**: En este módulo se procesarán los reportes obtenidos del módulo anterior y se extraerán las características relevantes de cada muestra. Estas características serán almacenadas para su posterior uso.
- **Módulo de clusterización**: Será necesario inicializar la base de datos cuando el sistema comience a funcionar. Una vez inicializada, este módulo será el encargado de recibir el flujo de muestras y las características extraídas de cada una de ellas, realizando los algoritmos necesarios para la determinación del cluster al que pertenece la muestra y la posterior indexación en la base de datos.
- **Módulo de visualización**: Una vez dispongamos de una base de datos donde podamos consultar las diferentes muestras de malware agrupadas en clusters, se construirá una aplicación web que permita el acceso a esta información y facilite el estudio de los clusters al analista.

En la siguiente imagen podemos observar el esquema global del sistema con los distintos módulos que lo componen.

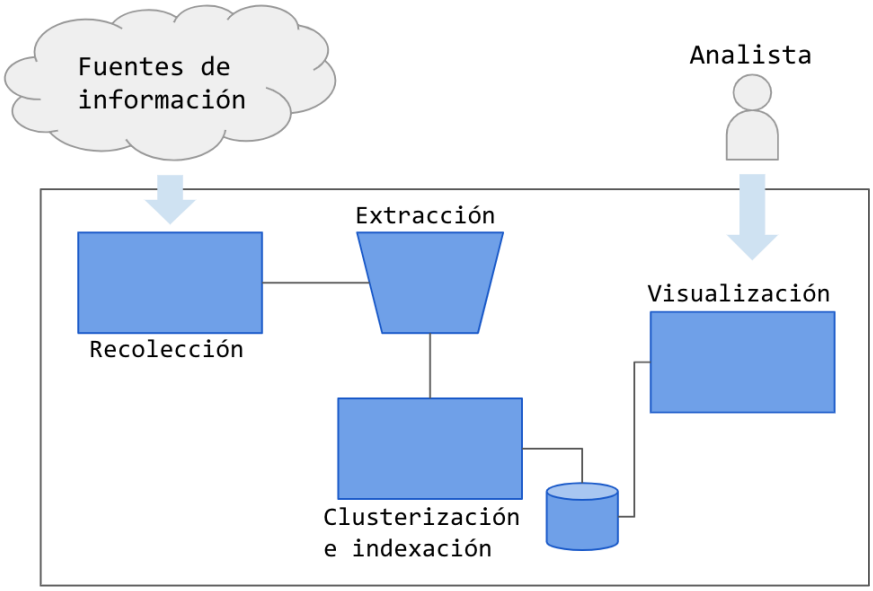

Figura 7. Módulos que componen el sistema. [fuente propia]

3.2. Obtención de fuentes

En este apartado se detalla el proceso seguido para la obtención de fuentes de información. Durante el desarrollo del proyecto nos centraremos en la recolección de muestras disponibles en servicios online de sandboxing.

El funcionamiento de este módulo consistirá en un pequeño programa capaz de crear una conexión, cada cierto intervalo de tiempo, con el servicio deseado y descargar el fichero que contiene la información del análisis de la muestra.

Hybrid Analysis ofrece un feed público en https://www.hybrid-analysis.com/feed?json, tan solo es necesario descargar el fichero en formato JSON y este contendrá varios análisis en su interior. Como podemos observar en la siguiente imagen, cada análisis nos ofrece información muy útil sobre el comportamiento de las muestras, en el próximo apartado veremos cómo extraer estas características.


```
{
    "md5": "09caba43d8ceec998217cf8103e5879e",
    "sha1": "e43d380b8340ed54b559c62270246123a3917d40",
    "sha256": "4ca867e34875012e2c47f54dab9c44b178cc52c24ac0e834315f6946404d2bc1",
    "tags": ["bladabindi", "rat"],
    "isinteresting": false,
    "analysis_start_time": "2018-12-24 15:03:33",
    "threatscore": 100,
    "threatlevel": 2,
    "threatlevel_human": "malicious",
    "avdetect": 85,
    "isunknown": false,
    "vxfamily": "Generic.MSIL.Bladabindi",
    "submitname": "09caba43d8ceec998217cf8103e5879e",
    "isurlanalysis": false,
    "size": 29696,
    "type": "PE32 executable (GUI) Intel 80386 Mono\/.Net assemb ...",
    "et_alerts_total": 2,
    "domains": ["emad1987.myq-see.com"],
    "hosts": ["141.255.157.17", "95.101.13.56", "2.17.213.199"],
    "hosts_geo": [{ ... }, { ... }, { ... }],
    "compromised_hosts": ["141.255.157.17"],
    "et_alerts": [{ ... }, { ... }],
    "environmentId": "110",
    "environmentDescription": "Windows 7 32 bit (HWP Support)",
    "isreliable": true,
    "reporturl": "\/sample\/4ca867e34875012e2c47f54dab9c44b178cc52c24ac0e834315f6946404d2bc1\/5c20e5437ca3e168557a6278",
    "vt_detect": 85,
    "ms_detect": 85,
    "process_list": [{ ... },
        {
            "uid": "00016415-00002144",
            "parentuid": "00016055-00003616",
            "name": "Trojan.exe",
            "normalizedpath": "%TEMP%\\Trojan.exe",
            "commandline": "",
            "sha256": "4ca867e34875012e2c47f54dab9c44b178cc52c24ac0e834315f6946404d2bc1",
            "av_label": "Generic.MSIL.Bladabindi",
            "av_matched": 61,
            "av_total": 71
        }, { ... }
    ],
    "extracted_files": [{
            "name": "Trojan.exe",
            "file_path": "%TEMP%\\Trojan.exe",
            "file_size": "29696",
            "type": "PE32 executable (GUI) Intel 80386 Mono\/.Net assembly, for MS Windows",
            "type_tags": [ ... ],
            "threatlevel": 2,
            "threatlevel_readable": "malicious",
            "av_label": "Generic.MSIL.Bladabindi",
            "av_matched": "61",
            "av_total": "71",
            "targetname": "09caba43d8ceec998217cf8103e5879e.exe"
        }, { ... }
    ]
}
```


Figura 8. Ejemplo de reporte en formato JSON. [fuente propia]

El CERT de Estonia dispone de un cuckoo online el cual ha sido posible utilizar para analizar muestras y obtener reportes. El servicio es abierto y se puede acceder en https://cuckoo.cert.ee/analysis/#.

VMRay es un servicio premium de pago que ofrece una sandbox bastante completa en la cual podemos realizar análisis de malware. En este caso bastará con recolectar los análisis que publican abiertamente en redes sociales como Twitter. También podremos realizar una búsqueda en Google para ver cuántos reportes han sido indexados. Algunos ejemplos de muestras obtenidas de este servicio:

```
Emotet: https://www.vmray.com/analyses/20c4ab220a6e/report/overview.html
Arkei/Vidar: https://www.vmray.com/analyses/d6a479704bd6/report/overview.html
Gootkit:
https://www.vmray.com/analyses/evasive-gootkit-banking-trojan/report/overview.html
```

3.3. Extracción de características

La extracción de características es la base fundamental y más crítica cuando hablamos de machine learning, los algoritmos trabajarán con estas características y la variación de estas influirá en los resultados obtenidos. Se deberá procesar los reportes obtenidos del módulo comentado en el apartado anterior y extraer los datos del comportamiento que sean relevantes a la hora de decidir si una muestra es parecida a otra, tal y como lo haría un analista.

Nuestra solución intenta conseguir agrupar diferentes muestras de malware en subconjuntos más pequeños o clusters. Para poder realizar esto es necesario que las features seleccionadas del análisis sean lo suficientemente buenas como para poder identificar los patrones que identifican un comportamiento en concreto. Por ejemplo, dos muestras de la misma familia de malware y que pertenecen a la misma campaña, tienen una alta probabilidad de compartir los dominios contactados por estas. No obstante, estos dominios seguramente cambien al poco tiempo, por lo que no debemos basarnos simplemente en ese criterio y aumentar las features que vamos a utilizar.

Tras observar los diferentes reportes podemos ver cómo hay ciertas características que no varían o lo hacen mínimamente a medida que la familia de malware evoluciona. Este es el caso de los procesos ejecutados en el sistema y los ficheros escritos en disco. Por lo tanto, podemos añadir estos comportamientos a la lista de features.

Feature	Descripción
`domains`	Lista de dominios contactados por la muestra de malware.
`processes`	Lista de procesos ejecutados en el sistema.
`files`	Lista de ficheros escritos en el disco.

Tabla 1. Features seleccionadas para el inicio del estudio.

Una vez hemos definido las features que vamos a utilizar para alimentar el sistema es el momento de la implementación. El módulo comenzará con un programa capaz de leer los reportes descargados de los servicios de sandbox. Debido a que todos los servicios que hemos utilizado presentaban un formato de reporte basado en ficheros JSON, la extracción de features consistirá en simplemente seleccionar los valores de las claves deseadas.

```
File: a4b492840299c3435b6edadf96fd4b572f8993d97e3fda00d912ba554a84e8ae

Size: 68096

File type: Word document

Domains:

     grupoperezdevargas.com

Processes:

    %PROGRAMFILES%\Microsoft Office\Office14\WINWORD.EXE
    %WINDIR%\System32\cmd.exe
    %WINDIR%\System32\WindowsPowerShell\v1.0\powershell.exe
    %TEMP%\427.exe
```

```
    %TEMP%\427.exe
 Files dropped:

    %TEMP%\427.exe
    %APPDATA%\Microsoft\Office\Recent\FILE0VC22598.LNK
    %APPDATA%\Microsoft\Office\Recent\index.dat
    %APPDATA%\Microsoft\Windows\Recent\CustomDestinations\BNZV4DDP8RJL4LDF
VZ2E.temp
```

Como podemos observar, en el esquema anterior se representa la información relevante de un reporte, se han resaltado las features que extraeremos para formar nuestro conjunto, quedando de la forma:

```
{'grupoperezdevargas.com',    '427.exe',    'cmd.exe',    'file0vc22598.lnk',
'winword.exe', 'index.dat', 'powershell.exe', 'bnzv4ddp8rjl4ldfvz2e.temp'}
```

3.4. Clusterización

Este apartado es el motivo principal del proyecto y en el cual se detalla el proceso, así como los algoritmos y técnicas utilizadas para conseguir realizar la agrupación de las muestras. Los datos de entrada que se introducen a este módulo son básicamente las features de las muestras de malware colectadas anteriormente.

A la hora de agrupar muestras será necesario hacer comparaciones entre ellas y obtener el grado de similitud. Este proceso consiste en recorrer todo nuestro conjunto de muestras e iterar sobre ellas dos a dos comparando sus features. El grado o coeficiente de similitud que obtenemos es conocido como **índice de Jaccard**. Este término estadístico es utilizado cuando se mide la similitud entre dos conjuntos finitos, en nuestro caso las features de las muestras. El cálculo del índice de Jaccard es igual a la división del tamaño de la intersección de los conjuntos entre el tamaño de la unión de ambos.

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

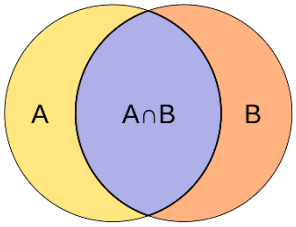

Figura 9. Intersección de dos conjuntos. [fuente]

Podemos implementar una función en nuestro código que calcule el índice de Jaccard dados dos conjuntos diferentes de features.

```
def jaccard(set1, set2):
     return float(len(set1.intersection(set2)))/float(len(set1.union(set2)))
```

Una vez implementada, tomaremos como ejemplo dos muestras diferentes pero pertenecientes a la misma familia y calcularemos el índice de Jaccard entre ambas para comprobar su similitud.

```
Sample1: ef0db47cebafe2a19c37e6c67d096de29875bb273364c146d72605c343810b8d

{'osslusturv.com', 'winword.exe', 'powershell.exe', '41e49512.exe', 'cmd.exe'}

Sample2: 4e745ee0d3286802fa5d73922e9735a62e040fec773cda0d7e5dafea85fbfe9b

{'osslusturv.com', 'dc469ec3.exe', 'winword.exe', 'powershell.exe', 'cmd.exe'}

Índice de Jaccard obtenido: 0.666 -> 67% de similitud
```

Cómo podemos observar, ambas muestras comparten ciertas features entre sí y por lo tanto obtenemos un índice de Jaccard mayor que 0.

El siguiente experimento que realizaremos será el de elegir una muestra al azar y calcular la verosimilitud con cada una de las muestras pertenecientes a un conjunto mayor. Los resultados con un índice mayor que 0 indicaran que las muestras comparten características y, por lo tanto, que podrían pertenecer al mismo cluster.

```
 Sim. Sample
 ---- ------------------------------------------------------------------
 0.67 4e745ee0d3286802fa5d73922e9735a62e040fec773cda0d7e5dafea85fbfe9b
 0.50 cdd840901672416a3ab4ccb8d6fc5d96e6d350006b2f5cd62033827117e6b1f6
```

```
0.43 ac3fbf02333d2b7b1bc821493d0532bb2a30a0bfc9343e9c690185e072ad7819
0.43 608c215893b99203b2d355253d42b14fe0bae98b22a891cfa2950c79d8b4dfe1
0.30 649e67f86adcacc3122e01bb922af166f6c15dd727e7acaf7bcefb9810739fb4
0.27 ba157ba0994e2444f188a09b6ceea2e09c5b62389c95290a6df2b1529240b3d3
0.27 3a063820969e256ec22f1902e525b7a213b5369cf58d8771f0919fcaa8fa5812
0.27 37e4a6a266f2c2605e8b5c8923512fde8518b3a36fadac8128c15dcf1aa4dd6d
0.25 ec9d26f29e14e8645faa93c0d2062baf75e19db7cbf42c1cecd44a83c413938f
0.23 fa2bd6f53ab4326aa404519807e7ee4e62809f935313105e6e893347985ac453
0.23 3f0682df796b3893938cec1d83909cc56b4f9107f1c0c40f05cb0dd68c32fdad
0.22 578e9d352e2b4fbaa5a540b8357e994e6c14330ae8193e84554c9ac6c1df7412
0.21 c9dbc841e4ad55c500cccaf4526ef40e5c07179f1579d2a5f199ef52144caa20
0.21 aec7b3af31247fd166c4af11606d48d05656580e7b55cf6a61ec9a7631753b4b
0.20 d1cd4900ab9cae0888c037c979410b20fac46a633b25a10d07d41133c0a791c5
0.20 cdd245ad3fcc0f29d35c18f78f8e0a12ccead958df91d16be3a3f4dcdee95c91
0.20 bec4333610d7bc5651c1edaead6d8463fc1afc4261f3b2f25a8338ccfb6b14ab
...
```

De esta forma, podemos conocer lo similar que es una muestra al resto de las contenidas en el *dataset*. Como se puede observar, la lista ha sido ordenada y por lo tanto la muestra con mayor similitud se encuentra en primera posición. Tras realizar un estudio manual de estos resultados, observaremos que existen muestras que presentan un cierto índice de similitud pero que no pertenecen a la misma familia. Estos casos se encuentran cuando bajamos a partir de una similitud del 20% para este escenario, es decir, si deseamos agrupar las muestras en subconjuntos deberemos establecer un valor de similitud como umbral (threshold).

El proceso que seguiremos a la hora de agrupar todas las muestras de un conjunto en diferentes clusters consistirá en la iteración de todo el conjunto y la comparación de muestras dos a dos para calcular su índice de Jaccard. Los resultados de este proceso pueden ser confusos si se leen directamente desde la salida del programa. Para ello utilizaremos representaciones gráficas que nos permitan un estudio más genérico.

NetworkX es una librería open source de Python que nos permitirá construir y manipular redes complejas. Utilizaremos la librería en nuestro código para representar los clusters formados del conjunto de muestras. Los nodos representarán las muestras y las uniones representarán la relación que existe entre muestras debido a su similitud. Además, utilizaremos GraphViz, ya comentado en el estado del arte, para la visualización del grafico resultante.

La función para iterar sobre las muestras y construir el gráfico será la siguiente. Esta generará un fichero “clusters.dot”.

```
def cluster_graph(dataset):
    graph = networkx.Graph()
    for sample in dataset:
        graph.add_node(sample)
    for sample1, sample2 in itertools.combinations(dataset, 2):
        jaccard_index = jaccard(dataset[sample1], dataset[sample2])
        if jaccard_index > 0.2:
            graph.add_edge(sample1, sample2, penwidth=7)
    write_dot(graph, "clusters.dot")
```

Y después podemos ejecutar el siguiente comando usando GraphViz para obtener una imagen.

```
>fdp -Tpng clusters.dot -o clusters.png
```

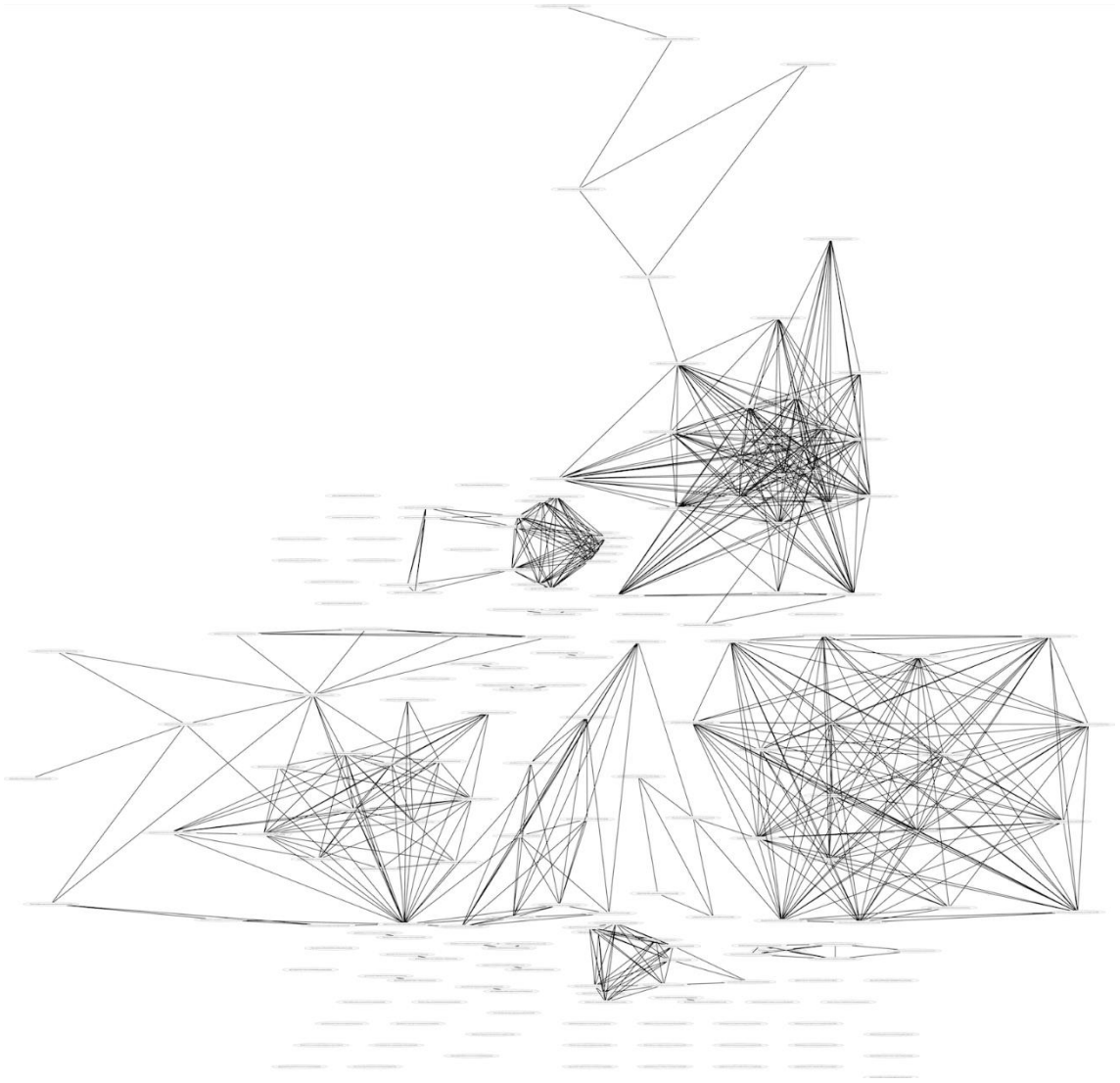

Figura 10. Clusters generados para el conjunto de muestras. (threshold = 0.2) [fuente propia]

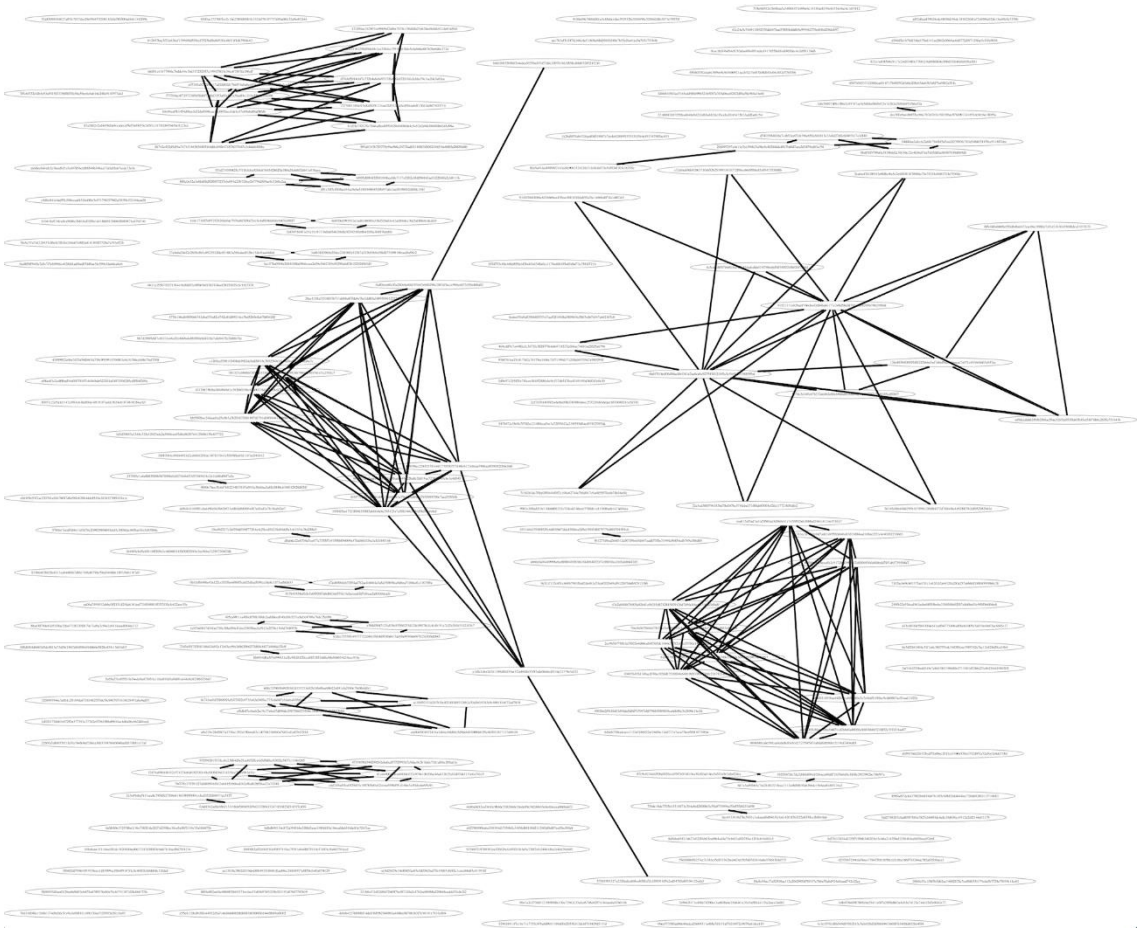

Figura 11. Clusters generados para el conjunto de muestras. (threshold = 0.4) [fuente propia]

Los resultados demuestran cómo el tamaño de los clusters varía con respecto al umbral establecido. Utilizando este parámetro se puede ajustar el sistema a nuestras especificaciones y obtener los resultados deseados.

No obstante, si queremos implementar este proceso en un sistema que funcione de forma automática y el que el número de muestras totales con las que se trabajará será mucho mayor, encontraremos un problema de complejidad no lineal. Iterar el conjunto de muestras dos a dos y calcular su similitud es viable únicamente en conjuntos reducidos y que no contienen cientos de miles de muestras.

Por ejemplo, en el experimento anterior se ha utilizado un conjunto de aproximadamente 200 muestras de malware donde el tiempo de computo ha sido despreciable. El problema es que el número de iteraciones necesarias para recorrer todo el conjunto aumenta de la forma:

$$nº\ iteraciones = \frac{n^2 - n}{2}$$

$$nº\ iteraciones\ para\ 200\ muestras \rightarrow \frac{200^2 - 200}{2} = 19900$$

Para valores de n grandes la formula se puede escribir como $\frac{n^2}{2}$, con lo cual, queda claro que no es la función de una recta sino de una parábola. No es recomendable implementar un sistema que no siga una complejidad lineal, ya que existirá un determinado momento en el cual el sistema se volverá ineficiente y agotaremos la capacidad de computo.

$$nº\ iteraciones\ para\ 1000\ muestras \rightarrow \frac{1000^2 - 1000}{2} = 499500$$

$$nº\ iteraciones\ para\ 100000\ muestras \rightarrow \frac{100000^2}{2} = 4999950000$$

Si queremos solventar este problema debemos aplicar alguna técnica que nos permita reducir el tiempo de computo. Una solución que podemos aplicar en este caso es la aplicación de **minhash**. Esta técnica nos permite obtener una aproximación al índice de Jaccard de forma mucho más eficiente y rápida.

La técnica de minhash consiste en aplicar k veces una función de hash a todas las features pertenecientes a una muestra. Por cada iteración de los k valores, ordenaremos todos los hashes resultantes y almacenaremos únicamente el de menor valor. Así, tras realizar el proceso completo obtendremos un conjunto de k hashes.

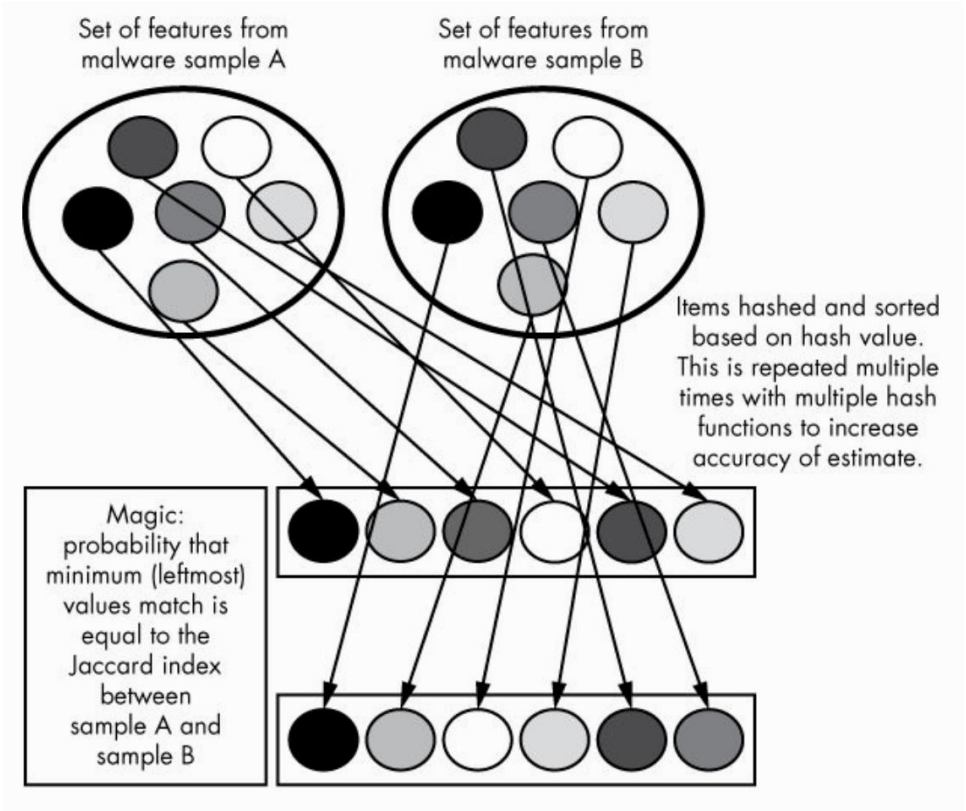

Figura 12. Aplicación de minhash. [fuente]

Una vez calculados todos los minhashes de la muestra, para obtener la aproximación del índice de Jaccard tan solo será necesario contar el número de hashes que coinciden con los minhashes de otra muestra y dividirlo entre k. La función implementada en el código será:

```
def jaccard_minhash(minhashes1, minhashes2):
    return (minhashes1 == minhashes2).sum() / float(MINHASH_K)
```

Los minhashes calculados usando $k = 16$ para las dos muestras utilizadas anteriormente en el estudio del índice de Jaccard serán:

```
[995477475, 250258220, 1439663527, 1105987225, 199906460, 68607355,
302883127, 10710982, 566603499, 415939851, 763188940, 584230738,
1725075463, 416883330, 1315267888, 599805747]

[584908856, 250258220, 964513170, 1105987225, 199906460, 68607355,
302883127, 10710982, 550402247, 666400445, 763188940, 584230738,
1211979044, 416883330, 1315267888, 599805747]
```

El resultado obtenido tras estimar el índice de Jaccard utilizando minhashes es 0.6875. Un 68% de similitud entre las muestras, comparado con el 67% obtenido con el índice real. Como era de esperar, aunque usando esta técnica ganemos en tiempo de computo, también debemos tener en cuenta el error introducido a la hora de estimar el índice. La aproximación de este error se puede expresar como:

$$\frac{1}{\sqrt{k}}$$

Cuanto mayor sea k menor será el error de esta estimación, pero mayor será la cantidad de hashes necesarios para calcular el minhash y por lo tanto aumentaremos la capacidad de computo.

$$k = 64 \rightarrow error = \frac{1}{\sqrt{64}} = 0.125 \rightarrow 12.5\%$$

$$k = 128 \rightarrow error = \frac{1}{\sqrt{128}} = 0.088 \rightarrow 8.8\%$$

$$k = 256 \rightarrow error = \frac{1}{\sqrt{256}} = 0.0625 \rightarrow 6.2\%$$

El valor $k = 256$ nos ofrece una buena relación entre un error relativamente bajo y una computación eficiente y rápida. Será este el que utilizaremos para implementar nuestro sistema basado en minhashes.

Otra técnica muy importante a la hora de implementar un sistema con complejidad lineal que pueda escalar a medida que crece el tamaño de muestras es la indexación en base de datos de los minhashes. De esta forma, cuando una nueva muestra entre en el sistema, se calcularán sus minhashes y se calculará su similitud únicamente con las que compartan algún hash en común. Esto nos permite ahorrar las iteraciones con todas las muestras del conjunto y disminuir drásticamente el tiempo utilizado a la hora de intentar agrupar una muestra en su cluster.

Para realizar esta indexación en base de datos utilizaremos una técnica denominada como **sketching** y que nos ayudará a comparar solo las muestras que tienen gran probabilidad de compartir características. Los sketches son simplemente hashes de los minhashes, de forma que, dividiremos el conjunto de minhashes en grupos más pequeños y aplicaremos una función de hash sobre estos grupos. Así se obtendrá un número más reducido de hashes por cada muestra y facilitará su indexación y búsqueda en la base de datos. Esta técnica se fundamenta en la colisión que se produce entre hashes cuando las muestras comparten ciertas características.

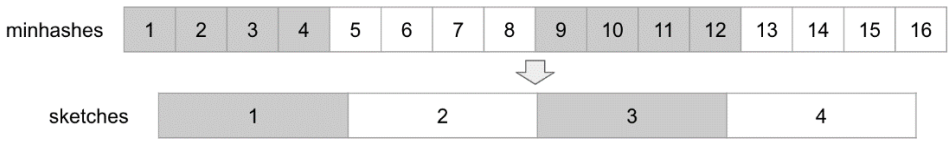

Figura 13. Aplicación de sketches. [fuente propia]

3.5. Base de datos

Como se ha comentado anteriormente, el sistema utilizará una base de datos para almacenar los sketches y disminuir así el tiempo de computo al agrupar las muestras. El esquema que seguirá la base datos estará compuesto por 3 tablas diferentes: una tabla para los sketches, una tabla para las muestras y una tabla para los clusters.

A continuación, se definirá el esquema de las tablas:

sketches		
Clave	**Tipo**	**Descripción**
`sketch`	INTEGER PRIMARY KEY	Clave primaria y única que almacenará cada uno de los sketches generados por las muestras.
`samples`	TEXT	Esta clave almacenará cada una de las muestras que han generado un sketch específico.
`cluster`	TEXT	Cada sketch tendrá un cluster asociado. Esta clave relaciona los diferentes clusters con los sketches.

Tabla 2. Esquema tabla sketches.

samples		
Clave	**Tipo**	**Descripción**
`sample`	TEXT PRIMARY KEY	Clave primaria y única que almacenará cada una de las muestras que contiene el conjunto total.
`minhash`	TEXT	Esta clave almacenará el minhash generado para cada una de las muestras del conjunto.
`features`	TEXT	Esta clave almacenará todas las features extraídas para cada una de las muestras del conjunto.
`date`	DATE	Fecha en la que se ha indexado la muestra en la base de datos.

Tabla 3. Tabla de muestras.

clusters		
Clave	**Tipo**	**Descripción**
`cluster`	TEXT PRIMARY KEY	Clave primaria y única que almacenará cada uno de los clusters generados a partir de los sketches.
`date`	DATE	Fecha de la creación del cluster en la base de datos.
`tag`	TEXT	Clave que almacenará una etiqueta identificativa en el cluster para facilitar su estudio.

Tabla 4. Tabla de clusters.

El cometido principal de la tabla de sketches será el de almacenar el conjunto global de todos los sketches generados a partir de aplicar minhashes a las features de las muestras y después realizar sketching sobre estos. De esta forma, la base de datos dispondrá de una visión global de todos los sketches generados y que a su vez estarán relacionados con las muestras y los clusters. Por cada entrada en esta tabla existirá un sketch, con las correspondientes muestras cuyos sketches han colisionado y el cluster al que pertenece el sketch.

La tabla de samples y la de clusters aportan información adicional a la tabla anterior. En la tabla de samples podemos encontrar las features en formato minhash y original que correspondes a cada muestra, así como la fecha en la que la muestra fue recolectada. La tabla de clusters nos permite añadir etiquetas para identificar clusters más fácilmente y mejorar el trabajo de analista, esta tabla también almacenará la fecha en la que fue creada el cluster.

4. Implementación del sistema

En este apartado del documento se detallará una sencilla implementación del sistema que se ha realizado para poder llevar a cabo pruebas de funcionamiento y estudios de la viabilidad. Cabe destacar que la implementación del sistema dependerá del uso que se quiera dar al mismo, en este caso veremos una implementación en un ordenador portátil de uso personal, pero sería también posible utilizar un servidor o cualquier sistema capaz de llevar a cabo los cómputos necesarios.

4.1. Inicialización de la base de datos

La base de datos que se utilizará será una instancia en local utilizando SQLite ya que es ligera y nos permite realizar las operaciones necesarias de forma rápida y eficaz. Además, utilizaremos la librería open source sqlite3 de Python para trabajar con ella desde el código.

Tras crear nuestra base de datos bajo el nombre de "*malware_analysis.db*" debemos crear las tablas necesarias que han sido comentadas en el apartado anterior.

```
CREATE TABLE sketches
(sketch INTEGER PRIMARY KEY, samples TEXT, cluster TEXT);

CREATE TABLE samples
(sample TEXT PRIMARY KEY, minhash TEXT, features TEXT, date TEXT);

CREATE TABLE clusters
(cluster TEXT PRIMARY KEY, date TEXT, tag TEXT);
```

Quedando una base de datos de la forma:

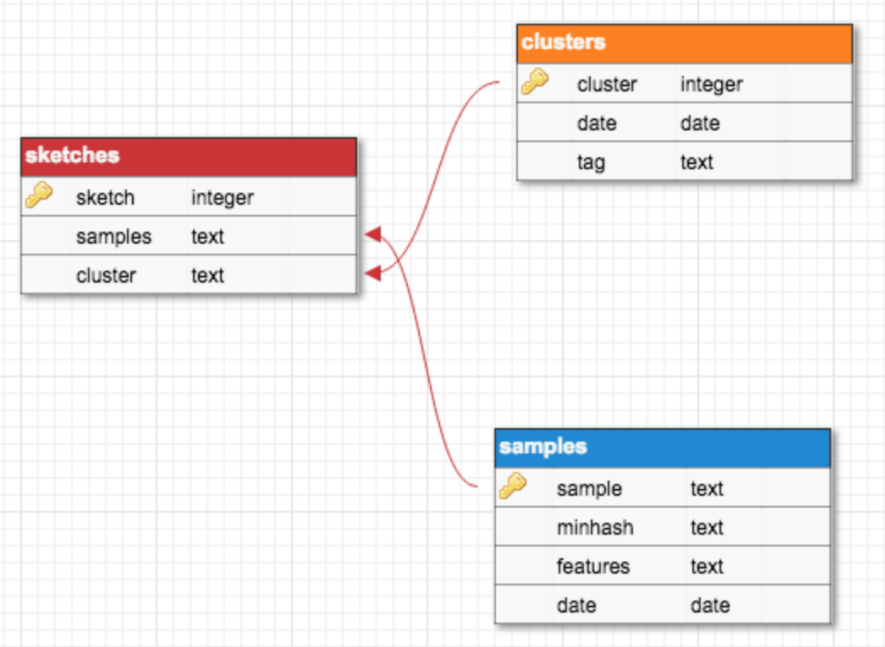

Figura 14. Base de datos inicializada. [fuente propia]

4.2. Indexación de las muestras

Una vez inicializada la base de datos, el sistema deberá procesar las muestras y calcular los minhashes y sketches. La forma en la que esté decidirá a que cluster debe pertenecer una muestra debe ser la siguiente.

```
 def to_cluster(sample, minhashes, sketches, features, date, db):
     db.execute("SELECT cluster FROM sketches WHERE sketch IN (%s)" %
','.join('?'*len(sketches)), sketches)
     clusters = db.fetchall()
     if clusters:
          stats = collections.Counter([i[0] for i in clusters])
          cluster = max(stats, key=stats.get)
          store_sample(sample, minhashes, sketches, str(cluster),
False, features, date, db)
     else:
          cluster = uuid.uuid4()
          store_sample(sample, minhashes, sketches, str(cluster),
True, features, date, db)
```

La función `to_cluster` será llamada cada vez que una muestra entre al sistema, a ella se le pasará como parámetro: la muestra, los minhashes, los sketches, las features, la fecha de la muestra y el conector a la base de datos. El parámetro más relevante es el de los sketches ya que con estos se consultará la base de datos, en caso de no devolver ningún valor, se creará un identificador de cluster nuevo y se llamará a la función `store_sample`. Esta función indexará los nuevos sketches junto a la muestra y el nuevo cluster, también insertará los datos correspondientes en el resto de las tablas. En caso de que la consulta por sketches devuelva algún valor, significará que ya existe uno o varios clusters que comparten similitud, por lo tanto, elegiremos aquel con mayor número de ocurrencias para los sketches de la nueva muestra y añadiremos estos al cluster existente.

Al ejecutar el sistema por primera vez, el rango de nuevos clusters creados será mayor ya que inicialmente no existe ninguno y los que se creen tendrán pocos sketches asignados. Esto no es problema debido a que el sistema será capaz de mejorar por cuenta propia a medida que se añadan más muestras a la base de datos. Esta característica nos permite conseguir un sistema autónomo y escalable.

En el siguiente ejemplo se llevará a cabo una prueba del sistema donde se indexará un conjunto de unas 100 muestras sobre una base de datos donde ya se ha indexado un conjunto mayor.

```
c5eb8b8e30b593d964c9e2cc0574b72d4f159b83ca64f224d2ace72e4a530dd7
     New cluster: 41687148-a997-48f2-9d52-0b59569e860f
7c5aecf5828b7d9847e0300b26f81eb96ba37f1056781bae38c632f6c761b70d
     Added to cluster: d6910fdd-c177-4de7-a07d-18b45bef73ba
ce3730c4bfdf77f292e6503a46b1dafb31abc90cb1497e9cfab2f61c639a2c1e
     Added to cluster: d0c203ee-8491-40ca-a8cb-ef94e2da9720
bb28b3c146f854a5b6886a7af0bd295a442be920428376315d3bb77d52550531
     Added to cluster: d6910fdd-c177-4de7-a07d-18b45bef73ba
079b6b2dfb4f45be0755e2de6d7b581d8086f9f2254891a9f0a8e0ac37827407
     New cluster: f8e7fd81-220d-4935-b7f2-f65263c37752
1a17c231dfa00427f6023768f3dafc009eb14ec971c18516e750f0df0593486b
     Added to cluster: d6910fdd-c177-4de7-a07d-18b45bef73ba
22e87093640d97c395f53ea8429fec6766b588f4cc3e8c1774ad32cf303016ec
```

```
    Added to cluster: 1eedcc99-5a10-459c-91bd-cce2794a4272
0844761b0a4a73440e32f2d2f553f95c03ee0cdbaaaca00a054251fa70cf5598
    Added to cluster: d7c70875-9926-4e4e-ba87-5bedd53a0fa0
2b9ec1a464323053003e3d4c234b1b5dc98c9491e04eb296c894cddfcada4cd0
    New cluster: 380af460-9d25-4ba3-bec8-42d4e96a6157
1d88b43b26b778f406dd06eee8d4eec83ddfec09888db71d7f3a19f23d37d0b4
    Added to cluster: 23c8c7f5-1b10-4c3d-bf7f-378e03f043a6
faa1393c5d5c5596e353d61361f836145fde1f0a9ca99374347044bf04a4a2a0
    Added to cluster: 23c8c7f5-1b10-4c3d-bf7f-378e03f043a6
1ddc3574b61ff5358791b3bf6f28ea1f2b468da438d165df35ba2083108e4421
    Added to cluster: 8c523c24-60e4-4436-bbbc-aa7d5cafcfc6
a66127069f4125d2e495d0e16c54c91573899e7bfd62cbac660b265a43e63faf
    Added to cluster: a1260f45-8fdd-490c-a307-01a647ba5ef7
45cc99e4099fdbd3bcab63d0617c1d05fa655739421aec8630ac702192bf184f
    New cluster: 05b792ef-200e-4dce-9dc8-9619bc095b0d
e8e7a3f8ecb2a0a3eb097371ca2c1fd79044b993310ec190e69886a97df49fdd
    New cluster: 4ecd40d0-0664-49ae-9d7e-6ddac62de6ad
25695eaaa49838b7289784c90a1f2c3d1d0502d8d3e6c0fbbb25243c9cdfb568
    New cluster: 3ef5cc4e-d555-4476-8607-ffee13a11f4e
   ...
```

Como se puede observar el sistema ha sido capaz de reconocer las muestras que presentan cierta similitud con muestras ya conocidas y las ha agrupado en su cluster correspondiente de forma autónoma. El resultado obtenido nos indica que el sistema ha sido capaz de agrupar el 50% de las muestras en clusters ya existentes y por lo tanto que esto reduciría el trabajo del analista.

En el apartado de pruebas se realizarás más ejecuciones para comparar resultados y tiempos.

4.3. Visualización del sistema

La visualización del funcionamiento del sistema y por lo tanto de los resultados, es un módulo más del diseño. Este nos permitirá conocer el rendimiento obtenido y la eficacia del sistema, así como mejorar el estudio de las muestras a la hora del análisis.

Para llevar a cabo la visualización se ha empleado un dashboard HTML que funcionará sobre una aplicación web. Esta aplicación utilizará la base de datos del sistema para devolver los resultados deseados e implementará las consultas mediante código PHP.

Gracias a este dashboard podemos tener un control sobre las muestras indexadas en la base de datos, los clusters creados, sketches generados para todas las muestras y etiquetas otorgadas a los clusters para el estudio de las familias de malware.

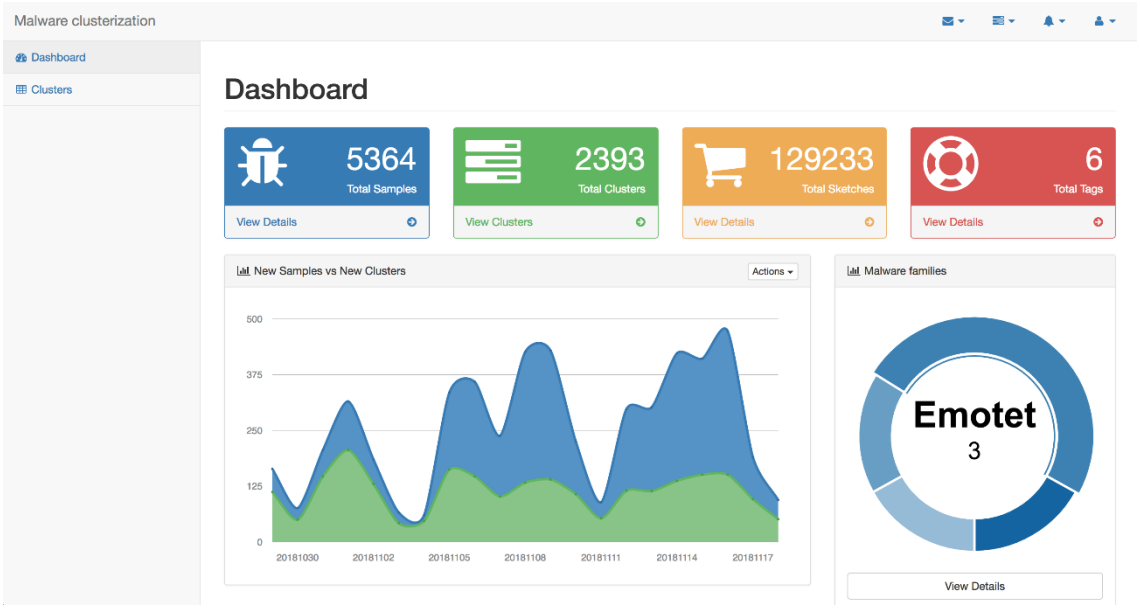

Figura 15. Dashboard principal del sistema. [fuente propia]

El dato más importante que se puede observar en la imagen es la tendencia hacia abajo que presenta la gráfica de nuevos clusters creados (grafica verde) a lo largo del tiempo. Tal y como comentábamos anteriormente, los primeros pasos del sistema no son excesivamente buenos pero el sistema es capaz de mejorar con el tiempo. En la gráfica anterior, el mejor resultado obtenido, en cuanto al número de muestras que se consiguen añadir a un cluster y que por lo tanto eliminan la necesidad de análisis, es superior al 76%.

La aplicación también nos permite estudiar los clusters en detalle, de esta forma podemos ver cuando fue su fecha de creación o que etiquetas se le han añadido tras su estudio. Como se puede observar en la siguiente imagen, la tabla solo muestra los clusters que contienen más de una muestra ya que el resto serán clusters recién creados y que aumentarán de tamaño con el tiempo o muestras individuales que se analizarán.

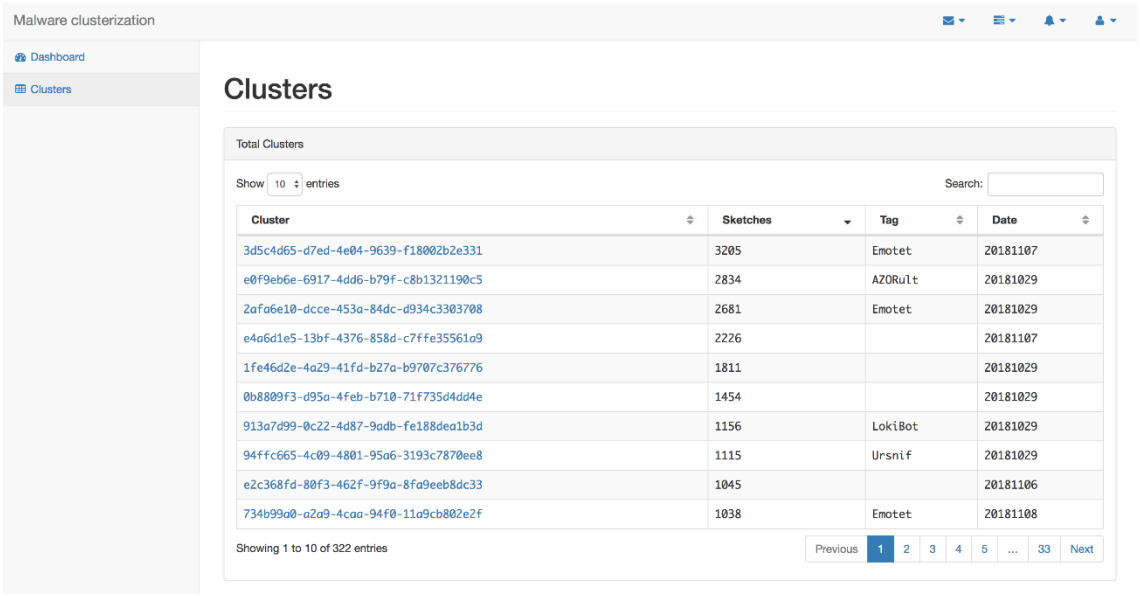

Figura 16. Vista para el estudio de clusters. [fuente propia]

Desde esta tabla se puede acceder al contenido de cada cluster donde se podrán ver todas las muestras que lo componen y las features o características de cada una de estas muestras. Además, el analista podrá añadir la etiqueta que identifique al cluster y evitar así futuros estudios innecesarios.

5. Pruebas

El apartado de pruebas aportará la información necesaria para conocer en que forma el sistema se comporta mejor o peor cuando se realizan modificaciones en el número de características elegidas. Para ello ejecutaremos varias veces el sistema variando el número de features seleccionado y compararemos los resultados obtenidos.

En primer lugar, calcularemos el rendimiento obtenido por nuestro sistema en la implementación del apartado anterior. Para ello se tendrá en consideración el número de muestras utilizadas, los clusters totales obtenidos, los clusters con más de una muestra y el tiempo transcurrido en la ejecución.

Resultados prueba 1	
Features	Dominios, procesos y ficheros.
Tiempo de ejecución	19.209s
N.º total de muestras	5364
Clusters totales	2393
Clusters (1+ muestras)	322

Tabla 5. Resultados obtenidos para la prueba 1.

Los resultados obtenidos demuestran que el tiempo de ejecución es suficientemente bueno y hemos conseguido resolver el problema afrontado con el índice de Jaccard. También se puede observar que la relación entre las muestras procesadas y los clusters con más de una muestra es bastante buena, por lo tanto, el sistema es capaz de mejorar el trabajo de un analista.

Resultados prueba 2	
Features	Tipo de fichero, dominios, procesos y ficheros.
Tiempo de ejecución	20.095s
N.º total de muestras	5364
Clusters totales	2233
Clusters (1+ muestras)	333

Tabla 6. Resultados obtenidos para la prueba 2.

Resultados prueba 3	
Features	Tipo de fichero, tamaño, dominios, procesos y ficheros.
Tiempo de ejecución	21.203s
N.º total de muestras	5364
Clusters totales	2336
Clusters (1+ muestras)	351

Tabla 7. Resultados obtenidos para la prueba 3.

A medida que se ha aumentado el número de características a contemplar en el cálculo de los minhashes, el número de clusters con más de una muestra ha aumentado. Esto nos indica que el sistema es capaz de agrupar mejor las muestras y los clusters son más precisos. Por lo tanto, si se desea implementar el sistema en un entorno real, la variación de las features es un aspecto importante a tener en cuenta.

6. Planificación

Este proyecto se debe a un trabajo de fin de máster sobre "Seguridad de las Tecnologías de la Información y de las Comunicaciones" realizado en la Universitat Oberta de Catalunya junto con la Universitat Autònoma de Barcelona, la Universitat Rovira i Virgili y la Universitat de las Illes Balears.

Durante el desarrollo del proyecto se han seguido diferentes fases organizadas en tareas para poder seguir una correcta organización lógica y una utilización eficiente del tiempo y recursos.

Listado de Tareas				
Tarea	**Día de inicio**	**Duración (Días)**	**Comienza**	**Finaliza**
Fase Inicial	**0**	**1**	**19/09/2018**	**20/09/018**
Asignación de proyecto. Comunicación de plazos y actividades.	0	1	19/09/2018	20/09/2018
Fase de análisis	**2**	**56**	**21/09/2018**	**15/11/2018**
Lectura del problema	2	7	21/09/2018	27/09/2018
Estudio de soluciones actuales	9	8	28/09/2018	05/10/2018
Recolección de material y estudio de técnicas de machine learning	17	4	06/10/2018	09/10/2018i
Estudio de algoritmos aplicables a la detección de malware	21	31	10/10/2018	09/11/2018
Planteamiento del sistema funcional	52	6	10/11/2018	15/11/2018
Fase de implementación	**58**	**24**	**16/11/2018**	**09/12/2018**
Sistema de detección utilizando los algoritmos estudiados	58	18	16/11/2018	03/12/2018
Pruebas y resultados	76	6	04/12/2018	09/12/2018
Fase final	**82**	**22**	**10/12/2018**	**31/12/2018**
Conclusiones	82	4	10/12/2018	13/12/2018
Documentación	86	18	14/12/2018	31/12/2018

Tabla 8. Tabla del listado de tareas.

Como se puede observar en el siguiente diagrama de Gantt, la duración total del proyecto ha sido de 104 días. La planificación se ha compuesto de 4 fases principales: inicial, análisis, implementación y final. Cada una de estas ha conllevado una serie de tareas tal y como queda reflejado en la tabla y diagrama.

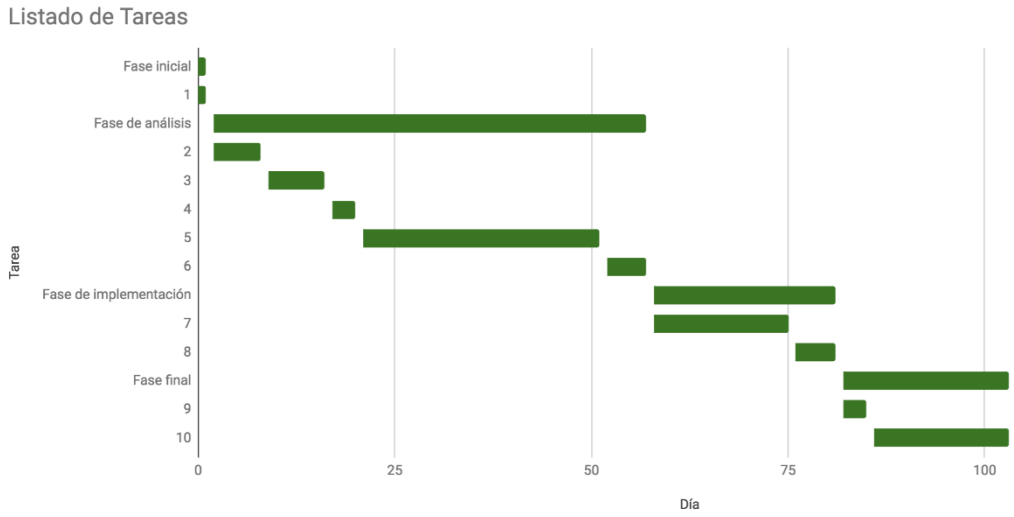

Figura 17. Listado de tareas (Gantt). [fuente propia]

7. Conclusiones y trabajos futuros

Finalizaremos el proyecto presentando las conclusiones obtenidas tras realizar nuestro estudio y observar los resultados obtenidos. También se expondrán los posibles trabajos futuros que podrían aplicarse al sistema para mejorar los resultados o añadir funcionalidades.

7.1. Conclusiones

Comenzaremos realizando una revisión sobre los objetivos propuestos en el primer apartado de la memoria para comprobar si se han cumplido.

- **Realizar un estudio previo del problema**. Uno de los principales objetivos consistía en realizar un estudio sobre las soluciones actuales que se aplican en la detección de malware. Posteriormente se realizó otro estudio con las posibles aplicaciones que se podían implementar en el sistema y como esto conseguiría resolver el problema. Tras realizar el estudio conseguimos encontrar una solución aplicable que nos permitía continuar con el proyecto.
- **Extracción de las características**. Las fuentes de información fueron analizadas, reconociendo los datos interesantes que se podían recolectar y las características que nos permitían agrupar las muestras por su similitud. Finalmente se consiguió extraer y estudiar las características que permiten a los algoritmos identificar y agrupar las muestras.
- **Implementación del sistema**. Como se ha podido comprobar en el apartado 4.3 de este documento, la implementación realizada nos permitió agrupar un gran conjunto de muestras de malware en sus correspondientes clusters. Esta implementación está abierta a mejoras, pero en primera instancia ha obtenido unos resultados muy buenos llegando a clusterizar el 76% de las muestras recibidas en un día.
- **Visualización de los resultados**. La aplicación web que se implementó para el estudio del sistema es una gran herramienta de trabajo que nos facilita el trabajo con la base de datos y además nos permite conocer el estado del sistema. Esta visualización aportará un gran valor para el analista de seguridad que necesite analizar una gran cantidad de muestras de malware.

Conclusiones personales:

Como conclusiones personales surgidas en la realización de este proyecto cabe destacar la oportunidad de aprender y poder experimentar con conceptos no conocidos previamente como pueden ser los algoritmos de machine learning y los métodos estadísticos. También es de agradecer la experiencia de llevar a cabo un trabajo que permita el estudio, análisis y desarrollo de un sistema completo, así como su documentación y revisiones.

En general, ha sido una experiencia enriquecedora y que supone un crecimiento a nivel personal que se podrá utilizar en futuros proyectos en los que participe.

7.2. Trabajos futuros

La posibilidad de mejoras o funcionalidades que este trabajo nos abre de cara al futuro es bastante amplia. Día a día, los sistemas informáticos que funcionan de forma autónoma y que son capaces de mejorar su rendimiento a medida que estos aprenden, son más abundantes y cada vez ocupan lugares más cercanos a nuestro entorno.

Mejoras:

Como ya vimos en el apartado de pruebas, donde hicimos variar las features que proporcionábamos como entrada al sistema, la modificación de las características recogidas para cada muestra hace mejorar el comportamiento final del sistema. Cuando aumentamos el número de features y estas son relevantes para la identificación de las familias, el número de clusters con arias muestras aumentaba.

Por lo tanto, queda abierta una posible línea de estudio en cuanto a las características empleadas se refiere. Será necesario estudiar nuevas fuentes de información ya que estas pueden contener nuevos datos interesantes para el análisis de malware. De la misma forma, será interesante estudiar la técnica **weighted minhash** que nos permite aplicar diferentes pesos a las características dependiendo de su importancia en el estudio de la muestra.

Funcionalidades:

Como funcionalidades que se pueden añadir al sistema, la más interesante quizás sea la aplicación de machine learning para la clasificación de los clusters. Esto quiere decir, que la funcionalidad debería ser capaz de conocer la familia de malware a la que corresponde un cluster empleando machine learning. Para esto se deberá estudiar las diferentes familias y etiquetar los clusters debidamente. Una vez tengamos suficientes estudios realizados, se usarán las características del cluster junto a su etiquetado para poder etiquetar nuevos clusters de forma automática.

Referencias

[1] Firmas y heurística: https://www.welivesecurity.com/la-es/2013/03/18/heuristica-antivirus-deteccion-proactiva-amenazas/

[2] Windows Defender - https://cloudblogs.microsoft.com/microsoftsecure/2017/12/11/detonating-a-bad-rabbit-windows-defender-antivirus-and-layered-machine-learning-defenses/

Malware detection using ML - https://cloudblogs.microsoft.com/microsoftsecure/2018/02/14/how-artificial-intelligence-stopped-an-emotet-outbreak/

[3] ANY RUN - https://www.bleepingcomputer.com/news/security/anyrun-an-interactive-malware-analysis-tool-is-now-open-to-the-public/

Índice de Jaccard - https://en.wikipedia.org/wiki/Jaccard_index

Minhash Python - http://mccormickml.com/2015/06/12/minhash-tutorial-with-python-code/

Malware Data Science - https://nostarch.com/malwaredatascience