UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

CARRERA DE CIENCIA DE LA COMPUTACIÓN

ANÁLISIS FORENSE DE MEMORIA PARA LA DETECCIÓN DE *MALWARE*

TESIS

Para optar el título profesional de Licenciado en Ciencia de la Computación

AUTOR

Alejandro Otero Henostroza

ASESOR

Jose Carlos Pazos Ortiz

Lima - Perú

2024

DECLARACIÓN JURADA

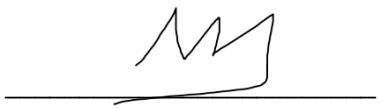
Yo, Jose Carlos Pazos Ortiz, identificado con DNI N° 48027423, en mi condición de persona responsable de validar la autenticidad de los trabajos de investigación y tesis de la Universidad de Ingeniería y Tecnología (en adelante, la Universidad), **declaro bajo juramento** lo siguiente:

Que el trabajo de investigación/tesis denominado: “ANÁLISIS FORENSE DE MEMORIA PARA LA DETECCIÓN DE MALWARE” ha sido elaborado bajo mi asesoría, por Alejandro Otero Henostroza, identificado con DNI N°75987588, para obtener el título profesional de Licenciado en Ciencia de la Computación.

El trabajo de investigación/tesis antes mencionado ha sido sometido a los mecanismos de control y sanciones anti plagio previstos en la normativa interna de la Universidad, encontrándose un porcentaje de similitud de 0%.

En fe de lo cual firmo la presente.

En Barranco, el 07 de noviembre de 2024



Firma del asesor

Dedicatoria:

A ese chico del colegio que buscaba mejorar el día a día de las personas, pero sobre todo a mi familia, especialmente a mis padres, por no dejarme descansar hasta cumplir mis metas.

Agradecimientos:

Las palabras se me van de la mano para expresar lo mucho que agradezco a quienes estuvieron 1 mensaje, como a quienes estuvieron 1 conversación entera. Gracias a cada uno de ustedes.

TABLA DE CONTENIDO

Pág.

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

RESUMEN

El aumento de ataques a través de *malwares* ha destacado la necesidad de una detección efectiva, pero lamentablemente, las herramientas actuales no logran satisfacer completamente estas necesidades mediante métodos tradicionales. Por este motivo, otros enfoques analíticos han ganado popularidad, como el análisis de memoria, también conocido como análisis forense de memoria. En este proceso, se genera un volcado de memoria que luego se analiza mediante herramientas como Volatility Framework. Esto permite a los analistas revisar toda la información y determinar la presencia de *malware*. Sin embargo, este procedimiento puede volverse complejo y puede llegar a consumir mucho tiempo. Por esta razón, se propone la utilización de modelos de *machine learning* y la creación de una base de datos. Este conjunto de datos permitirá entrenar de manera adecuada los modelos de clasificación propuestos, ya que la información recopilada proviene de entornos reales. El objetivo es automatizar el proceso de detección, lo que reduciría el tiempo de trabajo de los analistas. Adicionalmente, se realizaron experimentos para evaluar la efectividad de los modelos al hacer uso de protocolos de comparación, entre los cuales tenemos a los reductores de dimensionalidad, *cross validation* y métricas, como precisión, exactitud, etc. Se obtuvo que los modelos Random Forest y AdaBoost consiguieron los mejores resultados con un 85 % de precisión al ser entrenados con el *dataset* construido.

Palabras clave:

Malware; análisis forense de memoria; Volatility framework; *machine learning*

ABSTRACT

Memory forensics for malware detection

The increase in attacks through malware has underscored the need for effective detection, but unfortunately, current tools fall short of fully meeting these requirements through traditional methods. For this reason, other analytical approaches have gained popularity, such as memory analysis, also known as memory forensics. In this process, a memory dump is generated and then analyzed using tools like the Volatility Framework. This allows analysts to review all the information and determine the presence of malware. However, this procedure can become complex and may consume a significant amount of time. Therefore, the use of machine learning models and the creation of a database is proposed. This dataset will enable the proper training of the proposed classification models, as the collected information comes from real environments and provides accurate results. The goal is to automate the detection process, thereby reducing the time analysts spend on this task. Moreover, experiments were conducted to assess the effectiveness of the models using comparison protocols, including dimensionality reducers, cross-validation, and metrics such as precision, accuracy, among others. It was found that the Random Forest and the AdaBoost classifiers achieved the best results with a 85 % precision when trained with the constructed dataset.

Keywords:

Malware; memory forensics; Volatility framework; machine learning

CAPÍTULO I

INTRODUCCIÓN

1.1 Presentación del tema de investigación

Desde 2009 hasta 2018, el sistema operativo Windows experimentó un aumento en la cantidad de *malware* de 45 millones; en el caso de Android, este aumento fue de dos millones; mientras que en MacOS se observó una variación de 62 casos a 58.8 mil [1]. Adicionalmente, en el 2023 más de 6 billones de incidentes generados por *malware* fueron detectados a nivel mundial [2]. Incluso en los últimos meses, Microsoft observó un aumento de 275 % en ataques de *ransomware* de julio del 2023 a junio del 2024 [3]. Estos datos evidencian el papel fundamental que ha adquirido la seguridad en computación para contrarrestar estos ataques, que pueden involucrar el robo de información valiosa o la ejecución de binarios que borran datos, entre otros casos más [4]. El robo de credenciales, números de tarjetas y otra información sensible de usuarios u organizaciones se ha convertido en una tarea más sencilla para los *hackers*. Este tipo de amenazas puede manifestarse de diversas formas, siendo el *malware* uno de los ejemplos más destacados en años recientes. El *malware* es un *software* malicioso capaz de extraer información crítica de una computadora o, simplemente, causar daños significativos al afectar intencionalmente el equipo objetivo mediante la eliminación, ocultación, encriptación o corrupción de archivos y del sistema operativo, entre otros métodos [1, 5–7].

Existen diversas formas de llevar a cabo este proceso, dado que hay diferentes tipos de *malware*, cada uno con un funcionamiento distinto pero con el mismo propósito. Entre estos tipos se encuentran el *adware*, el *spyware*, el *virus*, entre otros [8]. Este proceso examina el comportamiento del *malware*. Para ello, existen tres tipos de

análisis: estático, dinámico y basado en memoria [9]. En el caso del análisis estático, se intenta extraer información y metadata para determinar si se trata de un archivo malicioso. Este método es rápido y sencillo para detectar *malware*, pero no es efectivo contra amenazas nuevas, ya que la información extraída puede no ser suficiente [10]. Por otro lado, en el análisis dinámico, se extraen más datos, como *logs* de redes, modificaciones de memoria, y otros, que pueden contribuir a no pasar por alto ningún resultado, es decir, tener un alto índice de detección. Es una visión general del comportamiento del *malware*. En este caso, se emplean diversas herramientas como Wireshark[11], IDA Pro[12], Lida[13], entre otros [10]. Para el último tipo, también conocido como análisis forense de memoria, se centra en comprender el detalle de los archivos maliciosos que residen en la memoria volátil. Esto se logra mediante un volcado de memoria que se extrae para su posterior análisis y obtener información sobre procesos en ejecución, conexiones abiertas, entre otros aspectos [14]. Lamentablemente, el análisis forense de memoria puede ser sumamente complejo y requiere mucho tiempo; por este motivo, están surgiendo nuevas alternativas, como el *machine learning*, para mitigar estos inconvenientes y mejorar los resultados [15]. El proceso de análisis de memoria implica la extracción del volcado de memoria, la utilización de herramientas como Volatility Framework[16], Rekall[17], Redline[18], entre otros [19], y la labor de un analista para determinar la presencia de *malware* en el equipo a través de la información recopilada por estas herramientas [20]. Cabe recalcar que para este trabajo se utilizará Volatility Framework, debido a que es la herramienta más utilizada en el análisis forense de memoria[20–22].

En este trabajo de investigación, se busca proponer un flujo para la detección de *malware* en máquinas virtuales con Windows. Este proceso implica, en primer lugar, la extracción del volcado de memoria de un entorno virtual, seguido del análisis de los datos mediante el uso de Volatility Framework. Posteriormente, se genera un *dataset* con la información recopilada y, finalmente, se procede al entrenamiento de modelos de *machine learning* para identificar el que obtiene mejores resultados. El

objetivo es automatizar cada etapa del proceso. Como trabajo futuro se busca crear una aplicación que tome como base esta automatización. El propósito de este *framework* será detectar *malware* en equipos con Windows de manera efectiva, reduciendo considerablemente el tiempo requerido para identificar *software* malicioso.

1.2 Descripción de la situación problemática

Según las estadísticas del AV-Test Institute [9], se detectaron 470,01 millones de instancias de *malware* en 2015, pero para 2021, esta cifra ascendió a 1312,54 millones. Incluso, a principios de 2022, se detectaron aproximadamente 30 millones de nuevos *malwares*. Por otro lado, la cantidad de infecciones por *malware* ha experimentado un rápido incremento, como se refleja en los 5.6 billones de ataques registrados en el año 2020 [23].

Estos números evidencian un aumento constante de los ataques cibernéticos, y la creación de nuevos tipos de *malware* subraya la necesidad de alternativas efectivas para detectarlos. Por lo tanto, este trabajo busca automatizar el análisis de memoria posterior al análisis realizado por Volatility Framework, utilizando modelos de *machine learning*. Esto se debe a que las soluciones actuales no son eficientes [24], ya que se basan en métodos tradicionales, como el análisis estático o dinámico, que no pueden detectar las variaciones de *malware* y consumen una cantidad considerable de recursos y tiempo.

1.3 Formulación del problema

Los métodos tradicionales ya no resultan tan efectivos en la detección de *malware*, ya que dependen de una base de datos que requiere actualizaciones constantes para comparar las características del archivo sospechoso. Sin embargo, esto

resulta ineficaz frente a *malwares* recientes que carecen de atributos registrados [25]. Por consiguiente, es necesario considerar nuevas propuestas, como los modelos de *machine learning*, los cuales emplean el análisis forense de memoria en su entrenamiento. Este enfoque permite extraer información relevante que puede ayudar a determinar si un archivo es o no un *malware*. No obstante, los conjuntos de datos utilizados para entrenar los clasificadores no se generan a partir de entornos reales, sino de ideales, donde se ejecutan solo los procesos esenciales del sistema operativo y del ejecutable, lo que sesga los datos. Estos entornos ideales se logran precisamente con los *sandbox*, como Cuckoo Sandbox [15, 25, 26], los cuales logran simular este ambiente para que se ejecute el *malware*, sin tener que propiamente crear una máquina virtual. Sin embargo, se busca acercarse lo más posible a la realidad al crear instancias virtuales, donde se puedan ejecutar más aplicaciones, junto al archivo malicioso, para simular un contexto real de un usuario.

1.4 Objetivos de investigación

- Objetivos generales
  - Mejorar el análisis de memoria a través de procesos automatizados.
  - Detectar *malware* de forma efectiva a través del análisis forense de memoria.
- Objetivos específicos
  - Automatizar el análisis al tener los modelos de clasificación entrenados que puedan realizar la detección de *malware*.
  - Construir un *dataset* en un entorno real y no con un entorno ideal para que los datos no sean sesgados.

1.5 Justificación

Malware es el principal recurso utilizado por los *hackers* para llevar a cabo sus ataques. Por ejemplo, un informe realizado por McAfee ATR Threat muestra un aumento significativo en las amenazas relacionadas con Powershell, *malware* para MacOS, *malware* para Office, entre otros, todo esto durante la segunda mitad del 2020. Asimismo, según un reporte de Kaspersky Security Network, se observaron nuevos intentos de robo de dinero mediante *malware* en aproximadamente 120,000 computadoras de diversos usuarios en la primera mitad del 2021 [27]. Además, según los últimos informes, se están desarrollando nuevas variantes y evoluciones, lo que dificulta su detección mediante métodos tradicionales al carecer de las firmas de estos nuevos *malwares* [26].

Detectar *malware* de manera eficiente es una necesidad con el incremento de incidentes que ocurren a diario. Por ello, entre los distintos tipos de análisis de *malware* disponibles, el análisis forense de memoria se destaca como el más efectivo, aunque también es el más complejo. Precisamente por esta razón, se busca automatizar una parte de este análisis utilizando modelos de *machine learning*, con el fin de obtener resultados precisos en un tiempo reducido. La idea es probar cinco modelos y utilizar aquel que obtenga puntajes más altos en las métricas al momento de detectar *malware*.

1.6 Alcance y limitaciones / restricciones

Debido a la complejidad que puede alcanzar la propuesta de automatizar todo el procesos de análisis forense de memoria, desde el volcado de memoria, hasta el resultado de si hay un *malware*, el alcance de esta investigación es automatizar el análisis de memoria post examinación por la herramienta Volatility Framework y

demostrar la importancia que tiene esto dentro del campo de detección de *malware*. La propuesta es que a través de modelos de *machine learning* se pueda identificar si un archivo es malicioso. En esta investigación no se está buscando la forma de eliminar el *malware*, solamente detectarlo. Eso se verá en futuros trabajos, así como el desarrollo de todo el *framework* para la detección.

Por otro lado, las limitaciones para esta investigación son, en primer lugar, que al ser un *software* malicioso que va evolucionando con el paso del tiempo, la información no siempre puede que esté actualizada y los métodos de detección propuestos puede que tengan que ser actualizados y/o cambiados por nuevos modelos de *machine learning* publicados, así como nuevos datos a considerar del volcado de memoria para entrenar dichos modelos a fin de conseguir resultados más certeros. En segundo lugar, al no tener una amplia experiencia en el campo, al ser un trabajo de pregrado y por la complejidad que puede llegar a alcanzar la detección de *malware*, el trabajo, para esta oportunidad, no va a poder alcanzar la idea principal de generar un *framework* completo de detección de *malware*, que sería la automatización del análisis forense de memoria. Finalmente, hay que considerar los ataques que se puedan generar en contra de los modelos, de la base de datos, etc. Debido a que, esto puede generar que los clasificadores produzcan *outputs* erróneos de forma intencionada o que la data se corrompa para que no se obtengan los resultados correspondientes [28].

1.7 Contribuciones

En esta investigación se contribuye esencialmente lo siguiente:

- Se propone un nuevo conjunto de datos, con diversos tipos de *malware*, y que se acercan lo más posible a la realidad. Esto con el fin de entrenar los modelos de *machine learning* de una forma más efectiva.

- Se genera un flujo para la detección de *malware* que utilizan clasificadores de *machine learning* para reducir el tiempo de análisis de archivos malignos al automatizar la clasificación.

La estructura de esta investigación se divide en cinco partes. En primer lugar, se realiza una revisión al estado del arte. Luego, en la Sección tres se explican los términos que se utilizan en el informe. En la Sección cuatro se detallan las partes del flujo de la detección de *malware*. Más tarde, en la Sección cinco se ven los resultados y su análisis. Finalmente, las conclusiones, donde se discuten los hallazgos y futuros trabajos.

CAPÍTULO II

REVISIÓN CRÍTICA DE LA LITERATURA

La mayoría de propuestas para la detección de *malware* se fundamentan en métodos tradicionales que abarcan el análisis estático y dinámico, así como en métodos basados en el aprendizaje, utilizando modelos de *machine learning* [29]. A continuación, se detallarán los trabajos relacionados con estos dos enfoques de detección que están vinculados a la propuesta de investigación, los cuales se van a examinar en la Sección 2.1 y 2.2. Asimismo, se revisarán las investigaciones relacionadas a la creación de una base de datos en la Sección 2.3 y, finalmente, se realizará el contraste con todo lo discutido en este capítulo en la Sección 2.4.

2.1 Métodos convencionales

El análisis estático detecta archivos maliciosos rápidamente y con un bajo consumo de recursos [30]. En [31], proponen el uso de técnicas de *hashing* (como el fuzzy-import hashing) para encontrar similitudes entre dos archivos, uno de los cuales es *malware* y el otro es la muestra a analizar. Sin embargo, aunque este enfoque genera resultados rápidos, por sí solo no es muy fiable debido a ciertas limitaciones, como los puntajes de similitud, ya que otros tipos de análisis podrían arrojar valores diferentes. Además, en [32, 33] se propone el uso de modelos de *machine learning* debido a la popularidad que ha ganado en los últimos años, como SVM, Random Forest entre otros. Los autores muestran que con un *dataset* adecuado, generado a partir del análisis estático, se pueden identificar diversas clases de *malware* con una alta precisión. Incluso, en [33] que se centra en los sistemas operativos de Android, indican que no son necesarias todas las características y que se puede emplear la

reducción de la dimensionalidad para obtener mejores resultados.

En [34, 35], los autores aprovechan las ventajas del análisis dinámico y estático para crear *datasets* y, posteriormente, entrenar modelos de *machine learning* para cada caso. En este tipo de investigaciones ya se introduce el concepto de *sandbox*, ellos utilizan Cuckoo, justamente para examinar ejecutables en un entorno seguro y recolectar sus características principales a fin de construir un *dataset*. Lamentablemente, los *malwares* son capaces de detectar cuándo se aplica el análisis por comportamiento al simplemente ejecutar algunas APIs, lo que impide la recopilación completa de información debido a sus mecanismos de defensa, lo cual se ve reflejado en su base de datos. Por ello, en futuros trabajos, buscan proponer entornor virtuales que no puedan ser detectados por los *malwares* o el uso de *deep learning* a fin de clasificar con mayor precisión. Además, en [36, 37], se enfocan en la recolección de llamadas de las APIs para formar un *dataset* que luego se utiliza para entrenar modelos de clasificación. Dado que a partir de estas llamadas, se puede entender el comportamiento de una aplicación y determinar si son maliciosas sus actividades. No obstante, los autores reconocen que este tipo de análisis tiene algunas limitaciones, ya que si el archivo malicioso detecta que se encuentra en un entorno virtual, como un *sandbox*, evita su ejecución para evitar ser detectado.

2.2 Métodos modernos basados en aprendizaje

El análisis de memoria ha ganado bastante popularidad en los últimos años, ya que las nuevas variantes de *malware* operan directamente en memoria, lo que les permite no dejar rastros y dificultar su detección. Por ello, una alternativa por la que se está optando en [38] es la de convertir el volcado de memoria, que se extrae con el análisis forense de memoria, en una imagen. De esta forma se puede aprovechar toda la información extraída del equipo. Asimismo, se está prefiriendo extraer

las características principales del volcado de memoria, de modo que se pueda determinar si un equipo está infectado. En [25], identifican la falta de buenos *datasets* para abordar este tema de investigación; por ello, como parte de su propuesta de detectar *malware* con clasificadores de *machine learning*, crean una fuente de datos con la que se entrenan los modelos. Para esto utilizan una herramienta de análisis forense de memoria llamada Volatility Framework, con la que adquieren atributos como llamadas a APIs, funciones DLL, inyecciones de código, conexiones en la red, *handles* y escalamiento de privilegios. Con esto, logran conseguir que SVM obtenga una precisión mayor al 98 %. Pero, las pruebas realizadas son en un entorno ideal, donde se ejecutan pocos procesos y uno de esos son las muestras a analizar, lo que facilita el análisis a los modelos. El mismo caso se da en [15, 26], donde proponen extraer algunas de las características principales que presenta el volcado de memoria, como las llamadas de APIs, las conexiones de red, los DLLs, entre otros para luego utilizar modelos de *machine learning* para ser clasificados.

En [38], los autores proponen una manera de analizar el volcado de memoria convirtiéndolo en una imagen RGB, y presentan un método para no almacenar imágenes de alta calidad con el objetivo de ahorrar espacio en el disco. Su *framework*, llamado *Memory-Resident Malware Deep Learning Detector*, automatiza la detección de *malware* al generar el volcado de memoria, transformarlo en una imagen y luego analizarlo mediante una combinación de una CNN (red neuronal convolucional) con una RNN (red neuronal recurrente). Sin embargo, a pesar de haber obtenido mejores resultados en comparación con otros estudios del estado del arte, esta propuesta consume muchos recursos y tiempo. Por otro lado, en las investigaciones [10, 24, 27, 39, 40], también se propone el uso de volcados de memoria como imágenes y/o el uso de datos sin procesar del *malware* en forma de imagen para entrenar modelos de clasificación de *machine learning* o redes neuronales. Aunque no logran obtener resultados tan prometedores como en [38], siguen representando una alternativa confiable para detectar *malware* en un período de tiempo menor.

2.3 *Datasets* de *malware* públicos

Debido a la escasez de bases de datos y *datasets* de *malware* que hay en la web para fines educativos, en [25, 41] crean su propio conjunto de datos para luego entrenarlos con modelos de clasificación de *machine learning*. Los *datasets* que ellos crean buscan contener las características principales que se puedan extraer de los volcados de memoria a fin de hacer un *dataframe* que no consuma mucho espacio para poder entrenar los modelos. En la investigación [25] se busca aprovechar seis *plugins* importantes que pueden conducirnos a identificar la presencia de un *malware*, los cuales son *privileges*, handles, APIs, inyecciones de código, conexiones de red y dlls. Por otra parte, en [41] se busca tener pocos atributos de solo cinco *plugins* a fin de obtener puntajes altos en la precisión. Los *features* que utilizan son inyecciones de código, módulos ldr, handles, vista de procesos y los APIs. Asimismo, todas sus pruebas las realizan en máquinas virtuales y no en *sandboxes*.

2.4 Consideraciones finales

Luego de haber analizado algunos de los trabajos relacionados a esta investigación, las diferencias principales se encuentran en la construcción de la base de datos. Por el lado de los modelos de *machine learning*, los clasificadores utilizados se repiten en varios de los trabajos ya mencionados, así como el uso de redes neuronales. Esto, debido a que en la mayoría de trabajos la propuesta gira entorno a la base de datos empleada para entrenar los modelos de *machine learning*. Esto incluye esencialmente el análisis utilizado para crear el *dataset* y la diversidad de *malwares* que se utilizan para el conjunto de datos.

Justamente, ahí es donde predomina esta investigación, al construir una base de

datos en un entorno lo más real posible, al ejecutar otros programas junto al *malware* para simular un escenario de un usuario común y al emplear diversos tipos de *malware* para las muestras, de modo que el *dataset* tenga diversidad.

CAPÍTULO III

MARCO TEÓRICO

Existen diversas ténicas para la detección de *malware*, pero para poder entender mejor cuáles son y por qué es que se está optando por el análisis forense de memoria sobre otros métodos, en la siguiente Sección se va a explicar estos métodos. En la Sección 3.1 se detallarán algunos de los tipos de *malware* que existen. Luego, en la Sección 3.2 se van a examinar las técnicas de anásis y detección de *malware*. Seguidamente, en la Sección 3.3 se describen los pasos del análisis forense de memoria. Después, en la Sección 3.4 se ilustra qué es *machine learning* y cómo es que se utiliza para la detección de *malware*.

3.1 *Malware*

Es un *software* malicioso que utilizan los *hackers* para atacar a sus víctimas. Esto con la inteción de ganar acceso a los recursos computacionales o de la red, corromper las operaciones del equipo y hasta recolectar información confidencial sin la autorización del usuario [42].

3.1.1 Tipos de *malware*

Existen diversos tipos de *malwares*, cada uno con un propósito distinto, pero todos con el fin de causar daño al usuario objetivo. Algunos de los más conocidos [43] y que fueron considerados en esta investigación son los siguientes:

- **Virus:** Es un programa que infecta otros programas y los modifica para alterarlos. Además, este se reproduce por sí solo [44].
- **Worm:** Es un tipo de *malware* que se exparce de una computadora a otra al transmitir copias de sí mismo a través del internet [45].
- **Spyware:** Es un ejecutable que recolecta información confidencial sin que el usuario se dé cuenta. Este programa es silencioso, puesto que no genera ningún indicio como para ser detectado [1].
- **Trojan:** Se les conoce por simular ser un *software* legítimo, pero al final es un *malware*. Se usa especialmente en la ingeniería social, de modo que los usuarios creen que están descargando un programa verídico [46].
- **RAT**: Es un tipo de Trojan que se centra en el acceso remoto. Esto permite al atacante poder comunicarse con el equipo infectado, ejecutar comandos, entre otras actividades [47].
- **Ransomware:** Es probablemente uno de los más peligrosos al prevenir que los usuarios puedan acceder a sus dispositivos e información. Suelen extorsionar, es decir, devolver su información a cambio de sumas de dinero [48].
- **Rogue:** Es una aplicación que engaña al usuario al hacerle creer que su equipo está infectado [49].

3.2 Detección de *malware*

Para entender las diversas técnicas de detección *malware* que existen, primero hay que tener en cuenta que los sistemas de detección se dividen en dos partes: análisis y detección [50].

3.2.1 Técnicas de análisis de *malware*

Tenemos cuatro tipos: estático, dinámico, híbrido y basado en memoria.

- El primero, hace referencia a analizar los archivos ejecutables sin hacerlos correr en el equipo, de modo que se pueda extraer información superficial que delaten si el ejecutable es un *malware*. Esto incluye hashes, firmas, etc 3.3.
- El análisis dinámico, también conocido como análisis de comportamiento, hace lo opuesto al anterior, en este caso sí se ejecuta el archivo, dentro de un entorno controlado, como máquinas virtuales, donde solo se busca observar *malware*. Esto permite que se puedan utilizar diversas técnicas, como monitoreo de funciones, para analizar cómo se comporta el *malware* 3.3.
- El análisis híbrido es una combinación del análisis estático y dinámico. Ambos tienen sus ventajas y desventajas, pero se complementan, de modo que se incremente la habilidad de detectar *malware* de forma asertiva 3.3.
- En cuarto lugar, se tiene el análisis basado en memoria que ha ganado bastante popularidad en los últimos años. Este enfoque analiza el volcado de memoria, con el malware en ejecución, obteniendo información detallada de los procesos, el sistema operativo, entre otros elementos de la computadora. A diferencia del análisis dinámico, el análisis de memoria se enfoca más en examinar lo que sucede dentro de una computadora y no en cómo afecta el resultado de la ejecución del *malware* al equipo. Este tiene dos partes: la adquisición y el análisis de memoria [50]. Esto se detalla en la Sección 3.3.

3.2.2 Técnicas de detección de *malware*

A grandes rasgos, se tienen dos métodos: basado en firmas y basado en heurísticas. Con respecto al primero, se utilizan los hashes y bytes de un archivo

malicioso y se compara con diversas muestras, de una base de datos de *malware*, para encontrar un *match* y así detectar el *software* malicioso. Esta técnica es efectiva; sin embargo, no funciona con nuevos *malwares* al no tener firmas que estén relacionadas. Por otro lado, el método basado en heurísticas, también conocido como basado en comportamiento, se analizan las acciones realizadas por el achivo en ejecución y se cataloga como malicioso o legítimo a partir de la fase de monitoreo. Este método sí es capaz de detectar *malware* ya antes reconocido y nuevas amenazas; sin embargo, tiene un alto *ratio* de falsos positivos y consume bastante tiempo en el monitoreo [50].

3.3 Análisis forense de memoria

Memory forensic se entiende como el análisis de memoria, el cual es un proceso en el que se investiga a profundidad la memoria volátil de un sistema, también conocida como RAM. Esto se debe a que algunos *malwares* esconden la data recolectada en la memoria[51]. Existen distintas herramientas para este fin, como Volatility o Rekall que son de las más usadas a nivel mundial [21], las cuales extraen data específica de los sistemas operativos, ya sea las llamadas a funciones, y posibles indicadores de compromiso de la memoria, lo cual ahorra tiempo, energía y especialmente conocimientos técnicos a un investigador para poder identificar si es que hay algún elemento sospechoso gracias a la comunidad que respalda dichas herramientas. Hay que tener en cuenta, que para que se desarrolle el análisis por parte de estas herramientas, se debe, primero, extraer el volcado de memoria que se puede realizar a través de diversas herramientas, pero que en este caso será DumpIt, porque de acuerdo a [21] es el programa que mejor logra recuperar una copia de la memoria. Pero, de no existir herramientas como Volatility o Rekall se vuelve aún más complejo el proceso de análisis, ya que se necesitaría de expertos que sepan interpretarla. Además, con el paso del tiempo, al haber más versiones de *software*, se necesitaría

un investigador que esté especializado en comprender la información de cada versión [52], lo cual dificulta aún más este proceso.

3.3.1 Proceso de análisis

Para realizar el análisis forense de memoria se suele utilizar una serie de etapas que son respaldadas por la comunidad forense [19, 21, 22, 53], el cual consiste principalmente de dos partes: adquisición y análisis, como se puede ver en la Figura 3.1. Dentro de estas, puede haber pasos adicionales, como la identificación de la evidencia o conclusiones.

Un ejemplo que emplee este diagrama de flujo se puede ver en [19], donde en el paso de adquisición se crea un entorno virtual y se extrae el volcado de memoria, puede ser con DumpIt. Hay que tener en cuenta que el volcado de memoria contiene información acerca de los procesos en ejecución, la sesión del usuario, conexiones de red abiertas, entre otros datos más. Todo esto en el momento que se extrajo la memoria y en un formato que debe ser procesado por una herramienta adicional [54]. Luego, para la parte del análisis, se parsea la data cruda obtenida con una herramienta, como Volatility Framework, y se examina el resultado, que también se puede realizar a través de un método adicional, como Signature Scanning. De esta forma se busca la presencia de un *malware* al hallar llamadas a funciones o datos sospechosos que no deberían estar almacenados en la memoria volátil.

3.4 *Machine Learning*

El campo de *machine learning* es una subclase del área de inteligencia artificial, donde la idea principal es que el sistema sea capaz de aprender y mejorar su desempeño automáticamente a partir de la experiencia y los datos, sin requerir una

FIGURA 3.1: Diagrama de flujo del análisis forense de memoria.

programación explícita para cada tarea específica. Por ello, la idea principal de los modelos de *machine learning* es que sean entrenados a partir de un algoritmo, de modo que puedan ser capaces de clasificar, predecir valores, etc. Este entrenamiento se genera a partir de un *input* que se le da al modelo y a partir de este paso, el programa es capaz de predecir un resultado a partir de la data inicial [43].

3.4.1 Entrenamiento supervisado y no supervisado

Los entrenamientos supervisados y no supervisados son tipos de técnicas utilizadas en *machine learning* para entrenar los modelos. La diferencia entre ambas

formas es que, la primera, utiliza etiquetas en el *dataset*, de modo que la data que se le da al modelo es previamente asociada a un resultado. Esto se da con el propósito de que se busque la función de mapeo más apropiada que, a partir de la información dada, el modelo pueda recibir x, como *input*, y devolver el y, como *output* [55], a fin de obtener el resultado correcto.
Por el lado del entrenamiento no supervisado, el *dataset* no contiene etiquetas, por lo que no hay un resultado o un atributo objetivo relacionado a las demás características. Los resultados obtenidos de este tipo de entrenamiento no se evaluan como correctos o incorrectos, sino que se busca comprender la estructura y la agrupación de los datos [55].

3.4.2 Detección de *malware*

Los modelos de *machine learning* son capaces de aprender distintos patrones de ataques y hasta de prevenir incidentes similares en el futuro. Sin embargo, aún es un reto de que sean capaces de detectar nuevas variantes. Por ello, deben ser entrenados de forma adecuada, es decir, con un *dataset* robusto que contenga tanto muestras malignas, como benignas [29].

Cuando hacemos referencia a la detección de *malware*, en términos de *machine learning* se le conoce como clasificación, donde los modelos identifican si es que la muestra analizada es benigna o maliciosa. Incluso, se han propuesto clasificadores que buscan trabajar con *datasets* multiclase, de modo que se pueda averiguar el tipo de incidente [32]. Para este proceso, algunos de los modelos más utilizados son [43, 55–57]

- Support Vector Machine [25]
- Decision Trees [26]

- Naives Bayes Classifiers [15]

Asimismo, para esta investigación se implementarán los clasificadores ya mencionadas, así como el AdaBoost Classifier [58] y el Random Forest Classifier [59]. Con respecto al SVM, es un modelo que fue creado para tareas de regresión y clasificación. Cabe recalcar, que para esta investigación se implementará el segundo tipo, que se le conoce como SVC. Además, su funcionamiento consiste en separar las diversas clases del *dataset* con una superficie que maximice el margen entre ambas. Posee diversos tipos de funciones de kernel para aplicar el algoritmo, como la lineal, que es la más usada, la polinomial, la gausiana, entre otras. Adicionalmente, por el lado de sus desventajas es un modelo que tiende a tener un costo computacional alto; sin embargo, posee una muy buena capacidad de generalización [60].

En el caso de los Decision Trees, es un modelo basado en árboles que representa de forma jerárquica las relaciones entre nodos. Desde la raíz hasta la hoja son un conjunto de decisiones que resultan en un valor de tipo booleano. Este modelo también funciona para problemas de clasificación y regresión. Una de sus ventajas es que es capaz de clasificar valores categóricos y numéricos. Pero, por el lado de sus desventajas, un *dataset* de gran tamaño de entrenamiento puede generar que la complejidad del árbol incremente mucho [61].

Por otra parte, Naives Bayes es un modelo de probabilidades, el cual utiliza el teorema de Bayes. Produce la probabilidad para cada caso y después predice el que tenga mayor probabilidad de ser el resultado. Entre algunas de sus ventajas tenemos que su algoritmo es simple de implementar y con poca complejidad computacional, aunque una de sus desventajas es que es sensible con características irrelevantes [62, 63].

En el caso del AdaBoost, es un modelo conocido como caja negra, al estar compuesto de varios clasificadores. Lo que hace este modelo es mejorar clasificadores

más débiles al juntarlos, es una técnica de *boosting*. Suele utilizar los Decision Trees como base. Lo que hace es actualizar los pesos bases a partir del rendimiento para que las instancias que estén rindiendo menos, puedan tener mejores resultados [58].

Finalmente, tenemos al Random Forest, el cual es un clasificador que intenta mejorar los puntajes obtenidos por el Decision Tree al funcionar mejor donde se tengan grandes cantidades de información, ya que construye muchos clasificadores para poder alcanzar una mayor precisión. Utiliza técnicas como AdaBoost y Bootstrapping para generar estos clasificadores, creando diferentes instancias para cada subconjunto de datos. Entre algunas de sus ventajas tenemos que provee métodos para estimar data faltante y que es eficiente en manejar grandes bases de datos. Por otro lado, algunas de sus limitaciones son que tiene problemas al lidiar con atributos multivariados y multidimensionales, y que tiende a hacer *overfitting* en problemas de regresión [59].

3.5 Consideraciones finales

En este capítulo se vio el detalle de los términos más relevantes que se van a utilizar a lo largo de la investigación. Por el lado de *malware* es importante entender qué tipos hay y cómo es que funcionan para que se tomen en consideración todas estas familias para incluirlas en la construcción de la base de datos. Por otro lado, la data con la que se arma el *dataset* es a través del análisis forense de memoria; por ello, es importante entender cómo funciona, puesto que de esa forma se extrae la información de los volcados de memoria. Finalmente, los modelos de *machine learning* son justamente los algoritmos que permiten automatizar el proceso de análisis, ya que estos clasificadores te indican si un archivo es maligno o benigno a partir del entrenamiento previo con la base de datos. Esto en vez de consumir tiempo de más con el análisis forense de memoria para identificar si es un *malware* la muestra a analizar.

CAPÍTULO IV

MARCO METODOLÓGICO

La propuesta de detección de *malware* está dividida en cinco pasos. En primer lugar, se utiliza DumpIt para generar el volcado de memoria de un entorno virtual Windows en el que se esté ejecutando la muestra maliciosa o benigna, el cual se verá el detalle en la Sección 4.1.1. Luego en la Sección 4.1.2, se extrae la información del volcado a través de nueve *plugins* del Volatility Framework. En tercer lugar en la Sección 4.1.3, con la información recolectada se construye el *dataset*, ya sea utilizando promedios, sumas de procesos, etc. Después en la Sección 4.1.4, se aplica la clasificación de *malware* con los modelos propuestos. Finalmente en la Sección 4.1.5, se elige el que obtenga mejores resultados y con ese se realizan las siguientes pruebas para identificar el archivo malicioso. En la Figura 4.1 se puede apreciar el gráfico del flujo.

4.1 Fases del flujo de detección de *malware*

Como se aprecia en la Figura 4.1 hay cinco partes que componen el flujo para la detección de *malware* con el análisis forense de memoria. Por ello se va a explicar cada paso de forma detallada en las siguientes subsecciones.

4.1.1 Adquisición de volcado de memoria

En esta fase, lo que se busca es utilizar una herramienta para extraer una copia de la memoria volátil, que en este caso será DumpIt como ya se mencionó, que refleja toda la data en el momento en que se generó la copia [21]. Esta herramienta, de

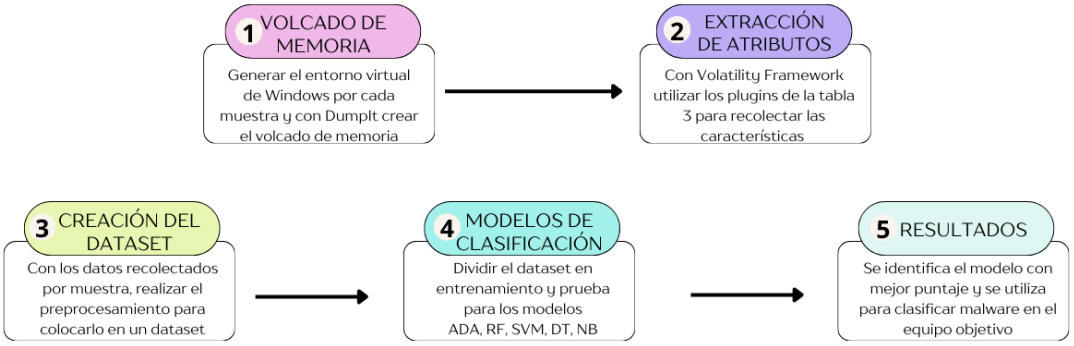


FIGURA 4.1: Flujo del proceso para la detección de *malware* divivido en cinco pasos: extraer el volcado de memoria, recolectar las características principales de este, generar el *dataset* con todas las muestras analizadas, entrenar los modelos y analizar los resultados.

acuerdo a la experimentación realizada por [21, 64] es de las mejores en la industria, al ser muy efectiva. El propósito de esta herramienta es adquirir evidencia de la memoria RAM al generar el volcado de memoria, mas no su análisis. Asimismo, es de código abierto y fue diseñada para Windows. Este proceso de extracción se hará para cada muestra, tanto maliciosa como benigna en un entorno virtual distinto para cada uno. En otras palabras, se creará una máquina virtual por cada muestra que se vaya a ejecutar, donde también habrán otros programas de uso diario, como Google Chrome, entre otros. Adicionalmente, las muestras malignas fueron descargadas de bases de datos en línea como repositorios de github, malwarebazaar, entre otros.

4.1.2 Extracción de características

Una vez extraída esta copia de la memoria volátil, se parsea la data con el apoyo de la herramienta, Volatility, y se pasa a guardar los resultados. Se utiliza este *framework*, porque es el más utilizado a nivel mundial en el ámbito del análisis forense de memoria. Es de código abierto y contiene más de 200 *plugins* que contienen una inspección profunda de la memoria volátil de los equipos para los sistemas operativos de Linux, Windows y Mac[20]. En la Tabla 4.1 se pueden ver los *plugins* utilizados para la extracción de las características. Se automatizó este proceso con un *script* de Python, en el cual se ejecutan, uno por uno, todos los *plugin* y cada uno de ellos genera un archivo de tipo csv que luego es guardado en una carpeta que se utilizará más adelante para crear la base de datos. Por lo tanto, por cada muestra se crea una carpeta que contiene nueve documentos.

Plugin	Detalle
malfind	Muestra los posibles procesos que contengan código inyectado
netscan	Escanea objetos de red
dlllist	Muestra los módulos cargados
handles	Lista todos los procesos con handles abiertos
privileges	Enseña los procesos con privilegios
modules	Muestra los módulos cargados del kernel
ldrmodules	Lista los módulos cargados
callbacks	Enseña los callbacks del kernel y sus notificaciones de rutina
pslist	Muestra los procesos que se estaban ejecutando

Tabla 4.1: *Plugins* con su detalle que se van a utilizar para extraer la información del volcado de memoria [65].

Por otro lado, con respecto a los *plugins* elegidos esto fue en base a distintos autores [15, 25, 26, 41, 43] que los utilizaron para crear sus bases de datos. Asimismo, algunos de estos *plugins*, como malfind y privileges, representan algunas de las vulnerabilidades que los *hackers* intentan explotar [66], ya sea, por ejemplo, para robar información confidencial, como credenciales. Este es otro motivo que se consideró para incluirlos dentro del proceso de extracción de características para, más adelante, construir el *dataset*.

4.1.3 Creación del *dataset*

En total se consiguen 28 atributos, los cuales se pueden apreciar en la Tabla 4.2, a partir de los *plugins* que ofrece Volatility para extraer la información necesaria para identificar si hay un *malware*. Este proceso se realizó con un *script* de Python para automatizar la construcción del *dataframe* con pandas que luego se usaría para el entrenamiento de los modelos.
El proceso para armar el *dataset* se divide en los siguientes pasos:

1. En primer lugar, se crean dos máquina virtuales, una para ejecutar programas benignos y otra para malignos. En el caso del primer entorno, se ejecutan aplicaciones, como Google Chrome, Whatsapp, entre otros. Por el lado de la segunda máquina, se ejecutan ese tipo de programas que se usan a diario, pero también se ejecuta un *malware*. Este se descarga de una base de datos en internet, ya sea de un repositorio de github o de otros sitios web. Cuando se haya ejecutado dicha aplicacion se extrae el volcado de memoria con DumpIt en cada máquina virtual.

2. Luego, este archivo que se consiguió de la RAM se envía a la tercera máquina virtual, donde se realiza la recopilación de características. Para trasladar los volcados de memoria, se utilizó la nube de Google, Drive, para que se pueda

tener acceso desde otro ambiente con tan solo estar conectado al internet. Ambos documentos se descargan y se utilizan los dos *scripts* creados en Python para automatizar el proceso de recolección de los atributos más importantes. Los *plugins* utilizados se pueden observar en la Tabla 4.1. El resultado de ejecutar estos *scripts* son nueve archivos en formato csv, por cada muestra, que luego se guardan en una carpeta, una para el maligno y otra para el benigno.

3. Cada carpeta que contiene los nueve documentos de la extracción de características, se guardan dentro del directorio *malware*, si es malicioso, o dentro de *benign*, si es benigna. De esta forma se tienen dos carpetas, las cuales contienen todos los directorios correspondientes a cada muestra de acuerdo al tipo de reputación que tiene, si es sospechoso o no. Esto se hizo con la intención de que cuando se vaya a iterar sobre cada carpeta de acuerdo al nombre, se pueda asignar el *label* correspondiente.

4. Al momento de tener ambas carpetas listas, se recopilan los datos más relevantes de cada archivo csv, por muestra, para crear una instancia con 28 variables. Este proceso se repite por cada muestra que se tenga y se coloca uno o cero de acuerdo a la clase que le corresponde a la instancia. En este caso, se le colocó el cero si era benigno y uno si era malicioso. Cabe recalcar que estos datos fueron elegidos, debido a que distintos autores [15, 25, 26, 41, 43] los consideraron para armar sus datasets y se consideraron, especialmente, los atributos que se repetían en más de 2 investigaciones.

5. Los atributos que se recopilan a partir de los archivos csv y lo que significa cada uno se pueden ver en la Tabla 4.2. Este proceso, también se realizó a través de un *script* de Python, el cual analizaba cada documento y extraía dicha información de forma automática. De esta forma se construyó el *dataset*. Los

scripts que se utilizaron para llevar esto a cabo se encuentran en un repositorio de github[1].

Lo más complicado de la elaboración de la base de datos es extraer las características de cada volcado de memoria con los *plugins* indicados, debido a que consume mucho tiempo y recursos; por ello, es que se optó por crear *scripts* de Python para automatizar el proceso.

[1]https://github.com/Oteranga/malware-detection.git

Atributo	Detalle
malfind.npid	Cantidad de procesos encontrados con malfind
malfind.unique	Cantidad de procesos únicos encontrados con malfind
privs.npid	Cantidad de procesos encontrados con privileges
privs.avg_pid	Promedio de procesos por tipo de privilegio realizado
privs.SeSystemEnvironmentPrivilege	Cantidad de procesos que hizo uso de SeSystemEnvironmentPrivilege
privs.SeSystemProfilePrivilege	Cantidad de procesos que hizo uso de SeSystemProfilePrivilege
privs.SeSystemtimePrivilege	Cantidad de procesos que hizo uso de SeSystemtimePrivilege
privs.SeTakeOwnershipPrivilege	Cantidad de procesos que hizo uso de SeTakeOwnershipPrivilege
privs.SeTcbPrivilege	Cantidad de procesos que hizo uso de SeTcbPrivilege
privs.SeTimeZonePrivilege	Cantidad de procesos que hizo uso de SeTimeZonePrivilege
privs.SeUndockPrivilege	Cantidad de procesos que hizo uso de SeUndockPrivilege
modules.nmodules	Cantidad de módulos encontrados con modules
dlllist.ndlls	Cantidad de dlls encontrados con dlllist
dlllist.avg_dlls	Promedio de dlls
callbacks.ncallback	Cantidad de callbacks encontrados con callbacks
pslist.npid	Cantidad de procesos encontrados con pslist
pslist.nppid	Cantidad de procesos padres encontrados con pslist
handles.nhandles	Cantidad de handles encontrados con handles
handles.avg_handlers	Promedio de handles
handles.ndirectory	Cantidad de directorios encontrados con handles
handles.nmutant	Cantidad de mutants encontrados con handles
handles.nthread	Cantidad de threads encontrados con handles
handles.nkey	Cantidad de keys encontrados con handles
handles.nsection	Cantidad de sections encontrados con handles
netscan	Cantidad de conexiones
ldrmodules.not_load	Cantidad de ldrmodulos no cargados
ldrmodules.not_init	Cantidad de ldrmodulos no inicializados
ldrmodules.not_mem	Cantidad de ldrmodulos que no están en memoria
LABEL	Etiqueta de la clase de la muestra

Tabla 4.2: Atributos del *dataset* con su detalle de qué significa cada uno.

4.1.4 Modelos de *machine learning*

Los clasificadores que se van a utilizar son cinco, los cuales ya se explicaron anteriormente, y son

- SVC [60]
- AdaBoost [58]
- Random Forest [59]
- Decision Tree [61]
- Naives Bayes [62]

Estos modelos que fueron implementados con la librería ScikitLearn [67], se les va a pasar el *dataset* a cada uno de ellos para que puedan ser entrenados y generar los resultados.
Estos modelos son los propuestos para esta investigación, debido a que en [15, 25, 26, 29, 32, 43] utilizan estos modelos por ser de los más conocidos dentro del contexto de *machine learning* y que son más efectivos al obtener mejores resultados. En el caso de AdaBoost, se propone su uso, debido a que como se explicó anteriormente, mejora el clasificador que reciba por lo que en este caso se utiliza para mejorar el rendimiento del Random Forest como un modelo aparte.

4.1.5 Análisis de resultados

En este último paso, se identifica el modelo de *machine learning* que tenga mejor puntaje de acuerdo a las métricas definidas y ese es el clasificador que se va a utilizar para las siguientes pruebas. Además, se verifica si es que se pudo clasificar de forma correcta las muestras al utilizar los protocolos de comparación, como el

cross validation y la reducción de dimensionalidad. Este apartado será detallado en la Sección 5.4, donde se comentará acerca de todos los hallazgos y dificultades que se obtuvieron en el proceso de conseguir los resultados.

CAPÍTULO V

RESULTADOS

En este capítulo se presentan los experimentos realizados con los modelos basados en *machine learning* y con la base de datos propuesta. Este apartado está dividido en tres secciones. Primero en la Sección 5.1, se detallarán los métodos de comparación que está compuesto por las métricas para medir el rendimiento, métodos de reducción de dimensionalidad para mejorar el desempeño de los clasificadores y *cross validation* con la finalidad de comprobar la efectividad de los modelos. Luego en la Sección 5.2, se verán los detalles sobre la implementación para la construcción de la base de datos. Después en la Sección 5.3, se muestran los resultados de la experimentación, al entrenar los modelos con el conjunto de datos. Finalmente, se tiene la discusión en la Sección 5.4, el cual es el apartado en el que se van a analizar todas las secciones.

5.1 Protocolos de comparación

Se emplearon varios métodos para evaluar el rendimiento de los modelos de clasificación propuestos. Esto se hizo con el objetivo de identificar el mejor modelo y confirmar que los resultados obtenidos no fueran producto de la distribución de los datos.

5.1.1 Métricas definidas

Para calificar los modelos de *machine learning* se emplearon cuatro métricas que permiten identificar cuál de los clasificadores es mejor a partir de los resultados

que obtenga cada uno. Los criterios que se utilizaron son los que se encuentran a continuación:

- Precisión: $\frac{TP}{TP+FP}$,
- Exactitud: $\frac{TP+TN}{TP+TN+FP+FN}$,
- *Ratio* de verdaderos positivos (*Recall*): $\frac{TP}{TP+FN}$,
- Puntaje F: $\frac{2\times(P\times R)}{P+R}$,

donde TP son los casos de verdaderos positivos, FP son los casos de falsos positivos, TN son los verdaderos negativos, FN son los falsos negativos, R es *recall* y P es precisión. Cabe recalcar que cuando se menciona un caso verdadero positivo, se hace referencia a las muestras que eran *malware* y fueron catalogadas de tal forma y por el lado de falso positivo es cuando un archivo que no era maligno es clasificado como una amenaza.

Estas métricas fueron seleccionados a partir de otras investigaciones [25, 26] que también los utilizan para comparar los distintos modelos de *machine learning* para la clasificación de *malware*.

5.1.2 Reducción de la dimensionalidad

Para aplicar la reducción de la dimensionalidad se aplicaron tres métodos: PLS [68], UMAP [69] y PCA [70]. No obstante, se realizó una curva ROC sin aplicar los reductores de dimensionalidad y poder comparar los resultados antes y después de aplicar este método. En la Figura 5.1 se pueden apreciar los resultados obtenidos por los modelos sin utilizar los reductores de dimensionalidad y en las Figuras 5.2, 5.3 y 5.4 se pueden observar los resultados obtenidos para cada modelo de clasificación al aplicar los reductores. Estas ténicas se utilizan para que los *embeddings* que se

encuentran en una determinada dimensión sean proyectados en un espacio de menor cantidad de características. De esta forma se reduce la complejidad al conjunto de datos al remover atributos que son redundantes, al comprimir los datos, entre otros [71]. Asimismo, en las Figuras 5.2, 5.3 y 5.4 se aprecian las leyendas donde se indican los valores de AUC[1] de cada modelo. Cabe recalcar, que para fines de esta investigación el número de componentes utilizado fue de seis para cada método aplicado, el cual se agrega como argumento al crear el objeto de estas técnicas.

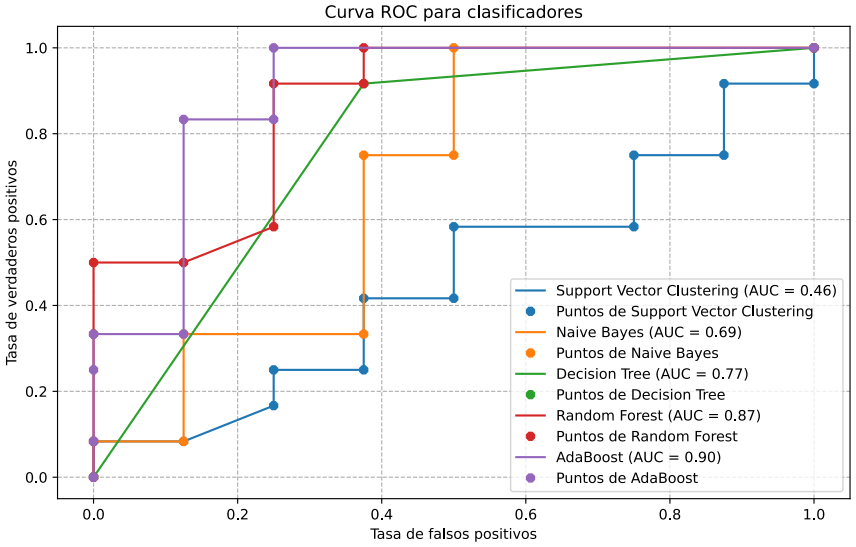


FIGURA 5.1: Curva ROC de los modelos implementados, donde se ve la relación entre el *ratio* de verdaderos positivos y de falsos positivos.

[1]Hace referencia al área bajo la curva. Mientras más cercano el puntaje esté a 1, es un mejor clasificador, pero mientras esté más cerca a 0, será un clasificador obsoleto. https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc

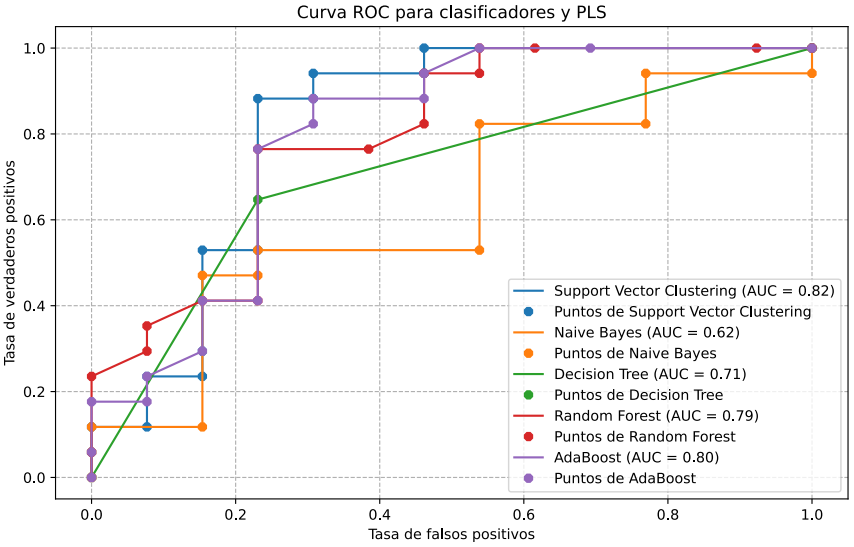


FIGURA 5.2: Curva ROC para PLS de los modelos implementados, donde se ve la relación entre el *ratio* de verdaderos positivos y de falsos positivos.

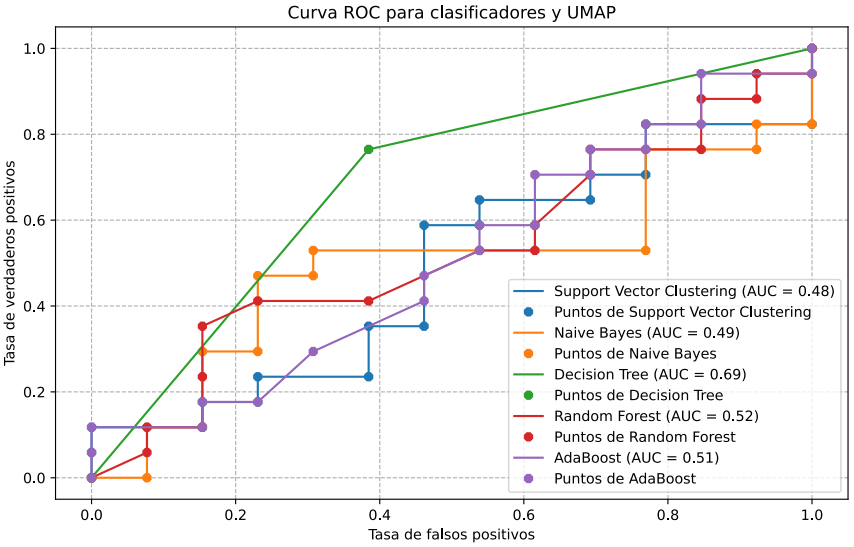


FIGURA 5.3: Curva ROC para UMAP de los modelos implementados, donde se ve la relación entre el *ratio* de verdaderos positivos y de falsos positivos.

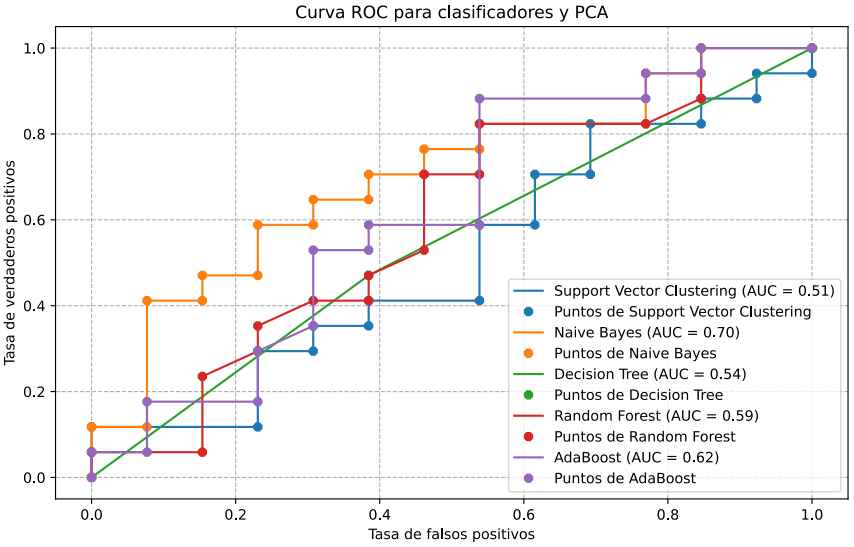


FIGURA 5.4: Curva ROC para PCA de los modelos implementados, donde se ve la relación entre el *ratio* de verdaderos positivos y de falsos positivos.

5.1.3 *Cross validation*

Se empleó la estrategia de KFold, en el cual se divide el *dataset* en K partes, donde una partición se utiliza para evaluar, y se entrena K veces [72]. En la Tabla 5.1 se puede ver el promedio y la desviación estándar obtenido con cada modelo de las K evaluaciones realizadas a partir de la base de datos construida. Para fines de esta investigación se utilizó este método con una partición de 10, es decir, K con un valor de 10, debido a que es el valor que más se recomienda utilizar al no presentar una varianza alta ni un sesgo marcado [73, 74].

En la Figura 5.5 se puede apreciar el *boxplot* de cada modelo, donde se ve de otra forma el promedio y la desviación estándar para cada caso. En este, se observa el rango en el que puede variar su puntaje de cada clasificador a partir de la desviación estándar y del promedio, representado por las líneas verdes. Esto significa que el

Modelos	Promedio	Desviación estándar
SVC [60]	0.66	0.13
Naives Bayes [62]	0.49	0.12
Decision Tree [61]	0.83	0.09
Random Forest [59]	**0.85**	**0.07**
AdaBoost [58]	**0.85**	**0.07**

TABLA 5.1: Promedio y desviación estandar de los puntajes obtenidos para evaluar el rendimiento y estabilidad de cada modelo al aplicar Kfolds con 10 particiones.

mejor puntaje posible sería el promedio más la desviación estándar, mientras que el peor caso sería el promedio menos la desviación estándar.

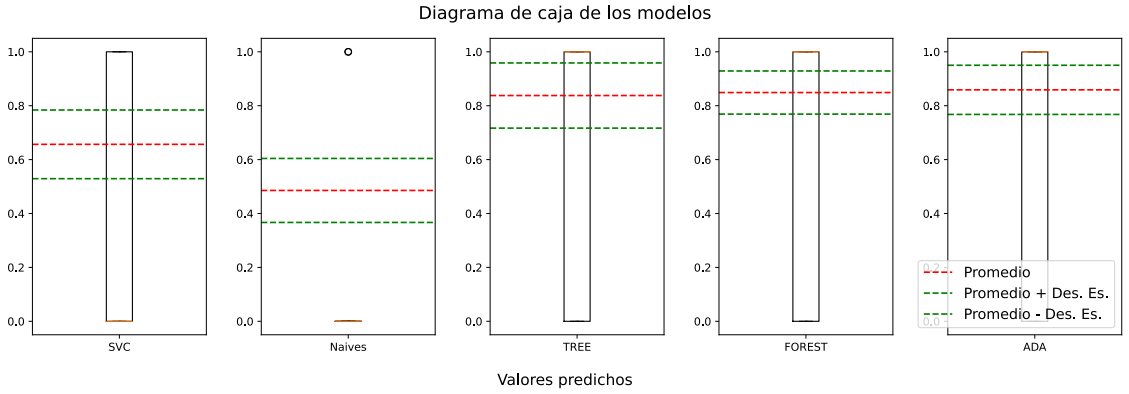


FIGURA 5.5: Los boxplots de cada modelo implementado con su promedio e indicando el valor máximo y mínimo posible a partir de la desviación estándar.

5.2 Detalles de implementación de la base de datos

Para el desarrollo de la base de datos se utilizaron tres tipos de máquinas virtuales en VirtualBox [75], de las cuales se extrajeron los volcados de memoria y se realizó el análisis de de cada muestra. En la Tabla 5.2 se pueden ver los detalles de cada entorno virtual.

	Máquina virtual 1	Máquina virtual 2	Máquina virtual 3
Sistema operativo	Ubuntu 20.04 LTS	Windows 10	Windows 10
SSD	50GB	40GB	40GB
RAM	8GB	2GB	2GB
Propósito	Análisis de volcados	Muestras benignas	Muestras malignas

TABLA 5.2: Detalles de las máquinas virtuales utilizadas para la creación de la base de datos.

Como se puede observar en la tabla 5.2, la primera máquina virtual se utiliza para realizar la extracción de características de los volcados de memoria, tanto malignos, como benignos. En el caso del segundo entorno, este se utiliza para ejecutar programas, como Google Chrome, Adobe, entre otros para generar las muestras benignas. Finalmente, en la última máquina virtual se ejecutan los mismos programas del segundo ambiente más el *malware* para crear la muestra maligna y así extraer el volcado de memoria.

5.2.1 Instancias evaluadas

Por otro lado, dentro de las muestras analizadas se tiene el mismo número de archivos maliciosos y benignos. Asimismo, los tipos de *malware* ejecutados son variados con el fin de que los modelos estén expuestos a las diversas características que pueda tener cada uno. Las clases que se utilizaron fueron las siguientes: Ransomware [48], Trojan [46], Rogues [49], Spyware [1], RAT [47], Worms [45] y Virus [44]. Todos estos tipos eran clasificados como malignos dentro del *dataset*, mas no por su familia de *malware*.

Igualmente, por el lado de los programas benignos que se utilizaron, en ambos tipos de muestras, para simular el ambiente más real posible fueron los siguientes:

Google Chrome, Microsoft Edge, Spotify, Skype, One Drive, Whatsapp, Adobe, Steam, Microsoft Office, Reproductor VLC, entre otras aplicaciones[2].

5.3 Experimentación

En la Tabla 5.3 se pueden observar los resultados obtenidos de haber ejecutado los modelos con la base de datos construida y se emplean las métricas propuestas.

Modelos	Precisión(%)	Exactitud(%)	Puntaje F(%)	Exhaustividad(%)
SVC	0.45	0.5	0.45	0.44
Naives Bayes	0.4	0.46	0.4	0.3
Decision Tree	0.8	0.8	0.8	0.79
Random Forest	**0.85**	**0.85**	**0.85**	**0.85**
AdaBoost	**0.85**	**0.85**	**0.85**	**0.85**

Tabla 5.3: Resultados con el *dataset* construido sin ninguna técnica aplicada, pero que fueron entrenados tres veces y se colocó el promedio en cada caso.

Para conseguir los resultados de la Tabla 5.3 se tuvo que hacer múltiples experimentos con los modelos propuestos con el fin de identificar los hiperparámetros que mejor guiaban el aprendizaje del modelo para el *dataset* creado.

Por el lado del SVC se utilizó el kernel de poly, parámetro de regularización igual a 0.5, grado de la función del kernel igual a 5, coef0 con valor de 2 y *break_ties* como verdadero.

En el caso de Naives Bayes, se usaron los valores por defecto.

Luego, con Decision Tree, también se usaron los valores por defecto.

[2]https://www.microsoft.com/es-es/store/most-popular/apps/pc

Por el lado de Random Forest, se utilizaron dos argumentos: 100 como número de estimadores y 42 para el *random_state*.

Finalmente, AdaBoost utilizó el clasificador Random Forest como modelo base con 100 como número de estimadores y 42 para el *random_state* y para los hiperparámetros del AdaBoost se usó el mismo número de estimadores, pero un *random_state* de 10.

5.4 Discusión

Durante la elaboración de la propuesta, desde los protocolos de comparación hasta la construcción de la base de datos, se hallaron varias dificultades y hallazgos que se van a detallar a continuación.

1. Las métricas empleadas permiten calificar los modelos de *machine learning* para poder compararlos e identificar el mejor de ellos. Estas son utilizadas en distintos trabajos a la hora de evaluar clasificadores [15, 25, 26, 43]. Asimismo, dentro de las técnicas de reducción de dimensionalidad, se usan especialmente para *datasets* con muchísimos atributos; sin embargo, se probó con el conjunto de datos creados para verificar si es que los resultados podían mejorar. Sin embargo, en las Figura 5.1 se aprecia que los modelos, especialmente el Random Forest y el AdaBoost, obtienen los mejores resultados, mayor valor de AUC, sin utilizar los reductores de dimensionalidad, con 0.87 y 0.90 respectivamente. En las Figuras 5.2, 5.3 y 5.4, los mejores valores de AUC, área bajo la curva, obtenidos son de Support Vector Clustering con 0.82 con el método PLS; Decision Tree con 0.69, con UMAP; y Naives Bayes con 0.70, con PCA. Esto nos demuestra que los modelos alcanzan un rendimiento óptimo sin los reductores de dimensionalidad. Además, con el *cross validation* se validan los

puntajes obtenidos por cada modelo, de modo que se puede analizar si hay espacio para mejoras en caso los puntajes no sean muy alentadores. Justamente, de acuerdo a los resultados obtenidos, se puede verificar que tanto el Random Forest como AdaBoost son los modelos que destacan al tener un promedio mayor en comparación a los demás de 0.85; no obstante, el Decision Tree no se queda atrás con un promedio de 0.83 respectivamente. Incluso la desviación estándar del Random Forest y del AdaBoost son las más bajas a comparación de los demás con un valor de 0.07, lo que significa que los modelos tienen un resultado estable y no tienden a variar. Esto también se puede evidenciar en la Figura 5.5, donde el diagrama de caja tanto del Random Forest y AdaBoost son muy similares.

2. Realizar la base de datos fue el proceso más complejo y que consumió más tiempo en todo el trabajo de investigación, puesto que consistía de los siguientes pasos principalmente: encontrar las muestras, crear la máquina virtual, extraer las características del volcado de memoria y juntar toda la data en un solo *dataset*. Inicialmente, se intentó llevar a cabo este procedimiento con Cuckoo Sandbox, ya que varios autores utilizaron esta herramienta [15, 25, 26, 43]; no obtante, se intentó descargar y ejecutar ese programa, pero no hubo resultado. Según los autores, el programa lo ejecutaron en equipos Windows 10 u 11 con procesador Intel, pero a pesar de replicar el mismo ambiente, incluso hasta con otros equipos como una Mac, no hubo resultado. Por ello, se buscó una alternativa para extraer los volcados de memoria y ahí fue donde se optó por usar DumpIt.

   Un detalle que se descubrió fue el hecho de que estos autores al utilizar Cuckoo Sandbox para ejecutar sus muestras maliciosas, estaban utilizando un entorno ideal, puesto que la herramienta solo analizaba el comportamiento de ese archivo maligno, mas no de otros procesos que en una computadora que utiliza un usuario común puede estar corriendo, ya sea Google Chrome, aplicaciones

de Microsoft Office, entre otras. Por este motivo, es que nació la idea de extraer los volcados de memoria de escenarios lo más cotidianos posibles con el fin de tener datos más precisos para que a la hora de entrenar los modelos de *machine learning*, los clasificadores realmente tengan que identificar la actividad maliciosa de cada muestra. No obstante, hay que entender los diversos motivos por los cuales una muestra real sería considerada más precisa que una ideal. Por un lado, los *sandboxes* al ser entornos controlados y simplificados, el *malware* es capaz de detectar que está siendo analizado, de modo que altera su comportamiento. Por otra parte, utilizar máquinas virtuales permite simular distintos entornos reales, es decir, mayor variabilidad en los datos, lo que permite que los modelos de *machine learning* detecten el *malware* en distintas situaciones. Además, generar volcados de memoria que tengan tanto actividades maliciosas como benignas permite que los modelos sean capaces de diferenciar de forma eficaz entre estos tipos de comportamientos. Adicionalmente, de los autores mencionados se identificaron los atributos que utilizan para que el *dataset* propuesto tenga todas estas características para que tenga más información para hallar si una muestra tiene datos sospechosos. De esta manera se obtienen resultados más certeros al buscar un bajo *ratio* de falsos positivos, ya que a veces los programas benignos pueden tener comportamientos sospechosos, pero eso no quiere decir que sea malicioso, justamente ahí es donde se encuentra el valor de estos modelos para poder diferenciar entre una muestra maligna y benigna. Entonces, durante la creación de las máquinas virtuales y la ejecución del *malware*, también se buscaba abrir otros programas para simular un entorno real. Mismo caso se dio para las muestras benignas, solo que sin la ejecución del archivo malicioso. Incluso, para que no sean los mismos programas, en unas muestras se ejecutaban una o dos aplicaciones más y en otras podían ser hasta cuatro o cinco para que la data no fuera la misma y pueda ser lo más variada posible. Una vez que se tenían los archivos csv de

cada instancia es que recién se podía proceder a armar el *dataset* al juntar los datos más importantes.

3. De lo observado en la experimentación, se puede ver que los modelos Random Forest y AdaBoost son los mejores al obtener 85 % de precisión. No obstante, hay que considerar el puntaje obtenido por el *cross validation*, donde la desviación estandar es de aproximadamente 0.07, lo que significa que ese resultado aún puede experimentar pequeñas variaciones. Asismimo, en las demás métricas se puede observar que estos modelos son los que consiguen mejores resultados, aunque cabe recalcar que el clasificador Decision Tree obtuvo 80 % de precisión, lo que lo ubica detrás del Random Forest y del AdaBoost.

CONCLUSIONES

Con el aumento de incidentes por *malware*, la detección de ellos ha ganado bastante importancia. Debido a que, los métodos tradicionales ya no son tan efectivos, se implementó en esta investigación modelos de *machine learning* y una base de datos a partir del análisis forense de memoria con la finalidad de poder detectar este tipo de archivos en un equipo de forma efectiva. Esto se logra evitando las desventajas asociadas con otros métodos, que incluyen lentitud, la falta de efectividad contra nuevas variantes que no están registradas en una base de datos, entre otros.

A partir de los experimentos, se observó que los modelos con mejores resultados a partir del entrenamiento con el *dataset* construido fueron el Random Forest y el AdaBoost. Estos clasificadores obtuvieron una precisión de 85 % y también consiguieron los mejores resultados en las demás métricas. Adicionalmente, se apreció que sin la aplicación de los métodos de reducción de dimensionalidad, los modelos con mejores resultados fueron el Random Forest y el AdaBoost con valores de 0.87 y 0.90. Esto respalda lo obtenido en las métricas y, además, nos demuestra que los reductores de dimensionalidad reducen considerablemente el área bajo la curva de los modelos sugiriendo que no es óptimo el uso de reductores de dimensionalidad para este *dataset*. De igual modo, al aplicar el *cross validation* usando k-folds de 10, los clasificadores con mejor puntaje fueron el AdaBoost junto al Random Forest con un resultado de 0.85, seguidos del Decision Tree con un promedio de 0.83. Esto nos permite sustentar el resultado obtenido al inicio de la precisión, ya que el Random Forest y el AdaBoost obtienen resultados superiores al de los demás modelos, especialmente en la prueba de *cross validation*, donde también consiguen un puntaje estable al tener una desviación estándar menor a 0.1. Por otra parte, pese a los problemas que se presentaron para utilizar la herramienta de Cuckoo Sandbox, se elaboró la base de datos para poder simular entornos lo más reales posibles, además de considerar los atributos más relevantes para el *dataset*, de modo que puedan ayudar en identificar

la presencia de un *malware*. También, se consideró una amplia variedad de tipos de *malware*, puesto que esto ayuda a que haya una mayor diversidad y que los modelos estén expuestos a más casos con el fin de que su entrenamiento sea más completo. De esta forma se busca que los clasificadores obtengan resultados que no sean sesgados y que tengan una mayor precisión a la hora de clasificar entre un archivo maligno y benigno. Esto justamente con la intención de corregir las limitaciones de las bases de datos comentadas en el estado del arte. Finalmente, como lecciones aprendidas de los hallazgos de esta investigación se destaca, principalmente, que no es opción limitarse a una sola herramienta, ya que uno puede perder muchísimo tiempo al intentar que funcione, pero en caso no logre ejecutarse, hay que tener alternativas para poder continuar con el trabajo. Ese fue el caso con la herramienta de Cuckoo Sandbox que finalmente tuvo que ser reemplazada por DumpIt para generar los volcados de memoria. De ese inconveniente fue de donde nació la propuesta de generar muestras lo más reales posibles para tener información lo más cercana posible a la realidad.

En futuros trabajos se busca construir una base de datos más extensa, con más atributos y más muestras, a fin de poder entrenar mejor los modelos y, además, poder aplicar una clasificación de multiclases, de modo que se pueda determinar el proceso de respuesta correspondiente a cada tipo de *malware*. Asimismo, se busca expandir esta investigación a otros sistemas operativos, como macos a fin de detectar archivos maliciosos con efectivididad, es decir, con un bajo ratio de falsos positivos. Por otro lado, la intención más adelante es realizar un *framework* para la automatización de todo el flujo de la detección y respuesta ante *malware* con el fin de ofrecerlo al público para proteger sus equipos y que funcione como un antivirus y antimalware.

REFERENCIAS BIBLIOGRÁFICAS

[1] A. R. D. Javaheri, M. Hosseinzadeh, "Detection and elimination of spyware and rasomware by intercepting kernel-level system routines," *IEEE Access*, 2018.

[2] A. Petrosyan. (2024) State of malware worldwide - statistics facts. [Online]. Available: https://www.statista.com/topics/8338/malware/#topicOverview

[3] M. Kapko. (2024) Microsoft reveals ransowmare attacks against its customers nearly tripled last year. [Online]. Available: https://www.cybersecuritydive.com/news/microsoft-customers-ransomware-attacks-triple/730011/#:~:text=Microsoft%20customers%20confronted%20nearly%20triple,July%202023%20and%20June%202024.

[4] A. A. A. M. M. A. Mohammed N. Alenezi, Haneen Alabdulrazzaq, "Evolution of malware threats and techniques: A review," *International Journal of Communication Networks and Information Security*, 2020.

[5] S. G. A. Bhardwaj, "Keyloggers: silent cyber security weapons," *Network Security*, pp. 14–19, 2020.

[6] H. G. S. L. M. M. K. J. A. Kumar, K. Kumar, "Keylogger awareness and use in cyber forensics," *Rising Threats in Expert Applications and Solutions*, 2022.

[7] M. F. Z. A. Ruhani, "Keylogger: The unsung hacking weapon," *Borneo International Journal eISSN 2636-9826*, 2023.

[8] S. B. P. G. M. Kalpesh, D. Kataria, "Real time working of keylogger malware analysis," *International Journal of Engineering Research Technology*, 2020.

[9] A. O. M. Dener, G. Ok, "Malware detection using memory analysis data in big data environment," *Applied Sciences*, 2022.

[10] N. J. A. u. R. K. S. S. H. Shah, A. R. Ahmad, "Memory forensics-based malware detection using computer vision and machine learning," *Electronics*, 2022.

[11] [Online]. Available: https://www.wireshark.org

[12] [Online]. Available: https://hex-rays.com/ida-pro/

[13] [Online]. Available: https://github.com/microsoft/lida

[14] N. J. A. u. R. K. S. S. H. Shah, A. R. Ahmad, "Memory forensics-based malware detection using computer vision and machine learning," *Electronics*, 2022.

[15] S. M. S. M. Azeem, "Automate memory forensics," 2022.

[16] [Online]. Available: https://www.volatilityfoundation.org

[17] [Online]. Available: http://www.rekall-forensic.com

[18] [Online]. Available: https://fireeye.market/apps/211364

[19] S. L. M. G. M. D. J. D. C. T. H. Nyholm, K. Monteith, "The evolution of volatile memory forensics," *Journal of Cybersecurity and Privacy*, 2022.

[20] M. F. M. J. A. A. M. S. G. R. A. Case, R. Maggio, "Hooktracer: Automatic detection and analysis of keystroke loggers using memory forensics," *Computers and Security*, 2020.

[21] G. C. J. Taylor, B. Turnbull, "Volatile memory forensics acquisition efficacy: A comparative study towards analysing firmware-based rootkits," *Proceedings*

of the 13th International Conference on Availability, Reliability, and Security, 2018.

[22] S. J. M. Parekh, "Memory forensic: Acquisition and analysis of memory and its tools comparison," *International Journal of Engineering Technologies and Management Research*, 2018.

[23] T. P. R. Chaganti, V. Ravi, "Image-based malware representation approach with efficientnet convolutional neural networks for effective malware classification," *Journal of Information Security and Applications*, 2022.

[24] M. Omar, "New approach to malware detection using optimized convolutional neural network," *arXiv*, 2023.

[25] K. A. Z. R. Sihwail, K. Omar, "An effective memory analysis for malware detection and classification," *Computers, Materials Continua*, 2021.

[26] A. M. D. U. L. R. A. S. M. Murthaja, B. Sahayanathan, "An automated tool for memory forensics," *2019 International Conference on Advancements in Computing*, 2019.

[27] P. V. M. Conti, S. Khandhar, "A few-shot malware classification approach for unknown family recognition using malware feature visualization," *Computers Security*, 2022.

[28] O. D. A. d. B. Rey Reza, Anqi Xu, "Adversarial examples in modern machine learning: A review," *arXiv*, 2019.

[29] V. V. J. Shaukat, S. Luo, "A novel deep learning-based approach for malware detection," *Engineering Applications of Artificial Intelligence*, 2023.

[30] C. F. Y. F. Ya Pan, Xiuting Ge, "A systematic literature review of android malware detection using static analysis," *IEEE Access*, 2020.

[31] L. Y. T. B. N. I.-O. Nitin Naik, Paul Jenkins, "Fuzzy-import hashing: A static analysis technique for malware detection," *Forensic Science International: Digital Investigation*, 2021.

[32] K. C.-C. . Y. C.-S. Zhang, S.-H., "Static pe malware type classification using machine learning techniques," *International Conference on Intelligent Computing and its Emergin Applications*, 2019.

[33] D. G. V. Syrris, "On machine learning effectiveness for malware detection in android os using static analysis data," *Journal of Information Security and Applications*, 2021.

[34] M. I. M. Ijaz, M. H. Durad, "Static and dynamic malware analysis using machine learning," *International Bhurban Conference on Applied Sciences and Technology*, 2019.

[35] S. S. R. C. C. Raghuraman, S. Suresh, "Static and dynamic malware analysis using machine learning," *Advances in Intelligent Systems and Computing*, 2019.

[36] M. A. A. H. S. Galal, Y. B. Mahdy, "Behavior-based features model for malware detection," *Journal of Computer Virology and Hacking Techniques*, 2015.

[37] I. Z. E. Amer, "A dynamic windows malware detection and prediction method based on contextual understanding of api call sequence," *Computers Security*, 2020.

[38] J. Z. Q. L. J. Liu, Y. Feng, "Mrm-dldet: a memory-resident malware detection framework based on memory forensics and deep neural network," *Cybersecurity*, 2023.

[39] R. V. G. G. R. I. A. A.-Gombe, S. Sudhakaran, "crgb_mem: At the intersection of memory forensics and machine learning," *Forensic Science International: Digital Investigation*, 2023.

[40] T. D. P. R. Chaganti, V. Ravi, "Image-based malware representation approach witht efficientnet convolutional neural networks for effective malware classification," *Journal of Information Security and Applications*, 2022.

[41] A. T. A. H. L. Tristan Carrier, Princy Victor, "Detecting obfuscated malware using memory feature engineering," *8th International Conference on Information Systems Security and Privacy*, 2022.

[42] S. S. E. Gandotra, D. Bansal, "Malware analysis and classification: A survey," *Journal of Information and Security*, 2014.

[43] K. Chumachenko, "Machine learning methods for malware detection and classification," 2017.

[44] J. S. J. Horton, "Computer viruses. an introduction," *Proceedings of the 20th Australasian Computer Science Conference*, 1997.

[45] R. Sharp, "An introduction to malware," 2007.

[46] D. K. M. Moffie, "Hunting trojan horses," *Proceedings of the 1st Workshop on Architectural and System Support for Improving Software Dependability*, 2006.

[47] J. M. Brettingen, "Development of a customized remote access trojan (rat) for educational purposes within the field of malware analysis."

[48] U. Bansal, "A review of ransomware attack," *International Conference on Secure Cyber Computing and Communications*, 2021.

[49] D. L. K. V. K. G. S. K. Deepali Yadav, Gautam Kumar, "Malware techniques and its effect: A survey," *International Conference on Communications and Cyber Physical Engineering*, 2021.

[50] K. A. Z. A. R. Sihwail, K. Omar, "A survey on malware analysis techniques: Static, dynamic, hybrid and memory analysis," *International Journal on Advanced Science Engineering Information Technology*, 2018.

[51] J. H. A. R. B. Ahmed, M. Shoikot, "Keylogger detection using memory forensic and network monitoring," *International Journal of Computer Applications*, vol. 177, 2019.

[52] J. S. R. Petrik, B. Arik, "Towards architecture and os-independent malware detection via memory forensics," *Computer and Communication Security*, 2018.

[53] J. León, "Análisis forense en entorno windows," 2017.

[54] R. J. R. Daniel Uroz, "On challenges in verifying trusted executable files in memory forensics," *Forensic Science International: Digital Investigation*, 2020.

[55] A. S. A. Sharma, A. Kaur2, "Supervised and unsupervised prediction application of machine learning," *International Conference on Cyber Resilience*, 2022.

[56] R. K. Judy Subramanian, "Detection and clasification of malware for cyber security using machine learning algorithms," *Eighth International Conference on Science Technology Engineering and Mathematics (ICONSTEM)*, 2023.

[57] H. J. I. H. S. G.M. Sakhawat Hossain, Kaushik Deb, "Pdf malware detection: Towards machine learning modeling with explainability analysis," *IEEE Access*, 2023.

[58] R. M. A. J. Hatwell, M. Medhat, "Ada-whips: explaining adaboost classification with applications in the health sciences," *BMC Medical Informatics and Decision Making*, 2020.

[59] S. S. A. Babu, "A brief survey on random forest ensembles in classification model," *Lecture Notes in Networks and Systems*, 2018.

[60] L. R.-M. A. L. J. Cervantes, F. Garcia-Lamont, "A comprehensive survey on support vector machine classification: Applications, challenges and trends," *Neurocomputing*, 2020.

[61] A. M. B. Taha, "Classification based on decision tree algorithm for machine learning," *Journal of Applied Science and Technology Trends*, 2021.

[62] H. K. I. Wickramasinghe, "Naive bayes: applications, variations and vulnerabilities: a review of literature with code snippets for implementation," *Soft Computing*, 2020.

[63] M. G. P. Chandra, M. Hajra, "Supervised classification algorithms in machine learning: A survey and review," *Emerging Technology in Modelling and Graphics*, 2019.

[64] H. Gutiérrez, "Evaluación de herramientas de software libre, para el sistema operativo windows, en la adquisición de evidencias de la memoria ram," *Publicaciones e Investigación*, 2022.

[65] [Online]. Available: https://volatility3.readthedocs.io/en/stable/

[66] M. O.-O.-A. A. E. A. Omer Aslan, Semih Serkant, "A comprehensive review of cyber security vulnerabilities, threats, attacks, and solutions," *Electronics*, 2023.

[67] [Online]. Available: https://scikit-learn.org/stable/index.html

[68] K. H. Tahir Mehmood, Solve Saebo, "Comparison of variable selection methods in partial least squares regression," *Journal of Chemometrics*, 2020.

[69] F. K.-M. C. Benyamin Ghojogh, Ali Ghodsi, "Uniform manifold approximation and projection (umap) and its variants: Tutorial and survey," *arXiv*, 2021.

[70] A. M. Basna Salih, "A review of principal component analysis algorithm for dimensionality reduction," *Journal of Soft Computing and Data Mining*, 2021.

[71] T. B. Papia Ray, S. Surender Reddy, "Various dimension reduction techniques for high dimensional data analysis: a review," *Artificial Intelligence Review*, 2021.

[72] M. C. Benyamin Ghojogh, "The theory behind overfitting, cross validation, regularization, bagging, and boosting: Tutorial," *arXiv*, 2019.

[73] T. H. R. T. Gareth James, Daniela Witten, *An Introduction to Statistical Learning*. Springer, 2013.

[74] K. J. Max Kuhn, *Applied Predictive Modeling*. Springer, 2013.

[75] [Online]. Available: https://www.virtualbox.org