Universidad Zaragoza

Trabajo de Fin de Grado

Prevalencia de tipos de ASEPs en malware de Windows

Prevalence of Types of ASEPs in Windows Malware

Autor

Carlos Borau González

Director

Ricardo Julio Rodríguez Fernández

ESCUELA DE INGENIERÍA Y ARQUITECTURA
2022

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe remitirse a seceina@unizar.es dentro del plazo de depósito)

D./Dª. Carlos Borau González ,

en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de Estudios de la titulación de Grado en Ingeniería Informática (Título del Trabajo)

Prevalencia de tipos de ASEPs en malware de Windows

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, a 22 de Junio de 2022

Fdo: Carlos Borau González

AGRADECIMIENTOS

A Ricardo, por ofrecerme la oportunidad de realizar un trabajo que me ha permitido desarrollarme como informático y del que puedo sentirme orgulloso. A mi familia, por creer siempre en mí, compartir mis logros y ayudarme a levantarme tras mis tropiezos. A mis amigos, por darme ánimos y por su apoyo incondicional. A todos los que me han acompañado en este viaje, a los que hoy están a mi lado y, en especial, a aquellos cuyo recuerdo me sigue dando fuerzas.

Para todos ellos, sólo tengo palabras de agradecimiento.

RESUMEN

Los puntos de inicio automático de ejecución (*Auto Start Execution Points*, ASEPs) son aquellos lugares del sistema operativo que permiten a un programa ejecutarse de forma automática sin la necesidad de que haya una interacción explícita con el usuario. En el ámbito de la ciberseguridad, es común que el *malware* (software malicioso) haga uso de estos elementos para garantizar su persistencia en un sistema comprometido durante el mayor tiempo posible.

Este proyecto se centra en diseñar un flujo de trabajo que permita estudiar la prevalencia de los ASEPs en malware de Windows a través de un sistema automatizado, capaz de obtener y procesar muestras de malware de diferentes fuentes, así como de coordinar diferentes máquinas encargadas de analizar dinámicamente su comportamiento para, posteriormente, categorizarlas en función de los resultados de dicho análisis.

Una vez finalizada la fase de experimentación del trabajo, se ha podido comprobar que el sistema de análisis desarrollado es capaz de llevar a cabo, de forma exitosa, el análisis y clasificación de la gran mayoría de muestras introducidas en el *pipeline* de análisis, ofreciendo un reporte detallado de los resultados de este proceso. Por otro lado, se ha podido constatar que el sistema diseñado ha logrado detectar en múltiples muestras el uso de diferentes tipos de ASEPs y, posteriormente, clasificarlos acertadamente. Durante el desarrollo del proyecto han surgido una serie de dificultades que han limitado el alcance original del estudio y para las cuales se ofrece un análisis de su impacto, así como diversas propuestas para solucionarlas.

ABSTRACT

Auto-start execution points (ASEPs) are those places in an operating system that allow a program to be executed automatically without explicit need for user interaction. In the field of cybersecurity, it is common for malware (malicious software) to make use of these elements to ensure their persistence in a compromised system for as long as possible.

This project's focus is the development of a workflow that allows studying the prevalence of ASEPS types in Windows malware through an automated system, capable of obtaining and processing malware samples from various sources, as well as coordinating different machines responsible for dynamically analyzing their behavior to subsequently categorize them.

Once the experimentation phase has been completed, it has been possible to verify that the analysis system developed is capable of successfully performing the analysis and categorization of the vast majority of samples fed to the analysis pipeline, providing a detailed report about the analysis results. In addition, it has been possible to verify that the designed system has managed to detect, and subsequently accurately classify the use of different types of ASEPs in various samples. During the development of this project a series of difficulties have arisen that have limited the original scope of the study. An analysis of the impact of each of these difficulties is provided, as well as some approaches on how to address them.

Índice General

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción y objetivos

Este capítulo introduce la motivación, objetivos, metodología y alcance del proyecto desarrollado, así como una breve descripción de la estructura del mismo en adelante.

1.1. Motivación

La seguridad informática es una disciplina que cada año adquiere mayor relevancia, tanto para las empresas como para los particulares [8]. De igual forma, las amenazas informáticas no dejan de crecer y volverse más complejas, y con ellas aumenta el riesgo de que las infraestructuras informáticas y los datos que contienen se vean comprometidos. Es en este contexto en el que se busca realizar estudios que permitan identificar los métodos más comúnmente empleados por el malware para infectar y persistir en un sistema, para así poder idear diversas estrategias de defensa contra nuevas amenazas [23].

De los diversos mecanismos que emplea el malware, los puntos de autoinicio de ejecución (ASEPs, del inglés) [27] son particularmente interesantes, pues permiten que programas se ejecuten sin la necesidad de que un usuario interactúe con ellos explícitamente, y, por el momento, no se han realizado estudios a cerca de la prevalencia de estos mecanismos en muestras de malware contemporáneas. Así pues, con este trabajo se pretende dar un primer paso en este campo proporcionando un sistema de código abierto con el que realizar análisis de forma rápida, distribuida y modular.

1.2. Objetivos, metodología y alcance

El objetivo de este proyecto es el de **estudiar la prevalencia de los ASEPs en malware de sistemas operativos Windows**. Para realizar este estudio se ha diseñado un *workflow* de análisis de malware que permite ejecutar muestras de malware en un entorno controlado para extraer sus trazas de ejecución. Estas trazas

se analizan posteriormente en busca de una serie de patrones de comportamiento con los que discernir si se utiliza algún tipo de ASEP en particular, según la taxonomía definida en la literatura [30]. Estos patrones de comportamiento se han obtenido a partir de una serie de trazas de ejecución sintéticas, creadas mediante una serie de pruebas de concepto desarrolladas para tal fin.

Para llevar a cabo la experimentación, se ha diseñado un sistema con diferentes componentes que permiten distribuir entre varias máquinas las tareas del *workflow* de análisis. Este sistema, que incorpora herramientas de análisis tanto dinámico como estático, integra la capacidad de obtener muestras de repositorios web así como del almacenamiento local, la generación de trazas de ejecución de estas muestras, y su clasificación en función de dichas trazas. Así pues, el sistema resultante puede emplearse para llevar a cabo distintos tipos de análisis sobre múltiples muestras de malware, obtenidas de diferentes repositorios y empleando diferentes soluciones de análisis simplemente modificando los componentes de manera apropiada.

El alcance del proyecto se ha visto reducido debido a una serie de limitaciones surgidas de la utilización de la herramienta escogida para el análisis y generación de trazas, por lo que no se ha podido realizar un estudio tan completo y riguroso como se planteó en un principio. Por otro lado, los esfuerzos se han centrado más en el desarrollo modular y distribuido del sistema de análisis, para así permitir que en un futuro se continúe este estudio de la forma más ágil y eficiente posible, permitiendo incluso cambiar las herramientas utilizadas sin mayor dificultad.

1.3. Estructura del documento

Este documento se encuentra dividido en 6 capítulos y un anexo. En el **Capítulo 2** se definen una serie de conceptos fundamentales para la correcta comprensión del resto del documento. El **Capítulo 3** trata el proceso del diseño del sistema de análisis, las distintas componentes de éste y las soluciones escogidas para cada tarea. En el **Capítulo 4** se detalla la fase de experimentación, las pruebas de concepto realizadas, así como la arquitectura del despliegue utilizada y los resultados obtenidos. En el **Capítulo 5** se hace mención de diversos proyectos relacionados con las herramientas empleadas para este trabajo. Por último, el **Capítulo 6** expone las conclusiones del trabajo y menciona aspectos a mejorar y desarrollar a futuro para continuar con el estudio. Al final del documento, el **Apéndice A** presenta un desglose de las horas dedicadas a las distintas fases del proyecto.

Capítulo 2

Conceptos previos

En este capítulo se introducen varios conceptos y definiciones necesarios para la comprensión del trabajo desarrollado, las herramientas empleadas en el mismo y los resultados y limitaciones de la experimentación que se ha llevado a cabo. Entre estos conceptos caben destacar aquellos relacionados con el análisis de malware, los diversos sistemas de análisis que se han estudiado como candidatos para incorporarse al workflow de análisis, y por último los múltiples tipos de ASEPs y su clasificación según la literatura.

2.1. Análisis de malware

El análisis de malware es un proceso por el cual se obtiene toda la información relevante a cerca de una muestra de código (tipo de archivo, librerías utilizadas, comportamiento en ejecución, firma digital, etcétera) para así determinar si se trata de software dañino o benigno [35]. A lo largo de las últimas décadas se han desarrollado múltiples métodos cada vez más innovadores y ágiles para llevar a cabo esta tarea, aunque todos pueden clasificarse bajo dos grandes categorías de análisis [28]:

- **Análisis estático**. Este tipo de análisis se centra en extraer información de una muestra sin ejecutar el código que contiene. Este tipo de aproximación, aunque rápida y simple, puede no ser suficiente para categorizar una muestra como maliciosa, especialmente si se trata de malware relativamente actual y cuya firma aún no ha sido registrada.

- **Análisis dinámico**. Las técnicas que se engloban en esta categoría de análisis son muy variadas, aunque todas presentan un rasgo común: el estudio del comportamiento del código durante su ejecución. Esto incluye no sólo la propia ejecución de la aplicación, sino también el tráfico de red, contenidos de memoria y cualquier otra interacción con la API del sistema operativo.

Ya que este tipo de análisis implica la ejecución del código, es necesario que se realice en un entorno aislado y controlado. Estos tipos de entornos se denominan *sandbox* [14] y permiten la monitorización de la ejecución de la muestra analizada, realizándose normalmente en una máquina virtual en lugar de una máquina física.

2.2. Sistemas de análisis

Para el desarrollo de este proyecto se va a entender un **Sistema de Análisis (de Malware)** como una estructura de componentes software que llevan a cabo una serie de tareas (automatizadas en la medida de lo posible) con el fin de, dado un conjunto de muestras de *malware*, llevar a cabo un estudio a cerca de sus características y comportamiento para emitir un veredicto acerca de su naturaleza una vez finalizado. Las diversas componentes de este sistema pueden estar implementadas con tecnologías diferentes, pero deben de ser capaces de interactuar entre sí sin problemas.

Una de las partes más relevantes de este proyecto ha sido el desarrollo de este sistema de análisis y clasificación de muestras de malware. El diseño, ajuste y experimentación con este sistema se detalla en profundidad en el **Capítulo 3** y el **Capítulo 4**.

2.3. Técnicas de análisis de *software*

Los sistemas de análisis de malware pueden emplear distintas técnicas para analizar muestras y posteriormente ofrecer un reporte de los resultados. Una de las técnicas que suelen emplear la mayoría de sistemas de testeo de software es el **Análisis de caja negra** [3]. Esta técnica consiste en comprobar la funcionalidad de un código sin tener en cuenta su estructura interna o detalles de implementación, únicamente centrando el foco del análisis en las entradas y salidas del sistema.

2.4. Puntos de Autoinicio de Ejecución (ASEPs)

El principal objetivo de estudio de este proyecto son los **Puntos de Autoinicio de Ejecución** (ASEPs, por sus siglas en inglés), definidos como el conjunto de mecanismos en un sistema operativo que un programa puede utilizar para habilitar la ejecución automática de código, sin que exista una invocación explícita previa de un usuario [34] [27].

Recientemente se ha propuesto una taxonomía de estos mecanismos [30], agrupándolos en cuatro categorías principales (*mecanismos de persistencia del*

sistema, *abuso del cargador de programas*, *abuso de aplicaciones* y *abuso del comportamiento del sistema*) en función de los métodos abusados por el malware para persistir en el sistema. La mayoría de estos mecanismos dependen del **Registro de Windows**.

El Registro de Windows es una base de datos jerárquica central que almacena datos críticos para la configuración del sistema, aplicaciones y dispositivos hardware [1]. Esta estructura se compone de "*colmenas*", cada una de las cuales tiene una clave raíz de la que nacen el resto de las claves de forma similar a un árbol. Dependiendo del ámbito de la configuración que almacena una clave, ésta se encontrará bajo una de las 5 claves raíz predefinidas:

- **HKEY_CURRENT_USER (HKCU)**: Contiene la información asociada al perfil del usuario que ha iniciado sesión en la máquina. Puede ser modificada con los permisos de un usuario estándar. Esta colmena será la preferente a modificar para el malware que se ejecuta sin permisos de administración.
- **HKEY_USERS (HKU)**: Contiene todos los perfiles de usuarios cargados actualmente en el equipo. Requiere permisos de administrador para ser modificada.
- **HKEY_LOCAL_MACHINE (HKLM)**: Contiene información relativa al sistema completo. Es necesario tener permisos de administrador para modificar las claves y valores de esta colmena. Muchos de los ASEPs basados en Registro de Windows modifican o crean claves bajo esta raíz.
- **HKEY_CLASSES_ROOT (HKCR)**: Contiene las correspondencias entre extensiones de ficheros y el programa a utilizar para abrirlos. Requiere permisos de administrador para ser modificada. Un tipo de ASEP particular (*Extension Hijacking*) hace uso de esta colmena para lograr persistencia.
- **HKEY_CURRENT_CONFIG (HKCC)**: Contiene información acerca del perfil hardware que el equipo utiliza al arrancar el sistema. Requiere permisos de administrador para ser modificada.

Por otro lado también se pueden encontrar ASEPs que dependen de rutas específicas del sistema de ficheros (principalmente carpetas del sistema o configuración particular de cada usuario). De forma similar a las claves del registro, dependiendo del ámbito de configuración serán necesarios (o no) privilegios de administración para emplear los mecanismos necesarios para asegurar la persistencia explotando estos ASEPs.

Capítulo 3

Diseño del sistema de análisis

En este capítulo se detalla el proceso de diseño del sistema de análisis de malware desarrollado y utilizado para la posterior experimentación, el *workflow* a partir del cual se ha realizado el diseño, las tareas en las que se divide y las componentes diseñadas para desempeñar cada una de estas tareas, y las herramientas escogidas para estas componentes.

3.1. *Workflow* de análisis de malware

Previamente al diseño del sistema completo de análisis de malware, se ha ideado un *workflow* genérico de análisis de malware. Para realizar la división en las tareas clave del *workflow* de análisis se ha partido de una situación inicial y se han considerado los procesos necesarios para llegar a un resultado final deseado.

- *Situación Inicial*: No hay muestras con las que trabajar.
- *Tarea 1*: Obtener y procesar muestras de malware.
- *Tarea 2*: Analizar el comportamiento de las muestras obtenidas y generar sus trazas de ejecución.
- *Tarea 3*: Clasificar las muestras analizadas en función de las trazas de ejecución generadas.
- *Resultado Final*: Muestras analizadas y categorizadas.

Así pues, de la división del workflow han surgido tres tareas principales: *Obtención de muestras*, *Generación de trazas de ejecución* y *Categorización de muestras*. El diseño del workflow realizado se muestra en la **Figura 3.1**. Esta figura recoge los distintos aspectos expuestos en los siguientes apartados de forma general, sin especificar qué tipo de tecnología/herramienta se ha escogido para llevar a cabo cada tarea. Al ser

un workflow modular, cualquiera de las tecnologías debería poder sustituirse por otra que fuera capaz de desempeñar la misma tarea, sin afectar al resto de la cadena. A continuación, se detalla el funcionamiento de cada una de estas tareas.

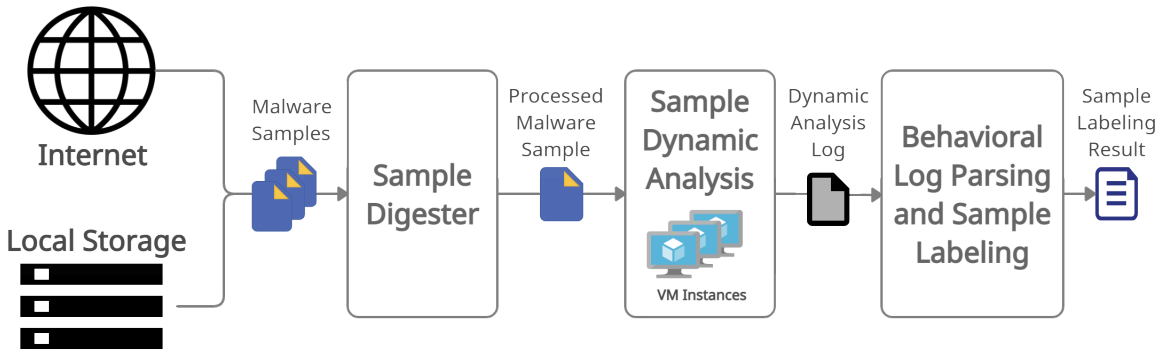


Figura 3.1: Workflow de análisis de malware diseñado

3.1.1. Obtención de muestras - *Digester*

La primera tarea que debe desempeñarse es la obtención de muestras de malware para su posterior análisis y clasificación. El componente que desempeña esta tarea se ha denominado **Digester**. El Digester se encarga de la ingesta y procesamiento de muestras de malware para ponerlas a disposición de la siguiente tarea del workflow. Cada muestra procesada se añade a una cola, a la que se accede en exclusión mutua, para ser posteriormente extraída cuando sea analizada.

Se ha considerado que el Digester debe ser capaz de aceptar muestras introducidas manualmente por un usuario en la cadena de análisis, pero también debe ser capaz de obtener muestras automáticamente. Es por esto que se ha diseñado este componente para poder procesar muestras de dos orígenes diferentes, como se explica a continuación:

Muestras introducidas desde el almacenamiento local

Las muestras procedentes del almacenamiento local son aquellas que un usuario introduce en la cadena de análisis manualmente. Para procesarlas se requieren los siguientes pasos:

- *Detección de la muestra*: En tiempo de ejecución se deben de poder detectar nuevas muestras añadidas por un usuario. Para ello se ha programado un proceso *"fisgón"* que, periódicamente y durante la ejecución del sistema de análisis, se dedica a consultar un directorio determinado en el que se espera que el usuario deposite muestras a analizar.

- *Comprobación de validez*: Para que una muestra se considere válida para analizar debe de cumplir una serie de requisitos, entre los cuales destaca el tipo de archivo (.exe, .dll ...).

- *Cálculo de la firma*: Para identificar una muestra de malware inequívocamente se procede a calcular su firma SHA256 y renombrarla con este valor. Esto permite descartar muestras ya analizadas, así como organizar los resultados finales de forma precisa.

- *Encolado de la muestra*: Si la muestra introducida no se encuentra entre las ya analizadas, se procede a encolarla en una cola de muestras disponibles para su análisis posterior.

Muestras descargadas de repositorios web

Existen múltiples repositorios online de los cuales se pueden descargar muestras de malware. Estos repositorios pueden ser abiertos o de pago y requerir o no identificación, entre otras características. Implementar la interacción con un repositorio web de malware permite la descarga de conjuntos relativamente grandes de muestras de malware, que además normalmente se encuentran ya clasificadas por tipo de archivo y con su firma SHA256 digital calculada.

A la hora de escoger un repositorio del que descargar muestras para el estudio se ha buscado un repositorio abierto, que ofrezca una API documentada con la que descargar fácilmente muestras y, sobre todo, que ofrezca una cantidad significativa de muestras de malware reciente periódicamente. De igual manera, otras características clave a la hora de agilizar el proceso de adquisición de muestras han sido: la capacidad de filtrar muestras por diferentes características (extensión, firma digital, fecha de publicación en el repositorio, etcétera) y la existencia (o no) de un límite a la descarga de muestras en un determinado periodo de tiempo.

La **Tabla 3.1** lista los diversos repositorios que se han valorado como alternativas de las que obtener las muestras a analizar, así como las principales características. Estas características se han utilizado para elegir el repositorio más adecuado para el estudio.

Repositorio	Requiere identificación	Presenta API	Ofrece filtrado de muestras	Límite de descargas	Muestras recientes	Histórico de muestras	De pago
VirusShare [31]	Sí	No	Sí	Sí	Sí	Sí	No
Hybrid-Analysis [16]	Sí	Sí	Sí	Sí	Sí	Sí	No
VirusTotal [33]	Sí	Sí	Sí	Sí	Sí	Sí	No
VirusSign [32]	Sí	No	No	Sí	Sí	De Pago	No
MalwareBazaar [22]	No	Sí	Sí	Recomendado	Sí	Sí	No

Tabla 3.1: Repositorios web de malware estudiados

De entre los repositorios listados, se ha escogido **MalwareBazaar** [22] puesto que se trata de un repositorio abierto de malware que presenta una API extensamente documentada, permite filtrado de muestras de malware y proporciona la capacidad de descargar grandes paquetes de muestras tanto recientes como del histórico.

Se ha añadido al Digester la capacidad de descargar muestras de MalwareBazaar a través de su API. En este caso, procesar estas muestras sólo requiere la consulta a la API de una lista de firmas SHA256 de las últimas muestras añadidas y descargar las muestras cuyas firmas no estén en la lista de muestras analizadas.

3.1.2. Generación de trazas de ejecución - Analyzer

Una vez se dispone de un conjunto de muestras de malware, la siguiente tarea es analizarlas para obtener sus trazas de ejecución. Para esta tarea se ha desarrollado el componente **Analyzer**. El Analyzer debe ser capaz de, dada una muestra de malware, ejecutar un análisis dinámico de la misma y proporcionar un fichero *log* con la traza de ejecución de la muestra una vez finalizado.

Al trabajar con muestras de malware, este análisis ha de realizarse en un entorno controlado y aislado, para evitar así que el código malicioso afecte a la máquina en la que corre el sistema de análisis [23]. Por esto, para este componente del *workflow* se busca utilizar una herramienta libre de análisis tipo sandbox [14]. Esta herramienta debe de permitir crear instancias virtuales de máquinas con una versión moderna de Windows (Windows 7 o 10 preferiblemente) en las que ejecutar la muestra a analizar. Otra característica clave es el *Tipo de API hook* que implementa, es decir, al método que emplea la herramienta para registrar las llamadas a la API realizadas por el software analizado. Dependiendo de si el método permite recopilar información a nivel de usuario, de núcleo, o de ambos se obtendrán resultados más o menos precisos [20]. Por otro lado también se ha de tener en cuenta los requisitos hardware y dependencias software que presenta la solución, así como los resultados que ofrece a cambio.

La **Tabla 3.2** lista las diferentes herramientas de generación de trazas de ejecución que se han considerado como candidatas para utilizar en el sistema de análisis, junto a una breve descripción, sus cualidades más remarcables y las características relevantes antes mencionadas.

Herramienta	**Última Versión Disponible**	Descripción	Cualidades Remarcables	**S.Os Invitados Soportados**	**Tipo de API hook**	**Requisitos**	**Precio**
Cuckoo Sandbox [12]	2.0.7 (2019)	Herramienta líder de análisis de malware dinámico automatizado. Desarrollada en Python. Permite realizar trazado de llamadas realizadas por procesos así como volcados de memoria y tráfico de red.	Adaptado para soportar la mayoría de soluciones de virtualización. Soporte para análisis distribuido. Extensa documentación de instalación y casos de uso. Permite utilizar máquinas físicas como "guests" para el análisis en vez de máquinas virtuales.	Windows 7 x64, Ubuntu 18.04 x32/x64	Usuario	Python 2.7 y librerías asociadas. Volatility para volcados de memoria. Postgresql u otro SGBD.	Gratuito
Cuckoo Sandbox 3 [13]	n/d	Rediseño de cuckoo sandbox basado en Python 3.	Mejora de rendimiento y capacidad así como nuevas funcionalidades respecto a la versión en Python 2.7	n/d	Usuario	n/d	n/d
Drakvuf [26]	0.8 (2022)	Herramienta de caja negra para el análisis de binarios a través de virtualización. Permite realizar trazado de llamadas realizadas por procesos así como volcados de memoria y tráfico de red.	Permite realizar el análisis del malware sin la necesidad de instalar software adicional en las máquinas virtuales, reduciendo así la posibilidad de detección.	Windows 7-8 x32/x64, Windows 10 x64, Linux 2.6.x-5.x x32/x64	Usuario y Kernel	Intel CPU con soporte para virtualización (VT-x) y Extended Page Tables (EPT). Xen 4.16. Python 3.	Gratuito
Drakvuf Sandbox [24]	0.18.1 (2021)	Herramienta de caja negra para el análisis de malware automatizado sin agente con el motor de Drakvuf por debajo. Permite realizar trazado de llamadas realizadas por procesos así como volcados de memoria y tráfico de red.	Permite realizar el análisis del malware sin la necesidad de instalar software adicional en las máquinas virtuales, reduciendo así la posibilidad de detección. Ofrece una interfaz web amigable así como una interfaz de línea de comandos que permite automatizar el proceso de análisis de malware. Ofrece un instalador para guiar el proceso de instalación.	Windows 7 x64, Windows 10 x64	Usuario y Kernel	Intel CPU con soporte para virtualización (VT-x) y Extended Page Tables (EPT). Máquina anfitriona con procesador de al menos 2 núcleos, 5GB RAM y con GRUB como gestor de arranque: Debian 10 Buster / Ubuntu 18.04 Bionic / Ubuntu 20.04 Focal. Virtualización anidada mediante: Xen / VMware Workstation Player / KVM	Gratuito
PyREbox [18]	(2019)	Entorno de pruebas basado en Qemu y Python enfocado a la ingeniería inversa de malware. Permite realizar trazado de llamadas realizadas por procesos así como volcados de memoria.	Permite inspeccionar una máquina virtual qemu en ejecución, su memoria, registros e instrumentalizar su ejecución mediante scripts. Cuenta con un Shell con comandos propios. Todavía se encuentra en desarrollo por lo que aspectos como la eficiencia tienen margen de mejora.	Cualquier Windows x32/x64 soportado por volatility	n/d	Arquitectura de las máquinas virtuales: x86 / x86_64. Sistema operativo de la máquina host: Fedora/CentOS/Debian	Gratuito
Cape [11]	2 (2022)	Entorno de pruebas derivado de cuckoo orientado al desempaquetado y extracción de la configuración del malware a analizar en entornos Windows aislados. Permite realizar trazado de llamadas realizadas por procesos así como volcados de memoria y tráfico de red.	Permite realizar clasificaciones de malware por familias a partir de reglas Yara de detección de firmas de comportamiento. Tiene la capacidad de evitar técnicas de sorteado que utiliza el malware para detectar la ejecución en un entorno virtualizado y no desplegar su auténtico potencial, ocultando así sus capacidades maliciosas al análisis. Permite utilizar máquinas físicas como "guests" para el análisis en vez de máquinas virtuales.	Windows 7 x64, Windows 10 x64	Usuario	Python 3. KVM como hipervisor. Sistema operativo de la máquina host: sistema nativo GNU/Linux (preferible Ubuntu 20.04 LTS)	Gratuito
Ether [2]	0.1 (2009)	Framework de análisis de software malicioso que hace uso de las extensiones de virtualización de Intel para permanecer transparente a este. Permite realizar trazado de llamadas realizadas por procesos	Realiza la traza de llamadas a la API de Windows x32 desplegándose al mismo nivel que el hipervisor, lo que permite monitorizar la máquina virtual sin contar con una presencia considerable en la misma.	Windows XP Service Pack 2	Usuario	Xen como hipervisor. Ejecución baremetal sobre un procesador Intel x64 con la extensión VT activada.	Gratuito
CaptureBAT [25]	Muerto	Herramienta de análisis de comportamiento de aplicaciones para la familia de Sistemas operativos WIn32. Permite realizar trazado de llamadas realizadas por procesos así como volcados de tráfico de red.	Monitoriza cambios de estado a nivel de kernel. Proporciona un mecanismo para excluir ruido que se da en un sistema en espera. Herramienta que se ejecuta en el entorno virtualizado.	Windows 2000, Windows XP, Windows Vista	Kernel	Parche de servicio en el sistema operativo en el que se ejecuta.	Gratuito
Malpimp [21]	2.0 (2013)	Herramienta de generado de trazas de llamadas a API avanzada diseñada para automatizar el proceso de ingeniería inversa de malware.	Permite una configuración avanzada mediante la inclusión o exclusión de DLL y API a monitorizar. Al ser una herramienta de línea de comandos ejecutada en el entorno virtualizado permite la automatización mediante scripts.	Windows 2003, Windows XP	Usuario	No tiene	Gratuito
Buster Sandbox [6]	1.92	Herramienta diseñada para analizar el comportamiento de los procesos, los cambios que estos realizan en el sistema y determinar si se trata de malware. Permite realizar trazado de llamadas realizadas por procesos así como volcados de tráfico de red.	Capaz de analizar cualquier tipo de fichero ejecutable, no sólo binarios. Puede ser ejecutado de forma automática desde línea de comandos. Herramienta que se ejecuta en el entorno virtualizado.	Windows 2000, Windows XP, Windows Vista, Windows 7, Windows 8	Usuario	Instalación previa de Sandboxie	Gratuito

Tabla 3.2: Herramientas de generación de trazas de ejecución de APIs estudiadas

Tras valorar todas las opciones se ha optado por **Drakvuf Sandbox** [24], un sistema de análisis de malware de caja negra [3] que utiliza el motor de **Drakvuf** [26] internamente. Las principales ventajas que ofrece esta solución son las siguientes:

- Baja probabilidad de que el malware detecte que está siendo analizado, puesto que Drakvuf no instala software adicional en las instancias virtuales.
- Interfaz disponible que permite automatizar el proceso de análisis a través de línea de comandos o mediante la creación de tareas de análisis con Python.
- Generación de un *log* que contiene las llamadas realizadas a la API de Windows por todos los procesos del sistema durante el análisis con los argumentos de estas llamadas y si las llamadas han tenido éxito.

El Analyzer ha sido programado para que sea capaz de interactuar con una instancia de Drakvuf Sandbox instalada, enviando muestras a analizar y obteniendo los *logs* generados como resultado. Los *logs* obtenidos se encolan en una cola FIFO para ser posteriormente empleados en la categorización de la muestra.

3.1.3. Categorización de la Muestra - Labeler

Tras haber obtenido las trazas de ejecución de una muestra, hay que analizarlas en busca de unos patrones de comportamiento determinados con los que categorizar la muestra. Esta es la función del **Labeler**. Este componente se encarga de, dada una traza de ejecución, realizar un análisis de la misma a través de expresiones regulares y patrones predefinidos para encontrar evidencias de un comportamiento que permita clasificar la muestra.

Se ha escogido implementar la funcionalidad de este componente a través de expresiones regulares por simplicidad y velocidad a la hora de analizar las trazas. Otras soluciones para este componente se mencionan en el **Capítulo 5**, así como posibilidades de desarrollo a futuro en el **Capítulo 6**.

3.2. Sistema de análisis completo

Para el diseño del sistema de análisis completo se han tenido en cuenta diferentes aspectos que no se abordaron durante el diseño del workflow de análisis:

- *Distribución de Carga de Trabajo*: Para acelerar el proceso de análisis es conveniente disponer de múltiples máquinas en las que se haya instalado la solución de análisis o clasificación escogida.

- *Almacenamiento de resultados*: Una vez se han obtenido los resultados de la clasificación, es de interés almacenar de forma comprimida tanto la muestra analizada como los *logs* generados y los resultados de la clasificación.

- *Coordinación de las componentes*: Para facilitar la interacción entre las diferentes componentes del sistema se ha creado una última componente que actúa como pieza central del mismo.

3.2.1. Componentes adicionales

A partir de las consideraciones anteriores se han desarrollado las siguientes componentes para el sistema completo:

- **Master**: Es la componente encargada de permitir a máquinas workers registrarse durante la ejecución del sistema para servir peticiones de análisis o clasificación. Es capaz de monitorizar su estado a través de latidos y reaccionar ante una caída de un worker.

- **AnalysisWorker**: Esta componente ofrece la funcionalidad del Analyzer en remoto a través de RPC. Se ejecuta en una máquina a parte con una instancia de la solución de análisis escogida instalada.

- **LabelingWorker**: Esta componente ofrece la funcionalidad del Labeler en remoto a través de RPC. Se ejecuta en una máquina a parte con una instancia de la solución de clasificación escogida instalada.

- **Storer**: Esta componente se encarga de almacenar las muestras analizadas junto con los *logs* generados y los resultados de la clasificación en un archivo comprimido.

- **Orchestrator**: Se trata de la pieza central del sistema, su función es la de gestionar y coordinar el proceso de análisis haciendo uso de las diferentes componentes. Puede configurarse para realizar el análisis y/o la clasificación de muestras de en local o remoto. Cuenta con un pool de procesos de análisis y clasificación que, de estar configurado como local realizan las tareas de análisis y clasificación, y en caso de estar configurados en modo remoto delegan estas tareas en los respectivos workers a los que están conectados.

3.2.2. Estructura final

El código del sistema completo, desarrollado en su totalidad en Python, se puede encontrar públicamente accesible y mediante licencia **GNU GPLv3**, en GitHub [4]. La Figura 3.2 resume la estructura final del sistema de análisis desarrollado para la experimentación.

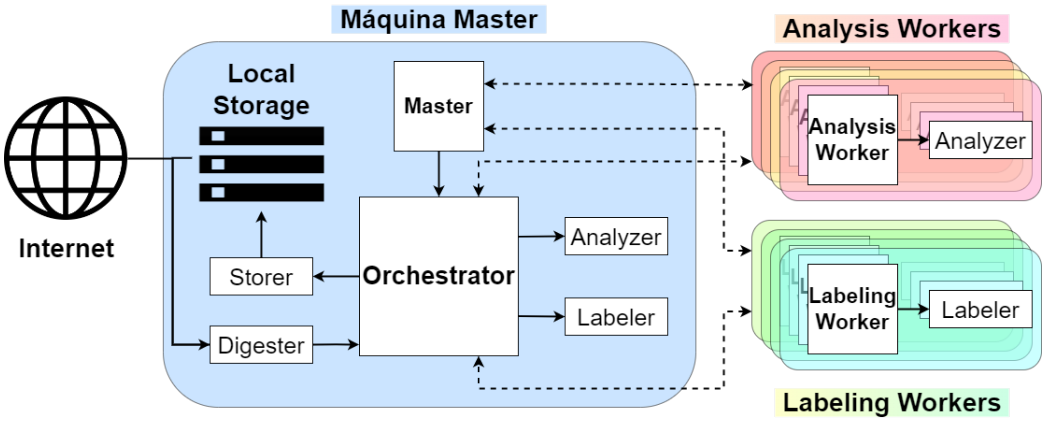


Figura 3.2: Estructura del sistema de análisis

Capítulo 4

Experimentación y resultados

En este capítulo se trata la fase de experimentación del proyecto. En concreto se ha empleado el sistema de análisis de malware desarrollado para realizar un estudio sobre la prevalencia de ASEPs en malware de Windows obtenido de un repositorio online de malware. En primer lugar se describen las pruebas de concepto implementadas para conocer los patrones de comportamiento a buscar. Después, se comenta la arquitectura del despliegue utilizada. Por último, se discuten los resultados obtenidos y las limitaciones encontradas.

4.1. Pruebas de concepto (POCs)

Antes de comenzar con la experimentación sobre malware real, ha sido necesario ajustar el Labeler para asegurarse que detecta los patrones de comportamiento deseados en las trazas que se generan en el Analyzer. Para realizar este ajuste, se han desarrollado una serie de aplicaciones POCs para cada tipo de ASEP que se busca detectar, analizándolas previamente para obtener sus respectivas trazas de ejecución de las que extraer los patrones de comportamiento a detectar para cada tipo de ASEP.

El desarrollo de estas POCs ha requerido una fase de estudio de la API de Windows, así como de la estructura del Registro de Windows y de las rutas del sistema de ficheros empleadas por dicho sistema operativo. El código fuente de las POCs en C++, así como los ejecutables compilados y las trazas de ejecución generadas, se pueden encontrar también en GitHub [5], públicamente accesibles y mediante la licencia **GNU GPLv3**.

Cabe destacar que la mayoría de los ejecutables generados requieren de privilegios de administrador para ejecutarse de forma correcta debido a los mecanismos empleados. Sin embargo, al intentar analizar estos ejecutables se ha podido constatar que la

solución escogida para el proceso de análisis dinámico (la herramienta Drakvuf, véase la **Subsección 3.1.2**) no es capaz de analizar muestras que requieren de elevación de privilegios para ejecutarse. Este problema va a limitar inevitablemente los resultados del análisis.

4.2. Arquitectura del despliegue

Para el despliegue del sistema de análisis se ha diseñado una arquitectura que aprovecha su capacidad para ejecutar las tareas de análisis y clasificación de manera distribuida. Esta arquitectura puede apreciarse en la **Figura 4.1**. A la hora de realizar la experimentación, sin embargo, sólo se ha dispuesto de una máquina para desplegar el sistema de análisis debido a la falta de recursos con los que trabajar. No obstante, puntualmente se ha empleado una segunda máquina para comprobar la capacidad de funcionamiento distribuido del sistema.

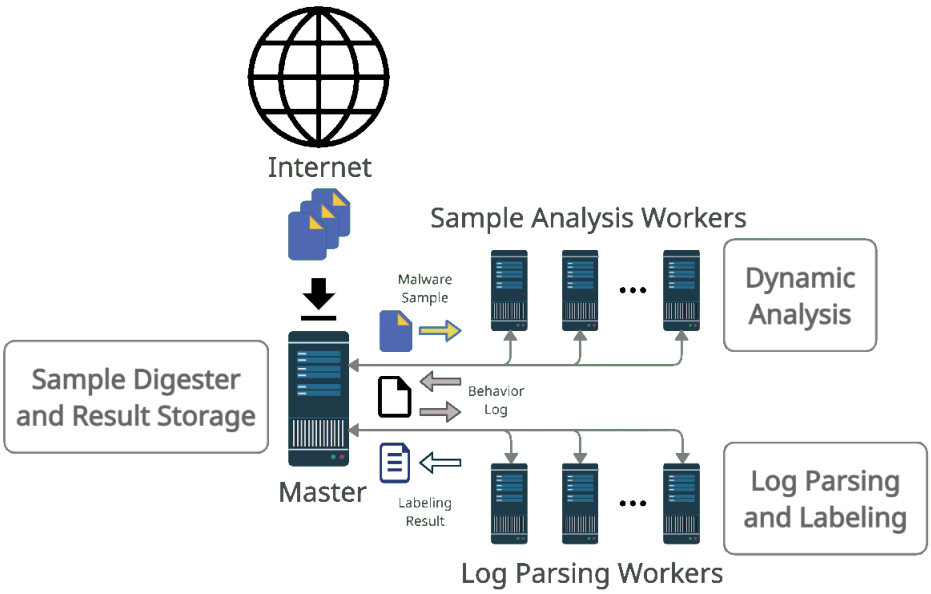


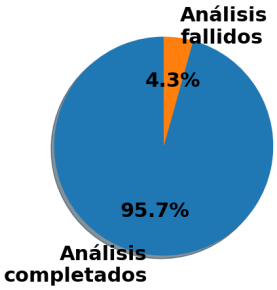
Figura 4.1: Arquitectura ideada para la experimentación

Cabe destacar que con las soluciones de análisis y clasificación escogidas no habría sido necesario disponer de "*Labeling Workers*", ya que el procesado de las trazas de ejecución mediante expresiones regulares es suficientemente rápido para no suponer una merma considerable del rendimiento de realizarse de forma local al Master. Por otro lado, sí hubiera sido interesante disponer de un número suficiente de "*Analysis Workers*", ya que por cada muestra se emplean unos 6 minutos aproximadamente en su análisis y generación de trazas de ejecución.

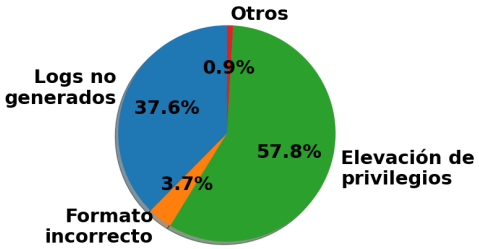
4.3. Resultados obtenidos y limitaciones

Una vez se ha ajustado el Labeler a partir de las pruebas de concepto y se ha desplegado el sistema de análisis con la arquitectura permitida, se ha procedido a descargar periódicamente muestras de malware con las que alimentar la cadena de análisis del repositorio MalwareBazaar (véase la sección **Sección 3.1.1**). Se ha comprobado que el sistema de análisis diseñado es capaz de permanecer durante días continuados analizando muestras sin generar ningún error ni colgarse. Al superar las 2500 muestras analizadas se ha detenido el sistema para realizar una valoración del rendimiento y calidad de los análisis y clasificaciones realizados, detallados en la **Figura 4.2**. En concreto:

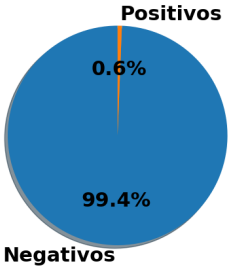
- Se han analizado un total de 2514 muestras de malware, a un ritmo de entre 5 y 6 muestras por hora. El 95,67 % (2405) de estas muestras se ha podido analizar correctamente.
- Del total de muestras analizadas, el 1,63 % (41) no han producido alguno de los *logs* necesarios para llevar a cabo la clasificación. Estos casos se consideran como fallos del sistema de análisis.
- Del total de muestras analizadas, el 0,16 % (4) no han podido ser analizadas por presentar un formato incorrecto de ejecutable. Estos casos no se consideran como fallos del sistema.
- El 2,5 % (63) de las muestras generadas requerían de privilegios de administración para poder ejecutarse. Como se ha expuesto anteriormente, Drakvuf Sandbox por defecto es incapaz de analizar este tipo de muestras. Estos casos se consideran una limitación del sistema de análisis.
- Del conjunto de muestras analizadas correctamente, el 0,62 % (15) de estas han sido clasificadas como uso positivo en ASEPs. Se ha podido confirmar que 7 de estas hacen uso de las "*Run Keys*", un tipo de ASEP bajo la categoría de "*System Persistence Mechanisms*"que crea una clave en el Registro de Windows. Otras 5 emplean el método de "*Startup Folder*", un tipo de ASEP bajo la misma categoría que consiste en añadir un ejecutable, o enlace al mismo en la ruta de la carpeta de Inicio de un usuario. Las 3 restantes utilizan el método "*COM Hijacking*", englobado en la categoría "*Program Loader Abuse*", que consiste en acceder a una clave del registro existente y modificar alguno de sus valores. Con esto se puede concluir que el sistema es capaz de detectar tanto ASEPs que dependen del Registro de Windows como de rutas específicas del sistema de ficheros.



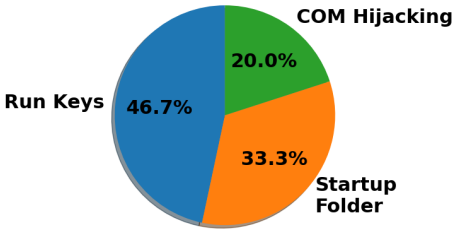
(a) Análisis completados/fallidos



(b) Motivos de fallo



(c) Uso de ASEPs positivo/negativo



(d) Distribución de ASEPs en los positivos

Figura 4.2: Gráficas de distribución de los resultados de la experimentación

4.3.1. Análisis de los resultados

Si bien se ha podido constatar que el sistema es capaz de detectar la utilización de ASEPs en muestras de malware, el bajo número de positivos es un posible indicativo de una carencia en el proceso de análisis y generación de trazas de ejecución, aunque también puede deberse a la presencia de malware evasivo [7].

Con esta hipótesis se ha procedido a examinar varias de las trazas generadas y catalogadas como negativas en presencia de uso de ASEPs y se han realizado los siguientes hallazgos:

- **Los argumentos de las funciones no siempre se pasan por valor**. Es decir, no siempre aparece el valor del argumento con el que se invoca a una función en la traza de ejecución, sino que a veces aparece la dirección de memoria en la que se almacena este valor (o sea, se pasa un puntero a una variable).

- **Las funciones que trabajan con el Registro de Windows no siempre emplean rutas completas o absolutas**. Se ha descubierto que a veces estas funciones parten de una clave ya abierta desde la que acceden a otra más baja en esa jerarquía.

Estos dos factores no se manifestaron en las trazas de las pruebas de concepto, con lo que no se tuvieron en cuenta a la hora de ajustar el Labeler. Este hecho combinado a la limitación de la herramienta para analizar muestras que requieren elevación de privilegios para ejecutarse podría explicar la baja tasa de positivos obtenidos. Por otro lado, se ha de tener en cuenta también que en ningún momento se ha contado con muestras etiquetadas previo análisis del sistema (es decir, no puede saberse si las muestras que se han clasificado como negativos emplean o no algún tipo de ASEP).

Capítulo 5

Trabajo relacionado

En este capítulo se mencionan distintos trabajos y estudios relacionados con el tema principal del proyecto y/o con las herramientas y soluciones empleadas en el sistema de análisis desarrollado. También se contextualizan las contribuciones aportadas con este proyecto al ámbito de investigación relacionado con el tema desarrollado.

El sistema de análisis desarrollado realiza el análisis de las trazas de comportamiento de las muestras, pudiendo correlar eventos dentro de un mismo *log* a través de expresiones regulares simples definidas por el usuario. En [29] se propone una solución de análisis de logs que emplea aprendizaje automático para realizar la correlación de eventos en el fichero *log* analizado. Esta aproximación es de gran interés si se tiene en cuenta la dificultad que puede llegar a suponer para una máquina comprender y aprender de una gramática compleja y cambiante, como puede ser la que presentan la mayoría de ficheros de log.

En [10] se ofrece una propuesta diferente, cuya aproximación es más útil cuando se han de correlar eventos descritos en diferentes *logs*. Así pues, esta solución se centra en la recolección, análisis y correlación entre logs permitiendo al usuario definir los patrones de comportamiento a buscar y las acciones a seguir en caso de detectarse dichos patrones.

Drakvuf Sandbox es la pieza central del sistema de análisis desarrollado, permitiendo realizar análisis dinámicos de muestras de malware y generar las trazas de ejecución correspondientes. En [19] se estudia una de las características más relevantes de Drakvuf: su capacidad para reducir su presencia en la máquina virtual en la que se ejecutan las muestras a analizar. También se ofrece un estudio de la capacidad de Drakvuf para aumentar los recursos destinados al proceso de análisis.

Relacionado con una de las características objeto del estudio anterior, en [9] se emplea Drakvuf para detectar *malware* evasivo, es decir, muestras de código malicioso que emplean diferentes métodos con el fin de no ser detectados por herramientas de análisis y/o protección en el sistema en el que se ejecutan. Este estudio posee una gran importancia respecto a la experimentación realizada en este proyecto, dado que el *malware* evasivo puede afectar significativamente a los resultados del estudio realizado.

En [15] se realiza una comparativa de distintas herramientas de análisis para el diseño de un sistema de análisis de malware empleando técnicas de introspección en máquinas virtuales, entre las que se encuentra Drakfuf. De forma similar, en [17] se realiza otra comparativa entre Drakvuf Sandbox y Cuckoo Sandbox, centrándose en las características y prestaciones de estas dos herramientas.

Pocos estudios se han realizado acerca de los puntos de autoinicio de ejecución desde que se definieran en 2004. En [34] se definen por primera vez estos mecanismos y se estudia el ciclo de vida del *spyware* (tipo de *malware* que recopila y transmite información de una máquina sin el conocimiento del usuario) en una máquina comprometida, monitorizando el uso de ASEPs para persistir en el sistema y continuar espiando a sus usuarios. Por otro lado, en [30] se propone por primera vez una taxonomía de los ASEPs en Windows. Es en esta taxonomía en la que se ha basado el desarrollo del componente Labeler del sistema de análisis diseñado.

Hasta ahora no se había realizado un estudio de la prevalencia de los ASEPs en malware de Windows, por lo que el proyecto desarrollado es un primer paso para conocer la verdadera extensión de la utilización de estos mecanismos en el malware moderno de sistemas operativos Windows.

Capítulo 6

Conclusiones y trabajo a futuro

En este capítulo se exponen las conclusiones extraídas del desarrollo del proyecto y la experimentación, así como posibles líneas de trabajo e investigación a futuro que permitan mejorar y emplear el sistema de análisis desarrollado.

6.1. Conclusiones principales

En este trabajo se ha desarrollado un sistema de análisis de muestras de software modular, con capacidad para distribución de tareas entre múltiples máquinas y monitorización de las mismas. La modularidad de este sistema permite cambiar fácilmente la implementación de cualquiera de las componentes gestionadas directamente por el Orchestrator (Digester, Analyzer, Labeler, Storer) sin necesidad de modificar ninguna otra componente del sistema. A su vez, la estructura del sistema permite utilizarlo para multitud de análisis dinámicos diferentes, y clasificar muestras según distintos criterios, simplemente ajustando los componentes correspondientes (Analyzer y Labeler, respectivamente). Se puede cambiar de igual manera la fuente de la que se obtienen las muestras (ajustando el Digester), o el método que se utiliza para almacenar los resultados (cambiando el Storer). Esta versatilidad permite que el sistema diseñado se pueda utilizar en cualquier estudio de análisis y clasificación de muestras de software en función del comportamiento de las mismas.

En lo referente al estudio acerca de la prevalencia de los ASEPs en malware de Windows, las limitaciones y problemas encontrados durante la fase de experimentación han limitado el alcance del mismo, no pudiendo llevarse a cabo de manera exhaustiva como se había planeado en un principio. Sin embargo, se ha constatado la detectabilidad de estos mecanismos a través de las trazas de ejecución generadas por una herramienta de análisis dinámico (concretamente, mediante Drakvuf Sandbox).

6.2. Trabajo futuro

A continuación se introducen una serie de ideas sobre las que se podría trabajar para expandir las capacidades del sistema desarrollado y mejorarlo.

Resiliencia

Si bien es cierto que el sistema de análisis desarrollado es capaz de registrar workers en tiempo de ejecución, monitorizar su estado y responder ante la pérdida de conexión con una de estas máquinas, no se ha llegado a implementar un mecanismo para soportar caídas de la máquina Master. Para ello, debe de programarse una funcionalidad en el máster que permita el registro de una máquina (o conjunto de máquinas) como réplica del máster, manteniendo en todo momento la coherencia entre estas y, en caso de caer el máster, una de las réplicas ocupe el lugar de este (mediante el algoritmo de elección de líder que se considere oportuno), tomando el control de los workers activos.

Traducción de punteros

Como se ha expuesto anteriormente en la **Subsección 4.3.1**, una de las limitaciones encontradas ha sido la presencia de punteros en las llamadas a la API en lugar del valor del argumento. Para solventar este problema es necesario contar con un volcado de la memoria virtual del proceso que ha realizado la llamada, para así traducir la dirección a la que apunta el puntero a un valor con el que se pueda trabajar en el proceso de clasificación. A pesar de que Drakvuf Sandbox proporciona un mecanismo de volcado de memoria, éste presenta una serie de errores que a día de hoy hacen poco fiable su utilización para este fin, e incluso directamente imposible.

Motor de análisis para la clasificación

Para la tarea de clasificación se han empleado expresiones regulares sobre las trazas de ejecución generadas por el proceso de análisis. Este método, pese a ser rápido y relativamente fácil de programar, puede no ser lo suficientemente preciso para llevar a cabo una clasificación de calidad. Por esto, una de las posibles mejoras a llevar a cabo consiste en implementar un motor de análisis con el que analizar las trazas de ejecución y buscar correlaciones entre llamadas, teniendo en cuenta los valores devueltos por las llamadas al sistema y los diferentes procesos que participan de los comportamientos analizados. Con esta aproximación se puede superar la segunda limitación mencionada en la **Subsección 4.3.1**.

Estudio exhaustivo de prevalencia de ASEPs

Una vez superadas las limitaciones encontradas, es de interés volver a realizar este estudio de prevalencia de ASEPs con un despliegue más ambicioso: utilizando múltiples máquinas para los procesos de análisis y clasificación, se puede aumentar la tasa de muestras procesadas y realizar un estudio con miles de muestras para estudiar la verdadera distribución de los puntos de autoinicio de ejecución en el malware actual de Windows. Este estudio además puede realizarse de manera longitudinal para observar la evolución de las técnicas ASEPs usadas a lo largo del tiempo.

Bibliografía

[1] A. Allievi y col. *Windows Internals, Part 2*. Developer Reference. Pearson Education, 2021. ISBN: 9780135462409.

[2] Monirul Sharif & Wenke Lee Artem Dinaburg Paul Royal. *Ether - Malware Analysis via Hardware Virtualization Extensions*. https://ether.gtisc.gatech.edu/index.html. Accedido en 17-06-2022.

[3] Boris Beizer. *Black-Box Testing: Techniques for Functional Testing of Software and Systems*. USA: John Wiley amp; Sons, Inc., 1995. ISBN: 0471120944. DOI: 10.5555/202699.

[4] Carlos Borau González. *Malware Analysis Workflow*. https://github.com/778280/Malware_Analysis_Workflow. Accedido en 17-06-2022.

[5] Carlos Borau González. *Windows ASEPs POCs C++*. https://github.com/778280/Win-ASEPs_Cpp/. Accedido en 17-06-2022.

[6] Buster. *Buster Sandbox - A designed to analyze the behaviour of processes and the changes made to system*. http://bsa.isoftware.nl/. Accedido en 17-06-2022.

[7] D'Elia, Daniele Cono & Coppa, Emilio & Palmaro, Federico & Cavallaro, Lorenzo. "On the Dissection of Evasive Malware". En: *IEEE Transactions on Information Forensics and Security* 15 (2020), págs. 2750-2765. DOI: 10.1109/TIFS.2020.2976559.

[8] Kenning Arlitsch & Adam Edelman. "Staying Safe: Cyber Security for People and Organizations". En: *Journal of Library Administration* 54.1 (2014), págs. 46-56. DOI: 10.1080/01930826.2014.893116.

[9] David Ekenstein Gustaf & Norrestam. "Classifying evasive malware". Tesis doct. Master's thesis, Lund University, 2017.

[10] Manage Engine. *Best log management for enhanced visibility into your network*. https://www.manageengine.com/products/eventlog/log-management-solution.html. Accedido en 20-06-2022.

[11] Cuckoo Foundation. *CapeSandbox - An Open Source software for automating analysis of suspicious files*. https://capev2.readthedocs.io/en/latest/index.html. Accedido en 17-06-2022.

[12] Cuckoo Foundation. *Cuckoo Sandbox - An open source software for automating analysis of suspicious files*. https://cuckoo.sh/docs/. Accedido en 17-06-2022.

[13] Cuckoo Foundation. *Cuckoo Sandbox 3 - A full rewrite of Cuckoo in Python 3*. https://hatching.io/cuckoo/. Accedido en 17-06-2022.

[14] Anup Greamo Chris & Ghosh. "Sandboxing and Virtualization: Modern Tools for Combating Malware". En: *IEEE Security & Privacy* 9.2 (2011), págs. 79-82. DOI: 10.1109/MSP.2011.36.

[15] Anssi Matti Helin. "Virtual machine introspection in malware analysis". English. Master's thesis. Aalto University. School of Science, 2016, págs. 58 + 6.

[16] *Hybrid-Analysis - Online Malware Analysis Service and Repository*. https://www.hybrid-analysis.com/. Accedido en 17-06-2022.

[17] Ilic, Slavisa & Gnjatović, Milan & Popovic, Brankica & Maček, Nemanja. "A pilot comparative analysis of the Cuckoo and Drakvuf sandboxes: An end-user perspective". En: *Vojnotehnicki glasnik* 70 (abr. de 2022), págs. 372-392. DOI: 10.5937/vojtehg70-36196.

[18] Cisco Talos Security Intelligence y Research Group. *PyREbox - A Python scriptable Reverse Engineering sandbox*. https://pyrebox.readthedocs.io/en/latest/. Accedido en 17-06-2022.

[19] Lengyel, Tamas K. & Maresca, Steve & Payne, Bryan D. & Webster, George D. & Vogl, Sebastian & Kiayias, Aggelos. "Scalability, Fidelity and Stealth in the DRAKVUF Dynamic Malware Analysis System". En: *Proceedings of the 30th Annual Computer Security Applications Conference*. ACSAC '14. New Orleans, Louisiana, USA: Association for Computing Machinery, 2014, págs. 386-395. ISBN: 9781450330053. DOI: 10.1145/2664243.2664252.

[20] Matthew Nunes & Pete Burnap & Omer Rana & Philipp Reinecke & Kaelon Lloyd. "Getting to the root of the problem: A detailed comparison of kernel and user level data for dynamic malware analysis". En: *Journal of Information Security and Applications* 48 (2019), pág. 102365. ISSN: 2214-2126. DOI: 10.1016/j.jisa.2019.102365.

[21] *Malpimp - An advanced API tracing tool and designed to automate the reverse engineering process*. https://securityxploded.com/malpimp.php. Accedido en 17-06-2022.

[22] *Malware Bazaar - Online Malware Repository*. https://bazaar.abuse.ch/. Accedido en 17-06-2022.

[23] Anoop Mohanta Abhijit & Saldanha. *Malware Analysis and Detection Engineering: A Comprehensive Approach to Detect and Analyze Modern Malware*. 2020. ISBN: 978-1-4842-6192-7. DOI: 10.1007/978-1-4842-6193-4.

[24] CERT Polska. *Drakvuf - An automated black-box malware analysis system with DRAKVUF engine under the hood*. https://drakvuf-sandbox.readthedocs.io/en/latest/. Accedido en 17-06-2022.

[25] The Honeynet Project. *CaptureBAT - A behavioral analysis tool of applications for the Win32 operating system family*. https://www.honeynet.org/projects/old/capture-bat/. Accedido en 17-06-2022.

[26] The Honeynet Project. *Drakvuf - A virtualization based agentless black-box binary analysis system*. https://drakvuf.com/. Accedido en 17-06-2022.

[27] Mark E Russinovich y Aaron Margosis. *Troubleshooting with the Windows Sysinternals Tools*. Microsoft Press, 2016.

[28] Abhijit Mohanta & Anoop Saldanha. *Malware Analysis & Detection Engineering*. Apress, 2020. DOI: 10.1007/978-1-4842-6193-4.

[29] Florian Skopik, Max Landauer y Markus Wurzenberger. "Online Log Data Analysis With Efficient Machine Learning: A Review". En: *IEEE Security & Privacy* 01 (2021), págs. 2-12.

[30] Daniel Uroz y Ricardo J. Rodríguez. "Characteristics and detectability of Windows auto-start extensibility points in memory forensics". En: *Digital Investigation* 28 (2019), S95-S104. ISSN: 1742-2876. DOI: 10.1016/j.diin.2019.01.026.

[31] *VirusShare - Online Malware Repository*. https://virusshare.com/. Accedido en 17-06-2022.

[32] *VirusSign - Online Malware Repository*. https://virussign.com/. Accedido en 17-06-2022.

[33] *VirusTotal - Online Malware, Url, IP and Domain Analysis Service and Repository*. https://www.virustotal.com/gui/home/upload. Accedido en 17-06-2022.

[34] Wang, Yi-Min and Roussev, Roussi and Verbowski, Chad and Johnson, Aaron and Wu, Ming-Wei and Huang, Yennun and Kuo, Sy-Yen. "Gatekeeper: Monitoring Auto-Start Extensibility Points (ASEPs) for Spyware Management". En: *Proceedings of the 18th USENIX Conference on System Administration*. LISA '04. Atlanta, GA: USENIX Association, 2004, págs. 33-46.

[35] Yong Wong, Miuyin & Landen, Matthew & Antonakakis, Manos & Blough, Douglas M. & Redmiles, Elissa M. & Ahamad, Mustaque. "An Inside Look into the Practice of Malware Analysis". En: *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. CCS '21. Virtual Event, Republic of Korea: Association for Computing Machinery, 2021, págs. 3053-3069. ISBN: 9781450384544. DOI: 10.1145/3460120.3484759.

Apéndice A

Dedicación

En este apéndice se realiza un desglose en fases de las horas totales invertidas en el desarrollo del proyecto a lo largo de los últimos meses, que aparece reflejado en la **Tabla A.1**. También se detalla la distribución de cada una de estas fases entre febrero y julio en la **Figura A.1**. Cabe destacar que en múltiples ocasiones se han solapado varias fases o estas se han llevado a cabo de forma interrumpida.

- *Investigación*: En esta fase se incluye el estudio de la literatura citada a lo largo del documento, así como de las herramientas consideradas como candidatas para la implementación de las diferentes componentes del sistema de análisis. Se han dedicado un total de 28 horas para esta fase.

- *Instalación de Herramientas*: Esta fase engloba las horas dedicadas a la instalación de la solución de análisis dinámico (Drakvuf Sandbox), corrección de problemas en la instalación e intercambio de comunicaciones con los desarrolladores de la herramienta (llegando a abrir varios *issues* en GitHub). Se han dedicado un total de 42 horas para esta fase.

- *Diseño de Pruebas de Concepto*: Esta fase se corresponde con la familiarización con la API de Windows y el diseño y testeo de las pruebas de concepto utilizadas para obtener las trazas de ejecución sintéticas con las que comparar las de la fase de experimentación. Se han dedicado un total de 55 horas para esta fase.

- *Diseño y Depuración del Sistema de Análisis*: Esta fase comprende el diseño y desarrollo de las diferentes componentes del sistema de análisis desarrollado para realizar el estudio. También el estudio de las APIs de las herramientas empleadas para integrarlas con las correspondientes componentes del sistema. Se han dedicado un total de 100 horas para esta fase.

- *Experimentación*: Durante esta fase se han realizado las pruebas de análisis y clasificación de muestras de malware de repositorios web con el sistema de análisis

desarrollado. También se ha llevado a cabo un análisis de los resultados obtenidos, así como de las limitaciones encontradas durante las pruebas. Se han dedicado un total de 90 horas para esta fase.

- *Redacción*: En esta fase se cuentan las horas empleadas en la redacción de este documento. Se han dedicado un total de 61 horas para esta fase.

- *Reuniones*: Aquí se recogen las horas empleadas en las reuniones realizadas a lo largo de los últimos 5 meses. Han sido un total de 18 reuniones, la mayoría de 20 minutos de duración. El total de horas empleadas en reuniones asciende a 7 horas.

Tabla de Horas Dedicadas al Proyecto	
Fase	**Horas Dedicadas**
Investigación	28 horas
Instalación de Herramientas	42 horas
Diseño de Pruebas de Concepto	55 horas
Diseño y Depuración del Sistema de Análisis	100 horas
Experimentación	90 horas
Redacción	61 horas
Reuniones	7 horas
Total	383 horas

Tabla A.1: Tabla de Horas Dedicadas al Proyecto

Actividad	Fecha de inicio	Fecha de fin	Febrero	Marzo	Abril	Mayo	Junio
Investigación	15-02-2022	12-03-2022					
Instalación de herramientas	15-03-2022	20-04-2022					
Diseño de pruebas de concepto	30-03-2022	01-06-2022					
Diseño y depuración del sistema de análisis	25-04-2022	10-06-2022					
Experimentación	20-05-2022	20-06-2022					
Redacción	06-06-2022	22-06-2022					
Reuniones	15-02-2022	22-06-2022					

Figura A.1: Diagrama de Gannt