Universidad Zaragoza
1542

Trabajo Fin de Grado

MEJORA DE LA DETECCIÓN DE MALWARE MEDIANTE LA MODIFICACIÓN PROFUNDA DE SISTEMAS DE SANDBOXING

Autor

José Carlos Ramírez Vega

Director

Antonio Sanz Alcober

Ponente

José Luis Salazar Riaño

Escuela de Ingeniería y Arquitectura de la Universidad de Zaragoza.
Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación.

Curso 2014-2015

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación).

D./Dª. José Carlos Ramírez Vega,

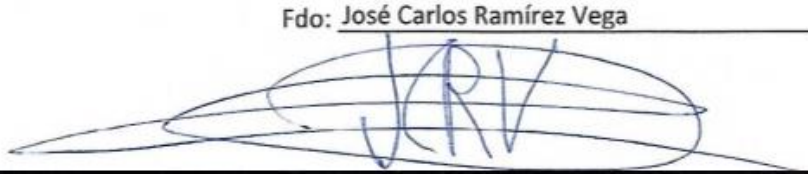
con nº de DNI 73020950 K en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster) Grado, (Título del Trabajo) Mejora de la detección de malware mediante la modificación profunda de sistemas de sandboxing.

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

Zaragoza, 25 de septiembre de 2015

Fdo: José Carlos Ramírez Vega



Agradecimientos

No creo que se pueda recordar a toda la gente que ha aportado su granito de arena durante estos años de Grado. A todos los compañeros de clase que he tenido, y a todo el claustro de profesores.

A Antonio Sanz por su tiempo, dirección y ayuda durante este trabajo, y por la guía en el mundo de la seguridad que desinteresadamente me ha dado.

A José Luis Salazar, por su gran ayuda a la hora de plasmar el trabajo realizado en la memoria de manera correcta.

A Mikael Keri por responder a mis dudas y permitirme aportar comentarios a su trabajo

A la ciudad del viento y a la de los lagos helados.

Al Equipo A, que siempre serán la Universidad. Disfrutad de Europa, no dudo que traeréis grandes historias.

A los Midori Peppers y el resto de la Vieja Guardia, siempre cerca. Ayer, hoy y mañana.

A mis abuelos, tíos y tía. La familia es lo que siempre está, sin dudar.

A Mari-Pi, has sido y serás las cuatro patas de mi mesa. El Apoyo con mayúscula.

A Pepito, que no podrá venir a la presentación pero no dejará de pensar en ella ni un minuto. Por tu preocupación, por ser un modelo inalcanzable, por tu esfuerzo, por tu mano siempre pendiente. Cuídate mucho de los mosquitos.

A Yai. Intentaré seguirte el ritmo, hasta que no haga falta que te lo sigan.

A un Pollito, que me hace volar más alto que las nubes. Espero dar la talla.

Mejora de la Detección de Malware Mediante la Modificación Profunda de Sistemas de Sandboxing

RESUMEN

Este trabajo ha sido realizado por José Carlos Ramírez Vega y dirigido por Antonio Sanz Alcober, siendo presentado como Trabajo de Fin de Grado de la titulación con nombre Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación, mención Telemática, de la Escuela de Ingeniería y Arquitectura de la Universidad de Zaragoza.

Se enmarca en el campo del análisis de código malicioso en los sistemas modernos, conocido como malware. Concretamente en el análisis realizado de manera dinámica, observando el comportamiento real en un sistema haciendo uso de Cuckoo Sandbox y VirtualBox. El objetivo principal es mejorar la detección de malware esquivo, el cual es capaz de comprobar su entorno y en caso de estar siendo analizado no desplegar su comportamiento malicioso.

Primero se definen conceptos claves necesarios para la comprensión del trabajo, como la tecnología a usar y su finalidad en este campo. Continúa con la explicación de algunas de los principales retos que se plantean a la hora de usar este tipo de análisis.

Entrando ya en el propio trabajo se narra un compendio ordenado de las técnicas más usadas por el malware moderno para esquivar los sistemas de análisis dinámico. A continuación se muestran las medidas elegidas a implementar, el razonamiento de la selección de cada una de ellas, posibles fallos y el resultado que se espera que produzca el sistema en un análisis real.

Después se dan detalles conceptuales sobre la implementación realizada en Python principalmente, aunque también se hace un uso muy ligero de shell script, batch script y AutoHotKey script. Se darán unas pequeñas indicaciones sobre el uso del software creado desde el punto de vista del usuario.

Finalmente se propone una metodología para comprobar la efectividad de la implementación propuesta, que consta de: elección de muestras, enfrentamiento con el sistema y definición de los casos de éxito. Los resultados obtenidos se muestran a través de unas gráficas resumen que son presentadas y comentadas.

Se cierra el trabajo con conclusiones y opiniones sobre trabajo futuro en el ámbito del análisis dinámico de malware y la ciberseguridad.

Malware Detection's Improvement through Deep Sandboxing Systems Modification

SUMMARY

This project have been performed by José Carlos Ramírez Vega and directed by Antonio Sanz Alcober. It is proposed as End of Bachelor's Thesis in the Bachelor of Engineering in Telecommunication Technologies and Services, major in Telematics, of the University of Zaragoza.

This Thesis belongs to the malicious code (malware) analysis field. More specifically it is part of the "dynamic analysis" group, which monitors the actual behavior of a sample inside a system using Cuckoo Sandbox and VirtualBox. The main goal of this project is to improve the detection of evasive malware, that is, malware that is aware of its environment and is able to decide whether or not to display its malicious behavior.

First, key concepts like technologies are defined. Then, the main challenges in this kind of analysis are exposed.

The main body of the thesis is organized as follows. An organized set of techniques used by modern malware to avoid dynamic analysis is presented. Then, measures are selected and implemented to cover these techniques. These measures are reasoned, exposing weaknesses and expected behavior during a real analysis.

After that, the software implementation is detailed. This have been done using mainly Python 2, but also shell script, batch script and AutHotKey script. Next, some guide lines about the usage of the software solution from the user's point of view is presented.

Finally a methodology to test the improvement of the solution is proposed. It contains elements like sample selection, testing against the system, and success definition. The results are explained using comments and graphs.

As closing lines. Conclusions and opinions about future work, the dynamic malware analysis and cybersecurity can be found.

Índice general

Contenido

Capítulo 1

Introducción

1.1 Motivación

En la actualidad millones de personas interaccionan a través de Internet cada día, y el número sigue creciendo. Es la llamada "era de la información". Se desea que todo el mundo esté conectado simultáneamente, haciendo las operaciones necesarias de manera absolutamente transparente y con las mayores facilidades para el usuario.

Su funcionamiento está basado en unas arquitecturas y protocolos diseñados cuando no se podía imaginar la expansión que ha sufrido y cuyo principal objetivo es poder dar mayor rendimiento, aprovechando el ancho de banda y otros recursos. En un mundo idílico este sería el enfoque adecuado, ya que aporta el máximo beneficio en prestaciones al conjunto de usuarios.

Con el tiempo se ha visto que no todo usuario en la red quiere convivir con los demás, ni tienen buenas intenciones, y las comunicaciones se han convertido en una mina de oro para nuevas modalidades de delincuencia.

Con el fin de lucrarse, los llamados ciber-delincuentes pueden intentar coordinar acciones ocultas a los usuario en máquinas externas (*botnets*), bombardear agresivamente con publicidad o directamente robar datos personales con los que suplantar a su propietario en trámites bancarios. Situaciones en las que no se pensó en el desarrollo original de Internet, pero que hoy ocurren con una frecuencia abrumadora.

Esto ha llevado a plantear activamente soluciones según se van descubriendo diferentes vulnerabilidades o "trucos" en los sistemas y en las redes que permitan actuar a estos delincuentes. En los últimos años la ciberseguridad se ha convertido en una de las principales preocupaciones de los grandes actores del mundo de las comunicaciones e Internet. Cada vez más, el resto de empresas y el público están empezando a ser conscientes de ello, a través de diferentes escándalos como el robo de fotografías personales a famosos o la filtración de datos confidenciales de servicios como Ashley Madison.

El mundo de la ciberseguridad es muy amplio. Comprende dominios tan dispares como las intrusiones en aplicaciones web, el bastionado de sistemas o el estudio con ingeniería inversa de malware o código malicioso. En los programas de ingeniería actuales no suele haber una gran oferta en estos campos. Lo más común en los grados de ingeniería es ofrecer una asignatura introductoria, y el TFG es una buena manera de profundizar en un tema concreto.

Estas son las razones que han llevado a la elección de la temática de este TFG. Poder profundizar en uno de los aspectos de la seguridad, en la que se inicia a los alumnos

durante el tercer curso del grado y el reto de enfrentarse a plataformas, conceptos y sistemas a los que no se está a veces tan acostumbrado.

Además empujará el desarrollo de la capacidad resolutiva frente a los problemas desconocidos y enfrentará al alumno al diseño de una solución software que interaccione con programas de terceros.

1.2 Objetivos

El objetivo principal ha sido diseñar un plan de mejora para la detección de *malware*, adaptando software existente que permite analizar de manera sistemática el comportamiento de archivos. Se ha centrado el enfoque en el malware evasivo moderno, que trata de comprobar si está ejecutándose en una máquina virtual o una *sandbox* para no mostrar actividad maligna en tal caso. Para ello, se crearán máquinas virtuales con características no convencionales y se adaptará la configuración de Cuckoo Sandbox haciendo más difícil detectar sus características de entorno.

Los sistemas operativos con los que se ha trabajado son: Ubuntu 14.04 LTS, para el *host* en el que se ejecuta Cuckoo, VirtualBox, etc., y Windows XP SP3, para las máquinas *guest* dentro de VirtualBox.

El proceso llevado a cabo ha consistido en:

- Investigación sobre virtualización, *sandboxing* y análisis de malware. Adquisición de una base conceptual sobre el entorno en el que se desarrolla el trabajo. Comprender las ventajas e inconvenientes que impone el uso de este tipo de sistemas. Saber determinar sus debilidades y cómo son explotadas por el *malware* moderno. Ser capaz de entender el proceso al que se somete una muestra potencialmente maliciosa cuando interactúa con estos sistemas.

- Análisis de protecciones frente a análisis dinámico usadas por *malware*. Conocer las diferentes características de los entornos de análisis y entender cómo una muestra de código malicioso puede percibirlas. Enumerar y caracterizar las técnicas usadas por el *malware* para detectar los diferentes indicios de análisis.

- Diseño de medidas para mejorar la detección de *malware* esquivo. Basado en lo descrito en las fases anteriores, se propone una serie de medidas que palían completa o parcialmente las técnicas usadas por el *malware* a la hora de ser analizado. La elección de estas medidas está enfocada a completar las características que las *sandboxes* ya ofrecen de por sí, y que permitan un funcionamiento normal y automatizado de las mismas.

- Implementación de la solución. Aplicación automatizada de las medidas diseñadas para la creación de máquinas virtuales y configuración de *sandboxes* para su uso en el análisis de *malware.*

- Verificación de las mejoras a través del análisis de muestras reales. Se comprobará la efectividad de la solución planteada realizando una batería de pruebas automatizadas, con muestras reales obtenidas de diferentes repositorios y bases de datos de Internet en busca de comportamiento anti-análisis. Primero se ha

planteado el escenario de pruebas, fijando SO (Sistema Operativo) del g*uest* y programas instalados. A partir de ahí se han analizado las muestras usando el sistema "mejorado" y el sistema sin medidas aplicadas. Después, se presenta una comparativa de análisis efectivos en el sistema mejorado.

Tras el desarrollo realizado se muestran una serie de conclusiones respecto al resultado conseguido y el uso de sistemas de *sandboxing* en el análisis de *malware* moderno, planteando posibles líneas futuras relacionadas con el TFG.

Para ayudar a la lectura de este documento se facilita un glosario de términos y siglas, ordenado alfabéticamente, en el Anexo H.

Capítulo 2

Estado del arte

Es necesario introducir conceptos básicos sobre el análisis de malware y las tecnologías básicas usadas para ello. No se profundizará excesivamente, pero se va a dar una visión clara sobre los sistemas virtualizados, las *sandboxes* y cómo se afronta hoy en día el análisis de código malicioso.

2.1 Virtualización y máquinas virtuales

Los conceptos de máquina virtual (VM, por sus siglas en inglés) y *sandbox* se basan en la llamada virtualización. La virtualización puede definirse como la creación a través de software de algún recurso ya sea hardware, un SO (Sistema Operativo), una zona de almacenamiento, etc., obteniendo una "interpretación virtual" del mismo. La capa de software encargada de esta virtualización se llama *hypervisor* o *Virtual Machine Monitor* (VMM). Esta capa divide todos los recursos disponibles de la máquina física como CPU, memoria, etc., en diferentes entornos de ejecución que el *hypervisor* se encarga de gestionar.

Uno de los usos más extendidos de la virtualización, aunque no el único, son las máquinas virtuales. Una VM es la simulación de un equipo completo virtualizado dentro de uno físico, denominando respectivamente *guest* o huésped, a la VM, y *host* o anfitrión, a la máquina física. Las VM pueden tener sus propios drivers para los distintos dispositivos de los que imita la presencia, pero es el *host* quien accede a estos recursos. Por ello se puede tener un gran número de *guests* dentro de un mismo *host*.

Otro uso, es el denominado como "VM de aplicación". En lugar de imitar un sistema completo, soporta un proceso al que le proporcionará un entorno de ejecución particular pudiendo ser usado en cualquier equipo y comportándose con las mismas características. El ejemplo más conocido de este caso es la Máquina Virtual de Java (JVM).

A pesar de la visión básica dada del concepto de virtualización, a la hora de aplicarlo a las VM existen dos posibilidades para alcanzar el objetivo: virtualización y emulación.

En la virtualización la VM se ejecuta directamente sobre el hardware de la máquina física, controlada por el *hypervisor* que a efectos prácticos añade una carga extra al *scheduler*. En cambio, en la emulación el hardware sobre el que se ejecuta la VM está completamente basado en software. Esto da mucha libertad en algunos casos, se pueden ejecutar sistemas y programas diseñados para una arquitectura completamente distinta, como plataformas obsoletas, al igual que resulta en una reproducción más fiel en ciertos aspectos del sistema. A cambio, tiene una penalización en rendimiento mucho mayor que el caso anterior, pero con los procesadores tan potentes que hay en el mercado hoy en día es más que asumible.

La gran expansión de este tipo de tecnología se explica por sí sola pensando en algunas de las ventajas de su uso, tanto en clientes como en servidores: la posibilidad de "volver atrás" cargando una *snapshot* (imagen del sistema) previa hace que las pruebas y el mantenimiento en servidores virtuales sea mucho más sencillo y ahorre coste; como ya se ha comentado puede haber varios *guest* en un mismo servidor físico, lo que genera un ahorro directo de hardware mediante lo que se denomina consolidación de servidores; también se puede concentrar el gasto en mantenimiento/actualización de ese equipo concreto, aumentando la fiabilidad del mismo; y un largo etc.

Una característica básica de las VM es que sus entornos están limitados en recursos por el propio *Hypervisor* y no afectan al resto de *guests* o al *host*. Con ello, se puede decir que en teoría estos entornos están completamente aislados y son indistinguibles de una máquina real. Pero a lo largo del tiempo se han ido viendo diferentes maneras de escapar de algunas VMs. Y por supuesto, ya que es la parte central de este trabajo, también de poder diferenciarlas de una máquina real. Uno de los ejemplos más recientes de "escapar" de una VM, es la vulnerabilidad llamada "VENOM" [1], que explota un *buffer-overflow* en el controlador del disquete de máquinas virtuales Xen, KVM y QEMU.

2.2 *Sandboxes*

Una vez introducida la virtualización y las VM como uso de ella, se debe entender el concepto de *sandbox* para ver como realmente están muy ligadas y ha impulsado lo que en el campo del análisis de malware se llama análisis dinámico.

Una *sandbox* es un mecanismo de seguridad usado para probar o verificar el comportamiento de un archivo, denominado muestra, de manera aislada del resto del sistema y desde la que se puede monitorizar la actividad de la muestra en cuestión. Una *sandbox* normalmente proporciona una parte limitada de espacio de disco, acceso a red u otros recursos del sistema y cualquier cosa creada o modificada por la muestra no será visible fuera de la *sandbox*, ni será guardada tras finalizar la ejecución del análisis.

Así vemos que una *sandbox* se entiende como un ejemplo de uso de virtualización. Como el objetivo de estos sistemas es el análisis y la monitorización, han de ser ejecutados en un nivel de privilegios que esté por encima, y fuera, del entorno a analizar. Es decir, al nivel del *hypervisor*.

Pero las *sandboxes* no solo se limitan a aislar ciertos procesos y recursos del sistema, si no que se puede aislar una VM entera dentro del sistema de *sandboxing*, con lo que toda su actividad quedará monitorizada y se obtendrá información muy valiosa a la hora de caracterizar el comportamiento de muestras potencialmente maliciosas. En estos casos la capacidad de las VM de volver a una imagen previa del sistema es de tremenda utilidad ya que simplifica el poder probar, una y otra vez, código potencialmente malicioso en un entorno elegido, sin que haya que reinstalar o formatear el equipo (VM).

Aquí se empieza a ver claramente el interés que pueden tener los productores de *malware* en detectar la virtualización o el *sandboxing*. Mientras no se pueda caracterizar una muestra, esta podrá seguir realizando su actividad maliciosa sin ser detectada. Actualmente lo más deseable para el *malware* es detectar el sistema de *sandbox* en lugar

de la VM, ya que muchos "objetivos deseables" de algunas muestras son sistemas virtualizados.

Aunque no solo el *malware* está interesado en saber si está siendo ejecutado en un entorno virtual. Otros casos en los que es interesante conocerlo pueden ser fabricantes de videojuegos para comprobar si sus jugadores están haciendo trampas gracias al uso de una VM. O para detectar la presencia de algunos de los llamados "*Rootkits*", que a través de la virtualización intentan dar al usuario una sensación de normalidad mientras llevan a cabo sus actividades maliciosas.

2.3 Analizando *malware* con *sandboxes*

Actualmente el *malware* se ha convertido en todo un mercado dentro de la *Deep Web*, moviendo mucho dinero. Con ello, los productores de *malware* no solo hacen cada vez muestras más complejas y elaboradas, sino que cada vez están más extendidas y las "familias", muestras con trozos de código o comportamientos muy parecidos, son mucho más grandes.

La conocida aproximación de detección de firmas se ha quedado insuficiente en muchos casos, viéndose impotente ante el llamado "*malware* polimórfico", una misma muestra que muta en una variedad de archivos con diferentes características pero mismo efecto, que ha llevado a la industria anti-*malware* a usar nuevos mecanismos de detección más rápidos y que no se vean afectados por técnicas de ofuscación de código [2]. Un buen ejemplo es el uso de los sistemas *sandbox* previamente explicados.

Comúnmente las técnicas de análisis de *malware* se dividen en análisis estático y dinámico. El análisis estático puede hacerse simplemente extrayendo cadenas de caracteres de interés en el ejecutable o librerías importadas, o más profundamente aplicar ingeniería inversa pasando el binario a ensamblador y tratar de entender su funcionamiento. En cambio, el análisis dinámico se basa en analizar la propia ejecución/comportamiento del binario. Esto puede hacerse mediante el uso de *debuggers*, viendo paso a paso el efecto de la ejecución de cada instrucción, o como en el caso de este trabajo usando *sandboxes*.

Las *sandbox* usan diferentes mecanismos para monitorizar la muestra ejecutada tras la introducción de un módulo en la máquina virtual, la cual intentará imitar a una víctima en todos los aspectos.

Los métodos de análisis más comunes en las *sandboxes* son los *hooks*. Estos han sido clasificados en [3], donde se da una explicación algo más detallada_

- *Hooks* a nivel de usuario: obtienen información de toda la actividad a nivel de usuario de las aplicaciones, asemejándose al funcionamiento de un *keylogger*. Son los más fáciles de detectar, y por ende, de engañar.

- *Hooks* a nivel de *kernel*: modificación del propio *kernel* del *guest* para implementar el seguimiento deseado. De forma que desde el nivel de usuario, donde se ejecuta la muestra, no sea fácil detectar su presencia. Se pueden

monitorizar operaciones básicas del sistema como el manejo de registros y de archivos, en otras palabras "ver lo que el *malware* le pide al sistema que haga".

- Emulación del sistema: modificación de un emulador de hardware de forma que se coloquen *hooks* en zonas de memoria concreta y poder registrar la actividad de *I/O* (*Input/Output)*, periféricos, etc.

Al ser el entorno de pruebas elegido un SO Windows, objetivo de la gran mayoría del *malware* actual, se necesitan tener ciertos conceptos del funcionamiento de su API (*Application Programming Interface*) y la llamada "API nativa" [4]:

- API de Windows: colección de rutinas en modo usuario utilizadas para interactuar con las funcionalidades básicas del SO, como acceso a recursos, funciones de red, o servicios de Windows. Está documentada en la plataforma de desarrollo de software de Windows, es la misma entre diferentes versiones del SO y está hecha para ser la manera óptima a través de la cual las aplicaciones interactúan con el SO.

- API nativa de Windows: aporta el interfaz de llamadas al sistema que se pueden realizar desde el modo usuario. En algunos casos cambia entre diferentes versiones del SO y no está oficialmente documentada ya que en principio no es la manera correcta por la que las aplicaciones deberían interactuar con el SO.

Normalmente las aplicaciones usan la API de Windows, la cual llama a la API nativa internamente. El interés en estas funciones de API nativa surge a raíz de que el *malware* suele usarlas directamente, en vez de las llamadas de la API común, para evitar que sean detectadas por algunas técnicas de análisis que se sitúan normalmente al nivel de llamadas de API común.

Un ejemplo del funcionamiento que hace el sistema de ambas podría ser: una aplicación a nivel de usuario llama a *WriteFile*, función de la API común contenida en *kernel32.dll*; la cual internamente llama a *NtWriteFile,* perteneciente a la API nativa y contenida en *ntdll.dll*; a partir de aquí pasaría la llamada al modo *kernel* que ejecuta finalmente la rutina.

La detección de *malware* se asemeja al juego "el gato y el ratón". El *malware* muestra nuevas técnicas que son estudiadas y añadidas a los sistemas de detección, después el *malware* evoluciona para evitar ser detectado o desarrolla nuevos comportamientos.

Como era esperable en esta evolución, han surgido diferentes maneras de hacer inefectivo el uso de *sandboxes*. Estos mecanismos no solo se basan en detección de la misma, sino también en aplicar medidas que no hagan viable su análisis automático mediante *sandboxing*. Algunas de estas técnicas de detección, que serán explicadas con detalle en el siguiente capítulo, pueden ser: buscar la interacción humana, comprobar el entorno en el que se ejecuta, o inspeccionar el propio sistema son algunas de las más usadas y que al "dar positivo" harían que el *malware* iniciara una rama de ejecución benigna que no levantase sospechas hasta que se agotase el tiempo de análisis o simplemente terminase su ejecución.

Algunas de las debilidades del análisis dinámico con *sandboxes* son: la ya citada limitación en el tiempo de análisis y en la potencia de procesado, comparándolo con un equipo real, o que solo se muestre la actividad de "un camino de ejecución". Esto último hace referencia a que las muestras pueden tener diferentes ejecuciones, no solo para mostrar actividad benigna como se ha visto, sino para desplegar diferentes modos de actividad maligna dependiendo de su entorno [5]. Con ello no se caracteriza la muestra de una manera tan detallada como se haría mediante el análisis estático del propio código si fuese posible, o se necesitaría la ejecución de varias VM con diferentes sistemas de análisis y diferentes características de entorno.

En el siguiente capítulo se enumeran algunas de las técnicas más usadas y sus posibles soluciones. Aun así, en algunos casos es posible cambiar el funcionamiento básico y obtener una con mejores resultados. Con esto se puede decir que no existe un sistema perfecto e infalible para detectar *malware* ya que, por ejemplo, los *hooks* son detectables [3].

Capítulo 3

Técnicas más comunes en el *malware* esquivo – avanzado

Las técnicas que se presentan a continuación son algunas de las más comunes que diferentes analistas, investigadores o profesionales han encontrado en los miles de muestras analizadas cada año. Conforme pasa el tiempo van variando, con lo que algunas técnicas que se pueden considerar "desfasadas" o apenas usadas actualmente han sido excluidas. Se puede encontrar información adicional más concreta, como las llamadas al sistema que se suelen usar o un listado exhaustivo de las claves de registro, en el Anexo A.

3.1 *Host Fingerprinting*

La lógica debajo de esta técnica es simple, "si el equipo en el que estoy siendo ejecutado no es el mismo que el que infecté, estoy siendo analizado". Por lo tanto, la muestra de *malware* incrustará uno o varios valores únicos del sistema que acaba de infectar en su propio binario de ejecución. Si se decide analizar esa muestra, comprobará que está ejecutándose en una máquina distinta y no mostrará comportamiento malicioso. Al reescribir parte de su binario el patrón de firma cambiará, haciendo imposible su detección por tal mecanismo [6].

Algunos de los valores únicos comúnmente elegidos son: GUID, dirección MAC, nombre de NetBIOS, una entrada de registro concreta, el *path* de ejecución o el nombre de usuario.

3.2 *Extended Sleeps*

Técnica sencilla enfocada a aprovechar el limitado tiempo de análisis dedicado a cada muestra de un sistema de análisis automático, con múltiples muestras que analizar. La muestra espera sin mostrar actividad un tiempo suficiente como para en caso de ser ejecutado en un entorno de análisis, este haya terminado. Así que aunque se utilicen varias *sandboxes* en paralelo para monitorizarla, no sería identificada [7].

Podría solucionarse modificando dinámicamente el tiempo que tiene como argumento la llamada al sistema o manipulando el reloj interno del sistema. Por ejemplo Cuckoo, incluye un sistema para evitar *extended sleeps*.

3.3 *Timing*

Una versión con cierto parecido a la anterior, pero más avanzada, son las llamadas "técnicas de *timing*".

Por ejemplo, si se obtiene el valor en milisegundos desde que se inició el sistema con *GetTickCount*(), después llama a *Sleep*(), y finalmente a *GetTickCount*() de nuevo, se puede comprobar restando ambos valores si en efecto ha pasado ese tiempo y la función de *Sleep*() no tiene un *hook* [8].

También se han visto algunos casos que seleccionan fechas concretas [3], por ejemplo el 25 de cada mes, y al comprobar la fecha del sistema llamar a *Sleep()* sino coincide. Así sucesivamente hasta que se dispara la condición y despliega su funcionamiento malicioso.

Otra mejora a esta técnica es leer valores del sistema más difíciles de suplantar como *Periodic Interrupt Timer*, ACPI (*Advanced Configuration and Power Interface) timer*, APIC *(Advance Programable Interface Controller) timer*, o ejecutar la instrucción "*rdtsc*", que devuelve un valor en ciclos del procesador [9].

En el caso de comprobar la instrucción "*rdtsc*", si la *sandbox* estuviese basada en un emulador en lugar de virtualización no sería posible detectarla, ya que la emulación replica completamente el hardware de una máquina incluyendo la CPU, con lo que tendrá su propia implementación de TSC (*Time Stamp Counter*) que sería coherente y similar al de una CPU física [10].

También se han observado muestras que comprueban la hora de manera externa al sistema [4], como podría ser conectándose a la página principal de Yahoo u otro servicio conocido.

En esta categoría también se pueden contar las muestras que intentan consumir los recursos del sistema de análisis, por ejemplo iniciando grandes bucles con operaciones básicas que a las máquinas de hoy en día no les representan apenas molestia, pero en el caso de sistemas de análisis con bajo procesado hace que termine antes el análisis que el bucle [11].

3.4 *Execution Path*

En las máquinas reales, el *malware* suele ejecutarse en directorios tales como archivos temporales, carpetas de descarga, etc. En cambio algunos analizadores directamente ejecutan estas muestras desde el directorio *root* u otros directorios poco frecuentes en situaciones reales.

Otra sencilla comprobación sobre el *path*, sea el que sea, es buscar cadenas potencialmente sospechosas de pertenecer a un sistema de análisis, como pueden ser: *sample, virus, sandbox, malware, test*… [12]

3.5 *Hiding Processes*

Las *sandboxes* suelen monitorizar la actividad de los procesos del sistema, cuándo se crean o terminan, intentando detectar actividades raras que se asocien con *malware*. Por ello ocultar su propio proceso es un mecanismo muy común.

Esconderse no solo se limita a ocultar proceso en sí. En algunos casos también es interesante evitar que su actividad de red sea percibida a través de analizadores de tráfico como Wireshark, netstat etc. Para ello, conseguir ocultarse del *driver* de WinPcap es una solución inteligente, vista en una muestra real [13].

3.6 *Interacción Humana*

Otro enfoque intenta detectar la presencia de un usuario real manejando la máquina, situación que no se da durante un análisis automático. Estas técnicas van desde la simple comprobación del movimiento del ratón en un periodo de tiempo, clicks, esperar a que el sistema sea reiniciado, a pedir *captchas* o interactuar con ventanas [3].

También se han empezado a ver implementaciones más avanzadas que esperan a que el usuario realice acciones como meterse en Facebook, su correo electrónico, o comprobar que en la máquina hay credenciales de redes sociales conocidas, historial de navegación, documentos, etc. [14].

3.7 Específicos del Entorno

Estas técnicas se basan en comprobar diferentes programas o partes del sistema que, o bien delaten la *sandbox* directamente, o sea raro que no posea una máquina real y por lo tanto considere que pueda estar siendo analizado.

En algunos casos con el fin de monitorizar el sistema, las diferentes *sandboxes* comerciales cargan DLLs o módulos propios específicos dentro del mismo, que si son detectados serán un indicio claro.

Algunas características de entorno más comúnmente comprobadas en las *sandboxes* son:

- No suele ejecutar las muestras con todos los privilegios, con lo que simplemente puede intentar realizar una acción que requiera privilegios altos para asegurarse [11].

- No suelen permitir que las muestras se conecten a Internet o limitan la conexión para evitar que se propaguen o envíen *spam*. Una muestra puede intentar descargar un archivo, a lo que la *sandbox* responderá devolviendo un error o generando un archivo automáticamente. Así que si comparase su *hash* con el del archivo que había previsto descargar lo detectaría, o al contrario, intentar acceder a un dominio web inexistente, en cuyo caso algunas *sandbox* generaría una página por defecto en lugar de un error [11].

Otra comprobación de entorno muy extendida es buscar aplicaciones concretas. En algunos casos no son una medida anti-análisis en sí, sino que esa muestra está hecha para atacar una versión de aplicación determinada, pero puede conseguir que no se clasifique la muestra como *malware* ya que no actuaría. Por ejemplo podrían comprobar si Internet Explorer tiene habilitadas extensiones de terceros o si tiene Java Runtime Environment o programas P2P como BitTorrent instalados, cosas muy comunes en máquinas reales [12].

También hay *malware* que utilizan cargadores de DLL distintos del más común con lo que si no está presente no se ejecutará [12].

Algunas muestras optan por reiniciar el sistema, lo que conlleva perder parte de las trazas de ejecución (evadiendo el análisis). Esto no es algo que suela pasar inadvertido en un análisis, así que no es un buen mecanismo anti-análisis.

También hay técnicas que directamente comprueban partes de la memoria. Por ejemplo comprobar zonas de memoria conocidas donde encontrar referencias a *hooks* colocados por ciertas *sandboxes* [15]. Esta última era usada por la conocida empresa Hacking Team y se descubrió su uso en producción tras el robo de información que sufrieron en julio de 2015.

Otra medida, que no solo comprueba si están analizándolo sino que directamente quita los posibles *hooks*, pasa por restaurar las direcciones de memoria originales de la SSDT leyéndolas de *ntoskrnl.exe* [4]. Esto puede hacerse incluso desde el nivel de usuario.

A modo de ejemplo, una técnica reactiva al entorno que no se ha considerado como "común" es el caso visto en ciertas muestras enviadas a una *sandbox* online conocida con conexión a Internet, Anubis. Filtran al exterior las direcciones IP de estos sistemas para poder añadirlas a una lista negra. Con ello futuras muestras del mismo fabricante de *malware* pueden evitar mostrarse en esos sistemas [4].

3.8 Técnicas Anti-VM

Aquí es necesario hacer subcategorías por la gran variedad que representan, con ello se distingue entre:

- Técnicas que comprueban procesos, archivos del sistema o registros.
- Técnicas que comprueban la memoria.
- Técnicas que comprueban hardware específico de VM.
- Técnicas que comprueban instrucciones específicas del procesador.

Como se va a ver a lo largo de este capítulo, la mayoría de las referencias, como cadenas de texto predefinidas, hacen objetivo a VirtualBox y VMware. La razón es sencilla: son los sistemas de virtualización más usados en la actualidad, y por lo tanto los principales objetivos del *malware* moderno.

- Procesos, archivos del sistema y registros:
    - Leer la ID del disco duro de la máquina y comparar con nombres típicos como VIRTUAL, VMWARE, VIRTUALBOX.

- Comprobar si el *ProductID* de la versión de Windows coincide con el de diferentes *sandboxes* ó VMs comerciales.

- Comprobar la *System BIOS Version* buscando nombres típicos como VIRTUALBOX o VMWARE.

- Buscar en los registros de servicios nombres como: VMTools, vmware, VirtualBoxMouse, VirtualBoxGuest, xennet, etc.

- Enumerar las claves relacionadas con las tablas ACPI como DSDT (*Differentiated System Descriptor Table)* y FADT *(Fixed ACPI Desciptor Table)*, buscando nombres como: VIRTUALBOX o xen.

- Comprobar la clave del *driver* de video buscando VMware SVGA II.

- Comprobar la clave del *Hard Drive Driver*: VMware, Virtual, IDE o Hard Drive.

- Comprobar una clave que contenga un GUID, el cual puede seguir un patrón conocido [4].

- Buscar software de apoyo como VMware tools [16] o VirtualBox Guest Additions [17].

- Procesos que en los nombres contengan las cadenas: VirtualBox, VirtualBoxService, VirtualBoxtray, vmware, tcpview, wireshark.exe...

- Buscar en los servicios de Windows cadenas como: vmci, vmdebug, vmmouse, VMTools, vmware...

- Comprobar el nombre de usuario en la máquina, y averiguar si es algo como: maltest, virus, malware, currentuser, sandbox, honey, vmware, snort...

- Buscar, en el directorio que contiene los *drivers* (%windir%\system32\drivers\), archivos que contengan cadenas conocidas.

- DLLs o ejecutables pertenecientes a VMs en el directorio system32.

- Comprobaciones de memoria. Aquí se encuentra el abanderado de las comprobaciones de virtualización, la técnica llamada RedPill de Joanna Rutkowska. Otras suelen conllevar una carga mucho mayor.

  - La técnica llamada RedPill hace objetivo a la IDT (*Interrupt Descriptor Table*) que en un ordenador físico reside en un rango memoria conocida pero en los *guest* se encuentran en otro (bajo y alto), para evitar conflictos con la tabla de descriptores entre *guest* y *host* ya que solo hay una tabla por procesador [18]. Eso evita que las interrupciones de una y otra se vean

afectadas entre ellas. A través de la instrucción en ensamblador SIDT se obtiene la dirección de memoria de la tabla para compararla con valores conocidos, pudiendo diferenciar entre un entorno virtualizado y uno real. Más adelante se aplicó la misma técnica en otras tablas como GDT y LDT [19]. Actualmente no es una buena solución ya que cuando hay más de un núcleo cada procesador las localiza en una posición distinta, con lo que diferentes ejecuciones darán diferentes valores [20]. También se ha visto que en algunos SO es posible proteger esas zonas de memoria, evitando que pueda obtenerse el valor [21].

- Técnicas que hacen objetivo al hardware virtual, aunque en los registros ya se puede encontrar bastante información relacionada por ejemplo con sus IDs:

    - Comprobar el tamaño del disco duro. Una máquina real no suele tener un espacio menor de 200 Gb, en cambio una *sandbox* sí [8].

    - Comprobar las direcciones MAC. Normalmente las VM suelen usar prefijos conocidos. Por ejemplo en VMware son 0x0569, 0x0C29, 0x1C14 y 0x5056 [22].

    - Obtener el número de núcleos. Es bastante común que a la VM se le asigne un solo núcleo de la máquina física cuando se realizan análisis. En cambio en máquinas reales lo más normal es encontrar varios [20].

    - Comprobar atributos de la pantalla como resolución, ratio de refresco y tamaño. Algunas VM los tienen fijados por defecto en todos los casos [23].

- Técnicas que comprueban instrucciones del procesador:

    - Comprobar a través de la instrucción CPUID si está activo el bit *hypervisor-present*, bit 31 del registro ECX. Acto seguido comparar el nombre de este con una lista de cadenas típicas [24].

    - Hay instrucciones concretas que tienen comportamiento diferente dependiendo de si se ejecutan en un entorno real o uno virtualizado. Por ejemplo no lanzando una excepción cuando debería [25].

    - Una instrucción en x86 puede ser alargada usando prefijos redundantes. Aunque se apilen varios se comporta como uno, siendo el tamaño máximo permitido 15 bytes. En una máquina real, el no respetar este máximo lanzaría una excepción, pero QEMU falla al hacer cumplir este límite, con lo que no lanza tal excepción [20].

Capítulo 4

Diseño de medidas

Las medidas se han dividido en bloques enfocados a combatir las técnicas citadas anteriormente. En cada bloque se expondrá: primero la técnica (o familia de técnicas) a las que está enfocada la medida, cómo les afectaría y consideraciones; segundo cómo se ha implementado en concreto la o las medidas; y tercero y último, comparativa entre antes y después, fortalezas y debilidades de la propuesta.

A la hora de evaluar la eficacia de una medida se han estudiado diferentes herramientas de detección de máquinas virtuales y *sandboxes* que la comunidad ha ido creando, tanto mecanismos encontrados en malware real como PoCs (*Proof of Concept* o Pruebas de Concepto). Estas herramientas han servido para ver de manera clara el funcionamiento de las técnicas y su respuesta ante diferentes medidas. La más representativa ha sido *Paranoid Fish* (PaFish) [15], que aglutina gran variedad de técnicas de diferente índole encontradas en muestras reales. Suele actualizarse con frecuencia, conteniendo por ejemplo las técnicas usadas por la empresa Hacking Team.

4.1 Entorno de trabajo

- VirtualBox (versión 4.3.10): conocido software de virtualización (x86 y AMD64/Intel64) de código abierto. Se puede ejecutar sobre Windows, Linux y Mac soportando una gran cantidad de sistemas operativos en sus máquinas virtuales. Permite configurar diferentes características de sus máquinas virtuales, tanto del sistema, la red, el disco o los periféricos disponibles. Aun así, al realizar configuraciones automatizadas mucha de la información relacionada con el hardware de la VM (inexistente) contiene valores por defecto, al igual que elementos como claves de registro de Windows.

- Cuckoo Sandbox (versión 1.2): sistema de *sandbox* de código abierto desarrollado en Python. Es un proyecto joven que fue anunciado y empezó su distribución en 2011. Muy vivo, con actualizaciones frecuentes que van añadiendo piezas de gran utilidad al sistema. Es capaz de analizar gran cantidad de archivos como ejecutables de Windows, DLLs (*Dynamic Link Library)*, archivos ZIP, java, diferentes *scripts*, archivos de Microsoft Office "y casi cualquier cosa" [26]. Tras un análisis devuelve resultados como trazas de llamadas al sistema realizadas por los procesos creados por la muestra, archivos creados, eliminados y descargados durante la ejecución, volcados de memoria de los procesos, capturas de tráfico en formato PCAP, capturas de pantalla durante la ejecución, volcado de memoria completo de la máquina, etc. En el Anexo B se puede ver una descripción con imágenes de un informe de ejemplo. Cuckoo soporta el uso de diferentes softwares de creación de máquinas virtuales como son VirtualBox, KVM, VMware o XenServer. Como se muestra en la figura siguiente

se usa la configuración de red *host-only* de VirtualBox, que crea un segmento virtual de red aislado a través del que comunicarse.

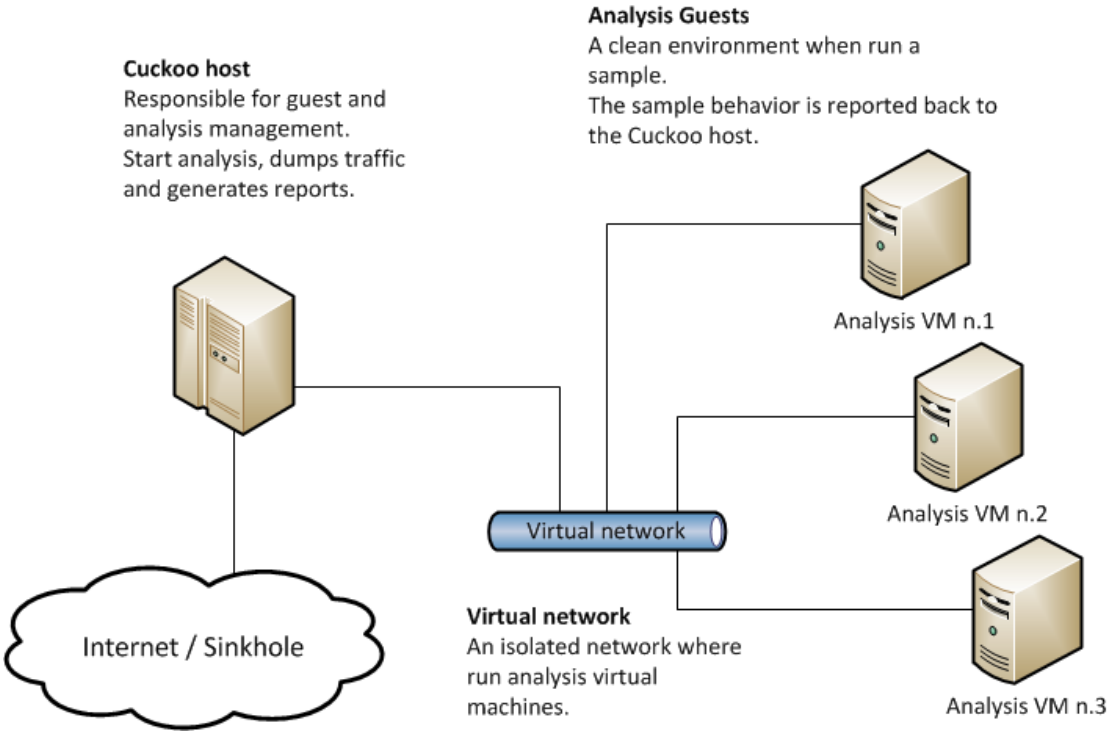


Figura 1: esquema de la arquitectura de Cuckoo. Obtenida de su web [26].

Cuckoo, a lo largo del tiempo, ha ido solucionando diferentes debilidades frente al *malware* esquivo. De esta forma, es capaz de evitar algunas técnicas por defecto, y no hará falta fortificar dichos aspectos.

- Posee un sistema para evitar las "*extended sleeps*" que intenten sobrepasar el tiempo del análisis [27]. Como ya se comentó, esto es una técnica muy básica, con lo que no es sorprendente que vaya por defecto fortificada. Se ha visto que hay casos en los que se comprueba de otras maneras que ese tiempo ha transcurrido correctamente, pero como Cuckoo da la lista completa de llamadas al sistema, se puede intentar contrastar con ejemplos conocidos y evitar caer en un falso negativo.

- Respecto al *path* en el que la muestra era ejecutada, en las primeras versiones de Cuckoo esto era una debilidad ya que la muestra la ejecutaba en un directorio con un nombre definido y fijo no común, con lo que podía ser detectado con métodos sencillos [28]. En versiones posteriores esto ha sido corregido haciendo aleatorio el nombre de este directorio.

- La ocultación de procesos no es tan problemática como en otros sistemas ya que "anota" cada nuevo proceso creado. Así aunque posteriormente sea ocultado, se conocerán los procesos lanzados.

- Hace uso de un módulo auxiliar llamado "human.py", su propio modo de imitar el comportamiento de un usuario en el sistema. Con este módulo el ratón se mueve

constantemente por la pantalla y hace diferentes clicks de vez en cuando. También interactúa con las posibles ventanas que aparecen, pulsando "aceptar", "ok", etc.

En cambio, aún hay pendiente por parte de Cuckoo un elemento importante: Cuckoo necesita que en la máquina *guest* se encuentre en ejecución "agent.py", proceso con el que se comunicará el sistema Cuckoo desde el *host*. De forma que podría comprobarse sencillamente si existe una ventana abierta con el nombre correspondiente.

Aquí se podría:

- Hacer que se ejecute sin abrir una consola nueva (cambiando la extensión de ".py" a ".pyw").

Aun así, sigue habiendo una debilidad. A pesar de que Python se extiende cada vez más, no es muy común encontrar en equipos de usuarios corrientes el proceso *python.exe* siendo ejecutado, la cual ocurrirá en la solución dada anteriormente. Se muestre o no la ventana, el *malware* puede listar los procesos y buscar entre ellos python o pythonw.

- Solución más definitiva: convertir "agent.pyw" en un ejecutable de Windows corriente que no llamase la atención frente al resto de procesos que se ejecuten en la máquina.

Esta fue la opción elegida inicialmente al plantear las medidas, ya que es la más definitiva. Sin embargo ha resultado ser más problemático de lo esperado, y no se ha conseguido que agent.py se ejecute correctamente al convertirlo en un ejecutable de Windows usando conocidas herramientas como PyInstaller [29] y Py2exe [30]. En este aspecto es necesario encontrar un sistema o herramienta que permita un correcto funcionamiento de agent.py al convertirlo a ejecutable.

4.2 Medidas frente a comprobaciones de entorno

- Evitar la instalación de VirtualBox Guest Additions.

La inclusión de esta característica de VirtualBox hace más cómodo interactuar entre la máquina virtual y el *host*, pero añade muchos archivos nombrados anteriormente en diferentes localizaciones y que son usados como objetivo en las comprobaciones del malware. Algún usuario recomienda instalarlo por comodidad durante la adecuación del *guest* y desinstalarlas posteriormente, pero es posible que no todos los archivos y claves se eliminen automáticamente.

La solución que se plantea frente a esto es por supuesto no instalar Guest Additions. Pero como sigue siendo necesaria una forma de interactuar con el *host* para añadir todo lo necesario en la preparación de la VM, se configurará un servidor FTP con vsftpd en la interfaz virtual creada para interactuar con Cuckoo en el *host* Ubuntu. Habilitando la descarga anónima no haría falta usuario y contraseña, esta inseguridad es asumible ya que estará en la red de la interfaz virtual.

Con esto se consigue una forma fácil y rápida de añadir diferentes archivos a la VM, sin necesidad de usar software sospechosos a los ojos del *malware*. El servidor FTP es desactivado durante los análisis.

- Una de las formas más comunes de detectar VirtualBox, sin contar con Guest Additions, son los numerosos registros de Windows con valores o nombres por defecto.

Este ha sido uno de los aspectos más estudiados y revisado por los diferentes investigadores o profesionales de la materia, con lo que a través de diferentes artículos y publicaciones se pueden encontrar detalladas descripciones de este tema.

No es difícil encontrar la mayoría, haciendo una simple copia de todas las claves y buscando entre ellas términos como "virtualbox", "VirtualBox" o similares. Pero esto no será un resultado fiable ya que hay varios casos en los que son números de serie, y similares, de valor fijo pero sin contener esas cadenas.

Siendo que este ha sido uno de los campos que más ha sido trabajado previamente, tras hacer las comprobaciones pertinentes de la existencia de estas claves, y ver cómo PaFish las reconocía. Se modificó un proyecto de código abierto [31] para adecuarlo a las necesidades del trabajo.

Este script, escrito en Python, utiliza la librería *dmidecode* para obtener información del *host*, que utilizará para rellenar una lista fija de las claves de registro de Windows que usan valores predeterminados. También añade información a la VM desde VBoxManage, el sistema desde el que se maneja VirtualBox en el *host*, como información de la tabla ACPI.

De esta forma, al crear un nuevo *guest* se personalizan los valores por defecto configurando información específica desde el exterior a través de VBoxManage: configuración personalizada de ACPI, información del vendedor de los discos duros y CD-ROM, información sobre DMI BIOS. Igualmente desde el interior modifica las claves de registro relacionadas con ACPI (DSDT, FADT y RSDT), BIOS (versión de la misma, fecha de lanzamiento y versión de la BIOS de la tarjeta gráfica). Con esto las comprobaciones comunes desde el interior del *guest* de VirtualBox son inutilizadas.

En lo previamente descrito hay que tener en cuenta que en algunos casos VirtualBox no soporta los datos extraídos directamente del *host* [17], de forma que se han ido aplicando en el propio código las restricciones encontradas. Aun así se requiere un estudio más exhaustivo para fijar el tamaño o formato de todos los campos. Otro comportamiento no deseado en esto casos viene a raíz de que la máquina física posea muchos núcleos de CPU, en cuyo caso la tabla DSDT del host será mayor de 64KB, que es el límite de VirtualBox.

- Una técnica no introducida en el listado de técnicas más comunes, pero de la que se ha hablado mucho, es la utilización de la información WMI (*Windows Management Instrumentation*) para detectar VirtualBox a través del UPnP (*Universal Plug and Play*).

La técnica cobró relevancia a raíz del robo de información a la empresa italiana Hacking Team, quienes la usaban en sus troyanos para evitar ser analizados. Esto puede comprobarse en una VM (Windows) de VirtualBox: ejecutando *wbemtest.exe*, introduciendo en espacio de nombres *"root\cimv2*", y haciendo la consulta "*select * from Win32_PnPEntity*", en la respuesta se encuentran varios identificadores con cadenas conocidas ("VirtualBox").

La solución presentada es bastante simple y tiene inconvenientes. Lo que se hace es desactivar el servicio UPnP a través de una clave de registro, con lo que no podría ser usado para ello… pero tampoco para otras tantas cosas, como por ejemplo verificar la autenticidad de la instalación de Windows. De forma que este también sería un apartado en el que poder profundizar.

4.3 Características de la máquina

- VirtualBox usa direcciones por defecto, por ejemplo MAC con prefijos conocidos, o cierto rango IP para las redes *host-only*. Aunque esto último no tiene por qué ser un indicador inequívoco de estar en una VM. El rango es 192.168.56.X, por defecto empezando en 101, 102, etc., con lo que podrí encontrarse en una red local.

La solución aquí es muy sencilla. Generar una dirección MAC aleatoria para cada máquina, mover la red del interfaz virtual a un rango distinto y hacer aleatorias las IPs que se asignen dentro de ese rango. También se usará uno de los DNS de Google como servidor DNS primario.

- Cuando se crea una VM por defecto, VirtualBox le asigna capacidades que en una máquina física hace muchos años que no son comunes, como son: 192MB de RAM, HDD de 10GB, un solo núcleo de CPU.

Esto se debe modificar para encajar más en los valores que suela tener una máquina hoy en día. Esto conlleva que el equipo que esté realizando el análisis necesite ser potente, más si se piensan hacer varios en paralelo. De esta manera se fija la memoria RAM en 2Gb y el tamaño del HDD en 250 Gb que son valores bastante estándar, además los núcleos de CPU en 3.

4.4 Imitación del comportamiento humano

Cuckoo trae un módulo encargado de esto de por sí. El campo del análisis de malware es reactivo a lo que hacen las muestras y viceversa, por lo que se ha visto que es posible determinar si el movimiento del ratón es el definido por una *sandbox* [7] [15] basándose en el uso de cadencias fijas, posiciones demasiado dispares en instantes muy cercanos, etc.

Por ello se decidió replantear este apartado, así que del *human-module* de Cuckoo se dejó activo solo el encargado de hacer click en las diferentes ventanas que muestren opciones ("aceptar", "ok", "next"...). A pesar de esto, hay otro aspecto que se debe cubrir en la imitación de un sistema "usado por un humano", y es que no esté "a estrenar" sin archivos, temporales, historiales de navegación, etc.

La puesta en acción de estas medidas se divide en dos partes: pre-análisis y durante el análisis.

- Pre-análisis: se generarán diferentes archivos (no vacíos) de texto en varios directorios diferentes de Windows, y se creará historial de navegación actual y previa. Esto implica que se navegará automáticamente a diferentes páginas usando Internet Explorer variando la hora y la fecha del sistema, de forma que se genere "historial" a lo largo de unos meses previos en diferentes días y un rango de horas.

- Durante el análisis: a través de un ejecutable generado a partir de un script de AutoHotKey, se han definido varias acciones "comunes" de un usuario cuando usa su equipo.

    - Crear un nuevo archivo de texto que se escribirá y guardará con nombre semi-aleatorizado.

    - Navegar manualmente por el sistema de archivos y copiar y eliminar archivos.

    - Navegar por Internet, autenticándose en diferentes webs como Facebook o Outlook.

    - Movimientos de ratón siguiendo un camino semi-aleatorio. De esta forma los puntos no son consecutivos ni fijos pero sí siguen una dirección con cadencia de movimiento aleatorizada. Igualmente clicks izquierdos (simple y doble) y derechos aleatorios.

Hay que decir que una implementación más fiel del comportamiento humano, como en este caso con navegación a Internet, etc., tiene una desventaja: los informes generados estarán "contaminados" con la actividad que realiza el "módulo humano".

4.5 Otras medidas dentro del *guest* OS

Se desactivan las actualizaciones y el *firewall* de Windows. En este caso se necesita que la máquina sea lo más vulnerable posible, pero en otros casos lo ideal sería imitar los sistemas víctima.

El nombre de usuario no debe ser indicativo de nada. Lo mejor es usar uno aleatorio introducido durante la instalación.

4.6 Técnicas conocidas no solucionadas

A pesar de todo lo nombrado anteriormente, hay ciertas técnicas de las más conocidas y usadas por el malware moderno ante las que el sistema es descubierto, sin contar varias PoCs que no son prácticas para el malware [32].

- Detección mediante sucesivas ejecuciones de la instrucción *rdtsc*, que devuelve el número de ciclos de CPU desde el último *reset* (contador TSC).

Inicialmente devolvía exactamente el número de ciclos de CPU, pero actualmente se va incrementando en una constante determinada por la frecuencia máxima a la que el procesador puede llegar a funcionar, para no variar en casos de ahorro de energía, etc. [10].

La detección se basa en que en una máquina física dos medidas consecutivas tendrán una diferencia muy pequeña, en cambio en sistemas virtualizados esta diferencia puede ser mayor, con picos, debido al *overhead* que implica la virtualización cómo ya se ha visto en secciones previas. Aunque depende del producto de virtualización, esta diferencia es realmente significativa si la máquina tiene asignados dos o más núcleos [10].

Frente a esto puede buscarse el producto de virtualización que menos afecte a este valor y asignarle solo un núcleo, esto último choca con otra medidas de comprobación simple presentadas anteriormente, que los equipos físicos no suelen tener un solo núcleo. O usar una VM basada en emulación en lugar de virtualización, ya que replican todo el hardware de una máquina, con lo que tienen su propia implementación del contador TSC y el número de ciclos entre dos *rdtsc* consecutivas se asemeja bastante al de una máquina física, por ejemplo QEMU [10].

La segunda de estas técnicas ante las que estas medidas no protegen:

- Cuckoo almacena las direcciones de vuelta (*return addresses*) en un espacio extra en conjunto con la información de los *hooks* en la TLS (*Thread-local storage*).

Como el espacio extra ocupado es conocido, basta con calcular el valor máximo y mínimo de la localización de estas direcciones y comprobar si la información obtenida de los *hooks* en la TLS, más el espacio extra, es una zona de memoria contenida entre el máximo y el mínimo. De ese modo se puede conocer si se está siendo analizado por Cuckoo [15].

Capítulo 5

Detalles sobre la Implementación

El software se ha diseñado para ser usado junto a Cuckoo y VirtualBox pero sin tener que trabajar con ellos durante la puesta a punto, relegando su uso a los análisis. Las configuraciones y dependencias son gestionadas por la solución propuesta. De esta forma todas las medidas planteadas son automatizadas de manera sencilla y clara, pudiendo generar un sistema VM-*Sandbox* bastionado y listo para usar.
En el desarrollo de esta implementación se ha hecho uso de un control de versiones basado en Git. Puede encontrarse todo el código y la información en el repositorio de GitHub habilitado para ello: https://github.com/Silverse

5.1 Ventajas

Quizás en un primer momento puede no parecer prescindible. Pero Cuckoo, al igual que otras partes de la solución propuesta, posee un buen número de dependencias. Además implica la gestión de algunos archivos de configuración que no hace automáticamente, ni con GUI *(Graphical User Interface)*. Aunque una vez se tiene claro qué se está haciendo no es complicado, al principio puede llevar más tiempo del esperado y siempre es engorroso.

Crear una máquina virtual en VirtualBox se antoja más fácil ya que posee un claro interfaz gráfico. El problema es que en ciertos aspectos no tiene la granularidad deseada y el usuario se ve obligado a tener que añadir configuración a través de VBoxManage (*command line interface*). Por ello, uno de los pasos iniciales del trabajo fue familiarizarse con estas dos herramientas, VirtualBox y Cuckoo, y saber cómo se podían gestionar de diferentes maneras y hasta dónde se podía configurar en cada una.

Finalmente, la aplicación del resto de medidas conlleva un número no despreciable de operaciones. La particularización de la solución en cada caso con valores de la VM, la elección del orden idóneo y la correcta aplicación de las medidas, es fundamental para el correcto funcionamiento. Todo ello hace necesario gestionar de manera automatizada y controlada el uso del sistema

Tras esta introducción, en la Figura 2 se puede ver lo que mostrará inicialmente la ejecución “main.py”:

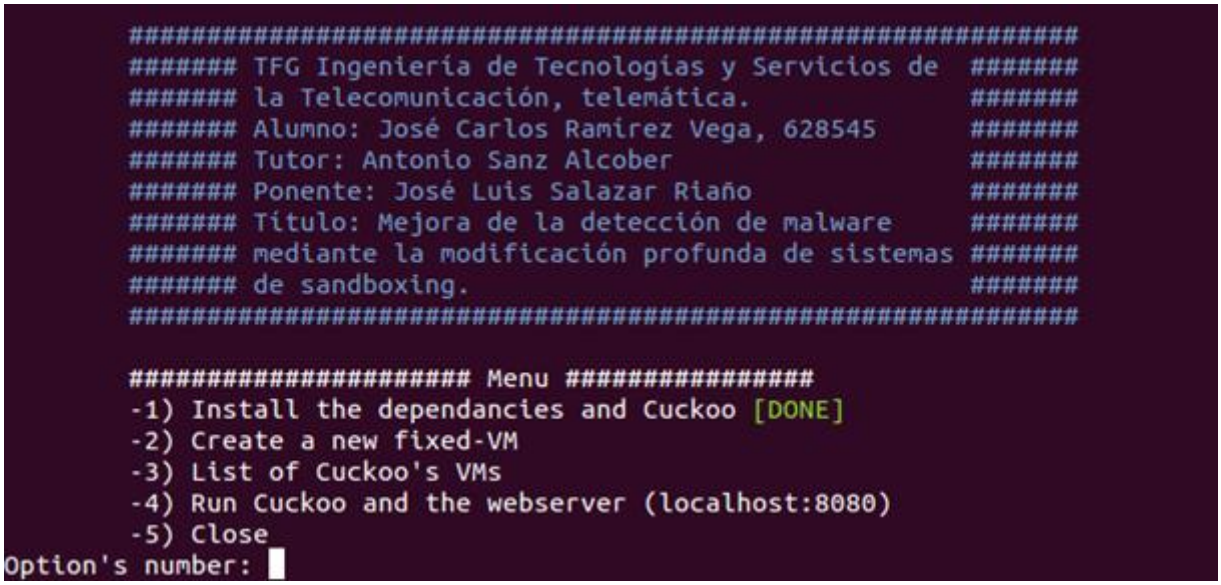


Figura 2: std-out al iniciar la ejecución.

La interfaz creada para el *Terminal* contiene una cabecera y las cinco opciones posibles, volviendo al menú tras realizar la opción seleccionada.

A la hora de seleccionar cada opción e introducir información, se han implementado ciertas comprobaciones de entorno y entrada. Por ejemplo ver si las dependencias han sido instaladas, que los nombres no contengan caracteres extraños, que se haya creado un usuario con los requerimientos para usar Cuckoo, etc. Ofreciendo en algunos casos asistencia para su creación, como en el usuario, y en otros un aviso de entrada no válida.

5.2 Funcionamiento

La primera opción, "*Install the dependancies and Cuckoo*", muestra a la derecha una comprobación del momento en el que se inicia el programa. Es imprescindible que se haya ejecutado la opción para el correcto funcionamiento del resto. Con ello se verá *"<- - Select this one*!!", en rojo, o *"[Done]"*, en verde, cada vez que se muestre el menú por pantalla. Es posible volver a seleccionarla e instalar otra vez las partes que lo permitan. Por ejemplo, los archivos de Cuckoo serían sustituidos por archivos por defecto. En el Anexo C se muestra la lista completa de las dependencias instaladas en esta opción.

La segunda opción, "*Create a new fixed-VM*", realiza el proceso necesario para generar una máquina de VirtualBox con características previamente fijadas por las medidas diseñadas o por compatibilidad con Cuckoo, como el interfaz *host-only*. También modifica los archivos de configuración de Cuckoo necesarios para detectar la VM y funcionar correctamente. En el Anexo D se listan los archivos de configuración modificados en este proceso y en el Anexo E se detallan las modificaciones y características relativas a la creación de un nuevo *guest* con la solución propuesta.

No se limita a ejecutar lo necesario para crear la VM con las características de VirtualBox deseadas. Guía al usuario durante la puesta a punto, la instalación del *guest* OS y las operaciones a realizar dentro de él, habiendo copiando los archivos necesarios a través del servidor FTP. En el *guest* solo habrá que ejecutar un *script,* aparte de instalar todo el software extra deseado en el momento seleccionado para ello.

Para obtener diferentes análisis efectivos, se deben seleccionar diferentes versiones antiguas de programas que comúnmente sean objetivo de software malicioso. Serán claros candidatos Microsoft Silverlight, Microsoft Office, Java o Flash.

La solución propuesta realiza automáticamente los apagados y encendidos de la máquina durante el proceso, con lo que el usuario solo tendrá que ejecutar algunos *scripts*. Para su uso continuado es recomendable entender el funcionamiento del software. Por ejemplo, si se apaga la VM y se quiere tomar una nueva *snapshot*, se den realizar ciertas tareas previas para preparar la VM otra vez. Por ejemplo, reescribir algunos registros del sistema.

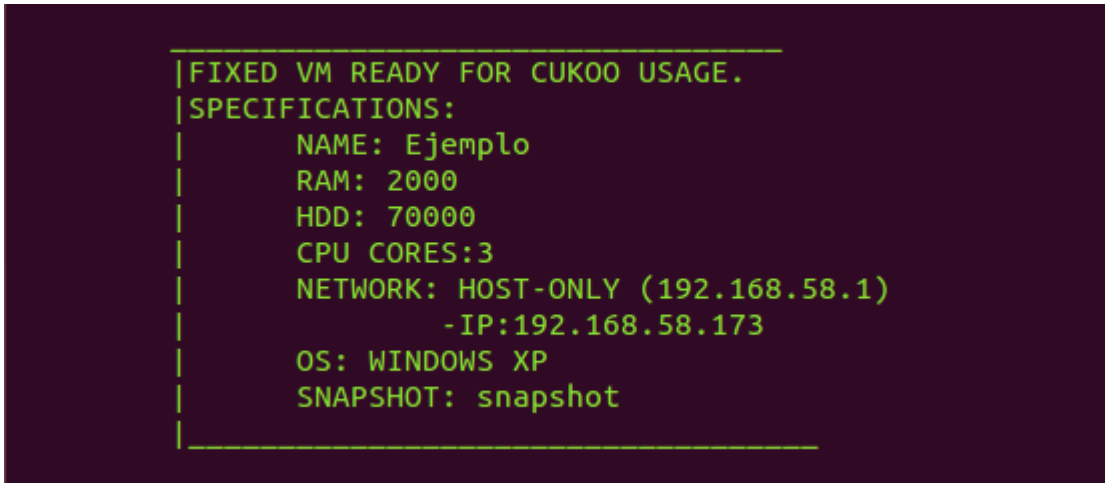


Figura 3: Resumen mostrado al terminar la creación de una VM.

En los scripts a ejecutar en el interior de la VM, se realizan dos pasos:

- Ejecutar un fichero ".bat" que instalará Python y tras ello iniciará un *script* con el mismo nombre, pero en Python.

- En el *script* en Python hay que seguir las instrucciones indicadas. Se encarga de ejecutar los instaladores imprescindibles y otros scripts para, por ejemplo, crear historial de navegación de relleno.

En cierto punto será necesario reiniciar la máquina, guiado por la solución propuesta siempre. Habrá que ejecutar otra vez "guide.py", pero eligiendo otra rama de ejecución para los ajustes post-reinicio. Culminará con la puesta en marcha del módulo de interacción humana ejecutado durante los análisis, y la toma de la *snapshot* del sistema.

La tercera opción del menú está enfocada al uso continuado, cuando se han añadido manualmente VMs a los archivos de configuración de Cuckoo. Listará las máquinas añadidas y sus características, y si alguna no está disponibles en VirtualBox o ha sido eliminadas, actualizará el fichero de configuración en consecuencia. Manteniendo así la cohesión de ambos sistemas.

Finalmente, sin contar la opción que termina la ejecución, se encuentra "*Run Cuckoo and the Webserver*". Ejecutará en dos terminales separados: primero el programa principal de Cuckoo, y segundo un servidor web al que se podrá acceder desde la dirección

"*localhost:8080*", establecida por defecto en los archivos de configuración. Este servidor web ofrece una interfaz cómoda e intuitiva para los análisis de Cuckoo, permitiendo añadir muestras o ver los informes de análisis previos.

Capítulo 6

Pruebas realizadas

Tras haber perfeccionado la implementación de las medidas elegidas se debe comprobar su eficacia frente a malware real. Previamente a enfrentar al sistema con las pruebas reales, hay que asegurarse de que las medidas implementadas se han hecho correctamente. Para ello se ha confiado en el realismo de la herramienta PaFish, analizando su ejecutable como si de una muestra de *malware* se tratase. En el Anexo F se muestra el log de salida de PaFish en ambos casos, el sistema bastionado y el básico.

6.1 Planteamiento de los escenarios

Para realizar la comparativa, se analizará el set de muestras con un sistema Cuckoo preparado a través de la solución propuesta y con un sistema Cuckoo básico, con valores por defecto y configuraciones sencillas sacadas en la documentación de Cuckoo.

El set de muestras de malware vivo usado para las pruebas procede de lo recopilado por el dominio Malc0de [33]. En concreto de muestras enviadas desde los Estados Unidos, uno de los principales objetivos de todas las campañas de malware.

Para poner a punto ambos sistemas y evitar que algunas muestras no desplieguen su comportamiento por falta de software, se ha añadido siguiendo las instrucciones facilitadas durante la instalación, el siguiente software complementario:

- Internet Explorer 8
- Adobe Acrobat Reader 5.0.5
- Microsoft Office Professional Edition 2003
- BitTorrent 4.3.3
- JRE 5.0
- Silverlight 1.0
- Utorrent 1.7.3
- Shockwave 7.03.015
- Firefox 1.0.3
- Adobe Flash Player 10
- Thunderbird 3.0

Por supuesto en ambos casos para hacer funcionar Cuckoo se necesitará:

- Instalar Python 2.7
- Instalar Python Imaging Library, ya que es necesario para las capturas de pantalla.
- Estar ejecutando “agent.py” en la versión sencilla o “agent.pyw” en la bastionada.

Durante la creación del sistema básico no se prestará atención a las indicaciones dadas previamente, con lo que se dejarán los valores de VirtualBox por defecto al realizar una instalación usando GUI:

- 192MB de RAM
- HDD de 10GB
- Un solo núcleo de CPU.
- MAC por defecto, con prefijo conocido.

También se utilizará como nombre de usuario, y de la máquina, "*malware"* y se instalará VirtualBox Guest Additions.

La revisión de resultados se ha enfocado a comprobar si en un sistema se despliegan más comportamientos que en el otro, y no en caracterizar completamente el funcionamiento de cada muestra. Se considerará que una prueba ha mostrado su comportamiento malicioso cuando descargue archivos en la versión bastionada y no en la básica. Por ello se centrará la comparativa en los archivos descargados y sus características, así como posibles anomalías presentes.

La razón principal por la cual no se han tenido en cuenta los *host* remotos contactados en los análisis o las peticiones DNS es la contaminación generada por el módulo humano. La mayoría del tráfico generado se puede conocer a qué corresponde, pero en algunos casos se crean situaciones confusas ya que, por ejemplo, Facebook carga contenido externo variable. Esto hace que no se pueda filtrar correctamente la contaminación generada en red ya que no siempre es la misma. En el Anexo G se puede ver un ejemplo de contaminación generada, no solo en red sino en todo el sistema.

De esta manera tras realizar todo el conjunto de análisis y haber obtenido los informes de ambos sistemas, se compararán en los resultados, y se generará un documento mostrando las diferencias entre ambos.

Definición de los casos de éxito:

- No descarga archivos en la versión bastionada: es *malware* que no contacta con el exterior o es *malware* anti-análisis capaz de superar las mejoras del sistema bastionado.

- Los mismos archivos son descargados en ambas versiones: no tiene comportamiento anti-análisis o lo tiene, ha superado las medidas de la solución propuesta y además sigue descargando archivos que simulen un comportamiento normal. Este segundo caso es algo enrevesado, pero el tipo de duda lleva a que se deba considerar como desfavorable.

- El nombre de los archivos descargados es el mismo, pero poseen diferente tamaño. Es algo relativamente común y dado que en el caso bastionado tendríamos más información, se podría considerar positivo. Aun así no se tienen indicios suficientes para afirmar tal cosa.

- Se alcanza el tiempo máximo de encendido de la VM: la muestra ha forzado que no pueda realizarse el análisis con normalidad, y sobrepasa tanto el tiempo de análisis como el límite estricto total.

    - Si la versión bastionada no alcanza el tiempo máximo pero la básica sí, la muestra posee comportamiento anti-análisis inefectivo contra la solución propuesta.

    - Si ambas versiones alcanzan el tiempo máximo no se puede afirmar que tenga comportamiento estrictamente anti-análisis, pero ha levantado sospechas que harán que sea revisada más a fondo.

- Nuevos archivos descargados en la versión bastionada: la muestra tiene comportamiento anti-análisis, ya que en la versión básica ha elegido una rama de ejecución en la que no despliega completamente su actividad maliciosa. En cambio, cuando es analizada por el sistema bastionado no es capaz de identificarlo como VM o *sandbox* y directamente opta por descargar el *payload* real que infecte completamente el equipo.

6.2 Resultados obtenidos

Se ha analizado un total de 200 muestras elegidas de manera aleatoria en la base de datos. De estos 200 análisis, en 12 casos ha ocurrido algún problema grave, como que VirtualBox deje de responder o el *host* tenga algún fallo, y no se ha terminado el análisis Estos casos no están relacionados con el comportamiento de la muestra. Estos 12 casos no son computados, con lo que tenemos un total de 188 análisis efectivos.

Las estadísticas extraídas de los informes son las siguientes:

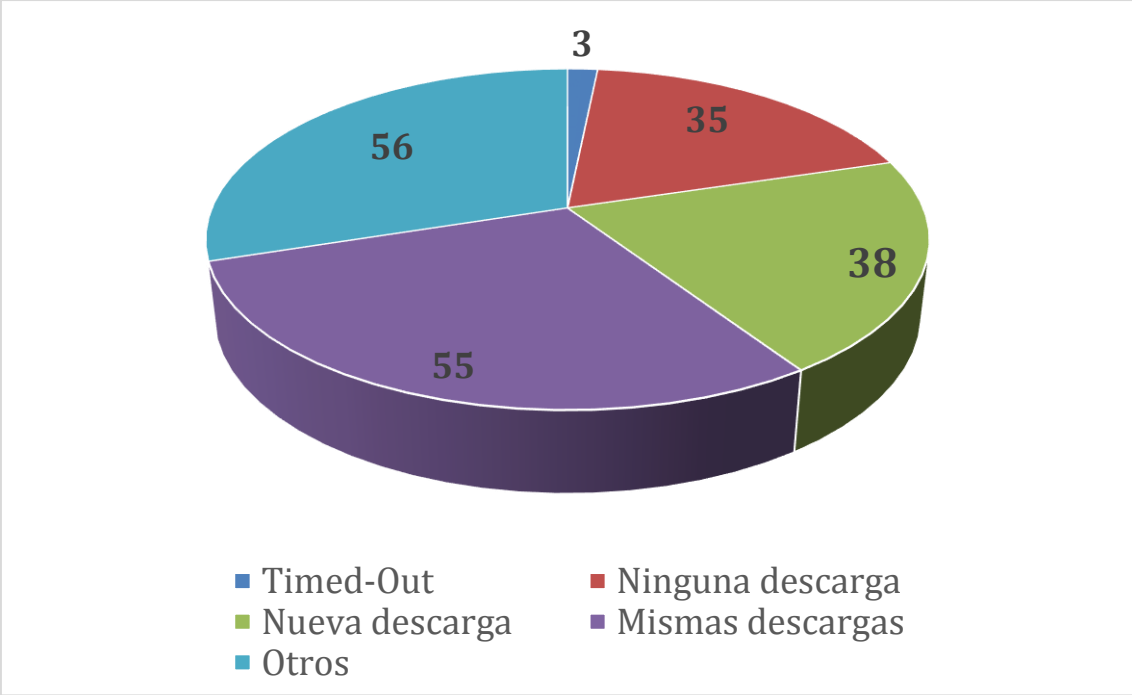


Fig 4: Gráfica de datos en bruto.

Llaman la atención sobre todo tres datos:

- Mismas descargas en 55 casos. Para evitar incertidumbre debería analizarse bien el comportamiento, de forma que si ha descargado un ejecutable consideremos que no posee capacidades anti-análisis. Pero como se ha expuesto en la sección anterior, se toma como caso desfavorable.

- Ninguna descarga en 35 casos. Este dato refleja exclusivamente los análisis con el sistema bastionado. A primera vista parece un dato muy preocupante, ya que da la sensación de que 35 muestras que han detectado el sistema.

- Otros, 56 casos. En este grupo entran los casos en los que se han descargado archivos con el mismo nombre pero diferente peso.

Para entender que el segundo dato no significa un fracaso, hay que ver cómo funciona Cuckoo. Durante algunos análisis no consigue seleccionar efectivamente las opciones de la GUI del instalador de la muestra. Puede ser porque esté en un idioma distinto del inglés, o por que use opciones o formatos no conocidos para el módulo humano, esto no es tan raro ya que este apartado de Cuckoo no es un software excesivamente complejo.

A esto hay que añadirle que cierta parte de las muestras analizadas, son sencillamente *AdWare*. Programas que poseen excesiva publicidad durante su uso. En estos casos no es raro que si no se continúa con la instalación, la muestra no descargue contenido por su cuenta y entre en el alarmante segundo dato. Por ello no se debe considerar automáticamente como *malware* evasivo a la totalidad de las muestras del segundo grupo.

Para poder conocer la efectividad real del sistema, no solo es necesario trabajar con un volumen mayor de muestras para eliminar casos marginales como instaladores que no continúan su desarrollo, también se debería trabajar sobre un ser de muestras que posean exclusivamente comportamiento anti-análisis, ya que varios de los casos expuestos generan incertidumbre al no poder estar seguros de si ha sido efectivo o no.

Tras haber descrito los posibles casos a considerar como negativos, se puede juzgar mejor la relación que tienen los casos de éxito con el total. De esta manera se replantean los resultados:

- Casos con con diferente peso, 56 casos. Como ya se ha comentado previamente, no se deben incluir en desfavorables pero tampoco se tienen suficientes indicios como para darlos completamente por favorables. Por ello podemos descartarlas.

- 38 casos con nuevos archivos. Representan éxitos rotundos, ya que demuestran un cambio de comportamiento en el que se muestra una naturaleza más agresiva.

- 35 casos sin descarga y 55 con las mismas descargas. De partida es el dato que muestra los fallos del sistema. Los que posean las mismas descargas puede que no tengan comportamiento anti-análisis, pero ante la duda se debe considerar que el sistema está siendo detectado.

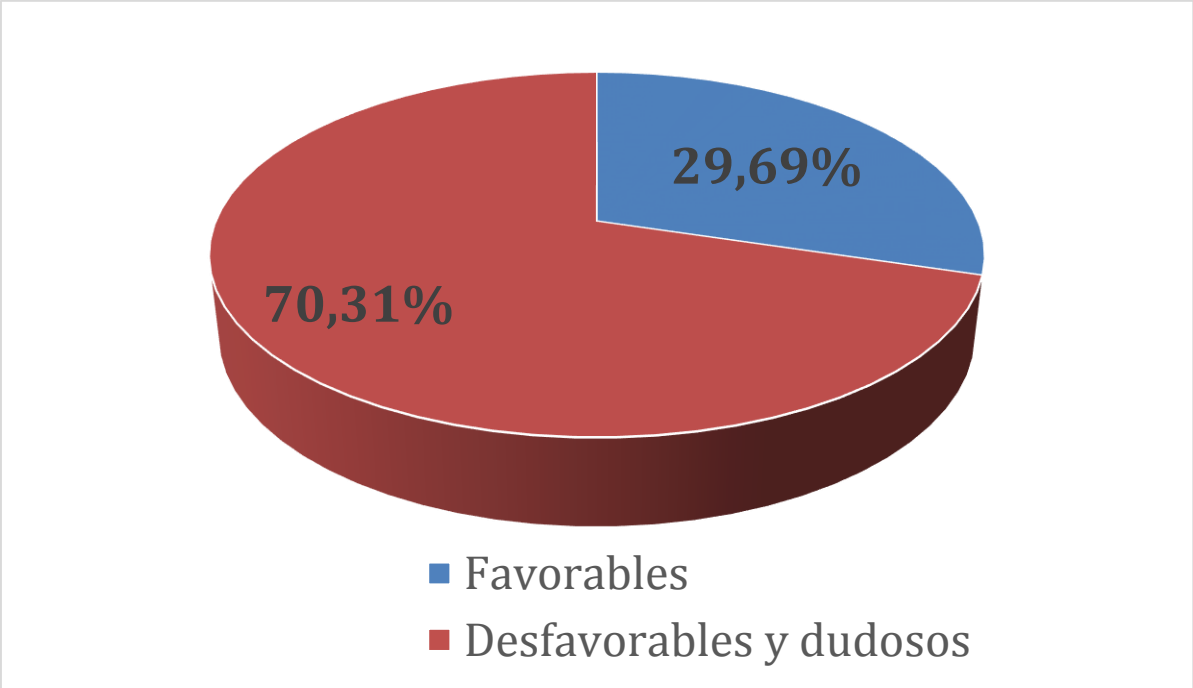


Fig 5: Resultados tras replantear los datos.

En esta segunda gráfica, habiendo aplicado el criterio derivado de un conocimiento más especifico del análisis con Cuckoo, muestra como la solución resuelve como mínimo un tercio de los casos. Aún así la incertidumbre sigue siendo demasiado alta como para sacar conclusiones definitivas, por lo que se debería replantear la metodología de pruebas para aportar mayor fiabilidad a los resultados, en lugar de dar por desfavorables los casos con duda.

Capítulo 7

Conclusiones y trabajo futuro

Al enfrentar el sistema contra pruebas reales se comprueba que buena parte de ellas siguen siendo dudosas. Sería preciso replantear la metodología de pruebas desde su inicio, partiendo de una selección cuidada en el sistema básico, de forma que solo se enfrente el sistema bastionado a muestras con comportamiento anti-análisis asegurado, comprobado tanto con tráfico de red como con archivos descargados.

Uno de los aspectos más importantes sobre los que continuar trabajando es conseguir que el módulo "agent.py" pudiera ser tratado como un ejecutable corriente. En una de las pruebas realizadas se pudo comprobar como una muestra sencillamente terminaba los procesos "python.exe" y "pythonw.exe". Este comportamiento no da lugar a dudas en la situación actual, ya que se trata de una base de datos de *malware* y podemos identificarlo como un comportamiento anti-análisis, pero si en lugar de terminar los procesos simplemente eligiese no hacer nada, pasaría fácilmente el análisis.

Se podría estudiar más profundamente el módulo de comportamiento humano para reducir la contaminación en los informes, además la forma más correcta de implementarlo sería integrándolo en el módulo "human.py" de Cuckoo. Podrían estudiarse diferentes funciones con las que elegir trayectorias para el movimiento de ratón, intentando mantener la coherencia correspondiente a los movimientos que realizaría un usuario. La navegación por internet podría mejorarse también, haciendo uso de una extensión de navegador que bloquee todo el contenido de terceros que las páginas visitadas tratasen de cargar. Esto llevaría a elaborar una lista fija de dominios visitados por el módulo durante el análisis, que al comparar con las DNS *requests* y *hosts* del informe, filtraría correctamente dejando solo las acciones de la muestra.

Otro aspecto del módulo humano que se podría trabajar más profundamente es la capacidad de dar sensación de uso. En la solución propuesta se limita a crear diferentes carpetas con archivos de textos distintos y a generar historial de navegación y cookies. Se debería conseguir generar información de uso en todos los programas instalados en el *guest*, al igual que obtener archivos de diferentes tipos, personalizar aspectos no relevantes del sistema como el fondo de pantalla, etc.

También sería interesante añadir mejores criterios a los casos de éxito de las pruebas. Para ello se debería mejorar el *script* con el que se extraen datos de manera automatizada de los informes, recogiendo más información relevante. Se deberían revisar exhaustivamente los informes y volcados de memoria correspondientes a muestras con comportamiento anti-análisis e identificar cómo reflejan las técnicas. Al final se deberían haber comprobado absolutamente todas las técnicas sobre la que se han diseñado mejoras, y saber identificarlas en un análisis. A partir de ahí se debería trabajar sobre PoCs o comportamientos que no se vean frecuentemente, pero que mejorarían la efectividad del sistema de cara al futuro.

Los requisitos establecidos conllevaban el uso de VirtualBox, software para el que Cuckoo se encuentra optimizado en su desarrollo. Esto no implica que no pueda dar buenos resultados con otros sistemas de virtualización o emulación. Un estudio pausado sobre las diferentes soluciones de virtualización y emulación existentes en el mercado, considerando concienzudamente las ventajas y desventajas que posean, podría resultar en una mejora de resultados.

Como se ha mostrado, el uso de *sandboxes* se ha convertido en un método cómodo y efectivo en la lucha contra el *malware* moderno, pero no infalible. Aun así la preparación de un sistema de *sandboxing* cuidado puede dar grandes resultados en la mayoría de los análisis.

Es necesario continuar investigando incesantemente e incentivar a los profesionales e investigadores a participar activamente, por ejemplo mediante sistemas de recompensa como los *bug bounties*. Como se ha visto en el caso de Hacking Team, el *malware* siempre tienen un truco bajo la manga o va un paso por delante (*0-day*) sin que se conozca, y no se puede confiar en que todos los días se filtren sus bases de datos masivamente.

Dado que los posibles fallos técnicos o "pistas" que dejan estos sistemas y permiten su detección están cada vez más controlados, y que el uso de sistemas de virtualización ha dejado de ser un tema de investigadores y ha sido adoptado por un gran número de sistemas de producción, el "descubrir la VM" está dejando de ser el punto central de las técnicas de evasión.

Estas técnicas se mueven hacia detección exclusiva para *sandboxes* o de interacción con el usuario, ya que no discrimina entre sistemas virtualizados y no pero presenta una eficacia nada despreciable frente a sistemas de análisis. Por ello cada vez se debería prestar más atención y esfuerzo al diseño de "módulos de comportamiento humano" que no solo realicen unos sencillos movimientos y clicks de ratón, en los que es fácil detectar patrones. Se debe tomar una concepción en la que se intente imitar una sesión real de un usuario, con todas las situaciones aleatorias que se dan, y la capacidad de responder a los intentos de interacción que pueda realizar el *malware*, desde cartelitos sencillos de "next, ok, accept", hasta *captchas*, o preguntas dirigidas.

De manera personal, la realización de este trabajo me ha llevado a darme cuenta de algo importante. A lo largo de la titulación, aparte de obtener conocimientos sobre los que poder cimentar el desarrollo profesional de los alumnos en los diferentes campos de estudio, también "aprendemos a aprender", se fomenta el desarrollo de pensamiento crítico y la capacidad para avanzar por nuestra cuenta en diferentes materias.

De esta forma podemos seguir nutriéndonos y ampliando nuestros horizontes sea cual sea el camino que se tome al terminar el grado. Nos ha dado la capacidad de saber y poder enfrentar los diferentes retos profesionales que nos esperan en el futuro.

Para mí, este trabajo ha sido una muestra de ello. Era un tema sobre el que no tenía conocimiento alguno, pero en el que he tenido la posibilidad de desarrollar mi TFG. Sobre él ahora tengo una perspectiva más clara y el conocimiento suficiente para poder seguir profundizando en ello, habiendo descubierto un campo profesional apasionante.

No ha sido el conocimiento en sí, sino poder trazar el camino hasta obtenerlo.

Bibliografía

[1] Geffner J. Sitio web: vulnerabilidad VENOM. Disponible en: http://venom.crowdstrike.com/. 2015.

[2] Vinod P. Laxmi V. "Survey of Malware Detection Methods". Disponible en: http://www.security.iitk.ac.in/contents/events/workshops/iitkhack09/papers/vinod.pdf. 2009.

[3] Bilogorskiy N. Sharma S. "Malware's Most Wanted. Anti-Sandbox malware techniques". Webinar. Disponible en: www.youtube.com/watch?v=4Cp2ZMAtog8. 2015.

[4] Lindorfer M. "Detecting Environment-Sensitive Malware" [Tesis de Máster]. Faculty of Informatics, Vienna University of Technology. 2011.

[5] Moser A. Kruegel C. Kierda E. "Exploring Multiple Execution Paths for Malware Analysis". Proceedings of the 2007 IEEE Symposium on Security and Privacy. 2007.

[6] Chubachi Y. Aiko K. "TENTACLE: Environment-Sensitive Malware Palpation". PacSec 2014. 2014.

[7] Lakhani A. "Malware Sandbox and Breach Detection Evasion Techniques". Dr. Chaos [www.drchaos.com/]. Disponible en: http://www.drchaos.com/malware-sandbox-and-breach-detection-evasion-techniques/. 6 de mayo de 2015 [acceso julio de 2015].

[8] Hoffman N. "VM Checking and Detecting". Blog personal [securitykitten.github.io/]. Disponible en: http://securitykitten.github.io/vm-checking-and-detecting/. 3 de diciembre de 2014 [acceso julio 2015].

[9] Pek G. Bencsath B. Buttyan L. "nEther: In-guest Detection of Out-of-the-guest Malware Analyzers". . In Proceedings of the Fourth European Workshop on System Security, EUROSEC'11. 2011.

[10] Ortega A. "rdtsc x86 instruction to detect virtual machines". Plug it, play it, burn it, rip it [blog.badtrace.com]. Disponible en: http://blog.badtrace.com/post/rdtsc-x86-instruction-to-detect-vms/. 22 de marzo de 2015 [acceso julio de 2015].

[11] Nasi E. "Bypass Antivirus Dynamic Analysis". Disponible en: http://packetstorm.wowhacker.com/papers/virus/BypassAVDynamics.pdf. 2014.

[12] Singh A. Bu Z. "Hot Knives through Butter: Evading File-based Sandboxes". BlackHat Conference U.S. 2013. 2013.

[13] Vasilescu M. Gheorghe L. Tapus N. "Practical Malware Analysis based on Sandboxing". RoEduNet Conference 13th Edition: Networking in Education and Research Joint Event RENAM 8th Conference. 2014.

[14] Kirat D. Vigna G. “BareCloud: Bare-metal Analysis-based Evasive Malware Detection”. Proceedings of the 23rd USENIX Security Symposium. 2014.

[15] Ortega A. Repositorio: herramienta PaFish. Disponible en: https://github.com/a0rtega/pafish. 2015.

[16] VMware support team. "Workstation User's Manual 7.1". VMware [www.vmware.com]. Disponible en: http://www.vmware.com/pdf/ws71_manual.pdf. 2010 [acceso julio de 2010].

[17] VirtualBox support team. "Oracle VM VirtualBox User Manual". VirtualBox web [www.virtualbox.org]. Disponible en: https://www.virtualbox.org/manual/. 2015 [acceso julio de 2015].

[18] Zidouemba A. "How does malware know the difference between the virtual world and the real world?" Snort's Vulnerability Research Team's Blog [vrt-blog.snort.org]. Disponible en: http://vrt-blog.snort.org/2009/10/how-does-malware-know-difference.html. 14 de octubre de 2009 [acceso julio de 2015].

[19] Quist D. Smith V. “Detecting the Presence of Virtual Machines Using the Local Data Table”. Disponible en: http://www.offensivecomputing.net/files/active/0/vm.pdf. Año.

[20] Singh S. “Breaking the Sandbox”. Disponible en: https://www.exploit-db.com/docs/34591.pdf. Año.

[21] Liston T. Skoudis E. “On the Cutting Edge: Thwarting Virtual Machine Detection”. Disponible en: http://handlers.sans.org/tliston/ThwartingVMDetection_Liston_Skoudis.pdf. 2006

[22] Joe Sandbox team. "VM and Sandbox Detections become more professional" Joe Sandbox Blog [joe4security.blogspot.com.es]. Disponible en: http://joe4security.blogspot.com.es/2012/08/vm-and-sandbox-detections-become-more.html. 2 de agosto de 2012 [acceso julio de 2015].

[23] Kang L. Xiaoning L. “Comprehensive Virtual Appliance Detection”. BlackHay Conference Asia 2014. 2014.

[24] Rin N. “Virtual Machines Detection Enhanced”. Disponible en: http://www.heise.de/security/downloads/07/1/1/8/3/5/5/9/vmde.pdf. 2013.

[25] Chen X. Andersen J. Morley Z. Bailey M. Nazario J. “Towards an Understanding of Anti-virtualization and Anti-debugging Behavior in Modern Malware”. International Conference on Dependable Systems &Networks. 2008.

[26] Cuckoo Foundation. "Cuckoo Sandbox User Manual". Cuckoo sandbox Book [cuckoo.readthedocs.org]. Disponible en: https://cuckoo.readthedocs.org/en/latest/. Octubre de 2014 [acceso julio de 2015].

[27] Kolbitsch C. Kirda E. Kruegel C. "The Power of Procrastination: Detection and Mitigation of Execution-Stalling Malicious Code". In Proceedings of the 18th ACM Conference on Computer and Communications Security. 2011.

[28] Ferrand O. "How to detect the Cuckoo Sandbox and hardening it?" 22nd EICAR Annual Conference. 2013.

[29] Goebel H. Zibricky M. Bajo G. Sitio web: herramienta PyInstaller. Disponible en: http://www.pyinstaller.org/. 2013

[30] Heller T. Retzlaff J. Hammond M. Sitio web: py2exe. Disponible en: http://www.py2exe.org/. 2013.

[31] Keri M. Repositorio: antivmdetection script. Disponible en: https://github.com/nsmfoo/antivmdetection. 2015.

[32] Reguera D. Repositorio: herramienta anticuckoo. Disponible en: https://github.com/David-Reguera-Garcia-Dreg/anticuckoo. 2015.

[33] Malc0de. Base de datos: malware vivo. Disponible en: http://malc0de.com/database/. 2015.

[34] Gyung M. Yin H. Hanna S. McCamant S. Dawn S. "Emulating Emulation-Resistant Malware". In Proceedings of the 2nd Workshop on Virtual Machine Security (VMSec'09). 2009.

[35] Bachaalany E. "Detect if your program is running inside a Virtual Machine". Code Project [www.codeproject.com]. Disponible en: www.codeproject.com/Articles/9823/Detect-if-your-program-is-running-inside-a-Virtual. 4 de abril de 2005 [acceso julio de 2015].

[36] Álvarez V. Sitio web: herramienta YARA. Disponible en: http://plusvic.github.io/yara/. 2015.

[37] Google Co. Sitio web: servicio VirusTotal. Disponible en: https://www.virustotal.com/. 2015.

[38] Volatility Foundation. Repositorio: Volatility Framework. Disponible en: https://github.com/volatilityfoundation/volatility. 2015.

Anexos

Anexo A - Información ampliada de las técnicas del capítulo 3

A.1 *Host Fingerprinting:*

Llamadas al sistema para la obtención del GUID:

- *GetVolumeNameForVolumeMountPoint()*
- *GetVolumeInformation()*
- *GetVolumNameByHandle()*
- *GetAdaptersAddresses().*

A.2 *Extended Sleeps*:

Posibles llamadas al sistema utilizadas:

- *SleepEx()*
- *NtDelayExecution()*

A.3 *Timing*:

Un ejemplo de esto, es un simple bucle llamando a *GetTickCount*(). *GetTickCount*() está implementado en Windows de forma que en lugar de hacer una llamada de sistema costosa, sencillamente lee el valor directamente de una página de memoria del *kernel* (el cuál es actualizado por el SO periódicamente).

Anubis realiza un monitoreo pesado, invocando funciones de log antes y después de cada llamada al sistema. De esta manera se puede implementar un gran bucle que llame a *GetTickCount*() sin que tenga un coste computacional apreciable en máquinas reales y en cambio cuando sea analizado por Anubis, haga dos llamadas de log y una lectura de registro por cada una. Convirtiéndolo en una carga suficientemente grande como para que el análisis termine mucho antes que el bucle [4].

A.4 *Execution Path*:

System Calls (SC) para la obtención del *execution path*:

- *mmioOpen()*
- *GetCommandLineA()*
- *GetModuleFileNameA()*

A.5 *Hiding Processes:*

La monitorización de procesos creados puede realizarse usando por ejemplo la rutina de *kernel PsSetCretaeProcessNotifyRoutine*.

A.6 Específicos del Entorno:

A través de *GetModuleHandleA*() o *Module32First*() y *Module32Next*() puede comprobar si entre los DLLs cargados hay cosas como: *sbiedll*.dll (Sandboxie), *vmcheck*.dll (Virtual PC)... y varios más, de forma que si el DLL está presente la muestra actuará en consecuencia.

En estos casos no tiene porque simplemente terminar su ejecución, si no puede intentar quitar el DLL usado por el sistema de análisis (haciendo que deje de monitorizar o controlar *hooks*) a través de *FreeLibrary*(). En este caso existen técnicas para proteger el sistema de análisis de que eliminen un DLL importante, una sencilla pero no completamente segura es cargar varias veces el DLL en cuestión de forma que su *Reference Count* será mayor que uno, y cuando se llame a *FreeLibrary*() reducirá este *Reference Count* pero no llegará a cero, que sería cuando de verdad eliminaría el DLL [20].

Esta solución no es completa porque de la misma manera que se llamaría varias veces a *LoadLibrary*(), puede llamarse a *FreeLibrary*() hasta que el *Reference Count* sea cero. Otras soluciones podrían ser poner un *hook* en las funciones de enumeración o directamente "esconder el DLL". Las funciones citadas hacen uso del *Process Environment Block*, que tiene tres listas de enlaces de los DLLs cargados, con lo que simplemente desvinculando el DLL en cuestión de estas, no se mostraría con las técnicas citadas ni se podría quitar con *FreeLibrary*() (estas funciones usan los datos de la PEB).

A.7 Técnicas anti-VM:

- Claves de registro, se abren usando *RegOpenKey*, *RegQueryValueEx*, *RegEnumKey*:
    - HKLM\SYSTEM\ControlSet001\Services\Disk\Enum; con valor 0 para leer la ID del disco duro de la máquina.
    - HKLM\HARDWARE\DEVICEMAP\Scsi\Scsi Port 0\Scsi Bus 0\Target Id 0\Logical Unit Id 0 con valor Identifier ; buscar identificadores (nombre del disco) como: vmware, vbox.
    - HKLM\Software\Microsoft\Windows\CurrentVersion; con valor ProductId, y comprobar el ID.
    - HKLM\HARDWARE\Description\System; con valor SystemBiosVersion.
    - HKLM\SOFTWARE\Microsoft; comprobar (enumerando) Hyper-V, VirtualMachine.
    - HKLM\SYSTEM\ControlSet001\Services;
    - HKLM\HARDWARE\ACPI\DSDT; HKLM\HARDWARE\ACPI\FADT; HKLM\HARDWARE\ACPI\RSDT ;
    - HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Class\{4D36E968-E325-11CE-BFC1-08002BE10318}\0000\DriverDesc .
    - HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Enum\IDE\DiskVMware_Virtual_IDE_Hard_Drive_____00000001\30303030303030303030303030303130\FriendlyName .

- HKEY_LOCAL_MACHINE\HARDWARE\DEVICEMAP\Scsi\Scsi Port 0\Scsi Bus 0\Target Id 0\Logical Unit Id 0\Identifier
- HKEY_LOCAL_MACHINE\HARDWARE\DEVICEMAP\Scsi\Scsi Port 1\Scsi Bus 0\Target Id 0\Logical Unit Id 0\Identifier
- HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Control\Class\{4D36E968-E325-11CE-BFC1-08002BE10318}\0000\DriverDesc
- HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Control\Class\{4D36E968-E325-11CE-BFC1-08002BE10318}\0000\ProviderName
- HKLM\SOFTWARE\MICROSOFT\CRYPTOGRAPHY\MACHINEGUID
- HKLM\SOFTWARE\VMware, Inc.\VMware Tools
- HKLM\SOFTWARE\oracle\VirtualBox Guest Additions

- *Drivers*:

  - *Mouse driver* (VMware): WINDIR%\system32\drivers\vmmouse.sys
  - Vbox(System32\drivers): VBoxMouse.sys, VBoxGuest.sys, VBoxSF.sys, VBoxVideo.sys

- System32/: VBoxDisp.dll, VBoxHook.dll, VBoxMRXNP.dll, VBoxOGLarrayspu.dll, VBoxOGLerrorspu.dll, VBoxOGLcrutil.dll, VBoxOGLfeedbackspu.dll, VBoxOGLpackspu.dll, VBoxoglpassthroughspu.dll, VBoxTray.exe, VBoxService.exe, VBoxControl.exe…

- Disco duro:

  - A través de *CreateFileA*() se puede obtener un *handler* a *PhysicalDrive0* y pasarlo a *DeviceIOCOntrol* con *dwIOControlCode* 7405C (IOCTL_DISK_GET_LENGTH_INFO) lo que dividiendo por 1073741824 dará el tamaño del disco en Gb.

  - También se puede comprobar si el disco, u otros dispositivos, pertenece a un sistema no deseado a través de las funciones *SetupDiGetClassDevsA*, *SetupDiEnumDeviceInfo* y *SetupDiGetRegistryProperty*, y comparar con las típicas respuestas, "virtual, vmware, hd…".

  - También se puede obtener el número de serie y fabricante con el comando *DFP_RECEIVE_DRIVE_DATA*

- Instrucciones del procesador:

  - Si ejecuta o no interrupciones o excepciones. Un ejemplo es la instrucción *ICEBP* (0xf1), un código de operación no documentado de la arquitectura x86. Se usaba antiguamente para *debugging* a nivel de hardware, pero en las máquinas modernas simplemente lanza una interrupción con vector 0x1, una versión no modificada de la VM QEMU usa esta instrucción para propósitos propios [34].

  - El caso más conocido de este tipo es la "*Backdoor* I/O" de VMware. Las instrucciones privilegiadas *IN* y *OUT* cuando son ejecutadas en un equipo

real en modo usuario generarán una excepción, pero VMware usa *IN* en un puerto especial (VX) que solo existe dentro de la VM como interfaz entre las VM y el software de VMware en sí, con lo que en ese caso no generará una excepción [12].

- Virtual PC también posee instrucciones con finalidad de *backdoor* que cumplen esta descripción [35].

Anexo B - Informe de un análisis realizado con Cuckoo

En este Anexo se va a describir y comentar la información que un informe de análisis muestra.

Primero se ofrece un pequeño resumen sobre el análisis en sí y la máquina virtual, y las condiciones en que se ha hecho.

FIXED	FIXED	VirtualBox	2015-09-03 00:42:57	2015-09-03 00:48:37
Machine	Label	Manager	Started On	Shutdown On

FILE	2015-09-03 00:42:57	2015-09-03 00:48:38	77 seconds	1.3
Category	Started On	Completed On	Duration	Cuckoo Version

Figura B.1

Justo después se ve otro cuadro resumen, esta vez sobre el archivo analizado. Información como tamaño, tipo, hashes etc. Los dos últimos son secciones muy interesantes, la primera es la aplicación de firmas Yara (firmas basadas es *strings* en vez de en *hashes* como otros sistemas de firmas) [36], y segundo el resultado de analizar el fichero en VirusTotal [37] que se despliega para ver los resultados detalladas de cada antivirus.

File name	ad0veupdates2o16.exe
File size	1592958 bytes
File type	PE32 executable (GUI) Intel 80386, for MS Windows
CRC32	19E8B87D
MD5	290a16be4671788afbff08ae44cc8551
SHA1	b53f1da0c7cd8b7d3c679d23807b71de2ce2e14a
SHA256	808aec02c17eb14a8e8325c41387a57e7b9aa1d49aa4071ee271c797f15f94df
SHA512	b42450853e82713d9bad15350b457d02690d051ccc8cf9227c19884df2ebf960d134a8ce5d2752ce0d20bb897d268c775c22fdb5387245563093d04eb9c1ac3a
Ssdeep	49152:dJZoQrbTFZY1iasFZQI5lTxr3BXLKfH5fLRFS:dtrbTA1DMrhKfp3S
PEiD	None matched
Yara	None matched
VirusTotal	Permalink VirusTotal Scan Date: 2015-09-02 13:57:02 Detection Rate: 9/57 (Expand)

Figura B.2

A continuación se ve la serie de capturas de pantalla del interior del *guest*, donde poder ver ciertos comportamientos de manera gráfica (en un análisis de 120 segundos se suelen obtener unas 60-70 capturas).

Screenshots

Figura B.3

En este nuevo bloque titulado *Static Analysis* se muestran aspectos al analizar el binario en sí de manera estática, como *Imports* y *Strings*.

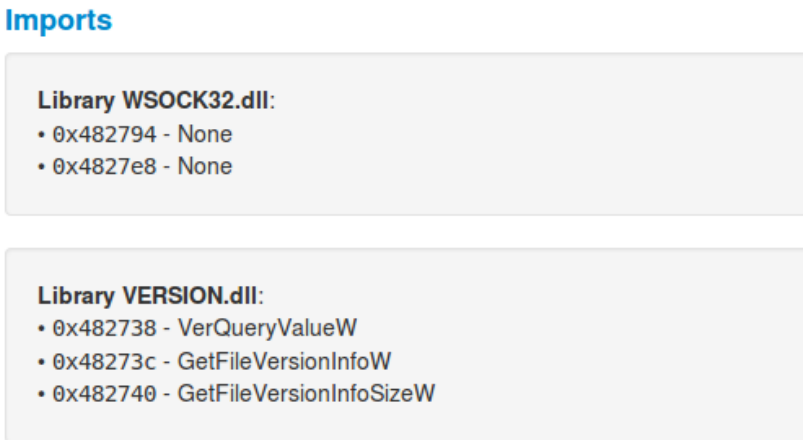


Figura B.4

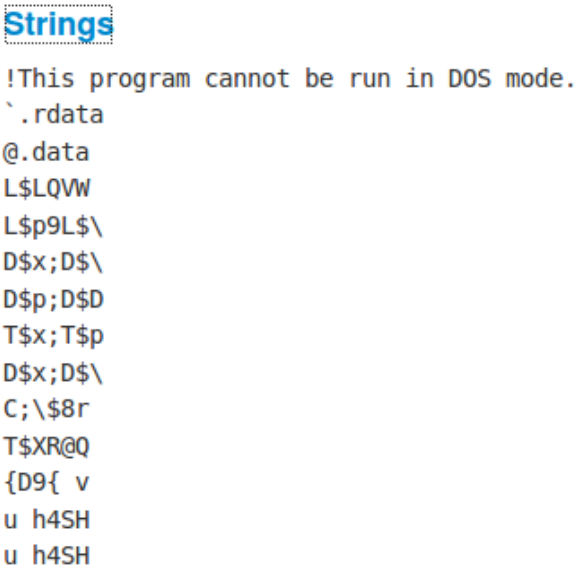


Figura B.5

Continúa con un bloque en el que da información de todos los archivos que ha creado o descargado el proceso objetivo. De cada uno muestra un cuadro de información como el nombrado al principio en la Figura B.2.

Dropped Files

gSHWKHmfD

File name	gSHWKHmfD
File size	858978 bytes
File type	data
MD5	53f9b2f4064986af55ddf4ede33404ac
SHA1	5125b2326a5cc4d65a3182a170f9f87a081ae3cb
SHA256	d63233b5a25b855fd72647ece17318e73354257dc4c602091f80137d4d8850ce
SHA512	4b7b45dcdc795283d2651964ed4db7feced43632330154ebf46c4102805db70a1d17ac94d1d1674fc67af431d
Ssdeep	24576:orU8f4Iyyllhokmq3Bj7Lg3fH/vTVfqWPLAerjtUNV:oZQI5lTxr3BXLKfH5fLRFS
Yara	None matched
VirusTotal	Search for Analysis

Windows.lnk

XAPJxyj.exe

Figura B.6 Solo se muestra un archivo desplegado

Aquí se encuentra el apartado sobre actividad de red, permite descargar la captura completa en formato PCAP o simplemente observar el resumen que contiene:

- IPs de los *hosts* con los que ha habido actividad

Network Analysis

Hosts Involved

IP Address
185.43.182.24
185.43.182.35
208.67.222.222

Figura B.7

- Peticiones DNS realizadas

DNS Requests

Domain	IP Address
www.gmail.com	216.58.210.133
tiles.cdn.mozilla.net	68.232.34.191
clients1.google.com	216.58.210.142

Figura B.8

- Peticiones HTTP

HTTP Requests

URL	Data
http://clients1.google.com/ocsp	POST /ocsp HTTP/1.1 Host: clients1.google.com User-Agent: Mozilla/5.0 (Windows NT 5.1; rv:40.0) Gecko/20100101 Firefox/40.0 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8 Accept-Language: en-US,en;q=0.5 Accept-Encoding: gzip, deflate Content-Length: 75 Content-Type: application/ocsp-request Connection: keep-alive 0I0G0E0C0A0 \x06\x05+\x0e\x03\x02\x1a\x05\x00\x04\x14\xf2\xe0j\xf9\x85\x8a\ x14J\xdd\x06\x16\x1b\xbc\xf6h\xb5v\xf5\x81\xb6\xbbb\x1a\xbaZ\x81/\x02\x08 \xd6i\xb3\x10\x91Xg

Figura B.9

Luego hay un sumario de la actividad de la muestra en el sistema durante el análisis, que muestra diferentes aspectos como son:

- Archivos modificados

Behavior Summary

Files
- C:\DOCUME~1\wef\CONFIG~1\Temp\ad0veupdates2o16.exe
- C:\DOCUME~1
- C:\Documents and Settings\wef
- C:\Documents and Settings\wef\CONFIG~1

Figura B.10

- Claves del Sistema modificadas

Registry Keys
- HKEY_CURRENT_USER\Control Panel\Mouse
- HKEY_CURRENT_USER\Software\AutoIt v3\AutoIt
- HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Policies\Explorer
- HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Policies\Explorer

Figura B.11

- *Mutexes*

Mutexes
- CTF.TimListCache.FMPDefaultS-1-5-21-1202660629-1284227242-1801674531-
- MSCTF.Shared.MUTEX.AEG

Figura B.12

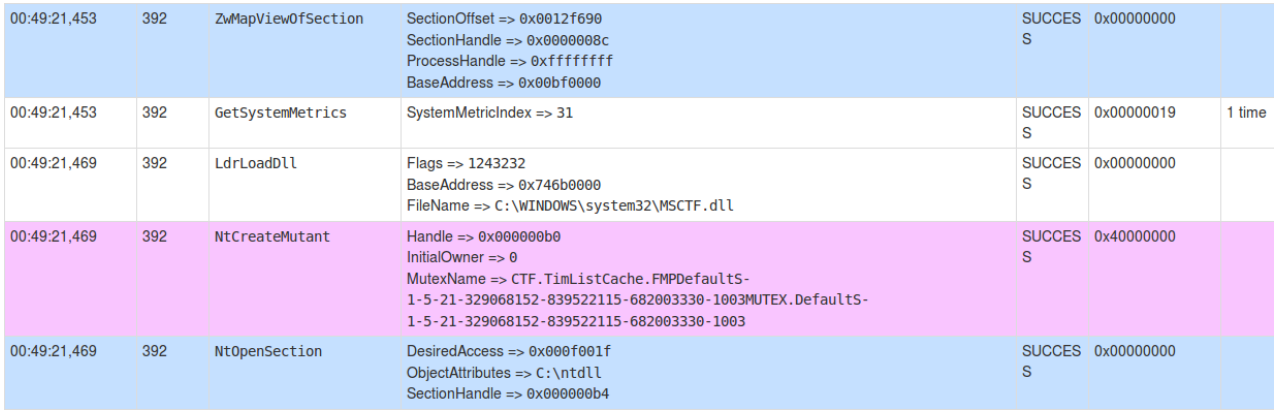
Después se encuentra un resumen de los procesos lanzados durante el análisis

Figura B.13

Cada uno de esos procesos puede ser desplegado y mostrar información ordenada de la actividad, diferenciando el objetivo de cada llamada al sistema realizada (usando la leyenda basada en colores que se muestra en la Figura B.13).

Timestamp	Thread	Function	Arguments	Status	Return	Repeated
00:45:28,500	2020	LdrGetDllHandle	ModuleHandle => 0x7c800000 FileName => KERNEL32.DLL	SUCCESS	0x00000000	

Figura B.14



00:49:21,453	392	ZwMapViewOfSection	SectionOffset => 0x0012f690 SectionHandle => 0x0000008c ProcessHandle => 0xffffffff BaseAddress => 0x00bf0000	SUCCESS	0x00000000	
00:49:21,453	392	GetSystemMetrics	SystemMetricIndex => 31	SUCCESS	0x00000019	1 time
00:49:21,469	392	LdrLoadDll	Flags => 1243232 BaseAddress => 0x746b0000 FileName => C:\WINDOWS\system32\MSCTF.dll	SUCCESS	0x00000000	
00:49:21,469	392	NtCreateMutant	Handle => 0x000000b0 InitialOwner => 0 MutexName => CTF.TimListCache.FMPDefaultS-1-5-21-329068152-839522115-682003330-1003MUTEX.DefaultS-1-5-21-329068152-839522115-682003330-1003	SUCCESS	0x40000000	
00:49:21,469	392	NtOpenSection	DesiredAccess => 0x000f001f ObjectAttributes => C:\ntdll SectionHandle => 0x000000b4	SUCCESS	0x00000000	

Figura B.15

Finalmente hay información obtenida del volcado de memoria realizado por Volatility [38].

Como extra, en algunos casos se obtiene un añadido a las secciones anteriores, en el caso de que haya situaciones que alteren parte del análisis, como es el caso de quitarle el *hook (unhook)* a una función, lo que no se guardará información sobre las llamadas a esa función a partir de ahí.

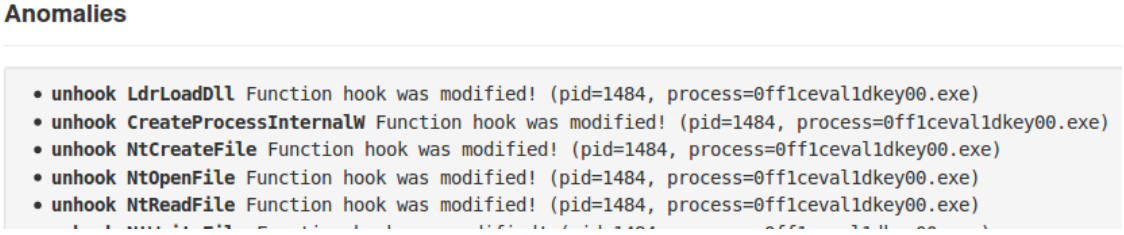


Figura B.16

Anexo C - Lista de dependencias

- Cuckoo
- VirtualBox
- Python 2
- Python Imaging Library
- Internet Explorer 8
- vsftpd
- volatility
- acpidump
- libcdio-utils
- ssdeep
- pydeep
- build-essential
- libjansson-dev
- libmagic-dev
- libtool
- eclipse-cdt-autotools
- yara
- yara-python
- tcpdump
- python-bson
- python-sqlalchemy
- python-jinja2
- python-magic
- python-pymongo
- python-gridfs
- python-bottle
- python-pefile
- python-chardet
- python-dmidecode
- python-dateutil
- python-dev

Anexo D - Cambios en ficheros de configuración

Enumeración de los valores añadidos o cambiados en cada uno de los archivos de configuración, el resto del archivo se deja por defecto.

Cuckoo:

"~/cuckoo/conf/cuckoo.conf":

- machinery = virtualbox
- memory_dump = on
- ip = 192.168.58.1
- port = 2042

"~/cuckoo/conf/auxiliary.conf":

- [sniffer]
- enabled = yes
- tcpdump = vboxnet0

"~/cuckoo/conf/virtualbox.conf":

- mode = gui
- path = /usr/bin/vboxmanage - Este *path* es extraído usando *whereis* por si acaso esté en otro directorio.
- machines = VM1,VM2,VM3 - Lista separada por comas de las VMs disponibles para Cuckoo
- Comentar el caso de ejemplo llamado cuckoo1 y eliminarlo de la lista "machines"
- [- Nombre de la VM]
- label = - Nombre de la VM.
- platform = windows
- ip = - @IP del *guest*.
- snapshot = - Nombre de la *Snapshot*
- tags = - Lista de etiquetas introducidas por el usuario.

"~/cuckoo/conf/reporting.conf":

- [reporthtml]
- enabled = yes

"~/cuckoo/analyzer/windows/modules/auxiliary/human.py"

- Comentar las líneas de la función *main*:
  - move_mouse()
  - click_mouse()

Very Secure FTP Daemon (vsftpd):
"/etc/vsftpd.conf":

- listen = yes

- anonymous_enable = yes
- dirmessage_enable = yes
- use_localtime = yes
- xferlog_enable = yes
- connect_from_port_20 = yes
- listen_address = 192.168.58.1 - Aunque en el inicio de la instalación del *guest* es 192.168.56.1, la dirección por defecto de *vboxnet0*.
- listen_port = 21

Levantamiento del *firewall* al iniciar el sistema.
"/etc/rc.local"

- Evitar *DDoS*: vigila los paquetes TCP con el *flag* SYN activado, descartando a partir del 15 proveniente de un mismo *host*. Lo que a efectos prácticos significa limitar el número de conexiones.

    - iptables -A INPUT -p tcp -i vboxnet0 -s <host_net> --syn -m connlimit --connlimit-above 15 --connlimit-mask 32 -j REJECT --reject-with tcp-reset

- Evitar *DDoS*: permite 20 nuevas conexiones antes de aplicar un límite de 30 nuevas conexiones por segundo.

    - iptables -A INPUT -m state --state RELATED,ESTABLISHED -m limit --limit 30/second --limit-burst 20 -j ACCEPT

- Evitar *spam*: tira todo el tráfico de entrada desde la VM dirigido al puerto de SMTP (25).

    - iptables -A INPUT -p tcp -i vboxnet0 -s <host_net> --dport 25 -j DROP

- Permitir la conexión al exterior de la VM:

    - iptables -A FORWARD -o eth0 -i vboxnet0 -s <host_net> -m conntrack --ctstate NEW -j ACCEPT

    - iptables -A FORWARD -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT

    - iptables -A POSTROUTING -t nat -j MASQUERADE

    - sysctl -w net.ipv4.ip_forward=1

Anexo E - Creación y modificaciones del *guest*

E.1 Valores generales

Tanto *in-* como *out- guest*.

- Memoria RAM 2 Gb.
- Tamaño HDD 250 Gb.
- Sistema operativo Windows XP SP3.
- Número de núcleos 3.
- Dirección MAC aleatorizada.
- Dirección IP aleatorizada en el rango 192.168.58.0/24.
- DNS 8.8.8.8

E.2 Valores específicos de la VM

- ACPI habilitado.
- IO-ACPI habilitado.
- Iniciar desde el lector DVD.
- Interfaz virtual en NIC 1.
  - vboxnet0 - IP 192.168.58.1
- Almacenamiento IDE:
  - Puerto 0, HDD formato "vdi".
  - Puerto 1, DVD-drive con el *path* de la imagen ISO del SO a instalar.
- *Snapshot* con el nombre elegido.

E.3 Modificaciones *out-guest*

Usando VBoxManage y tomando los valores de la máquina física.

- VBoxInternal/Devices/pcbios/0/Config/DmiBIOSFirmwareMajor
- VBoxInternal/Devices/pcbios/0/Config/DmiBIOSFirmwareMinor
- VBoxInternal/Devices/pcbios/0/Config/DmiBIOSReleaseDate
- VBoxInternal/Devices/pcbios/0/Config/DmiBIOSReleaseMajor
- VBoxInternal/Devices/pcbios/0/Config/DmiBIOSReleaseMinor
- VBoxInternal/Devices/pcbios/0/Config/DmiBIOSVendor
- VBoxInternal/Devices/pcbios/0/Config/DmiBIOSVersion
- VBoxInternal/Devices/pcbios/0/Config/DmiBoardAssetTag
- VBoxInternal/Devices/pcbios/0/Config/DmiBoardBoardType
- VBoxInternal/Devices/pcbios/0/Config/DmiBoardLocInChass
- VBoxInternal/Devices/pcbios/0/Config/DmiBoardProduct
- VBoxInternal/Devices/pcbios/0/Config/DmiBoardSerial
- VBoxInternal/Devices/pcbios/0/Config/DmiBoardVendor
- VBoxInternal/Devices/pcbios/0/Config/DmiBoardVersion
- VBoxInternal/Devices/pcbios/0/Config/DmiChassisAssetTag
- VBoxInternal/Devices/pcbios/0/Config/DmiChassisSerial
- VBoxInternal/Devices/pcbios/0/Config/DmiChassisType
- VBoxInternal/Devices/pcbios/0/Config/DmiChassisVendor
- VBoxInternal/Devices/pcbios/0/Config/DmiChassisVersion
- VBoxInternal/Devices/pcbios/0/Config/DmiOEMVBoxRev
- VBoxInternal/Devices/pcbios/0/Config/DmiOEMVBoxVer

- VBoxInternal/Devices/pcbios/0/Config/DmiProcManufacturer
- VBoxInternal/Devices/pcbios/0/Config/DmiProcVersion
- VBoxInternal/Devices/pcbios/0/Config/DmiSystemFamily
- VBoxInternal/Devices/pcbios/0/Config/DmiSystemProduct
- VBoxInternal/Devices/pcbios/0/Config/DmiSystemSKU
- VBoxInternal/Devices/pcbios/0/Config/DmiSystemSerial
- VBoxInternal/Devices/pcbios/0/Config/DmiSystemUuid
- VBoxInternal/Devices/pcbios/0/Config/DmiSystemVendor
- VBoxInternal/Devices/pcbios/0/Config/DmiSystemVersion
- VBoxInternal/Devices/piix3ide/0/Config/PrimaryMaster/ModelNumber
- VBoxInternal/Devices/piix3ide/0/Config/PrimaryMaster/SerialNumber
- VBoxInternal/Devices/piix3ide/0/Config/PrimaryMaster/FirmwareRevision
- VBoxInternal/Devices/piix3ide/0/Config/SecondaryMaster/ATAPIVendorId
- VBoxInternal/Devices/piix3ide/0/Config/SecondaryMaster/ATAPIRevision
- VBoxInternal/Devices/piix3ide/0/Config/SecondaryMaster/ATAPIProductId
- VBoxInternal/Devices/piix3ide/0/Config/SecondaryMaster/ATAPISerialNumber
- VBoxInternal/Devices/acpi/0/Config/AcpiOemId
- VBoxInternal/Devices/acpi/0/Config/AcpiCreatorId
- VBoxInternal/Devices/acpi/0/Config/AcpiCreatorRev

E.4 Modificaciones *in-guest*

- Claves de registro:
    - HKLM\HARDWARE\ACPI\DSDT\
    - HKEY_LOCAL_MACHINE\HARDWARE\ACPI\DSDT\
    - HKEY_LOCAL_MACHINE\HARDWARE\ACPI\FADT\
    - HKEY_LOCAL_MACHINE\HARDWARE\ACPI\RSDT\
    - HKEY_LOCAL_MACHINE\HARDWARE\DESCRIPTION\System /v SystemBiosVersion
    - HKEY_LOCAL_MACHINE\HARDWARE\DESCRIPTION\System /v VideoBiosVersion
    - HKEY_LOCAL_MACHINE\HARDWARE\DESCRIPTION\System /v SystemBiosDate
    - HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\services\Plug Play /v Start
    - HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\Current Version\WindowsUpdate\Auto Update

- Desactivar el *firewall* de Windows.

Anexo F - Resultados finales aplicando PaFish

Para comprobar la efectividad de la implementación, previamente a las pruebas se ha sometido el ejecutable de la herramienta PaFish a análisis a través de cuckoo.

F.1 Sistema básico

[pafish] Start
[pafish] Windows version: 5.1 build 2600
[pafish] CPU vendor: GenuineIntel
[pafish] CPU VM traced by checking the difference between CPU timestamp counters (rdtsc)
[pafish] CPU VM traced by checking the difference between CPU timestamp counters (rdtsc) forcing VM exit
[pafish] Sandbox traced using mouse activity
[pafish] Sandbox traced by checking username
[pafish] Sandbox traced by checking disk size <= 60GB via DeviceIoControl()
[pafish] Sandbox traced by checking disk size <= 60GB via GetDiskFreeSpaceExA()
[pafish] Sandbox traced by checking if NumberOfProcessors is less than 2 via raw access
[pafish] Sandbox traced by checking if NumberOfProcessors is less than 2 via GetSystemInfo()
[pafish] Sandbox traced by checking if pysical memory is less than 1Gb
[pafish] VirtualBox traced using Reg key HKLM\HARDWARE\DEVICEMAP\Scsi\Scsi Port 0\Scsi Bus 0\Target Id 0\Logical Unit Id 0 "Identifier"
[pafish] VirtualBox traced using Reg key HKLM\HARDWARE\Description\System "SystemBiosVersion"
[pafish] VirtualBox traced using Reg key HKLM\SOFTWARE\Oracle\VirtualBox Guest Additions
[pafish] VirtualBox traced using Reg key HKLM\HARDWARE\Description\System "VideoBiosVersion"
[pafish] VirtualBox traced using Reg key HKLM\HARDWARE\ACPI\DSDT\VBOX__
[pafish] VirtualBox traced using Reg key HKLM\HARDWARE\ACPI\FADT\VBOX__
[pafish] VirtualBox traced using Reg key HKLM\HARDWARE\ACPI\RSDT\VBOX__
[pafish] VirtualBox traced using Reg key HKLM\SYSTEM\ControlSet001\Services\VBoxGuest
[pafish] VirtualBox traced using Reg key HKLM\SYSTEM\ControlSet001\Services\VBoxMouse
[pafish] VirtualBox traced using Reg key HKLM\SYSTEM\ControlSet001\Services\VBoxService
[pafish] VirtualBox traced using Reg key HKLM\SYSTEM\ControlSet001\Services\VBoxSF
[pafish] VirtualBox traced using Reg key HKLM\SYSTEM\ControlSet001\Services\VBoxVideo
[pafish] VirtualBox traced using Reg key HKLM\HARDWARE\DESCRIPTION\System "SystemBiosDate"
[pafish] VirtualBox traced using driver file C:\WINDOWS\system32\drivers\VBoxMouse.sys
[pafish] VirtualBox traced using driver file C:\WINDOWS\system32\drivers\VBoxGuest.sys

```
[pafish] VirtualBox traced using driver file
C:\WINDOWS\system32\drivers\VBoxSF.sys
[pafish] VirtualBox traced using driver file
C:\WINDOWS\system32\drivers\VBoxVideo.sys
[pafish] VirtualBox traced using system file C:\WINDOWS\system32\vboxdisp.dll
[pafish] VirtualBox traced using system file C:\WINDOWS\system32\vboxhook.dll
[pafish] VirtualBox traced using system file C:\WINDOWS\system32\vboxmrxnp.dll
[pafish] VirtualBox traced using system file C:\WINDOWS\system32\vboxogl.dll
[pafish] VirtualBox traced using system file
C:\WINDOWS\system32\vboxoglarrayspu.dll
[pafish] VirtualBox traced using system file C:\WINDOWS\system32\vboxoglcrutil.dll
[pafish] VirtualBox traced using system file
C:\WINDOWS\system32\vboxoglerrorspu.dll
[pafish] VirtualBox traced using system file
C:\WINDOWS\system32\vboxoglfeedbackspu.dll
[pafish] VirtualBox traced using system file
C:\WINDOWS\system32\vboxoglpackspu.dll
[pafish] VirtualBox traced using system file
C:\WINDOWS\system32\vboxoglpassthroughspu.dll
[pafish] VirtualBox traced using system file C:\WINDOWS\system32\vboxservice.exe
[pafish] VirtualBox traced using system file C:\WINDOWS\system32\vboxtray.exe
[pafish] VirtualBox traced using system file
C:\WINDOWS\system32\VBoxControl.exe
[pafish] VirtualBox traced using MAC address starting with 08:00:27
[pafish] VirtualBox traced using device \\.\VBoxMiniRdrDN
[pafish] VirtualBox traced using VBoxTray windows
[pafish] VirtualBox traced using its network share
[pafish] VirtualBox traced using vboxservice.exe process
[pafish] VirtualBox traced using vboxtray.exe process
[pafish] VirtualBox device identifiers traced using WMI
[pafish] Cuckoo hooks information structure traced in the TLS
[pafish] End
```

F.2 Sistema bastionado

```
[pafish] Start
[pafish] Windows version: 5.1 build 2600
[pafish] CPU vendor: GenuineIntel
[pafish] CPU VM traced by checking the difference between CPU timestamp counters
(rdtsc)
[pafish] CPU VM traced by checking the difference between CPU timestamp counters
(rdtsc) forcing VM exit
[pafish] Sandbox traced using mouse activity
[pafish] Cuckoo hooks information structure traced in the TLS
[pafish] End
```

Anexo G - Comparativa de informes generados por Cuckoo

Con el fin de apreciar la contaminación producida por el módulo humano implementado, aquí se muestra parte de los informes generados al analizar un sencillo archivo de texto, que solo contiene unas cadenas de texto aleatorias formadas por números, letras y '-', llamado "seriales.txt".

G.1 Sistema básico:

Network Analysis

- Nothing to display.

Behavior Summary

Files

- C:\DOCUME~1
- C:\DOCUME~1\jjjj
- C:\DOCUME~1\jjjj\CONFIG~1
- C:\DOCUME~1\jjjj\CONFIG~1\Temp
- C:\DOCUME~1\jjjj\CONFIG~1\Temp\seriales.txt
- C:\WINDOWS\system32\msctfime.ime

Mutexes

- CTF.TimListCache.FMPDefaultS-1-5-21-1409082233-436374069-1957994488-1003MUTEX.DefaultS-1-5-21-1409082233-436374069-1957994488-1003
- ShimCacheMutex
- MSCTF.Shared.MUTEX.EPF

Registry Keys

- HKEY_CURRENT_USER\Software\Policies\Microsoft\Windows\System
- HKEY_LOCAL_MACHINE\Software\Microsoft\Command Processor
- HKEY_CURRENT_USER\Software\Microsoft\Command Processor
- HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Nls\Locale
- HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Nls\Locale\Alternate Sorts
- HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Nls\Language Groups
- HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\IMM
- HKEY_USERS\S-1-5-21-1409082233-436374069-1957994488-1003\Software\Microsoft\Windows NT\CurrentVersion\AppCompatFlags\Layers
- HKEY_CURRENT_USER\SOFTWARE\Microsoft\CTF
- HKEY_LOCAL_MACHINE\Software\Microsoft\CTF\SystemShared

- HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Policies\Explorer
- HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Policies\Explorer
- HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\ShellCompatibility\Applications\NOTEPAD.EXE
- HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\ShellCompatibility\Objects\{20D04FE0-3AEA-1069-A2D8-08002B30309D}
- HKEY_CLASSES_ROOT\CLSID\{20D04FE0-3AEA-1069-A2D8-08002B30309D}\InProcServer32
- HKEY_CLASSES_ROOT\Drive\shellex\FolderExtensions
- HKEY_CLASSES_ROOT\Drive\shellex\FolderExtensions\{fbeb8a05-beee-4442-804e-409d6c4515e9}
- HKEY_CLASSES_ROOT\Directory
- HKEY_CLASSES_ROOT\Directory\
- HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer
- HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\
- HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Policies\System
- HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\Advanced
- HKEY_CLASSES_ROOT\Directory\\ShellEx\IconHandler
- HKEY_CLASSES_ROOT\Directory\\Clsid
- HKEY_CLASSES_ROOT\Folder
- HKEY_CLASSES_ROOT\Folder\Clsid
- HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\User Shell Folders
- HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\FileExts
- HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\FileExts\.txt
- HKEY_CLASSES_ROOT\.txt
- HKEY_CLASSES_ROOT\txtfile
- HKEY_CLASSES_ROOT\txtfile\CurVer
- HKEY_CLASSES_ROOT\txtfile\
- HKEY_CLASSES_ROOT\txtfile\\ShellEx\IconHandler
- HKEY_CLASSES_ROOT\SystemFileAssociations\.txt
- HKEY_CLASSES_ROOT\SystemFileAssociations\text
- HKEY_CLASSES_ROOT\SystemFileAssociations\text\ShellEx\IconHandler
- HKEY_CLASSES_ROOT\txtfile\\Clsid
- HKEY_CLASSES_ROOT\SystemFileAssociations\text\Clsid
- HKEY_CLASSES_ROOT*
- HKEY_CLASSES_ROOT*\Clsid
- HKEY_CURRENT_USER\Keyboard Layout\Toggle
- HKEY_CURRENT_USER\SOFTWARE\Microsoft\CTF\LangBarAddIn\
- HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\CTF\LangBarAddIn\

Processes

- cmd.exe PID: 480, Parent PID: 296
- NOTEPAD.EXE PID: 504, Parent PID: 480

G.2 Sistema bastionado:

DNS requests

Name	@IP
www.microsoft.com	72.247.212.64
www.facebook.com	31.13.83.8
fbstatic-a.akamaihd.net	185.43.180.152
vassg141.crl.omniroot.com	84.53.132.72
i.s-microsoft.com	104.83.36.46
html5shim.googlecode.com	64.233.184.82
www.googletagservices.com	216.58.210.162
partner.googleadservices.com	216.58.211.194
securepubads.g.doubleclick.net	216.58.211.194
ads.rubiconproject.com	104.83.13.31
optimized-by.rubiconproject.com	62.67.193.41
secure-assets.rubiconproject.com	104.83.13.31
static.xx.fbcdn.net	31.13.83.4
fbcdn-profile-a.akamaihd.net	84.53.132.170
scontent-mad1-1.xx.fbcdn.net	31.13.83.4
pixel.facebook.com	31.13.83.8
4-edge-chat.facebook.com	31.13.83.8
login.live.com	131.253.61.82
auth.gfx.ms	23.54.88.70
sc.imp.live.com	104.83.200.133
account.live.com	65.54.187.24
account.microsoft.com	64.4.54.25
assets.onestore.ms	104.83.36.185
ajax.aspnetcdn.com	68.232.34.200
mem.gfx.ms	23.223.83.137
cid-c0967adb2414c24f.users.storage.live.com	134.170.107.48
www.google.es	216.58.210.131
ssl.gstatic.com	216.58.211.227
img.youtube.com	216.58.211.238
clients1.google.es	216.58.211.227
es.bab.la	85.25.30.170
ajax.googleapis.com	216.58.211.202
www.google-analytics.com	216.58.210.174
stats.g.doubleclick.net	64.233.184.155
tpc.googlesyndication.com	216.58.210.161
pagead2.googlesyndication.com	216.58.211.226

Behavior Summary

Files

- C:\DOCUME~1
- C:\DOCUME~1\dfgh
- C:\DOCUME~1\dfgh\CONFIG~1
- C:\DOCUME~1\dfgh\CONFIG~1\Temp
- C:\DOCUME~1\dfgh\CONFIG~1\Temp\seriales.txt
- C:\WINDOWS\system32\msctfime.ime

Mutexes

- CTF.TimListCache.FMPDefaultS-1-5-21-329068152-839522115-682003330-1003MUTEX.DefaultS-1-5-21-329068152-839522115-682003330-1003
- ShimCacheMutex

Registry Keys

- HKEY_CURRENT_USER\Software\Policies\Microsoft\Windows\System
- HKEY_LOCAL_MACHINE\Software\Microsoft\Command Processor
- HKEY_CURRENT_USER\Software\Microsoft\Command Processor
- HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Nls\Locale
- HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Nls\Locale\Alternate Sorts
- HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Nls\Language Groups
- HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\IMM
- HKEY_USERS\S-1-5-21-329068152-839522115-682003330-1003\Software\Microsoft\Windows NT\CurrentVersion\AppCompatFlags\Layers
- HKEY_CURRENT_USER\SOFTWARE\Microsoft\CTF
- HKEY_LOCAL_MACHINE\Software\Microsoft\CTF\SystemShared
- HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Policies\Explorer
- HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Policies\Explorer
- HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\ShellCompatibility\Applications\NOTEPAD.EXE
- HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\ShellCompatibility\Objects\{20D04FE0-3AEA-1069-A2D8-08002B30309D}
- HKEY_CLASSES_ROOT\CLSID\{20D04FE0-3AEA-1069-A2D8-08002B30309D}\InProcServer32
- HKEY_CLASSES_ROOT\Drive\shellex\FolderExtensions
- HKEY_CLASSES_ROOT\Drive\shellex\FolderExtensions\{fbeb8a05-beee-4442-804e-409d6c4515e9}
- HKEY_CLASSES_ROOT\Directory
- HKEY_CLASSES_ROOT\Directory\CurVer
- HKEY_CLASSES_ROOT\Directory\

- HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer
- HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\
- HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Policies\System
- HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\Advanced
- HKEY_CLASSES_ROOT\Directory\\ShellEx\IconHandler
- HKEY_CLASSES_ROOT\Directory\\Clsid
- HKEY_CLASSES_ROOT\Folder
- HKEY_CLASSES_ROOT\Folder\Clsid
- HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\User Shell Folders
- HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\FileExts
- HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\FileExts\.txt
- HKEY_CLASSES_ROOT\.txt
- HKEY_CLASSES_ROOT\txtfile
- HKEY_CLASSES_ROOT\txtfile\CurVer
- HKEY_CLASSES_ROOT\txtfile\
- HKEY_CLASSES_ROOT\txtfile\\ShellEx\IconHandler
- HKEY_CLASSES_ROOT\SystemFileAssociations\.txt
- HKEY_CLASSES_ROOT\SystemFileAssociations\text
- HKEY_CLASSES_ROOT\SystemFileAssociations\text\ShellEx\IconHandler
- HKEY_CLASSES_ROOT\txtfile\\Clsid
- HKEY_CLASSES_ROOT\SystemFileAssociations\text\Clsid
- HKEY_CLASSES_ROOT*
- HKEY_CLASSES_ROOT*\Clsid

Processes

- cmd.exe PID: 3700, Parent PID: 4052
- NOTEPAD.EXE PID: 1248, Parent PID: 3700

Anexo H - Glosario de términos y siglas

- **0-day:** vulnerabilidad de una aplicación o sistema que no es conocida por el público o el fabricante.
- **ACPI** (*Advanced Configuration and Power Interface*): estándar para proporcionar al sistema operativo una forma de descubrir, configurar y monitorizar hardware. Basado en la existencia de diferentes tablas de datos.
- **ACPI *timer***: temporizador de alta precisión contenido en las tablas ACPI.
- **Anubis**: *sandbox on-line* basada en QEMU.
- **API** (*Application Programming Interface*): conjunto de funciones ofrecidas por un sistema para poder ser usado a través de otro software.
- **APIC** (*Advance Porgrammable Interrupt Controler*) ***timer***: temporizador vinculado al controlador de interrupciones.
- **AutoHotKey**: Herramienta de código abierto para la creación de macros y automatización en Windows. Utilizando un lenguaje de *scripts* permite interactuar con diferentes elementos de Windows y automatizar clicks, movimientos de ratón, uso de teclado.
- **Ashley Madison**: red social de parejas dirigida principalmente a personas que ya tienen una relación. En julio de 2015 fue víctima de un robo de información, entre ella los datos de más de 37 millones de cuentas de usuarios.
- ***Backdoor*:** mecanismo que permite el acceso indirecto a un sistema.
- ***Botnet***: conjunto de ordenadores que se pueden ejecutar de forma autónoma y automática manejándolos de manera remota.
- ***Buffer Overflow***: error producido en un programa mientras se escriben datos a un buffer por el cual se pasa de los límites de esto y escribe información en zonas de memoria colindantes.
- ***Bug Bountie***: plan por el cual una compañía o propietario de un software o sistema ofrece una recompensa económica a quien encuentre y reporte de manera privada vulnerabilidades no conocidas.
- **Captchas** (*Completely Automated Public Turing Test to tell Computers and Humans Apart*): conjunto de *tests* para comprobar si el sistema está siendo usado por una persona.
- **Configuración *Host-Only***: o solo-anfitrión. Modo de red virtual de VirtualBox en el cual se aísla el segmento que conecta a la VM de la red local del *host* y se comunica a través de este.
- **CPUID**: instrucción de los procesadores x86 para obtener detalles sobre el procesador en sí.
- ***Debugger***: programa usado parar comprobar el funcionamiento, paso a paso, de un segundo programa.
- ***Deep web***: Internet profunda. Se refiere a la porción de internet que no es indexada por los motores de búsqueda comunes
- **DLL** (*Dynamic Link Library*): biblioteca de enlace dinámico. Archivos ejecutables cargados bajo demanda de un programa por parte del SO que contienen funciones que este desee usar.
- ***Driver***: componente software para que permite la comunicación entre un dispositivo hardware y el resto del sistema.
- **DSDT** (*Differentiated System Descriptor Table*): Tabla perteneciente al estándar ACPI.
- **ECX**: registro del procesador.
- **FADT** (*Fixed ACPI Descriptor Table*): Tabla perteneciente al estándar ACPI.

- **FTP**: *File Transfer Protocol*.
- **GDTR** (*Global Descriptor Table Registry*): registro de la GDT (*Global Descriptor Table*), estructura de datos usada en la arquitectura x86 en la que se definen características de diferentes zonas de memoria usadas durante la ejecución de un programa, incluyendo direcciones base, tamaño y privilegios de acceso.
- **Git**: software de control y mantenimiento de versiones de código fuente.
- ***GetTickCount***: función de Windows que devuelve la cantidad de milisegundos desde que el sistema fue inicializado.
- **GUI** (*Graphical User Interface*): interfaz visual usado para interactuar con un Sistema.
- **GUID** (*Globally Unique Identifier*): Identificador Único Global. Número pseudo-aleatorio que no garantiza ser único, aunque en la práctica se puede considerar como tal. Usado para diferentes aplicaciones software como identificador.
- **Hacking Team**: compañía italiana especializada en la venta de herramientas de vigilancia e intrusión. El 5 de Julio de 2015 se filtraron más de 400 Gb de información interna, entre ella sus productos y listas de clientes.
- ***Hash***: conjunto de bytes de tamaño fijo al que se puede mapear un conjunto arbitrario de bytes de cualquier tamaño.
- ***Hook***: sistema que intercepta una llamada al sistema o evento pudiendo modificar su comportamiento.
- **I/O***: Input and Output*.
- **ID**: identificador.
- **IDTR** (*Interrupt Descriptor Table Registry*): registro de la IDT (*Interrupt Descriptor Table*), tabla de vectores de interrupción en x86. Usada por el procesador para determinar la correspondencia entre interrupciones y excepciones.
- ***In-Guest***: acciones tomadas en el interior de la máquina virtual, el *guest*.
- ***Kernel***: programa encargado de manejar peticiones de I/O y traducirlas a instrucciones del procesador.
- ***Keylogger***: programa que guarda cada pulsación de tecleado realizada por el usuario en su sistema.
- **KVM** (*Kernel-based Virtual Machine*): solución de virtualización para *hosts* Unix, y *guests* Unix y Windows. Creada y mantenida por Qumranet. En el *front-end*, utiliza una versión modificada de QEMU.
- **LDT** (*Local Descriptor Table*): estructura de datos con las mismas funcionalidades que la GDT. La diferencia entre ambas radica en que la GDT es usada para acceder segmentos usados por cualquier programa, y la LDT es usada para acceder a segmentos usados por una aplicación en particular.
- **NetBIOS** (*Network Basic Input/Output System*): provee servicios de la capa de sesión.
- **Netstat**: herramienta por línea de comandos para manejar y mostrar conexiones de red.
- **NIC**: *Network Interface Card*.
- **Ofuscación de código**: Se refiere a realizar una serie de cambios en el código fuente de forma que no se altere su funcionamiento pero sea muy difícil de interpretar su funcionamiento al observar el código.
- ***Out-Guest***: acciones tomadas en el exterior de la máquina virtual, el *host*, pero cuyo objetivo es afectar al *guest*.
- ***Path***: localización única en el sistema de archivos.
- **PEB** (*Process Environment Block*): estructura de datos de los sistemas Windows NT, cuyo objetivo es ser usada solo por el sistema operativo. Contiene información a

aplicar a lo largo de un proceso concreto, incluyendo contexto, parámetros de inicio, etc.

- **PoC** *(Proof of Concept)*:
- **Py2exe**: herramienta para convertir un *script* en Python a un ejecutable de Windows que no necesite el intérprete Python.
- **PyInstaller**: herramienta para convertir un *script* en Python a un ejecutable de Windows que no necesite el intérprete Python
- **QUEMU**: *hypervisor* de código abierto que realiza virtualización basada en hardware (emulación).
- **Rama de ejecución**: sección del código fuente que se ejecuta en los casos en los que haya una condición en la que elegir el punto en el que se continúa.
- ***Rootkit***: tipo de *malware* que da acceso privilegiado de manera continua a ciertas partes del equipo infectado, manteniéndose oculto al administrador del mismo.
- **RSDT** (*Root System Descriptor Table*): Tabla perteneciente al estándar ACPI
- ***Sandboxes***: plural de *sandbox*.
- **Sandboxie**: solución *sandbox* que aísla entornos en los que ejecutar una aplicación concreta.
- ***Sandboxing***: hecho de realizar la función de un sistema *sandbox*.
- ***Scheduler***: componente de los SO que se encarga de repartir el tiempo disponible de un procesador entre los procesos que pueden ser ejecutados.
- ***Script***: programa normalmente simple que suele ser interpretado.
- **SGDT** (*Store Global Descriptor Table*): instrucción que consulta el GDTR
- **SIDT** (*Store Interrupt Descriptor Table*): instrucción que consulta el IDTR.
- ***Snapshot:*** imagen del sistema. Copia del estado completo de un sistema.
- **SO**: Sistema Operativo. OS, *Operative System* en inglés.
- **SSDT** (*System Service Descriptor Table*): tabla interna de punteros en sistemas Windows con información sobre *hooks* del sistema.
- **Tcpview**: utilidad de monitorización de red para Windows.
- **TFG**: Trabajo de Fin de Grado.
- **TLS**: métodos para manejar memoria estática o global en un hilo.
- **TSC** (*Time Stamp Counter*): registro en los procesadores x86. Cuenta el número de ciclos de CPU desde el último *reset*.
- **UPnP** (*Universal Plug and Play*): protocolos de red que permiten a dispositivos de red descubrirse entre ellos y establecer servicios de red.
- **VboxManage**: herramienta a través de comandos usada en VirtualBox para configurar, gestionar o monitorizar los diferentes elementos de este software de virtualización.
- **Virtual PC**: software de virtualización de Windows.
- **VirtualBox Guest Additions**: complementos software para ser usados dentro del *guest* en VirtualBox con el objetivo de facilitar su manejo.
- **VM**: *Virtual Machine*.
- **Vsftpd** *(Very Secure FTP Daemon)*: servidor FTP (*File Transfer Protocol*) para entornos Unix, bastante ligero y rápido de configurar.
- **WinPcap**: librería de Windows que implementa pcap, API para capturar tráfico de red.
- **Wireshark**: conocido analizador de paquetes de código abierto.
- **WMI** (*Windows Management Instrumentation*): infraestructura para gestionar datos y operaciones en sistemas Windows.

- **x86**: conocida arquitectura de microprocesadores. También se usa para denominar al conjunto de instrucciones que estos usan. Ha evolucionado en los microprocesadores x86-64.
- **Xen**: solución de virtualización de código abierto para *host* Unix únicamente y *guests* Unix y Windows. Desarrollado por la Universidad de Cambridge.