Universidad Zaragoza

Trabajo Fin de Grado

Detección de puntos de extensión de autoinicio en sistemas Linux

Auto-Start Extensibility Point Detection on Linux Systems

Autor

Carlos Navarro Gascón

Director

Ricardo Julio Rodríguez Fernández

Grado en Ingeniería Informática
ESCUELA DE INGENIERÍA Y ARQUITECTURA
2022/2023

AGRADECIMIENTOS

A Ricardo por ayudarme siempre que lo necesitaba y por confiar en mi hasta el final.

A mi familia por apoyarme en todo momento.

RESUMEN

Los puntos de extensión de autoinicio (en inglés, *Auto-Start Extensibility Point*, ASEPs), son los mecanismos mediante los cuales los programas pueden iniciarse automáticamente, sin necesidad de interacción explícita de un usuario. Estos componentes son habitualmente utilizados por el software malicioso (*malware*), ya que sirven como método de persistencia en el sistema que pretenden infectar. Este hecho hace que su estudio sea interesante en el campo de ciberseguridad.

Cuando un dispositivo se ha infectado con malware, se realiza un análisis forense con objeto de detectarlo. Dentro del análisis forense, el análisis forense de la memoria consiste en volcar la memoria del sistema afectado en un archivo, para que pueda ser posteriormente analizado con las herramientas apropiadas. En este proyecto se pretende desarrollar una herramienta capaz de clasificar los procesos de un volcado de memoria de un sistema Linux y en función del ASEP que ha provocado su ejecución.

Para lograr este objetivo, en este trabajo primero se han estudiado los ASEPs existentes en sistemas Linux. Después, se ha desarrollado un plugin para la herramienta de análisis forense *Volatility*. Para la evaluación, se han desarrollado pruebas de concepto de infección en los diferentes ASEPs encontrados en Linux y se ha comprobado su correcta detección usando la herramienta desarrollada. Todos los ASEPs han sido detectados correctamente, verificando el funcionamiento de la herramienta. Esta herramienta se ha publicado libremente bajo licencia GNU/GPLv3 para que pueda ser usada y extendida por la comunidad.

ABSTRACT

Auto-start extensebility points (ASEPs), are the mechanisms by which programs can start automatically, without the need for explicit user interaction. These components are commonly used by malicious software, as they serve as a method of persistence on the system they intend to infect. This fact makes its study interesting in the field of cybersecurity. This fact makes its study interesting in the field of cybersecurity.

When a device has been infected with malware, a forensic analysis is performed to detect it. Within forensic analysis, memory forensics consists of dumping the memory of the affected system into a file, so that it can be later analyzed with the appropriate tools. This project aims to develop a tool capable of classifying the processes of a memory dump of a Linux system and depending on the ASEP that has caused its execution.

To achieve this objective, in this work we have first studied the existing ASEPs in Linux systems. Later, a plugin has been developed for the forensic analysis tool *Volatility*. For the evaluation, infection proofs of concept have been developed in the different ASEPs found in Linux and their correct detection has been verified using the developed tool. All ASEPs have been correctly detected, verifying the operation of the tool. This tool has been released freely under the GNU/GPLv3 license so that it can be used and extended by the community.

Índice

Lista de Figuras

Capítulo 1

Introducción y objetivos

1.1. Motivación

Una de las estrategias más comunes utilizadas por los ciberataques es el uso de *malware* para infectar sistemas informáticos, lo que supone una amenaza para ellos. Según las estadísticas de *AV-Test* [1], la cantidad total de malware crece con el paso de los años, por lo que el estudio y desarrollo de herramientas que contribuyan a contrarrestarlo resulta de gran interés.

Basándose en el análisis de diferentes muestras de malware se han definido una serie de pasos, denominados tácticas, que definen los objetivos que los atacantes pretenden cubrir durante el ciclo de vida del programa. Estas tácticas son las siguientes [2]:

1. *Acceso inicial*: Establecer un primer contacto con el sistema infectado.
2. *Ejecución*: Determinar cómo se ejecuta el código malicioso en el sistema.
3. *Persistencia*: Decretar métodos para que el programa malicioso perdure en el sistema afectado.
4. *Escalada de privilegios*: Aumentar los privilegios del programa para acceder a nuevos recursos o sistemas.
5. *Evasión de la defensa*: Técnicas para evitar ser detectados por los mecanismos de defensa del sistema infectado.
6. *Acceso a credenciales*: Obtener las credenciales del sistema para expandir el control de recursos.

7. *Descubrimiento*: Obtener información sobre el contexto del sistema y la red del sistema afectado.
8. *Movimiento lateral*: Técnicas para pivotar entre otros sistemas de la red.
9. *Recolección*: Determinar cómo se almacena la información obtenida.
10. *Exfiltración*: Definir cómo se envía la información obtenida al atacante.
11. *Comando y control*: Técnicas usadas por el atacante para ejercer control sobre el sistema infectado.

De todas estas tácticas, este trabajo se va a centrar en la fase de *persistencia*. Para que el malware permanezca en el sistema afectado, se necesitan utilizar mecanismos que garanticen que su ejecución perdure a lo largo del tiempo. Los puntos de extensión de autoinicio (*Auto-Start Extensibility Point*, ASEPs) [3, 4] son una parte clave en este aspecto, pues son características propias del sistema operativo cuya función consiste en ejecutar programas automáticamente al iniciar el sistema sin necesidad de interacción del usuario.

Según el Instituto Nacional de Estándares y Tecnología de Estados Unidos, existen cuatro fases cuando se va a resolver un incidente informático [5]. La primera, es la *fase de preparación*, en la que además de prevenir incidentes garantizando que las redes, sistemas y aplicaciones utilizados son suficientemente seguros, también se debe predisponer de herramientas que permitan gestionar el incidente en caso de que ya se haya producido. La segunda, es la *fase de detección y análisis*, en la que se trata de descubrir si se ha producido un incidente, y una vez detectado se realiza un análisis del mismo determinando su origen, alcance y modo de actuación. En tercer lugar, se encuentra la *fase de contención, erradicación y recuperación*, donde se frena la expansión del incidente y se recolecta información sobre este, tras lo que se procede a eliminar la amenaza y se garantiza que todo funciona correctamente. Por último, se encuentra la *fase de actividad post-incidente*, en la que se estudia la información obtenida sobre el incidente y se trata de corregir los fallos encontrados de cara a futuro.

En particular, este trabajo se enfoca en la fase de análisis forense, ya que el objetivo es facilitar la identificación de las causas del incidente. Existen varios tipos de análisis forense [6], entre los que destacan:

- *Análisis forense de ordenadores*: Abarca técnicas variadas para analizar

ordenadores como puede ser el sistema ficheros, la procedencia de archivos o la búsqueda de cadenas de caracteres.

- *Análisis forense de software*: Esta categoría incluye análisis forense digital, de sistemas operativos y de aplicaciones. Su cometido involucra examinar el contenido del entorno que se desea analizar, incluyendo logs, monitorización de eventos de seguridad y ficheros de configuración. En este grupo también se puede incluir el análisis forense de bases de datos.

- *Análisis forense de multimedia*: Su objetivo consiste en recuperar y analizar vídeos e imágenes.

- *Análisis forense de dispositivos/IoT*: Este grupo se centra en recolectar evidencia de dispositivos de diferentes escalas, puesto que los procedimientos utilizados en ordenadores no son adecuados para ellos. Tales dispositivos pueden ir desde móviles hasta drones y consolas de videojuegos.

- *Análisis forense de redes*: En esta categoría se examina el tráfico de la red en sus diferentes niveles de protocolos en busca de evidencia de actividades maliciosas.

- *Análisis forense de malware*: En este grupo se engloban las técnicas utilizadas para identificar, examinar e investigar el código malicioso de una máquina para determinar características como su método de entrada, propagación e impacto en el sistema.

- *Análisis forense de la memoria*: Consiste en extraer la memoria RAM de un sistema en su estado vigente para posteriormente realizar un análisis de ella utilizando herramientas específicas.

Este trabajo se centra en el análisis forense de memoria, que es de gran utilidad debido a que los datos almacenados en el disco duro pueden haber sido dañados o eliminados, algunos datos sólo residen en memoria, y además el análisis inicial puede tener un dominio más acotado (la capacidad de almacenamiento en memoria es más limitada que comparada con el almacenamiento en disco).

1.2. Objetivos, metodología y alcance

El objetivo de este proyecto es *desarrollar un plugin para Volatility capaz de clasificar los procesos ejecutados por los diferentes ASEPs en sistemas Linux* para

facilitar la identificación de posibles procesos maliciosos en estos sistemas operativos. Volatility [7] es un framework de referencia en el análisis forense de memoria. Esta idea está inspirada en el plugin *winesap* de Volatility, que realiza la detección de ASEPs para sistemas Windows [4].

En primer lugar, se han estudiado los diferentes ASEPs disponibles en sistemas Linux. Para desarrollar el plugin correctamente se ha llevado a cabo una metodología ágil (SCRUM), que ha ayudado a obtener un prototipo rápido de la herramienta. También se ha utilizado una metodología experimental, creando sistemas virtualizados con VirtualBox "infectados" con pruebas de concepto. Estas pruebas de concepto están compuestas por programas desarrollados con el propósito de instalarse en cada uno de los ASEPs identificados en los sistemas estudiados. Posteriormente, se ha obtenido un volcado de memoria que consecuentemente será analizado utilizando el plugin desarrollado para Volatility dentro del marco de este TFG. El código desarrollado se ha publicado en `Github` con licencia libre [8].

1.3. Estructura del documento

Este documento se encuentra dividido en 6 capítulos y dos apéndices. En el Capítulo 2 se definen algunos conceptos necesarios para garantizar la comprensión correcta del resto del documento. El Capítulo 3 describe un análisis de los diferentes puntos de extensión que presentan los sistemas Linux y las técnicas empleadas para su detección. El Capítulo 4 explica los detalles de implementación para desarrollar la herramienta propuesta. El Capítulo 5 detalla la fase de experimentación llevada a cabo para comprobar que el plugin funciona correctamente. Por último, el Capítulo 6 expone las conclusiones obtenidas tras la realización del proyecto. Finalmente, en el Apéndice A se presenta un desglose de las horas dedicadas a las distintas fases del proyecto y en el Apéndice B se muestra un ejemplo de salida generada al ejecutar el plugin desarrollado.

Capítulo 2

Conceptos previos

En este capítulo se introducen algunos conceptos necesarios para la correcta comprensión del documento. En primer lugar, se explica en qué consiste un *análisis forense*. A continuación, se explica en qué consiste la herramienta `Volatility`, sobre la que se va a desarrollar el plugin propuesto. Después, se define en qué consisten los puntos de extensión de autoinicio y cuáles son sus características principales. Por último, se detallan algunos datos relevantes sobre la gestión de los procesos del sistema operativo Linux.

2.1. Análisis forense

Se conoce como análisis forense a la práctica de recolección, análisis y reporte de datos digitales. Estas prácticas sirven para detectar y prevenir ciberataques siempre y cuando dejen trazas de evidencia que se almacenen en la máquina afectada. Durante el proceso se utilizan técnicas de recuperación y análisis de datos, por lo que debe ser realizado por personal especializado en el tema [9].

El análisis forense puede clasificarse en diferentes tipos dependiendo de como se realice [6]. Este trabajo se enfoca en el análisis forense de memoria, en el que se examina la memoria RAM de la máquina afectada en el momento del incidente, ya que no está garantizado que los datos almacenados en disco permanezcan íntegros. Además, hay ciertos datos que solo son posibles de encontrar en la memoria y su dominio está más acotado, ya que posee menos datos para analizar que los disponibles en disco. Para ello, se realiza lo que se conoce como *volcado de memoria*, que consiste en extraer la memoria del sistema en un archivo tras haberse producido el incidente informático, de

forma que se pueda analizar su estado actual.

Debido a la complejidad para manejar este tipo de datos conviene utilizar algún software específico. Concretamente, para la realización de este proyecto se ha utilizado Volatility [7], que es un framework de código abierto, escrito en Python y especializado para análisis forense.

2.2. El framework Volatility

`Volatility` [7] es una herramienta de análisis forense que permite trabajar con volcados de memoria en sistemas Windows, Linux y macOS. `Volatility` permite analizar volcados de memoria de forma fácilmente interpretable por el usuario y dispone de una comunidad muy activa, con una extensa colección de plugins que permiten analizar y obtener información forense de diferente índole. Para trabajar con los volcados en crudo, `Volatility` utiliza una *tabla de símbolos* que se debe extraer previamente del sistema y añadir al programa para que el análisis forense pueda ser realizado correctamente. Esta tabla almacena ciertos datos de interés, como son las posiciones de memoria de estructuras relevantes del sistema, de forma que `Volatility` pueda interpretar los datos binarios del volcado y traducirlos en sus estructuras correspondientes.

`Volatility` posee varios *plugins* por defecto para cada sistema con diversas funcionalidades. Todos ellos pueden ser examinados ya que se trata de un framework de software libre, lo que puede ser de ayuda para familiarizarse con el programa. Además, la estructura interna de clases que lo forman se puede aprovechar para facilitar la realización de muchas tareas.

Actualmente `Volatility` cuenta con dos versiones: la versión 2, que funciona utilizando `Python 2`, y la versión 3, cuyo código está implementado en `Python 3`. Concretamente, para desarrollar el plugin propuesto en este trabajo se ha utilizado la versión 3.

2.3. Clasificación de ASEPs

Los puntos de extensión de autoinicio (ASEPs) son las técnicas mediante las que un sistema operativo permite que un programa sea iniciado automáticamente sin

interacción explícita del usuario [3]. Por ejemplo, algunos programas como pueden ser servicios del sistema o tareas programadas son ejecutadas durante el inicio del sistema, cuando el usuario inicia sesión o tras la ocurrencia de ciertos eventos. Desde el punto de vista de la ciberseguridad el estudio de los ASEPs es especialmente interesante, ya que es el método utilizado en los ciberataques para garantizar la persistencia de ejecución de código malicioso en el sistema infectado.

Estas técnicas se pueden clasificar en varias categorías, dependiendo de la característica del sistema de la que abusan los programas maliciosos para persistir en el sistema [4]. En primer lugar, se encuentran los *mecanismos de persistencia del sistema*, que son los propios mecanismos que el sistema utiliza para inicializar tareas y servicios para su propio funcionamiento. En segundo lugar, se encuentra el *abuso del cargador de programas*, que consiste en lanzar programas no deseados cuando el usuario ejecuta otro manualmente, sin que este se percate de ello. En tercer lugar, se encuentra el *abuso de aplicaciones*, donde se aprovecha la capacidad de ciertos programas legítimos para expandir su funcionalidad mediante plugins con la finalidad de ejecutar persistentemente el código malicioso. Finalmente, se encuentra el *abuso del comportamiento del sistema*, donde se abusa de determinadas características del sistema para ejecutar programas.

Algunos puntos de extensión solo pueden ser escritos por programas con suficientes privilegios. Además, algunos permiten que los programas que ejecutan tengan permisos elevados mientras que otros están limitados a los permisos del propio usuario que haya iniciado sesión.

Los puntos de extensión pueden requerir que se refresque el sistema de alguna manera para ejecutar un programa tras ser añadido, mientras que otros no lo necesitan. El ámbito de ejecución de un ASEP es de sistema si cualquier programa puede interactuar con aquel que haya sido ejecutado por dicho ASEP, de lo contrario su ámbito es de aplicación. Finalmente, el ámbito de configuración de un ASEP puede ser de nivel de sistema o de usuario, dependiendo de si afectan a todo el sistema globalmente o únicamente a la sesión del usuario registrado.

2.4. Procesos en Linux

Los sistemas Linux funcionan internamente gracias a la existencia de estructuras de memoria que almacenan información relevante para gestionar su ejecución, las cuales

se pueden encontrar tras realizar un volcado de memoria. Particularmente, para este trabajo las estructuras relativas a los procesos son de especial interés, ya que son el objetivo a analizar. En Linux, cada proceso que esté siendo ejecutado en el sistema tiene asociada una estructura `task_struct` [10] en memoria que almacena información variada del proceso como su estado de ejecución, identificador de proceso o el puntero a su proceso padre.

Entre todos estos campos que se pueden encontrar en el `task_struct` destaca uno, denominado `cgroup` (*control groups*). Los `cgroups` son una característica de los sistemas Linux que proporciona al sistema un mecanismo para dividir los procesos en grupos jerárquicos de tareas con un propósito específico. Esto permite la creación de grupos de procesos con propiedades similares y definir los recursos que cada uno de ellos utiliza, como la cantidad de memoria o de CPU que consumen. La importancia de este campo se encuentra en que permite conocer la procedencia de la invocación de algunos procesos, lo que es útil para encontrar qué método han utilizado para iniciarse, como se explica posteriormente en el Capítulo 5.

Capítulo 3

Tipos de ASEPs en Linux

En este capítulo se realiza un análisis de los tipos de ASEPs posibles en Linux. En primer lugar, se describen y categorizan los posibles ASEPs en sistemas Linux y se hace un breve análisis de su funcionamiento. A continuación, se explica la estrategia que se ha utilizado para detectar cada uno de ellos.

3.1. Análisis de ASEPs en entornos Linux

Antes de comenzar con el propio desarrollo, se ha realizado un análisis de los puntos de ejecución automáticos disponibles en Linux. El criterio utilizado para la elección de los ASEPs ha sido utilizar únicamente aquellos que son comunes a todas las distribuciones de Linux (o al menos a las más populares), ya que se pretende desarrollar una herramienta genérica para todos los sistemas Linux. Se va a realizar un estudio de cada ASEP individualmente, mostrando cuál es su funcionalidad y cómo la desempeñan, viendo el alcance de cada uno de ellos y explicando cómo un atacante puede utilizarlos para ejecutar programas de forma persistente. En la Tabla 3.1 se muestra una taxonomía de los ASEPs considerados para este trabajo, siguiendo la propuesta establecida en [4].

En la columna "Permisos de escritura" se detalla si el ASEP en concreto puede ser modificado manualmente, o si por el contrario requiere de algún programa con privilegios elevados para hacerlo. La columna "Privilegios de ejecución" especifica si el ASEP ejecuta procesos con *privilegios de ejecución elevados* o si por el contrario está limitado a los *privilegios del usuario* que haya accedido al sistema. En la columna "Rastreado en análisis forense de memoria" se indica si el ASEP puede ser detectado

o no en un análisis forense de memoria.

Como se ha descrito anteriormente en la Sección 2.3, algunos ASEPs requieren que se refresque el sistema para lanzar un proceso tras ser instalado en él, lo que puede incluir acciones como reiniciar el sistema por completo. En la columna "Refresco del sistema" se especifica este requerimiento mínimo de refrescar el sistema para que se ejecute el nuevo proceso instalado, que puede ser a distintos niveles (*de servicio*, *de terminal*, o *de sesión de usuario*). El ámbito de un ASEP es de *sistema* si cualquier programa del sistema puede interactuar con el programa definido en el ASEP; de lo contrario, su ámbito es de *aplicación*. Esta información se especifica en la columna "Ámbito de ejecución". Finalmente, algunos ASEPs tienen diferente "Ámbito de configuración", ya que pueden ser configurados a *nivel de usuario* o a *nivel de sistema*, lo que afecta al sistema por completo o únicamente a la sesión del usuario respectivamente.

En concreto, se han analizado 7 ASEPs cubiertos en 3 categorías diferentes: *mecanismos de persistencia del sistema*, *abuso de aplicaciones* y *abuso del comportamiento del sistema*. A continuación se describen cada uno de los ASEPs identificados en más detalle.

Tabla 3.1: Taxonomía de ASEPs de Linux y resumen de sus características

	Características					
Puntos de extensión de autoinicio de Linux	Permisos de escritura	Privilegios de ejecución	Rastreado en análisis forense de memoria	Refresco del sistema	Ámbito de ejecución	Ámbito de configuración
Mecanismos de persistencia del sistema						
Servicios de systemd	no	sistema	sí	reiniciar servicio	sistema	sistema
Cron (usuario)	sí	usuario	sí	no necesario	aplicación	aplicación
Cron (administrador)	no	sistema	sí	no necesario	sistema	sistema
Fichero rc.local	no	sistema	sí	reiniciar servicio	sistema	sistema
Abuso de aplicaciones						
Fichero .bashrc	sí	usuario	sí	iniciar nueva shell	aplicación	aplicación
Fichero .profile	sí	usuario	sí	reiniciar sesión de usuario	aplicación	aplicación
Abuso de comportamiento del sistema						
Módulos del núcleo	no	sistema	sí	no necesario	sistema	sistema

3.1.1. Mecanismos de persistencia del sistema

Servicios de `systemd`

Los servicios de `systemd` son programas que se ejecutan en segundo plano cuando se inicia el sistema. A pesar de que su cometido consiste en poner en marcha determinadas funcionalidades del sistema, Linux ofrece la posibilidad de crear nuevos servicios, lo cual puede ser aprovechado por un atacante para ejecutar malware en segundo plano.

La creación de nuevos servicios de `systemd` requiere permisos de administrador y sus privilegios de ejecución son a nivel de sistema. Para efectuar su ejecución basta

con iniciar el servicio creado por medio de un comando. Su ámbito de ejecución y de configuración son a nivel de sistema.

Tareas periódicas de `cron`

`Cron` es un programa por defecto de los sistemas basados en UNIX que ofrece al usuario la posibilidad de crear y ejecutar tareas periódicas. Un atacante puede crear un programa malicioso y almacenarlo en la máquina víctima y a continuación, conseguir que se ejecute regularmente en ciertos períodos de tiempo o al iniciar el sistema añadiéndolo a la lista de tareas periódicas de `cron`. Esto se puede conseguir simplemente modificando el fichero de configuración de `cron`. Concretamente, existen dos tipos de ficheros `cron`: el de usuario (normalmente situado en la ruta «`/var/spool/crontabs/nombre-del-usuario`») y el de administrador (normalmente situado en «`/etc/crontab`»).

El `cron` de usuario no requiere privilegios específicos para ser sobreescrito, mientras que el de administrador requiere tener privilegios de administrador para ser modificado. Los privilegios de ejecución de los programas ejecutados por `cron` son equivalentes a los del propio usuario, por lo que aquellos que sean inicializados por el `cron` del administrador tendrán privilegios de nivel de sistema. No es necesario reiniciar el sistema para que el programa añadido se ejecute (a no ser que se haya especificado lo contrario al programar la tarea), ya que `cron` se encuentra constantemente comprobando si hay alguna tarea pendiente. Tanto el ámbito de ejecución de los programas como el de configuración se corresponden con el del usuario que lo alberga.

Fichero `rc.local`

RC Local es un servicio de `systemd` específico que viene preinstalado en los sistemas Linux. Su función consiste procesar el fichero `rc.local` (normalmente situado en «`/etc/rc.local`») y ejecutarlo al iniciar el sistema. Por tanto, modificando este fichero se puede ejecutar cualquier programa deseado durante el arranque.

Este fichero se encuentra protegido, por lo que para modificarlo es necesario tener privilegios de administrador y también se ejecuta con privilegios de administrador. Para ejecutar los programas añadidos, al tratarse de un servicio de `systemd`, basta con reiniciar este servicio para que los cambios realizados se produzcan, por lo que no es necesario reiniciar el sistema por completo. Tanto su ámbito de ejecución como de

configuración son a nivel de sistema.

3.1.2. Abuso de aplicaciones

Ficheros de configuración de terminal

Cada vez que se inicia una sesión de terminal, se inicia un proceso que se encarga de configurarla para que funcione correctamente. En este proceso se realizan varias tareas como pueden ser la inicialización de ciertas variables de sistema y otros parámetros. Dependiendo de su función, los ficheros que se leen al iniciar una terminal se dividen en dos grupos:

- Fichero `.profile`: Este fichero se encuentra en el directorio raíz del usuario. Cada vez que el usuario inicia sesión, el sistema procesa este archivo y ejecuta todas sus instrucciones. Su función consiste en inicializar variables de entorno para preparar la sesión del usuario, pero puede ser modificado para añadir otras instrucciones.
- Fichero `.bashrc`: Comparte características con el fichero `.profile`, pero en este caso el fichero no se procesa cuando el usuario inicia sesión en el sistema, sino cada vez que se inicia una nueva sesión de shell (por ejemplo, al abrir una nueva terminal). Su función consiste en inicializar variables para presentar la terminal al usuario, pero igualmente se puede modificar para ejecutar los procesos que se deseen.

Como se ha comentado, estos ficheros se pueden modificar para añadir las instrucciones deseadas, que se ejecutarán al iniciar una nueva terminal. De esta manera, un atacante puede editar estos archivos para lanzar cualquier proceso. Sus privilegios de ejecución son los mismos que los del usuario que haya iniciado sesión y ambos ficheros pueden ser sobreescritos sin permisos de administrador. En el caso del fichero `.bashrc`, para refrescarlo basta con iniciar una nueva shell; mientras que para el fichero `.profile`, es necesario reiniciar la sesión del usuario. Tanto su ámbito de ejecución como de configuración son a nivel de aplicación.

3.1.3. Abuso de comportamiento del sistema

Módulos del núcleo

Todo sistema UNIX posee un núcleo, que es la base sobre la que se sustenta y se encarga de gestionar los recursos básicos, como el hardware, la memoria y las llamadas de sistema. Linux permite la creación de módulos del núcleo para expandir sus funcionalidades, por lo que un atacante podría crear su propio módulo que se iniciaría en el momento de cargar el núcleo.

Estos módulos se ejecutan con privilegios de todo el sistema y su creación se encuentra limitada a usuarios con permisos de administrador. Para invocar un nuevo módulo añadido no es necesario reiniciar el sistema, ya que se puede realizar mediante la ejecución de un comando (que requiere privilegios de administrador). Como se ejecutan a nivel del núcleo, sus ámbitos de ejecución y configuración abarcan todo el sistema.

3.2. Detección de los ASEPs

3.2.1. Detección de servicios de `systemd`

En la Sección 3.1 se ha comentado que Linux permite crear nuevos servicios que ejecuten cualquier programa que se desee. Cuando se crea un nuevo servicio este puede recibir cualquier nombre, por lo que si un atacante crea un servicio para ejecutar código malicioso puede darle un nombre que podría pasar desapercibido a ojos del usuario, ya que el sistema ejecuta procesos no malignos mediante esta característica. Por este motivo cualquier proceso que haya sido invocado mediante un servicio de `systemd` se encuentra clasificado en este grupo, a excepción de `cron` y `rc.local` (véase la Sección 3.2.2).

Para detectar este ASEP se debe examinar la estructura `task_struct` del proceso, pues los procesos que sean invocados a partir de uno de estos servicios mantienen esta información en ella. Concretamente, esta información se encuentra en el campo `cgroup` de esta estructura, por lo que al examinarla se puede determinar si el proceso ha sido invocado mediante un servicio de `systemd`.

3.2.2. Detección de `cron` y `rc.local`

En la Seccion 3.1, se ha detallado que tanto el administrador de procesos `cron` como el fichero `rc.local` son controlados por servicios de `systemd`, que gestionan su invocación y método de actuación. Por este motivo su método de detección es muy similar, pero en este caso también se debe examinar el nombre del servicio que invocó al proceso. Así pues, para reconocer procesos iniciados por cada servicio se debe examinar la cadena de caracteres que lo define. En el caso de `cron` la cadena a buscar es `cron.service`, mientras que en `rc.local` es `rc-local.service`.

Nótese que estas nomenclaturas son las utilizadas por defecto por estos servicios. En caso de que cualquiera de ellas se encuentre modificada, este método de detección fallaría, por lo que no es un método de detección completamente certero.

3.2.3. Detección de ficheros de configuración de shell

En este caso, no existe un artefacto específico que indique si un proceso ha sido invocado por uno de estos ficheros. No obstante, con la información que se conoce se puede tratar de identificar de manera exitosa. En el caso del fichero `.profile`, ejecutado durante el proceso de inicio de sesión, se crea un ámbito (`scope`) denominado `session-[id-de-la-session].scope`. Los ámbitos son estructuras de jerarquización de los `cgroups`, por lo que si un proceso se encuentra bajo este ámbito se sabe que ha sido invocado durante el inicio de sesión y por tanto es susceptible de haber sido invocado por el fichero `.profile`.

Por otro lado, los procesos invocados por `.bashrc` suelen estar asociados al servicio del terminal que utilice el sistema. Sin embargo, debido a que cada versión de Linux utiliza un servicio de terminal específico no resulta un medio adecuado para reconocer estos procesos. En cambio, una característica común a todos los sistemas Linux es que estos procesos estén agrupados en el servicio dedicado al usuario, el cual sigue el patrón `user@[UID-del-usuario].service`.

3.2.4. Detección de módulos del núcleo

Cuando se inicia un sistema Linux, en primera instancia se crea un proceso llamado *idle*, cuyo identificador de proceso (`PID`) es 0 y es la base para cargar todos recursos necesarios durante el arranque. En primer lugar, este proceso gestiona la carga del

núcleo y al terminar crea dos procesos hijos para completar la inicialización. El primer proceso que crea es el denominado *init*, el cual tiene `PID` con valor 1 y es el padre de todos los demás procesos que se inicien en el entorno del usuario. Por otro lado, se crea el proceso *kthreadd*, con `PID` 2, el cual funciona en el espacio del núcleo y es responsable de planificar la ejecución de los hilos del núcleo.

Por tanto, los módulos del núcleo se inician bajo el proceso *kthreadd*, ya que como su nombre indica su objetivo es operar a nivel de núcleo. Por el contrario, el resto de procesos ejecutados en el nivel de usuario, son hijos del proceso *init*. Conociendo esta información, para agrupar los programas ejecutados por módulos del núcleo simplemente se debe examinar su estructura `task_struct` en busca de su campo *parent*, donde se almacena un puntero al proceso padre. Si al acceder a este puntero se descubre que el proceso padre del proceso tiene `PID` con valor 2 (*kthreadd*), se puede concluir que dicho proceso ha sido ejecutado por un módulo del núcleo.

3.3. Extensión a otros sistemas operativos como macOS

macOS es un sistema operativo cuyo núcleo, de código abierto, es XNU, el cual guarda muchas similitudes con los sistemas Linux, ya que ambos están basados en UNIX.

En los sistemas macOS existen ciertos mecanismos de persistencia que funcionan de forma similar a los de Linux. Por ejemplo, macOS dispone del planificador de tareas `cron` con un funcionamiento similar al de Linux que permite la generación de tareas periódicas. También se encuentran ficheros de configuración de terminal, que pueden ser igualmente abusados para persistencia de malware. A diferencia de Linux, macOS no dispone de servicios de `systemd` para iniciar procesos en segundo plano, sino que utiliza el servicio `launchd` para realizar una función semejante con algunas diferencias. A pesar de no compartir núcleo, macOS también dispone de extensiones para gestionar su funcionamiento e igualmente pueden ser abusadas como mecanismo de persistencia.

En la propuesta inicial se pretendía profundizar más en este sistema operativo. Sin embargo, no ha sido posible por falta de tiempo y se plantea como trabajo futuro.

Capítulo 4

Diseño e implementación del plugin `linusap`

En este capítulo se especifica la implementación llevada a cabo para desarrollar el plugin `linusap`. En primer lugar, se detalla el diseño empleado para desarrollar el plugin. En segundo lugar, se explica cómo lleva a cabo su funcionamiento. Por último, se muestra el repositorio donde se encuentra almacenado el código del plugin.

4.1. Diseño

La Figura 4.1 muestra un diagrama en formato UML donde se puede observar la estructura formada por las diferentes clases que conforman el código de `linusap`. Se ha utilizado un patrón de diseño *Strategy* [11], ya que se ha considerado que se adapta adecuadamente a las necesidades del plugin y de esta forma el código queda más organizado y escalable en caso de que se quieran añadir nuevas funcionalidades. A continuación se da una breve explicación de la funcionalidad de cada una de estas clases:

4.1.1. Clase PluginInterface

`Volatility` ofrece la interfaz `PluginInterface` para permitir que los desarrolladores creen nuevos plugins que sean compatibles con el framework. Particularmente se han implementado dos métodos de esta interfaz:

- Método `get_requirements`: Se utiliza para comprobar que el plugin puede ser

ejecutado correctamente. Para ello, se verifica que el volcado de memoria que se está analizando cumple los requisitos del plugin. En el caso de `linusap`, se comprueba que el sistema es el esperado (Linux) y que la arquitectura del sistema es compatible (x86-64). Esta restricción de arquitectura podría relajarse, puesto que las estructuras de los procesos en Linux 32 y 64 bits no han variado, en cuanto a lo que se ha usado en este proyecto para la detección. Como trabajo futuro, se plantea la evaluación del plugin en arquitecturas de 32 bits para verificar su correcto funcionamiento.

- Método `run`: En este método se realiza la propia acción del plugin y se ejecuta tras realizar las comprobaciones de compatibilidad. Principalmente se encarga de definir la estructura de los datos mostrados en la salida, que se organizan en forma de tupla, y el resto del trabajo se deja relegado a las demás clases.

4.1.2. Clase Linusap

Es la clase principal que da nombre al plugin desarrollado y en la que se realiza la implementación de la interfaz `PluginInterface`. Además de los métodos implementados, se añade uno nuevo denominado `generator` en el que se genera la salida según el formato definido en el método `run`.

4.1.3. Clase LinuxASEPs

Representa la interfaz genérica que se utiliza en el patrón *Strategy*. Para cada uno de los puntos de extensión mencionados en la Sección 3.1 se implementa un *ConcreteStrategy* con su lógica correspondiente. Por un lado, se encuentra el método `detect`, que se encarga de aplicar las técnicas de detección de la Sección 3.2 individualmente para cada ASEP. Por otro lado, se encuentra el método `getASEPName` que devuelve el nombre relativo al ASEP detectado, lo cual es necesario para mostrar en la salida.

4.1.4. TaskInfo

Se trata de una estructura que almacena cierta información relevante de los procesos encontrados en el volcado de memoria. Esta información es utilizada por la clase `LinuxASEPs` para realizar la detección de los ASEPs correctamente.

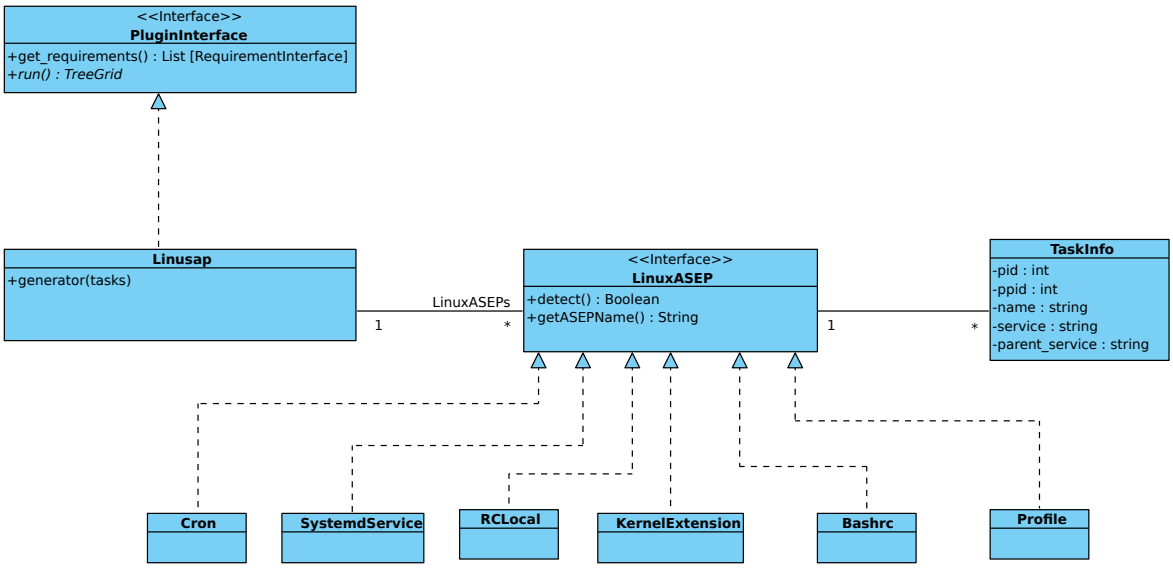

Figura 4.1: Diagrama UML de clases

4.2. Implementación

La Figura 4.2 muestra un diagrama de secuencia en formato UML que describe el caso de uso de la ejecución de `linusap`. En primer lugar, se realiza la llamada al plugin y tras ello se invoca al método `get_requirements` que realiza las comprobaciones de compatibilidad con el volcado de memoria que se pretende analizar. Si el volcado no cumple con los requisitos necesarios, se devuelve un mensaje de error indicando lo ocurrido; de lo contrario, se continúa con la ejecución normal del plugin.

A continuación, se invoca al método run donde se define el formato de salida del plugin y se invoca al método `linux.pslist`. Este método es un plugin que se encuentra por defecto al instalar `Volatility` y cuya función consiste en devolver una lista de los procesos en memoria compuestos por su `task_struct` (véase la Sección 2.4).

Esta lista de procesos es recibida por el método `generator`, que se encarga de iterar sobre cada uno de ellos para detectar su ASEP correspondiente. Cada proceso se almacena en la estructura `TaskInfo`, que adquiere la información relevante para realizar el reconocimiento. Después, se crean los diferentes estrategias concretas relativas al patrón *Strategy* para la detección individual de cada ASEP. Para cada estrategia, se llama al método `detect` correspondiente, que devolverá un booleano indicando si el ASEP concreto es el responsable de la ejecución de la tarea que se está procesando. Si este método devuelve `True`, se obtiene el nombre del ASEP correspondiente y se añade a la salida; si devuelve `False`, se procede a la detección del siguiente ASEP (en la Figura 4.2 se muestra únicamente el procesamiento de tres ASEPs por simplicidad,

pero el proceso es equivalente para el resto). Finalmente, si ninguno de los métodos consigue detectar el ASEP se determina su valor como "indefinido".

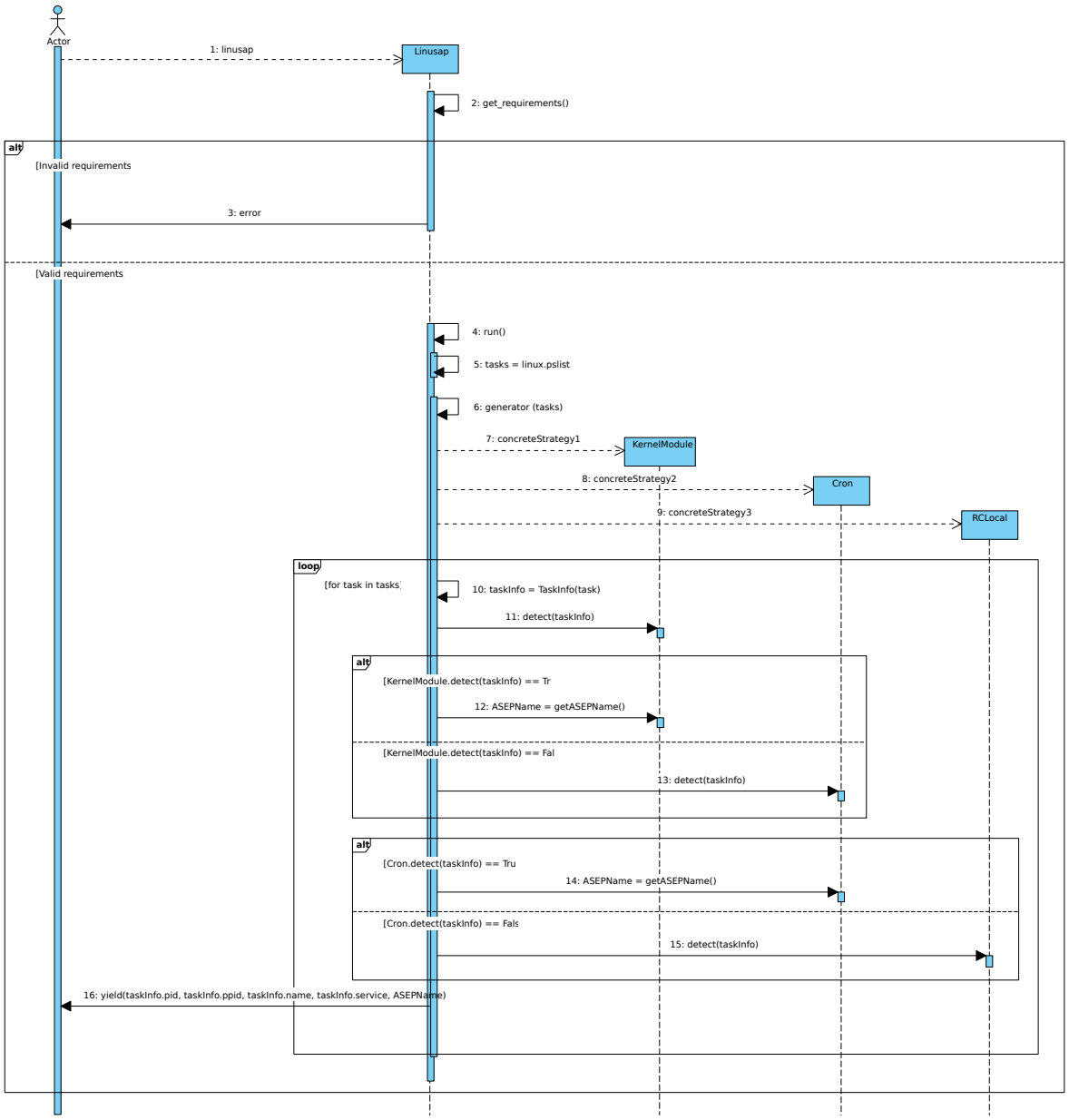

Figura 4.2: Diagrama UML de secuencia

4.3. Repositorio del plugin

Todo el código se ha publicado en `Github` en un repositorio público [8]. La herramienta es de uso libre bajo licencia GNU/GPLv3 de forma que pueda ser usada y extendida por la comunidad.

Capítulo 5

Validación experimental

En este capítulo se detalla la fase de experimentación llevada a cabo para comprobar el funcionamiento del plugin desarrollado. Primero se describe el funcionamiento de las pruebas de concepto utilizadas para la experimentación y posteriormente se describen los experimentos realizados, junto a una discusión de los resultados obtenidos.

5.1. Pruebas de concepto desarrolladas

Una vez realizado un análisis de los ASEPs con los que se va a trabajar, se han realizado pruebas experimentales para comprobar el funcionamiento de `linusap`. Para ello, se han creado una serie de *pruebas de concepto* con las que se van a realizar varios experimentos.

El objetivo de estas pruebas de concepto es instalar una aplicación conocida en los ASEPs identificados. Posteriormente, se realizan varios volcados de memoria y se procesan utilizando `Volatility` y `linusap` con objeto de detectar dicha aplicación a través de los ASEPs considerados en este proyecto.

Teniendo en cuenta los requisitos de las pruebas de concepto, se ha concluido que no es necesario que este malware simulado realice ninguna tarea específica. La única condición imprescindible es que el programa se mantenga ejecutado en segundo plano en la memoria del sistema, pues de esta forma mediante `Volatility` se puede obtener información útil sobre estos procesos. Por tanto, el programa a utilizar como prueba de concepto contiene únicamente un bucle infinito, lo que garantiza que nunca termine y concluyentemente su presencia perdure en memoria.

5.2. Experimentos y discusión

Para evitar problemas que pudieran corromper la máquina de pruebas, se ha utilizado VirtualBox para la experimentación con el objetivo de trabajar en un entorno aislado. Además de trabajar con sistemas virtualizados y así no comprometer la integridad de la máquina original en la que se trabaja, VirtualBox ofrece también otras ventajas como la posibilidad de realizar volcados de memoria de cualquier máquina virtual que se encuentre en estado de ejecución, lo que facilita el proceso para obtener la captura posteriormente analizada con `Volatility`. Concretamente, para llevar a cabo la experimentación se ha instalado una máquina virtual con un sistema operativo Debian 10 sobre una arquitectura de AMD de 64 bits.

ASEP de Linux	**Detección**
Servicios de `systemd`	✓
`Cron` *(usuario)*	✓
`Cron` *(administrador)*	✓
Fichero `rc.local`	✓
Fichero `.bashrc`	✓*
Fichero `.profile`	✓*
Módulos del núcleo	✓

Tabla 5.1: Resultados de detección de ASEPs en los experimentos realizados.

En la Tabla 5.1 se muestra el resultado de la experimentación realizada. Como se muestra, todos los métodos de detección de ASEPs descritos en la Sección 3.1 han funcionado correctamente. Cabe comentar el caso de los *ficheros de configuración de shell* (véase la Sección 3.1.2). A pesar de que las pruebas de concepto para estos ASPEs sí que son detectadas correctamente, otros procesos del sistema no invocados por estos ASEPs se marcan como procesos lanzados por alguno de estos ASEPs (es decir, existen falsos positivos). Se ha comprobado que este hecho se debe a que el método de detección utilizado es demasiado genérico, puesto que engloba un ámbito mayor del esperado. Como trabajo futuro, se pretende estudiar cómo minimizar este número de falsos positivos al mínimo posible (idealmente, ninguno).

Un ejemplo de la salida generada por la ejecución del plugin desarrollado se muestra en el Apéndice B.

Capítulo 6

Conclusiones

En los últimos años las muestras de malware para infectar sistemas informáticos han ido aumentando con el paso del tiempo. Para que estos programas maliciosos consigan subsistir en el sistema infectado necesitan utilizar mecanismos de persistencia que permitan su ejecución prolongada. Los puntos de extensión de autoinicio son los mecanismos utilizados por el sistema para ejecutar procesos automáticamente sin interacción explicita del usuario, por lo que son el principal método de persistencia utilizado por los ciberatacantes. Por este motivo el desarrollo de herramientas que ayuden a la detección de estos puntos de extensión son un bien necesario en el campo de la ciberseguridad.

En este trabajo se ha creado un plugin para la herramienta de análisis forense `Volatility` cuyo objetivo es detectar los puntos de extensión responsables de la ejecución de los procesos en sistemas Linux. El código desarrollado se ha publicado con licencia libre en `Github` para que sea examinado y expandido por la comunidad. Para verificar el funcionamiento de la herramienta, se han llevado a cabo pruebas de concepto compuestas por programas ejecutados por los diferentes puntos de extensión de forma que se ha comprobado si su reconocimiento se efectúa satisfactoriamente.

Como trabajo futuro, se plantean diferentes acciones:

- *Eliminación de falsos positivos*: Las pruebas de concepto creadas para verificar el funcionamiento del plugin han sido superadas con éxito para todos los puntos de extensión planteados. Sin embargo, en el caso de los *ficheros de configuración de terminal* la salida mostraba otros programas incorrectamente clasificados en este grupo, debido a que el método de detección utilizado es demasiado genérico. Como trabajo futuro se pretende reducir este número de falsos positivos al mínimo

posible (idealmente ninguno).

- *Detección de nuevos ASEPs*: La lista de ASEPs planteados en este trabajo pretendía ser común para todas las distribuciones de Linux disponibles (o al menos, las más populares). No obstante, este hecho no quita que estos sistemas dispongan de otros métodos de persistencia que pueden ser abusados por los ciberatacantes. Por esta razón, como trabajo futuro se plantea expandir el funcionamiento del plugin para todos los puntos de extensión disponibles en las diferentes distribuciones de Linux.

- *Expansión a otras arquitecturas*: Las pruebas de validación se han enfocado utilizando un sistema virtualizado cuya arquitectura es x86-64. Por tanto, como trabajo futuro se plantea expandir y comprobar su funcionalidad en otras arquitecturas utilizadas por sistemas Linux.

- *Expansión a sistemas macOS*: Originalmente, en este trabajo se pretendía expandir la funcionalidad de la herramienta para sistemas macOS, pero por falta de tiempo finalmente no ha sido posible. Por tanto, como trabajo futuro se propone desarrollar una herramienta que cubra las necesidades de detección de ASEPs en sistemas macOS.

Capítulo 7

Bibliografía

[1] AV-TEST GmbH. AV-TEST Security Report 2019/20. [Online, `https://www.av-test.org/fileadmin/pdf/security_report/AV-TEST_Security_Report_2019-2020.pdf`.], 2020. Accessed on November 24, 2022.

[2] Kris Oosthoek and Christian Doerr. SoK: ATT&CK Techniques and Trends in Windows Malware. In Songqing Chen, Kim-Kwang Raymond Choo, Xinwen Fu, Wenjing Lou, and Aziz Mohaisen, editors, *Security and Privacy in Communication Networks*, pages 406–425, Cham, 2019. Springer International Publishing.

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

[4] Daniel Uroz and Ricardo J. Rodríguez. Characteristics and Detectability of Windows Auto-Start Extensibility Points in Memory Forensics. *Digital Investigation*, 28:S95–S104, April 2019.

[5] Paul Cichonski, Thomas Millar, Tim Grance, and Karen Scarfone. Computer Security Incident Handling Guide. techreport SP 800-61 Rev. 2, National Institute of Standards and Technology (NIST), September 2012. Special Publication (NIST SP).

[6] Tina Wu, Frank Breitinger, and Stephen O'Shaughnessy. Digital forensic tools: Recent advances and enhancing the status quo. *Forensic Science International: Digital Investigation*, 34:300999, 2020.

[7] Volatility Foundation. Volatility 3. [Online; `https://github.com/volatilityfoundation/volatility3/`], 2022. Accessed on November 24, 2022.

[8] Carlos Navarro Gascón. Linusap. [`https://github.com/Lulay7/linusap.git`], 2022. Accessed on November 24, 2022.

[9] Dr. Ajay Prasad Dr. Jeetendra Pande. *Digital forensics*. Uttrakhand Open University, 2016.

[10] Robert Love. *Linux Kernel Development, Second Edition*. Sams Publishing, 2005.

[11] Johnson Ralph Vlissides John Gamma Erich, Helm Richard. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series, 1995.

Apéndice A

Dedicación

En este apéndice se realiza un desglose de las horas invertidas en el desarrollo del proyecto divididas en diferentes fases, que se puede ver reflejado en la tabla A.1. También se especifica la distribución temporal de cada una de estas fases en la figura A.1. Cabe destacar que algunas de las fases se han solapado y otras se han llevado a cabo de forma ininterrumpida. A continuación, se describe brevemente cada una de las fases:

- *Investigación*: En esta fase se engloban todas las horas dedicadas a la búsqueda de información entre los documentos citados que han ayudado a comprender y desarrollar el trabajo correctamente. Se ha dedicado un total de 90 horas a esta fase.
- *Instalación de herramientas*: En esta fase se recogen las horas empleadas en la instalación de las diferentes herramientas utilizadas a lo largo del trabajo, así como la posterior solución de errores de configuración surgidos. En esta fase se ha empleado un total de 25 horas.
- *Familiarización con el entorno de trabajo*: En esta fase se determinan las horas dedicadas a la adaptación al uso de las herramientas utilizadas para el desarrollo del protecto, principalmente consumidas en la experimentación con la herramienta `Volatility`. Se ha dedicado un total de 35 horas a esta fase.
- *Diseño e implementación del plugin*: En esta fase se incluyen todas las horas empleadas para diseñar la estructura del plugin y a la implementación de su código. Se ha dedicado un total de 50 horas a esta fase.
- *Experimentación y pruebas de concepto*: Esta fase hace mención de las horas

dedicadas a la creación e instalación de las pruebas de concepto desarrolladas para los diferentes ASEPs y al estudio y realización de múltiples experimentos para calibrar el funcionamiento del plugin desarrollado. Se ha dedicado un total de 30 horas a esta fase.

- *Redacción*: En esta fase se recogen las horas dedicadas a la memoria del proyecto. En total se han dedicado 90 horas a esta fase.

- *Reuniones*: En esta fase se determinan las horas dedicadas a las reuniones para consultar dudas con el director del trabajo. El total de tiempo dedicado a esta fase es de 3 horas.

Fase	**Horas dedicadas**
Investigación	90 horas
Instalación de herramientas	25 horas
Familiarización con el entorno de trabajo	25 horas
Diseño e implementación del plugin	50 horas
Experimentación y pruebas de concepto	30 horas
Redacción	90 horas
Reuniones	3 horas
Total	313 horas

Tabla A.1: Tabla de horas dedicadas al proyecto

Figura A.1: Diagrama de Gannt

Apéndice B

Salida del plugin

```
$ python3 vol.py -f /home/lulay/Uni/TFG/Dumps/linux-aseps.elf linux.linusap
Volatility 3 Framework 2.3.0
```

PID	PPID	COMM	SERVICE	ASEP
1	0	systemd	init.scope	SYSTEMD_SERVICE
2	0	kthreadd	NONE	UNDEFINED
3	2	rcu_gp	NONE	KERNEL_MODULE
4	2	rcu_par_gp	NONE	KERNEL_MODULE
5	2	kworker/0:0	NONE	KERNEL_MODULE
6	2	kworker/0:0H	NONE	KERNEL_MODULE
7	2	kworker/u2:0	NONE	KERNEL_MODULE
8	2	mm_percpu_wq	NONE	KERNEL_MODULE
9	2	ksoftirqd/0	NONE	KERNEL_MODULE
10	2	rcu_sched	NONE	KERNEL_MODULE
11	2	rcu_bh	NONE	KERNEL_MODULE
12	2	migration/0	NONE	KERNEL_MODULE
13	2	kworker/0:1	NONE	KERNEL_MODULE
14	2	cpuhp/0	NONE	KERNEL_MODULE
15	2	kdevtmpfs	NONE	KERNEL_MODULE
16	2	netns	NONE	KERNEL_MODULE
17	2	kauditd	NONE	KERNEL_MODULE
18	2	khungtaskd	NONE	KERNEL_MODULE
19	2	oom_reaper	NONE	KERNEL_MODULE
20	2	writeback	NONE	KERNEL_MODULE
21	2	kcompactd0	NONE	KERNEL_MODULE
22	2	ksmd	NONE	KERNEL_MODULE
23	2	khugepaged	NONE	KERNEL_MODULE
24	2	crypto	NONE	KERNEL_MODULE
25	2	kintegrityd	NONE	KERNEL_MODULE
26	2	kblockd	NONE	KERNEL_MODULE
27	2	edac-poller	NONE	KERNEL_MODULE
28	2	devfreq_wq	NONE	KERNEL_MODULE
29	2	watchdogd	NONE	KERNEL_MODULE
30	2	kswapd0	NONE	KERNEL_MODULE
48	2	kthrotld	NONE	KERNEL_MODULE
49	2	ipv6_addrconf	NONE	KERNEL_MODULE
50	2	kworker/u2:1	NONE	KERNEL_MODULE
59	2	kstrp	NONE	KERNEL_MODULE
95	2	kworker/0:2	NONE	KERNEL_MODULE
96	2	ata_sff	NONE	KERNEL_MODULE
97	2	scsi_eh_0	NONE	KERNEL_MODULE
98	2	scsi_eh_1	NONE	KERNEL_MODULE
99	2	scsi_tmf_0	NONE	KERNEL_MODULE
100	2	scsi_eh_2	NONE	KERNEL_MODULE
101	2	scsi_tmf_2	NONE	KERNEL_MODULE
102	2	scsi_tmf_1	NONE	KERNEL_MODULE
103	2	kworker/u2:2	NONE	KERNEL_MODULE
104	2	kworker/u2:3	NONE	KERNEL_MODULE
105	2	kworker/0:3	NONE	KERNEL_MODULE
107	2	kworker/0:1H	NONE	KERNEL_MODULE

114	2	ttm_swap	NONE	KERNEL-MODULE
116	2	irq/18-vmwgfx	NONE	KERNEL-MODULE
192	2	kworker/u3:0	NONE	KERNEL-MODULE
194	2	jbd2/sda1-8	NONE	KERNEL-MODULE
195	2	ext4-rsv-conver	NONE	KERNEL-MODULE
228	1	systemd-journal	systemd-journald.service	SYSTEMD-SERVICE
258	1	systemd-udevd	systemd-udevd.service	SYSTEMD-SERVICE
267	1	systemd-timesyn	systemd-timesyncd.service	SYSTEMD-SERVICE
390	1	cupsd	cups.service	SYSTEMD-SERVICE
391	1	udisksd	udisks2.service	SYSTEMD-SERVICE
392	1	alsactl	alsa-state.service	SYSTEMD-SERVICE
393	1	systemd-logind	systemd-logind.service	SYSTEMD-SERVICE
394	1	accounts-daemon	accounts-daemon.service	SYSTEMD-SERVICE
395	1	rsyslogd	rsyslog.service	SYSTEMD-SERVICE
396	1	ejemplo-systemd	test.service	SYSTEMD-SERVICE
399	1	cron	cron.service	CRON
401	1	dbus-daemon	dbus.service	SYSTEMD-SERVICE
402	1	wpa_supplicant	wpa_supplicant.service	SYSTEMD-SERVICE
403	1	NetworkManager	NetworkManager.service	SYSTEMD-SERVICE
404	1	avahi-daemon	avahi-daemon.service	SYSTEMD-SERVICE
405	1	ModemManager	ModemManager.service	SYSTEMD-SERVICE
407	1	anacron	anacron.service	SYSTEMD-SERVICE
408	399	cron	cron.service	CRON
420	408	sh	cron.service	CRON
425	404	avahi-daemon	avahi-daemon.service	SYSTEMD-SERVICE
426	420	ejemplo-cron.sh	cron.service	CRON
429	1	cups-browsed	cups-browsed.service	SYSTEMD-SERVICE
430	1	polkitd	polkit.service	SYSTEMD-SERVICE
459	390	dbus	cups.service	SYSTEMD-SERVICE
481	1	dhclient	networking.service	SYSTEMD-SERVICE
527	1	dhclient	networking.service	SYSTEMD-SERVICE
556	1	unattended-upgr	unattended-upgrades.service	SYSTEMD-SERVICE
557	1	ejemplo-rclocal	rc-local.service	RC-LOCAL
562	1	gdm3	gdm.service	SYSTEMD-SERVICE
576	1	sshd	ssh.service	SYSTEMD-SERVICE
826	1	exim4	exim4.service	SYSTEMD-SERVICE
871	1	rtkit-daemon	rtkit-daemon.service	SYSTEMD-SERVICE
882	1	upowerd	upower.service	SYSTEMD-SERVICE
893	1	packagekitd	packagekit.service	SYSTEMD-SERVICE
980	1	colord	colord.service	SYSTEMD-SERVICE
1009	562	gdm-session-wor	session-3.scope	PROFILE
1013	1	systemd	init.scope	BASHRC
1014	1013	(sd-pam)	init.scope	BASHRC
1026	1	gnome-keyring-d	session-3.scope	PROFILE
1030	1009	gdm-wayland-ses	session-3.scope	PROFILE
1032	1013	dbus-daemon	dbus.service	BASHRC
1034	1030	gnome-session-b	session-3.scope	PROFILE
1043	1034	ejemplo-profile	session-3.scope	PROFILE
1044	1034	ejemplo-profile	session-3.scope	PROFILE
1070	1034	gnome-shell	session-3.scope	PROFILE
1078	1013	gvfsd	gvfs-daemon.service	BASHRC
1083	1013	gvfsd-fuse	gvfs-daemon.service	BASHRC
1086	1070	Xwayland	session-3.scope	PROFILE
1098	1013	at-spi-bus-laun	at-spi-dbus-bus.service	BASHRC
1103	1098	dbus-daemon	at-spi-dbus-bus.service	BASHRC
1105	1013	at-spi2-registr	at-spi-dbus-bus.service	BASHRC
1109	1013	pulseaudio	pulseaudio.service	BASHRC
1116	1013	gnome-shell-cal	dbus.service	BASHRC
1117	1013	gvfs-udisks2-vo	gvfs-udisks2-volume-monitor.service	BASHRC
1124	1013	gvfs-afc-volume	gvfs-afc-volume-monitor.service	BASHRC
1125	1013	evolution-sourc	evolution-source-registry.service	BASHRC
1130	1013	gvfs-goa-volume	gvfs-goa-volume-monitor.service	BASHRC
1134	1013	goa-daemon	dbus.service	BASHRC
1149	1013	goa-identity-se	dbus.service	BASHRC
1151	1013	gvfs-mtp-volume	gvfs-mtp-volume-monitor.service	BASHRC
1159	1013	gvfs-gphoto2-vo	gvfs-gphoto2-volume-monitor.service	BASHRC
1163	1034	gsd-power	session-3.scope	PROFILE
1164	1034	gsd-print-notif	session-3.scope	PROFILE
1167	1034	gsd-rfkill	session-3.scope	PROFILE
1170	1034	gsd-screensaver	session-3.scope	PROFILE
1172	1034	gsd-sharing	session-3.scope	PROFILE
1175	1034	gsd-sound	session-3.scope	PROFILE
1178	1034	gsd-xsettings	session-3.scope	PROFILE

1184	1034	gsd-wacom	session-3.scope	PROFILE
1186	1034	gsd-smartcard	session-3.scope	PROFILE
1193	1034	gsd-a11y-settin	session-3.scope	PROFILE
1194	1034	gsd-clipboard	session-3.scope	PROFILE
1196	1034	gsd-color	session-3.scope	PROFILE
1197	1034	gsd-datetime	session-3.scope	PROFILE
1199	1034	gsd-housekeepin	session-3.scope	PROFILE
1201	1034	gsd-keyboard	session-3.scope	PROFILE
1205	1034	gsd-media-keys	session-3.scope	PROFILE
1206	1034	gsd-mouse	session-3.scope	PROFILE
1219	1	gsd-printer	session-3.scope	PROFILE
1260	1013	evolution-calen	evolution-calendar-factory.service	BASHRC
1262	1034	tracker-miner-a	session-3.scope	PROFILE
1271	1034	evolution-alarm	session-3.scope	PROFILE
1272	1034	gsd-disk-utilit	session-3.scope	PROFILE
1274	1034	tracker-extract	session-3.scope	PROFILE
1276	1034	gnome-software	session-3.scope	PROFILE
1282	1013	tracker-store	tracker-store.service	BASHRC
1316	1013	dconf-service	dbus.service	BASHRC
1320	1013	evolution-addre	evolution-addressbook-factory.service	BASHRC
1364	1	fwupd	fwupd.service	SYSTEMD_SERVICE
1672	1013	gnome-terminal-	gnome-terminal-server.service	BASHRC
1680	1672	bash	gnome-terminal-server.service	BASHRC
1682	1680	ejemplo-bashrc.	gnome-terminal-server.service	BASHRC
1714	2	ejemplo-kernel.	NONE	KERNEL_MODULE
1715	258	systemd-udevd	systemd-udevd.service	SYSTEMD_SERVICE