ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

UNIVERSIDAD DE CANTABRIA

Trabajo Fin de Grado

Estudio y aplicación de técnicas especializadas para la securización de máquinas virtuales

(Study and application of specialized techniques for the securization of virtual machines)

Para acceder al Título de

Graduado en Ingeniería de Tecnologías de Telecomunicación

Autor: José Javier Gómez Lastra

Octubre - 2017

GRADUADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

CALIFICACIÓN DEL TRABAJO FIN DE GRADO

Realizado por: José Javier Gómez Lastra
Parte I. Director del TFG: Alberto Eloy García Gutiérrez
Título: "Estudio y aplicación de técnicas especializadas para la securización de máquinas virtuales"
Title: "Study and application of specialized techniques for the securization of virtual machines"

Presentado a examen el día: Lunes 30 de Octubre de 2017.

para acceder al Título de

GRADUADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

Composición del Tribunal:
Presidente (Apellidos, Nombre): Marta García Arranz
Secretario (Apellidos, Nombre): Alberto Eloy García Gutiérrez
Vocal (Apellidos, Nombre): Roberto Sanz Gil

Este Tribunal ha resuelto otorgar la calificación de: .....

Fdo.: El Presidente

Fdo.: El Secretario

Fdo.: El Vocal

Fdo.: El Director del TFG
(sólo si es distinto del Secretario)

Vº Bº del Subdirector

Trabajo Fin de Grado Nº
(a asignar por Secretaría)

Agradecimientos.

Me gustaría dedicar este trabajo a todos los que, durante estos años, me han ayudado y apoyado, a los que han hecho posible que finalice todo esto.

A mi familia, pareja y amigos, que me han apoyado mucho durante estos últimos años en los momentos más difíciles, no dejando nunca de creer en mí. De verdad, aguantáis más de lo necesario.

A mis compañeros de clase, con los que he compartido estos últimos cinco años, y con los que espero compartir muchos más.

Por último, y no menos importante, a Alberto, mi tutor de proyecto, guiándome y ayudándome para realizar este proyecto, que ha sido más problemático de lo esperado.

Índice General

Índice de figuras

Resumen

Durante los últimos años, la computación Cloud se está volviendo cada vez más popular. Tanto es así, que grandes empresas como Microsoft, Google y Amazon, entre otras, utilizan los servicios en la nube. En estos entornos se almacenan grandes cantidades de información, y mucha de dicha información es muy delicada o valiosa. En los entornos virtuales, esta información está comprometida y es de vital importancia protegerla. Este proyecto nace de la necesidad de comprobar el estado de la seguridad en los entornos virtuales, en particular del elemento central, el hipervisor.

En este trabajo se describen las principales técnicas de ataques a entornos virtualizados, así como alguna de las soluciones planteadas y su prevención mediante su estudio en una nube privada preparada al efecto.

Aprovechando el uso de este entorno aislado, el trabajo muestra diferentes maneras de atacar un entorno virtualizado, especialmente a su hipervisor, buscando las vulnerabilidades existentes a través del uso de diferentes herramientas. Como prueba de concepto el trabajo muestra el resultado de las pruebas de diferentes ataques asociados a las vulnerabilidades encontradas previamente, realizados sobre diferentes sistemas operativos y comprobar si están solucionados o no.

Abstract

Over the past few years, Cloud computing has become increasingly popular. So much so, that big companies like Microsoft, Google and Amazon, among others, use the services in the cloud. In these environments large amounts of information are stored, and much of this information is very sensitive or valuable. In virtual environments, this information is compromised and it is vitally important to protect it. This project is born from the need to check the security status in virtual environments, in particular the central element, the hypervisor.

This paper describes the main techniques of attacks on virtualized environments, as well as some of the solutions proposed and their prevention through their study in a private cloud prepared for this purpose.

Taking advantage of this isolated environment, the work shows different ways of attacking a virtualized environment, especially its hypervisor, searching for the existing vulnerabilities through the use of different tools. As proof of concept the work shows the result of the tests of different attacks associated with the vulnerabilities previously found, made on different operating systems and to check if they are solved or not..

Capítulo 1.
Introducción y objetivos.

En este capítulo se comenzará con una breve introducción al contenido del trabajo. A continuación, se tratan las motivaciones para la realización del trabajo, se enumeran los objetivos a cumplir con el trabajo realizado, y por último, se explica brevemente la estructura del documento en sí.

1.1 Introducción.

La computación Cloud se está volviendo cada vez más popular debido a su agilidad, flexibilidad y rentabilidad. Tanto es así, que grandes empresas como Microsoft, Google y Amazon, entre otras, utilizan los servicios en la nube. Así mismo, usuarios particulares y pequeñas o medianas empresas (PYMES) utilizan los servicios en la nube, tanto para modelo de negocio (tiendas online, páginas web informativas) como para almacenamiento de datos.

Además de guardar datos, o trasladar servicios a la nube, los entornos virtualizados tienen multitud de utilidades como, por ejemplo: probar sistemas operativos; usar software que no está disponible en nuestro sistema operativo, o sólo disponible en sistemas operativos obsoletos; experimentar en el sistema operativo que corra dentro del entorno virtualizado; testeo de software en producción; o crear un servidor web, VPN, o de correo. Otra utilidad interesante es la del estudio de programas o software sospechoso de ser malicioso. En un entorno virtualizado se puede colocar código y programas para ver cómo se comporta. Una vez que se completa dicho análisis, el entorno puede ser eliminado sin riesgo para el entorno real que lo contiene.

Debido al aumento de su uso, así como la importancia de los datos que pueden contener, como por ejemplo los datos privados de usuarios, los números de tarjetas, documentación, etc., cada vez hay más ataques orientados a los entornos virtualizados, siendo cada vez más sofisticados. Esto hace que sea muy complicado implementar una defensa frente a ellos. A pesar de que la seguridad en la nube ha sido un área de investigación durante la última década, todavía existen muchos desafíos abiertos para lograrlo. Para controlar la seguridad en la nube, es crucial para desarrolladores, proveedores y usuarios entender los riesgos para tomar precauciones, implementar técnicas de seguridad ya existentes o desarrollar nuevas herramientas.

Hay que tener en cuenta que los entornos virtualizados no estaban orientados en sus orígenes hacia criterios relacionados con la seguridad, lo que ha hecho que incorporar a estas alturas dichos criterios resulte, cuando menos, un gran reto. Pese a que la seguridad

en estos entornos ha aumentado considerablemente, ésta aún no está suficientemente madura, al menos desde el punto de vista de los ataques más avanzados que ahora se están planteando.

Bajo esta perspectiva, es necesario entender los ataques que ya existen para comprender la mecánica de los mismos, y así llegar a proponer mecanismos mediante los cuales hacerles frente. Por ello, en este trabajo se prueban diferentes ataques dirigidos específicamente hacia entornos virtualizados, en particular hacia el hipervisor, para observar cómo se desarrolla la mecánica de ataque. Para ello es necesario establecer un entorno de experimentación basado en técnicas de virtualización, completamente aislado, para así evitar caer en ningún tipo de ciber-delito, ni que el tráfico pueda ser considerado "sospechoso" en la red, en este caso, la red de Unican (Universidad de Cantabria).

1.2 Motivación.

La única política de seguridad viable para la protección de la información es aquella que cubre todas las posibles brechas de seguridad. Los cibercriminales, cuando realizan ataques dirigidos contra un entorno virtualizado, tratan de abrirse camino de cualquier modo posible. Por lo tanto, intentan comprometer todas las vías de acceso posibles, por lo que la seguridad de un sistema completo equivale a tener la certeza de que se ha cubierto su frente menos protegido. Las estrategias de seguridad que no están debidamente protegidas contra todos los vectores de ataque posibles, resultan completamente inútiles frente a los ataques avanzados.

La mejor manera de conocer el funcionamiento de dichas amenazas y poder prevenirlas posteriormente es llevando a cabo estos ataques en primera persona. Realizar esto puede ser un problema, ya que realizar estas pruebas sobre sistemas reales puede resultar en un delito siempre que no haya consentimiento expreso firmado por parte de los propietarios de los elementos comprometidos. Por ello, se plantea este trabajo, un laboratorio aislado, que permita analizar vulnerabilidades como si se tratase de un entorno físico real, pero bajo el supuesto de un sistema virtualizado mediante técnicas de Cloud.

1.3 Objetivos.

De acuerdo con lo anterior, el objetivo de este Trabajo es describir las principales técnicas de ataque a entornos Cloud, así como su solución y prevención mediante la implementación de un laboratorio virtualizado y aislado que implemente una pequeña nube (Cloud) sobre la que aplicar los conceptos anteriores. Dentro de esta definición, se pueden identificar una serie de objetivos secundarios, entre los que están:

- Identificación de un servicio basado en el concepto de nube (Cloud) que presente vulnerabilidades explotables, así como un sistema operativo que permita recrear el servicio, actuando como servidor sobre el que poder alojarlo.
- Identificación de las principales herramientas utilizadas en la búsqueda y explotación de vulnerabilidades.
- Creación de un entorno virtual aislado de la red para la experimentación e implementación de las vulnerabilidades explotables identificadas.
- Conocer la explotación de los ataques, así como el alcance de los mismos.
- Poner en práctica la defensa correspondiente a los ataques practicados.

1.4 Organización del documento.

Este documento está organizado en seis capítulos congruentes entre sí, para dar el mayor enfoque posible tanto desde el punto de vista práctico como teórico de todos los aspectos tratados a lo largo del trabajo.

Para ello, tras este apartado de **Introducción y objetivos**, el capítulo **Entornos Virtualizados y Vulnerabilidades** explica el concepto de entorno virtualizado, sus componentes, servicios y ventajas de su uso, así como los vectores de ataque a los entornos virtualizados y los desafíos planteados por la virtualización.

A continuación, el capítulo **Vulnerabilidades del hipervisor** hace un repaso de las vulnerabilidades asociadas con la explotación de fallos relacionados con las funcionalidades del hipervisor, se incluyen algunos ejemplos de ataques y se finaliza con las herramientas utilizadas para la realización del trabajo, con el fin de conocerlas mejor.

Después, tras todo este desarrollo teórico, el capítulo **Aspectos Prácticos** expone todos los detalles relacionados con el desarrollo del proyecto, con sus diferentes etapas.

Para finalizar, se encuentran los capítulos de **Conclusiones**, donde se hará una valoración a partir de los resultados del proyecto, y **Aplicaciones y Líneas Futuras**, donde se incluirán tanto ejemplos de uso del proyecto, como las posibles continuaciones que puede tener el proyecto.

Capítulo 2.
Los Entornos Virtualizados.

Ya en 1974, Popek y Goldberg [1] definieron una máquina virtual como "un duplicado eficiente y aislado de una máquina real". También dieron tres condiciones necesarias para alcanzar dicho objetivo:

- Eficiencia: la virtualización no inducirá una disminución significativa en el rendimiento. Por lo tanto, la mayor cantidad de instrucciones no debe requerir una intervención del administrador de máquina virtual (VMM).
- Control de recursos: el VMM debe tener un control completo sobre los recursos virtualizados.
- Equivalencia: un programa debe comportarse de la misma manera en una máquina virtual y en su contrapartida física.

Pero ahora esta definición se nos queda demasiado pequeña. Por ello, utilizaremos la definición dada por Ramos [2]: "La virtualización se define como un marco que divide los recursos del dispositivo del entorno de la ejecución, permitiendo la pluralidad del entorno mediante el uso de una o más técnicas como el tiempo compartido, la emulación y el particionamiento". En sí, la virtualización no es otra cosa que la emulación del software y/o hardware que ejecuta otro software. El ambiente que se imita se llama Máquina Virtual (VM). Con la virtualización, uno o más sistemas operativos, junto con sus aplicaciones, se ejecutan en la parte superior del hardware virtual.

Cada instancia de un sistema operativo y sus aplicaciones se encapsula en una máquina virtual aislada del resto, llamada "máquina virtual invitada". Al estar encapsulado, las máquinas virtuales son totalmente portátiles, se utilizan y se administran independientemente del resto de máquinas virtuales. Esto permite que los recursos de una máquina física, incluidos procesadores, memoria, almacenamiento y canales de E/S se compartan entre varias máquinas virtuales simultáneamente, a la vez que se preserva el aislamiento. El componente de software que crea, gestiona y supervisa las máquinas virtuales es el hipervisor, también denominado Virtual Machine Monitor (VMM), que además controla el flujo de instrucciones entre los sistemas operativos invitados y el hardware físico, como CPU, almacenamiento en disco, memorias y tarjetas de interfaz de red. Además, monitoriza y controla los recursos de la máquina física, asignando lo que se necesita para cada máquina virtual y asegurándose que las mismas no se interrumpan unas a otras.

2.1 Tipos de virtualización.

La virtualización ofrece muchas posibilidades y, según las necesidades se distinguen cuatro tipos de virtualización:

- Virtualización de procesos, que proporciona una interfaz entre una aplicación y el sistema subyacente, lo que permite crear una aplicación sin preocuparse por el sistema en el que se va a ejecutar.
- Virtualización de almacenamiento, que fusiona el almacenamiento físico de múltiples dispositivos de almacenamiento en red para que parezcan ser un único dispositivo.
- Virtualización de red, que combina los recursos informáticos en una red dividiendo el ancho de banda disponible en canales independientes que se pueden asignar a un servidor o dispositivo en tiempo real.
- Virtualización de servidor, que permite que se ejecuten simultáneamente varios sistemas operativos en una máquina física. Esta categoría es, de lejos, la aplicación más común de la tecnología hoy en día, y se considera el principal impulsor del mercado.

Cuando se utiliza el término "virtualización", es muy probable que la gente se refiera a la virtualización de servidores.

2.2 Virtualización de servidores.

Existen varias maneras de implementar la virtualización de servidores, ya sea interactuando con el hardware subyacente, el sistema operativo o a través de un hipervisor. En este trabajo nos vamos a centrar en este último.

Los hipervisores se clasifican dentro de dos grupos [3]:

- Tipo 1: los hipervisores de tipo 1 proporcionan las mismas interfaces hardware que las que proporciona el hardware de la máquina física. Esto permite que los sistemas operativos y aplicaciones que se ejecutan no necesitan modificarse para que la virtualización funcione, siempre y cuando los sistemas operativos y las aplicaciones sean compatibles con el hardware subyacente. Este tipo de hipervisor se ejecuta con el mayor nivel de privilegios. Xen[4] y VMWare ESX[5] son hipervisores de este tipo.
- Tipo 2: los hipervisores de tipo 2 proporcionan interfaces al sistema operativo huésped sobre el que está instalado el hipervisor para que el sistema operativo invitado pueda usar en lugar de las interfaces de hardware usuales. Este tipo de hipervisor se ejecuta como cualquier otro programa dentro del sistema operativo. QEMU[6] y VirtualBox[7] son hipervisores de este tipo.

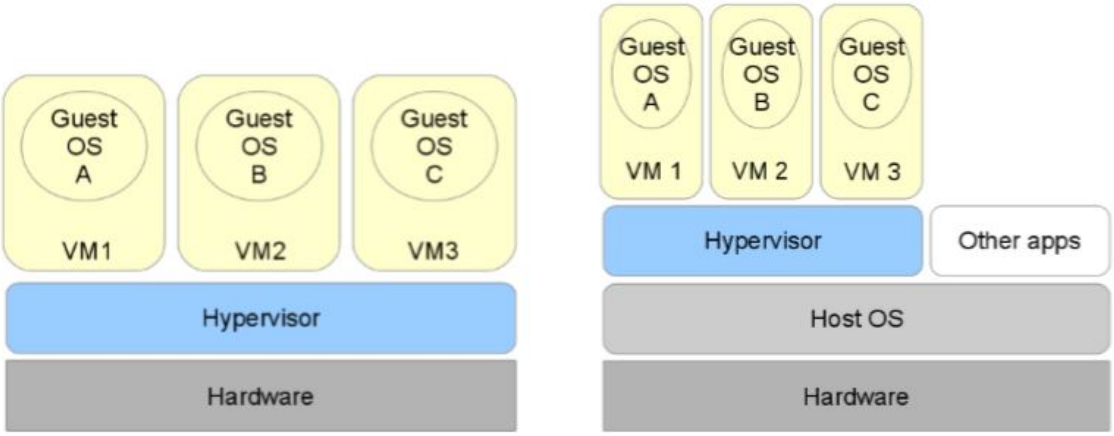

Figura 1: Hipervisores de tipo 1 y 2 respectivamente [8].

Además de la clasificación por tipo de hipervisor, se pueden distinguir tres técnicas principales de virtualización basada en el hipervisor:

- Virtualización total: en este caso, uno o más sistemas operativos invitados comparten recursos del hardware de la máquina física anfitriona. La presencia del hipervisor es transparente desde el punto de vista de los huéspedes.
- Paravirtualización: los núcleos de los sistemas operativos se modifican para hacerlos capaces de comunicarse con el hipervisor subyacente a través de las hypercalls. Las hypercalls pueden ser vistas como las equivalentes a las llamadas de sistema operativo no virtualizado.
- Emulación: al igual que en la virtualización total, la emulación permite que los sistemas operativos no modificados se ejecuten, pero los recursos vistos por el sistema operativo invitado son completamente simulados por el software. Esto permite ejecutar un sistema operativo compilado en una arquitectura diferente de la arquitectura del anfitrión. Esto resulta en un rendimiento menor que con las otras dos técnicas.

2.3 La computación en la nube.

Aunque hay muchas y diversas definiciones de la computación en la nube, el NIST (National Institute of Standards and Technology)[9] nos proporciona la definición más completa:

"La computación en la nube es un modelo para permitir el acceso a la red omnipresente y conveniente a un conjunto compartido de recursos computacionales configurables (por ejemplo, redes, servidores, almacenamiento, aplicaciones y servicios) que se pueden aprovisionar y liberar rápidamente con un esfuerzo mínimo de administración o interacción del proveedor de servicios".

Normalmente, es un modelo de pago por uso o cargo por uso. Este modelo de nube está compuesto de cinco características esenciales:

- Auto-servicio bajo demanda: Un usuario puede en todo momento decidir qué aplicaciones usar, y elegir entre las que son gratuitas o las de pago.
- Amplio acceso a la red: Los servicios están disponibles a través de la red, y se accede por medio de mecanismos estándar y desde plataformas heterogéneas, por ejemplo ordenadores, teléfonos móviles o tablets.
- Puesta en común de los recursos: los recursos proporcionados a los usuarios
- Elasticidad:
- Servicio medido:

Dentro de la definición del concepto de la computación en la nube se encuentra la definición de los tres modelos de servicio de la computación en la nube, los cuales están sobre dos capas más básicas, como se muestra en la Figura 2. Los tres servicios nombrados antes son: Infraestructura como servicio (IaaS), Plataforma como servicio (PaaS) y Software como servicio (SaaS).

- Infraestructura como servicio (IaaS): Este modelo de servicio es el nivel más bajo que se puede proporcionar a un cliente, el cual tiene un acceso controlado a la infraestructura virtual. Utilizando dicho acceso, el cliente puede instalar el sistema operativo y el software de aplicación, pero no tiene control sobre el hardware físico y tiene que gestionar los aspectos de seguridad.
- Plataforma como servicio (PaaS): En este modelo de servicio, el sistema operativo y todas las herramientas asociadas ya están instalados para el cliente, los cuales también son administrados por el proveedor de servicios en la nube. Los clientes tienen la libertad de instalar herramientas adicionales aunque el proveedor de servicios retiene el control de la infraestructura.
- Software como servicio (SaaS): Este modelo se centra en el nivel de aplicación y abstrae al usuario de los detalles de la infraestructura y la plataforma. Por lo general, a las aplicaciones se aprovisionan a través de interfaces de cliente ligero, como los navegadores o incluso aplicaciones móviles.

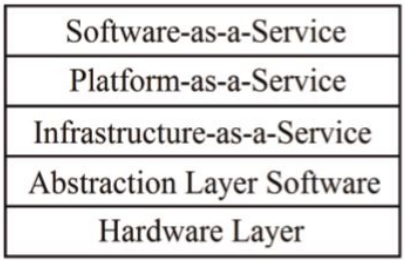

Figura 2: Capas de la Computación en la nube [10].

Como se puede apreciar en la Figura 2, en la base de la computación en la nube se encuentra la capa del hardware, que contiene los procesadores, la memoria y los

dispositivos físicos de almacenamiento. Y sobre ésta se encuentra la capa de abstracción de software del hipervisor.

Al final, la computación en la nube está orientada a ofrecer servicios, de lo que se obtiene el concepto de "cloud-as-a-service" o "la nube como servicio". Tras muchos análisis y debates, Linthicum[11] recoge y describe, dentro de once categorías principales, los diferentes tipos de servicios que son ofrecidos, los cuales se muestran en la Figura 3:

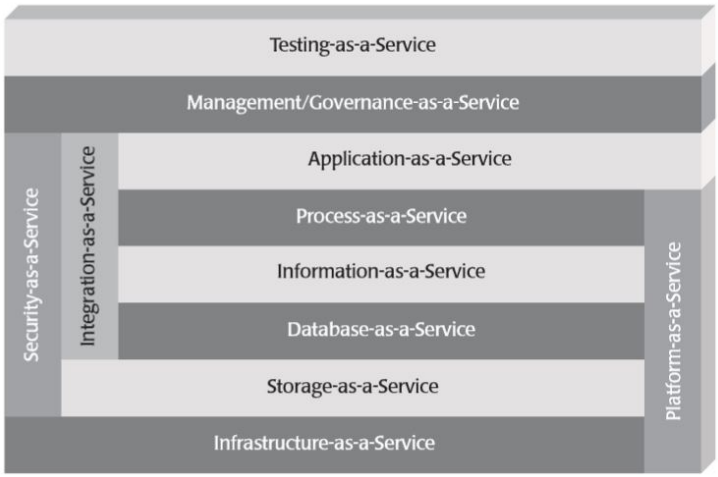

Figura 3: Componentes de la computación en la nube. [11]

1. Storage-as-a-service: es la capacidad de aprovechar el almacenamiento existente físicamente en un sitio remoto. Es el componente más primitivo de la computación en la nube y es aprovechado por la mayoría de los demás servicios ofrecidos.
2. Database-as-a-service: proporciona la capacidad de aprovechar los servicios de una base de datos remota, compartirla y tenerla funcionando como si fuese local.
3. Information-as-a-service: es la capacidad de consumir cualquier tipo de información alojada remotamente a través de una interfaz definida, como por ejemplo una API.
4. Process-as-a-service: es un recurso que puede enlazar varios recursos remotos juntos, como servicios y datos, alojados en el mismo recurso de computación en la nube, o de manera remota.
5. Application-as-a-service: es cualquier aplicación que se entrega a través de una plataforma web, como un navegador, por ejemplo.
6. Platform-as-a-service: es una plataforma completa, que incluye desarrollo de aplicaciones, interfaces, bases de datos, almacenamiento, etc., entregado a través de una plataforma remotamente alojada a los clientes.
7. Integration-as-a-service: este servicio incluye la mayoría de las características y funciones, pero se entregan al usuario como un servicio.
8. Security-as-a-service: es la capacidad de ofrecer servicios básicos de seguridad de manera remota a través de internet.

9. Management/governance-as-a-service: es cualquier servicio bajo demanda que ofrece la capacidad de administrar uno o más servicios en la nube. Normalmente estos servicios gestionan cosas simples como topología, utilización de recursos, administración en tiempo real, e incluso pueden aplicar políticas definidas sobre datos y servicios.
10. Testing-as-a-service: es la capacidad de probar sistemas usando software de prueba y servicios que se alojan remotamente. Estos servicios tienen la capacidad también de probar otras aplicaciones en la nube y sitios web.
11. Infrastructure-as-a-service: es la capacidad de acceder remotamente a los recursos informáticos. En esencia, se alquila un servidor físico para lo que necesita el cliente y, a efectos prácticos, es su centro de datos.

Estos servicios mencionados, son entregados de diferentes maneras, las cuales se concentran en cuatro modelos de presentación, los cuales son:

- Nube privada: en este tipo de nube, la infraestructura es propiedad, o está alquilada, por una sola organización, y el uso es exclusivo de dicha organización.
- Nube comunitaria: en este tipo de nube, tanto la propiedad, como el uso de la infraestructura es compartida por varias organizaciones, y soporta una comunidad específica de dichas organizaciones que tienen intereses compartidos como, por ejemplo, requisitos de seguridad.
- Nube pública: en este tipo de nube, la propiedad de la infraestructura es de una organización, que vende servicios en la nube al público en general, o a una empresa.
- Nube híbrida: aquí, la infraestructura de la nube es la combinación de dos o más nubes, ya sean privadas, comunitarias, o públicas, que siguen siendo nubes individuales, pero que están enlazadas entre sí por una tecnología propietaria o estandarizada que permite la portabilidad de datos y de aplicaciones.

2.4 Vectores de ataque.

La aparición de los entornos virtualizados ha contribuido a la llegada de nuevos escenarios de ataque. En primer lugar, un entorno virtualizado es el equivalente de una máquina física real, por lo que un atacante puede estar dispuesto a tomar el control de la máquina virtual usando las mismas herramientas que para atacar la máquina real. Por otro lado, con la adición de la capa de virtualización se puede dar al atacante nuevas formas de alterar su objetivo.

De hecho, como se muestra en la Figura 4, para un hipervisor de tipo 2 las vulnerabilidades en la capa de virtualización pueden permitir que los ataques se realicen desde una máquina virtual contra otra, contra el hipervisor o el sistema operativo del host. La imagen sería similar para un hipervisor de tipo 1, salvo que el sistema host no tiene sistema operativo bajo el hipervisor, por lo que el vector de ataque correspondiente también se elimina. Si hay alguna herramienta de administración del

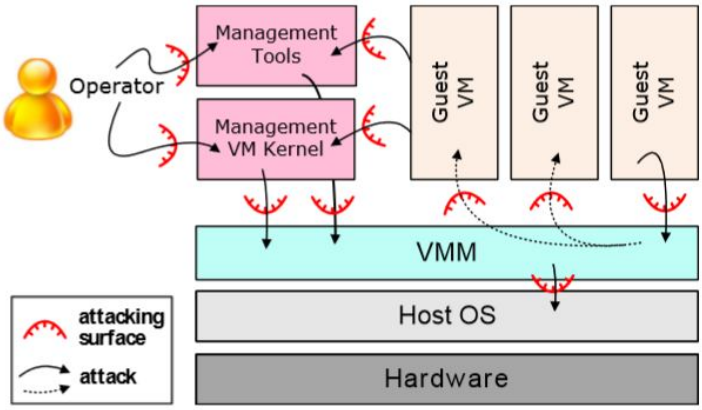

Figura 4: Vectores de ataque a entornos virtualizados. [12]

sistema, también se consideran como un vector de ataque potencial. Aunque no aparezcan en la imagen, otros vectores de ataque son los ataques remotos, los que vienen de fuera del entorno virtualizado, tanto al hipervisor, como a cualquier máquina virtual invitada, como al hardware subyacente. Garfinkel y Rosenblum [13] dan una lista de desafíos planteados por la virtualización:

- Escalabilidad: la virtualización permite la creación rápida y sencilla de nuevas máquinas virtuales. Por lo tanto, las políticas de seguridad de una red, como la configuración o las actualizaciones, tienen que ser lo suficientemente flexibles para manejar un rápido aumento en el número de máquinas.
- Transitoriedad: las máquinas virtuales a menudo se agregan o se eliminan de una red. Esto puede obstaculizar los intentos de estabilizar la red y sus elementos. Por ejemplo, si una red es infectada por un gusano, será más difícil determinar con precisión qué máquinas están infectadas y limpiarlas, cuando estas máquinas sólo existen durante breves periodos de tiempo en la red. Del mismo modo, las máquinas infectadas, o todavía vulnerables, pueden reaparecer incluso después de que se pensase que la infección se había eliminado.
- Ciclo de vida del software: la capacidad de restaurar una máquina virtual en un estado anterior plantea muchos problemas de seguridad. De hecho, las vulnerabilidades de la máquina que ya habían sido solucionadas (parches de programas, servicios desactivados, contraseñas antiguas…) pueden volver a aparecer. Además, restaurar una máquina virtual en un estado anterior permite que un atacante pueda volver a reproducir ataques anteriores, lo que hace obsoleto cualquier protocolo de seguridad basado en el estado de la máquina en un momento dado.
- Diversidad: en una organización donde las políticas de seguridad se basan en la homogeneidad de las máquinas, la virtualización aumenta el riesgo de tener muchas versiones del mismo sistema al mismo tiempo en la red.
- Movilidad: una máquina virtual se considera como cualquier otro archivo en un disco duro, que se pueden copiar a otro disco o a otra máquina. Esta característica

que se considera como un beneficio de la virtualización, también añade restricciones de seguridad, porque garantizar la seguridad de una máquina virtual se convierte en equivalente a garantizar la seguridad de cada máquina en la que ha estado.

- Identidad: los métodos habituales usados para identificar máquinas (como direcciones MAC) no son necesariamente eficientes con las máquinas virtuales. Además, la movilidad aumenta aún más las dificultades para autenticar al propietario de una máquina virtual.
- Duración de los datos: un hipervisor capar de guardar el estado de sus máquinas virtuales puede contrarrestar los esfuerzos realizados por un invitado para borrar datos sensibles de su memoria. De hecho, siempre puede haber una versión de copia de seguridad de la máquina virtual que contiene los datos.

Capítulo 3. Vulnerabilidades del hipervisor.

En este apartado se explican las vulnerabilidades que la capa del hipervisor presenta. Además, se exponen varios ejemplos de ataques al hipervisor, y las herramientas utilizadas para el desarrollo del Trabajo.

3.1 Vías de ataque al hipervisor.

No es de extrañar que la manera más obvia de atacar a un entorno Cloud es conseguir acceso al hipervisor, ya que es quien controla todas las máquinas virtuales que corren dentro de dicho entorno. Para la virtualización con hipervisores de tipo 1, no ha habido ataques conocidos a un hipervisor debido a su naturaleza, ya que va incrustado en el hardware [14]. De lo contrario, existen dos tipos de ataques al hipervisor: los ataques que se realizan a través del sistema operativo anfitrión y los ataques que se realizan a través de un sistema operativo invitado.

3.1.1 Ataques al hipervisor a través del sistema operativo anfitrión.

Estos ataques se utilizan explotando las vulnerabilidades del sistema operativo anfitrión en el que se ejecuta el hipervisor. Debido a que la virtualización con hipervisores de tipo 1 requiere un hardware específicamente configurado, la mayoría de las implementaciones de virtualización se realizan con la arquitectura alojada. Con las vulnerabilidades y los fallos de seguridad existentes en los sistemas operativos actuales, se pueden realizar ataques para obtener el control del sistema operativo anfitrión.

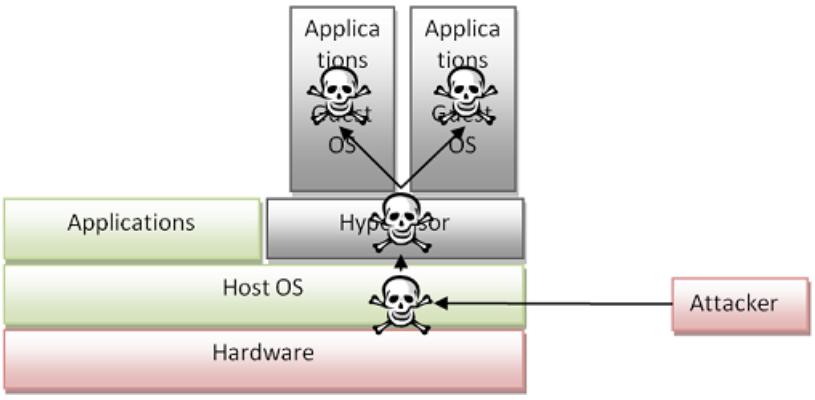

Figura 5: Propagación de los ataques desde el sistema operativo anfitrión

Dado que el hipervisor es simplemente una capa que se ejecuta en la parte superior del sistema operativo anfitrión, una vez que el atacante tiene control del sistema operativo anfitrión, el hipervisor está realmente comprometido. Por lo tanto, los privilegios de administrador del hipervisor permitirán al atacante realizar cualquier actividad maliciosa en cualquiera de las máquinas virtuales invitadas alojadas por el hipervisor. Esta propagación de ataques del sistema operativo alojado al hipervisor, y luego a las máquinas virtuales invitadas se muestra en la Figura 5.

3.1.2 Ataques al hipervisor a través del sistema operativo invitado.

El objetivo de este tipo de ataques es utilizar el sistema operativo de una máquina virtual invitada para obtener acceso no autorizado a otras máquinas virtuales o al hipervisor. Esto también se conoce como VM escape o ataques de jailbreak, en los cuales el atacante esencialmente "escapa" del confinamiento de la máquina virtual invitada en capas que de otro modo son desconocidas para la máquina virtual invitada. Este es el ataque más plausible en el hipervisor, ya que normalmente un atacante sólo puede comprometer una máquina virtual de manera remota en un entorno Cloud, ya que el sistema operativo subyacente (en el caso de que el hipervisor sea de tipo 2) es teóricamente invisible.

Sin embargo, dado que muchas máquinas virtuales comparten los mismos recursos físicos, si el atacante puede encontrar cómo los recursos virtuales de su máquina virtual se asignan a los recursos físicos, podrá realizar ataques directamente sobre los recursos físicos reales. Al modificar su memoria virtual de una manera que explota la forma en que los recursos físicos se asignan a cada máquina virtual invitada, el atacante puede afectar a todas las máquinas virtuales, al hipervisor y, potencialmente, a otros programas en dicha máquina virtual. La Figura 6 muestra la relación entre los recursos virtuales y los recursos físicos, y cómo el atacante puede atacar al hipervisor y a otras máquinas virtuales invitadas co-alojadas en el mismo entorno virtualizado.

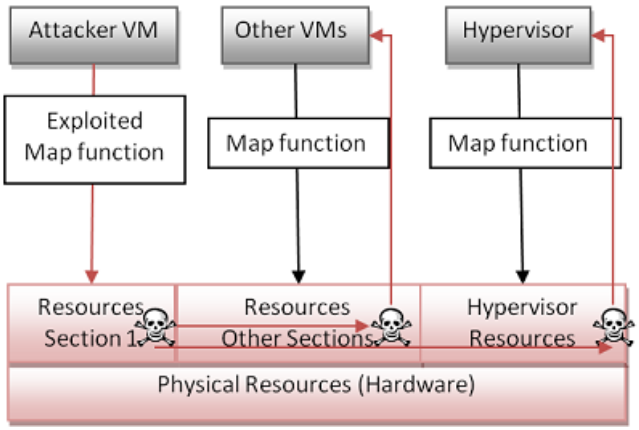

Figura 6: Ataque al hipervisor a través de una máquina virtual invitada.

3.2 Vulnerabilidades del hipervisor.

Una explotación exitosa de una vulnerabilidad conduce a un ataque que puede obstaculizar la confidencialidad, integridad o disponibilidad del hipervisor o una de sus máquinas virtuales invitadas. Cada informe CVE indica explícitamente el tipo de violación de seguridad que puede conducir a una combinación de las tres propiedades indicadas anteriormente. Aproximadamente el 50% de las vulnerabilidades reportadas hasta el momento pueden conducir a brechas de seguridad en los tres frentes. El segundo factor más común de explotación es provocar una indisponibilidad de los hipervisores (Denial of Service).

Diego Pérez-Botero [15], para una mejor comprensión de las diferentes vulnerabilidades, considera once funcionalidades que un hipervisor proporciona y asigna vulnerabilidades a éstas:

1. CPUs Virtuales.
2. Multiprocesamiento simétrico (SMP).
3. Unidad de gestión de la memoria blanda (soft MMU).
4. Mecanismos de interrupción y temporización.
5. E/S y conexión de red: E/S y redes.
6. E/S paravirtualizada.
7. Salidas de VM.
8. Hypercalls.
9. Administración de VM.
10. Software de gestión remoto
11. Complementos del hipervisor

Las primeras seis categorías presentan las vulnerabilidades referentes a la infraestructura de hardware virtualizada, que las máquinas virtuales requieren para funcionar de manera adecuada. Las categorías siete y ocho muestran las vulnerabilidades referentes a los mecanismos de delegación de operaciones sensibles al hipervisor de las máquinas virtuales. La categoría nueve se refiere a las vulnerabilidades de las instalaciones que el hipervisor necesita para administrar el estado de la propia máquina virtual. La categoría diez habla sobre las vulnerabilidades del software de gestión remota, mientras que la última categoría explica los problemas de los módulos adicionales en el hipervisor.

3.2.1 CPUs Virtuales.

El hipervisor asigna a cada máquina virtual invitada conjunto de CPUs virtuales (vCPU). El estado de cada una de estas vCPUs se guarda y se carga desde el área de invitado-estado de la Estructura de Control de Máquina Virtual (VMCS) de su respectiva máquina virtual. El hipervisor debe manejar los estados de registro de manera adecuada y programar las tareas de las vCPUs en las CPUs físicas mientras realiza las traducciones necesarias en ambos sentidos, ya que las vCPUs deben reflejar las acciones de una CPU física para cada una de las instrucciones de lenguaje máquina.

El CVE-2010-4525 es un ejemplo de vulnerabilidad en este sentido, ya que describe que se produjo una divulgación de contenido de memoria del hipervisor a través de los registros de las vCPUs debido a una inicialización incompleta de las estructuras de datos de las vCPUs, donde uno de los campos de relleno no se eliminó por completo. Dado que la memoria de la estructura de datos se asigna en el espacio del kernel, el campo de relleno podría terminar con información de las estructuras de datos utilizadas previamente por el hipervisor.

3.2.2 Multiprocesamiento Simétrico (SMP):

Los hipervisores pueden alojar máquinas virtuales invitadas que tienen la capacidad de utilizar SMP, lo que conlleva la posibilidad de que dos o más vCPUs pertenecientes a una única máquina virtual invitada estén programadas en paralelo a los núcleos físicos de la CPU. Este modo de operación añade mucha complejidad a la gestión del estado de las máquinas virtuales invitadas, y requiere precauciones adicionales en el momento de decidir el nivel de privilegio actual de la vCPU. Las vulnerabilidades en esta funcionalidad surgen de la hipótesis de que el código del hipervisor sólo es válido para procesos de subproceso único.

Por ejemplo, CVE-2010-0419 hace referencia a un error que permitía a los procesos de nivel de privilegio más bajo, ejecutar instrucciones privilegiadas cuando el SMP estaba habilitado. Para ello, se invocaba una instrucción de E/S legítima en un hilo, e intentaba reemplazarla por una privilegiada en otro hilo después de comprobar la validez de la primera, pero antes de que se ejecutase.

3.2.3 Unidad de gestión de la memoria blanda (soft MMU):

Las máquinas virtuales invitadas no pueden tener acceso directo a la MMU, ya que les permitiría tener acceso a la memoria perteneciente al hipervisor y la de otras máquinas virtuales. Bajo la ausencia de una MMU de hardware con reconocimiento de virtualización, como el EPT (Extended Page Table), el hipervisor ejecuta una soft MMU para mantener una tabla de páginas para cada máquina virtual. Cada modificación de mapeo es interceptada por la MMU blanda para ajustar las tablas de las páginas en consecuencia.

Las vulnerabilidades en la implementación de soft MMU blanda son peligrosas, ya que pueden conducir a la divulgación de datos en espacios de direcciones arbitrarios, como un segmento de memoria de otra máquina virtual invitada o del hipervisor. Por ejemplo, CVE-2010-0298 hace referencia a una vulnerabilidad del emulador KVM, que utiliza el nivel de privilegio más alto cuando accede a la memoria de una máquina virtual invitada. Esto permitía que una aplicación no privilegiada engañara a KVM para que ejecutase una instrucción maliciosa que modifica la memoria de espacio de kernel de la misma máquina virtual utilizando una instrucción de E/S a la gestión de la memoria.

3.2.4 Mecanismos de interrupción y temporización.

Un hipervisor debe emular los mecanismos de interrupción y temporización que son proporcionadas por la placa base a una máquina física. Estos mecanismos incluyen el temporizador de intervalo programable (PIT), el controlador de interrupción programable avanzada (APIC) y los mecanismos de petición de interrupción (IRQ).

Un ejemplo de vulnerabilidad en este caso es el CVE-2010-0309, en el que se describe cómo una máquina virtual provocaba una denegación de servicio completo al sistema operativo anfitrión debido a la falta de validación de los datos contenidos en las estructuras de datos relacionadas con el PIT.

3.2.5 E/S y conexión de red:

El hipervisor también emula las E/S y las redes. En Xen y KVM, la emulación de dispositivos es posible a través de la división del trabajo, al tener dos tipos de controladores de dispositivo. Los controladores front-end residen dentro de las máquinas virtuales invitadas y se ejecutan en el nivel de privilegios más alto, lo que proporciona el nivel de abstracción que espera el sistema operativo invitado. Sin embargo, dichos controladores tienen acceso al hardware físico, ya que el hipervisor debe mediar los accesos de los usuarios a los recursos compartidos. Por lo tanto, en el hipervisor residen unos controladores back-end que se comunican con los front-end para realizar las operaciones solicitadas. A su vez, los controladores de back-end hacen cumplir las políticas de acceso y multiplexan los dispositivos reales.

La emulación de dispositivos se implementa generalmente en lenguajes de nivel superior (por ejemplo, C y C ++), y los ataques más elaborados están habilitados por la expresividad de lenguajes de nivel superior como C. Por ejemplo, CVE-2011-1751 describe un error que se utilizó para desarrollar el ataque Virtunoide [16]. QEMU intentó desconectar en caliente cualquier dispositivo que los programadores deseaban. La falta de limpieza de estado por parte de algunos dispositivos virtuales resultó en oportunidades de uso después de su liberación, donde las estructuras de datos que se estaban usando por un dispositivo virtual desconectado seguían en la memoria, y podían se secuestradas mediante código ejecutable por un atacante.

3.2.6 E/S Paravirtualizada.

Las máquinas virtuales invitadas paravirtualizadas ejecutan kernels invitados modificados que son virtualizados y usan APIs de hypercalls especiales para interactuar directamente con el hipervisor. La paravirtuailzación de las operaciones de E/S disminuye el número de transiciones entre la máquina virtual invitada y el hipervisor, lo que el rendimiento se ve afectado muy positivamente. Para ello, se requiere unos controladores front-end y back-end especiales, que no son necesariamente desarrollados por el mismo responsable de la emulación del dispositivo.

Las vulnerabilidades asociadas a las E/S paravirtualizadas son muy parecidas a las asociadas a las E/S emuladas, ambas se basan en las interacciones entre los controladores

front-end y back-end y entre los controladores back-end y el mundo exterior. Por ejemplo, el CVE-2008-1943 describe una vulnerabilidad en Xen que permitió a los controladores front-end paravirtualizados causar denegación del servicio y posiblemente ejecutar código arbitrario con privilegios.

3.2.7 VM Exits.

Las VM Exits son el mecanismo utilizado por el hipervisor para interceptar y llevar a cabo las operaciones invocadas por las máquinas virtuales invitadas que requieren de privilegios de la raíz de Virtual Machine eXtensions (VMX). Dichas interfaces de máquina virtual al hipervisor son dependientes de la arquitectura (por ejemplo, el código es diferente para las arquitecturas x86 que para las arquitecturas AMD64), y normalmente se implementan en lenguajes de programación de bajo nivel, como lenguaje ensamblador.

El hecho de que el código de manejo de VM exits no posea estructuras de datos muy ricas significa que las vulnerabilidades apenas tienen efectos que puedan explotarse, a excepción de la denegación de servicio de la máquina virtual huésped o de las máquinas virtuales invitadas. Por ejemplo, todos los campos de la VMCS tienen una codificación de campo de 32 bits única, que elimina las vulnerabilidades comunes que surgen de la entrada de tamaño variable, como los flujos del buffer.

Según el CVE-2010-2938, solicitar un vaciado completo de la VMCS de una máquina virtual invitada causaría que todo el sistema anfitrión se bloqueara al ejecutar Xen en una CPU sin la funcionalidad de la Tabla de Páginas Extendida (EPT). La razón de esto era que Xen trataría de acceder a los campos de VMCS relacionados con la EPT sin verificar primero el soporte de hardware para esos campos, permitiendo que las aplicaciones de la máquina virtual invitada privilegiadas activaran un ataque completo de denegación de servicio en ciertos anfitriones en cualquier momento.

3.2.8 Hypercalls.

Las hypercalls son las análogas a las llamadas del sistema de un sistema operativo. Mientras que las VM exits son específicas de la arquitectura, las hypercalls son específicas del hipervisor (por ejemplo Xen, KVM, etc.) y proporcionan una interfaz a través de la cual las máquinas virtuales invitadas pueden solicitar acciones privilegiadas del hipervisor, como consultar la actividad de la CPU, administrar las particiones del disco duro, y crear interrupciones virtuales.

Las vulnerabilidades de las hypercalls pueden presentar a un atacante, que controla una máquina virtual invitada, una forma de obtener privilegios escalonados sobre los recursos del sistema anfitrión. Como se presenta en el CVE-2009-3290, KVM permitía a las máquinas virtuales invitadas no privilegiadas emitir hypercalls a la MMU. Puesto que las estructuras de comando de la MMU deben ser pasadas como un argumento a esas hypercalls por su dirección física, tienen sentido solo cuando son emitidas por un proceso del kernel. Al no tener acceso al espacio de direcciones físicas, los llamantes podían pasar

direcciones aleatorias como argumentos, que bloqueaban la máquina virtual invitada o, en el peor de los casos, leían o escribían en segmentos de memoria del espacio del kernel de la máquina anfitriona.

3.2.9 Administración de máquinas virtuales (configurar, iniciar, pausar y detener VM).

Las funcionalidades de administración de las máquinas virtuales constituyen el conjunto de operaciones administrativas básicas que debe soportar un hipervisor. La configuración de las máquinas virtuales invitadas se expresa en términos de sus dispositivos virtuales asignados. El hipervisor debe iniciar, pausar y detener las máquinas virtuales que sean fieles a las configuraciones declaradas por el proveedor de la nube. Las imágenes del kernel deben ser descomprimidas en la memoria e interceptadas por el dominio de administración al iniciar una máquina virtual.

Por ejemplo, el CVE-2007-4993 indica que el arranque de Xen para las imágenes paravirtualizadas utilizaba instrucciones exec()[1] de Python para procesar el archivo de configuración del kernel, dando lugar a la posibilidad de ejecutar código Python arbitrario dentro del kernel. Con ello, el atacante podría engañar al kernel para que emita un comando que desencadenaría en la destrucción de otra máquina virtual co-alojada.

3.2.10 Software de gestión remoto.

Software de gestión remota: estos programas suelen ser aplicaciones web que se ejecutan como un proceso en segundo plano, y no son esenciales para la correcta ejecución del entorno virtualizado. Su propósito es generalmente facilitar la administración del hipervisor a través de interfaces web y consolas virtuales orientadas a la red.

Las vulnerabilidades en estas aplicaciones se pueden aprovechar desde cualquier sitio y pueden conducir a un control total sobre el entorno virtualizado. El CVE-2008-3253 describe un ataque Cross-Site Scripting[2] en una consola de administración remota, que expuso todas las acciones de administración de una máquina virtual de Xen a un atacante remoto tras robar las cookies de autenticación de la víctima.

3.2.11 Complementos del hipervisor.

Ciertos tipos de hipervisores, como Xen y KVM tienen diseños modulares que permiten extensiones a sus funciones básicas. Los complementos de hipervisor aumentan la probabilidad de presencia de vulnerabilidades, ya que el código del hipervisor es mayor.

---

[1] Instrucción que se utiliza para ejecutar un programa externo al código en el que se ejecuta.

[2] Es un tipo de agujero de seguridad típico de las aplicaciones web, que permite a una tercera persona inyectar en páginas web visitadas por el usuario código Javascript o en otro lenguaje similar, evitando medidas de control como la "Política del mismo origen".

El CVE-2008-3687 describe una oportunidad de desbordamiento en uno de los módulos de seguridad opcionales de Xen, que da como resultado un escape de un dominio no privilegiado directamente al hipervisor.

3.3 Ataques al hipervisor.

Después de repasar la lista de desafíos a los que se enfrenta el hipervisor, se va a pasar a explicar algunos ejemplos típicos de ataques propuestos por Studnia [8] dirigidos a sistemas virtualizados o usando propiedades de virtualización para corromper una máquina.

3.3.1 Detección de un entorno virtualizado.

Aunque se supone que la virtualización proporciona al sistema operativo invitado un duplicado idéntico al de un sistema real, esto no es del todo cierto. Esto no es completamente cierto, y es posible detectar si hay un hipervisor ejecutándose debajo del sistema operativo. Esto es útil tanto para atacantes como para defensores. Por un lado, un atacante puede comprobar si el sistema al que está atacando está virtualizado y actúa en consecuencia. Por otro lado, un usuario dispuesto a comprobar la integridad de su máquina puede comprobar la presencia de un hipervisor instalado contra su voluntad.

Primero, un hipervisor se puede detectar comprobando el tiempo de ejecución de algunas instrucciones porque el procesamiento de dichas instrucciones requiere más tiempo si lo procesa un hipervisor que si lo procesa un sistema no virtualizado. Sin embargo, este método sólo funciona si las mediciones se realizaron previamente sobre un sistema operativo idéntico en un sistema real físico.

Otro método de detección de un entorno virtualizado basado en las mediciones del tiempo necesario para acceder a los Buzones de Traducción (TLB). Una vez que los buffers se han llenado con algunos datos conocidos, se llama a la instrucción CPUID que activa la limpieza de al menos una parte de la TLB si el hipervisor la recibió. Entonces, comparando los tiempos de acceso al TLB medido previamente y después de llamar a la CPUID, se puede establecer la presencia de un hipervisor. Una ventaja de este enfoque es que no requiere mediciones previas.

3.3.2 Identificación del hipervisor.

Ferrie [17] describe los procesos que utilizó para identificar con precisión cuál de los seis hipervisores (VMWare, VirtualPC, Parallels, Bosh, Hydra o QEMU) fue utilizado. Para ello, utilizó instrucciones específicas que no son utilizadas por algunos hipervisores de la misma manera que en un sistema real. Esto puede llevar a excepciones específicas de cada hipervisor o, por el contrario, a la ausencia de excepción. Entre los ejemplos conocidos de estos métodos, se puede citar RedPill [18] y ScoopyDoo [19].

Una vez que un hipervisor se identifica correctamente, un atacante puede adaptar su escenario de ataque a las vulnerabilidades conocidas como características de ese hipervisor. Dichas técnicas también permiten la creación de virus dirigidos sólo a los sistemas en los que se está ejecutando un tipo específico de hipervisor. Cabe señalar que algunos de estos métodos permiten detectar e identificar varios hipervisores populares. Sin embargo, en la mayoría de los casos se instalan legítimamente en una máquina, por lo que es comprensible que no intenten ocultarse.

3.3.3 Brecha en el aislamiento.

Uno de los principales objetivos de un hipervisor es el correcto aislamiento de las máquinas virtuales alojadas, lo que significa que una máquina virtual invitada no pueda tener más recursos de los que se ha concedido. Sin embargo, una mala configuración del hipervisor puede permitir que un atacante salga del aislamiento. Esto puede llevar a:

- Denegación del Servicio (DoS): una máquina virtual consigue utilizar toda la capacidad de computación del host físico, impidiendo que las otras máquinas virtuales se ejecuten correctamente.
- System halt: una instrucción específicamente diseñada hace que la máquina virtual o el hipervisor se bloquee.
- VM escape: esta categoría engloba todas las situaciones en las que un atacante obtiene acceso a la memoria ubicada fuera de la región asignada a la máquina virtual dañada. El atacante puede acceder a la memoria de otras máquinas virtuales, e incluso del sistema host, y puede leer, escribir o ejecutar su contenido. Esto puede llevar a una toma de control completa de otra máquina virtual, del hipervisor o del sistema operativo host, para un hipervisor de tipo 2.

En los ataques documentados de VM escape dirigidos a Xen, VMWare y KVM se describe cómo un atacante fue capaz de ejecutar código personalizado con los privilegios más altos, por lo que se hizo cargo de todo el sistema. Este tipo de ataques explota algunas vulnerabilidades en el código fuente o en el diseño del hipervisor. Ormandy [20] describe cómo se envían secuencias aleatorias de código mediante la técnica del fuzzing a diferentes hipervisores. Normalmente éste es el primer paso para descubrir nuevas vulnerabilidades explotables.

El fuzzing es una técnica de pruebas de software, normalmente automatizado o semiautomatizado, que implica proporcionar datos inválidos, inesperados o aleatorios a las entradas de un programa de ordenador. Entonces se monitorizan las excepciones tales como caídas, aserciones de código erróneas, o para encontrar filtraciones de memoria potenciales.

3.3.4 Rootkits basados en máquinas virtuales.

Los ataques descritos anteriormente se basaban en algunos rasgos característicos o fallos de diseño de los hipervisores, pero los VMBRs (Virtual Machine Based Rootkits)

son diferentes. En este tipo de ataques, el atacante utiliza algunas características de la virtualización asistida por hardware de procesadores x86 para instalar un hipervisor debajo del sistema operativo objetivo, poniéndolo en una máquina virtual.

Como el atacante controla el hipervisor puede monitorizar todo el sistema operativo virtualizado. Algunas implementaciones de VMBR pueden virtualizar el sistema operativo al vuelo, como BluePill [21], que no necesita instalación, y otras pueden resistir el reinicio de la máquina, como SubVirt [22], por ejemplo.

Por último, Rutkowska [23] y su equipo fueron capaces de instalar BluePill bajo Xen después de subvertirlo bajo una de sus máquinas virtuales, lo que quiere decir, pusieron el propio hipervisor en una máquina virtual (virtualización anidada).

3.4 Herramientas

En este apartado se va a realizar una breve introducción a las herramientas utilizadas en el desarrollo del proyecto, para conocerlas antes de explicar su uso en el mismo. Para utilizar las herramientas se ha decidido instalar la distribución KaliLinux, que reúne dentro de un único sistema operativo una gran cantidad de herramientas para la realización de test de penetración.

3.4.1 Microsoft Visual Studio [24]

Microsoft Visual Studio es un entorno de desarrollo integrado (IDE) para sistemas operativos Windows. Soporta múltiples lenguajes de programación, tales como C++, C#, Visual Basic .NET, Java, Python Ruby y PHP, al igual que entornos de desarrollo web, como ASP.NET MVC, Django, etc., a lo cual hay que sumarle las nuevas capacidades online bajo Windows Azure en forma del editor Monaco.

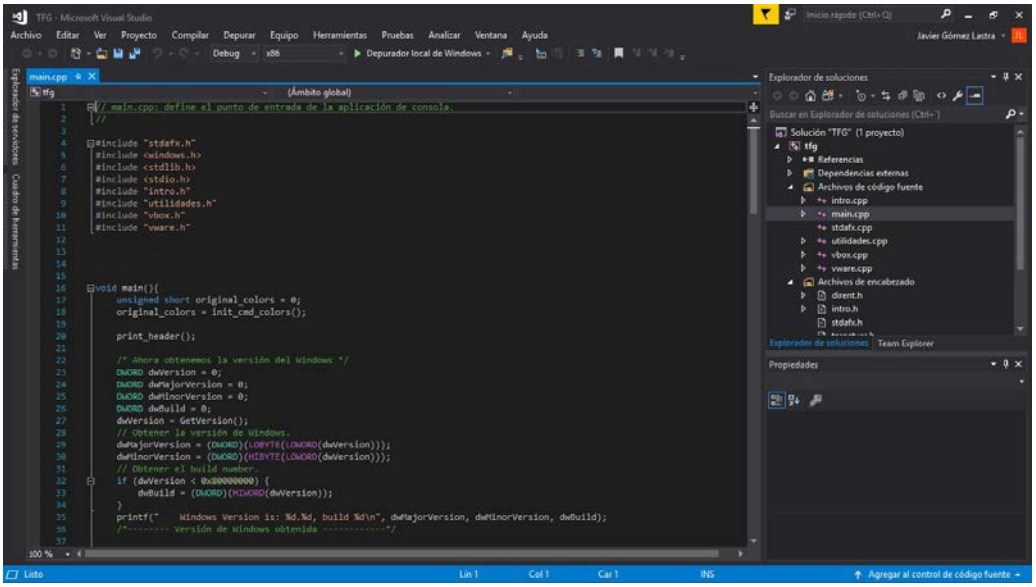

Figura 7: Vista de desarrollo de Microsoft Visual Studio.

Para la realización de este proyecto se ha utilizado la versión Visual Studio Community 2017 para la programación del malware desarrollado y utilizado. Este malware está desarrollado en C++, para una mayor flexibilidad a la hora de desarrollarlo, y compilado en este IDE.

3.4.2 Kali Linux [25]

Kali Linux es una distribución basada en Debian GNU/Linux diseñada principalmente para la auditoría y seguridad informática en general. Se trata de un proyecto de código abierto que es mantenido y financiado por Offensive Security [26]. Mati Aharoni y Devon Kearns, ambos pertenecientes a Offensive Security, desarrollaron la distribución a partir de la reescritura de BackTrack [27], que se podría definir como la antecesora de Kali Linux.

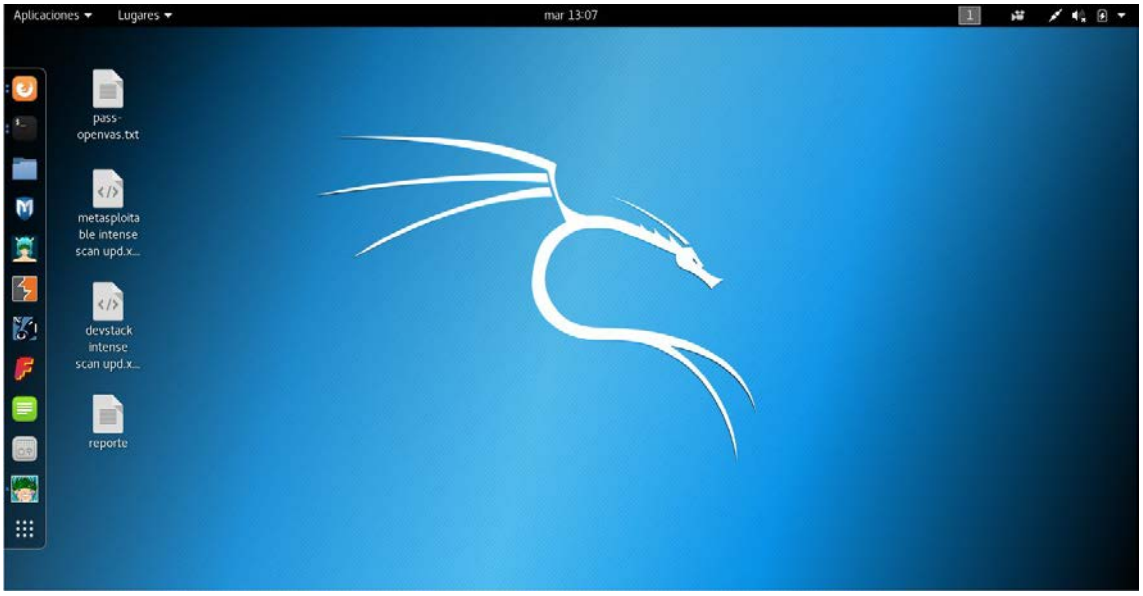

Figura 8: Escritorio de Kali Linux

Kali Linux trae preinstalados más de 600 programas, incluyendo Nmap [28](un escáner de puertos, más adelante explicado), Wireshark [29] (un sniffer) y John de Ripper [30] (un crackeador de passwords). Kali puede ser usado desde un Live CD, live-usb y también puede ser instalado como sistema operativo principal.

Kali está desarrollado en un entorno seguro. El equipo de Kali está compuesto por un grupo pequeño de personas de confianza, son quienes tienen permitido modificar paquetes e interactuar con los repositorios oficiales. Todos los paquetes de Kali están firmados por cada desarrollador que lo compiló y publicó. A su vez, los encargados de mantener los repositorios también firman posteriormente los paquetes. Kali se distribuye en imágenes ISO compiladas para diferentes arquitecturas (32/64 bits y ARM).

3.4.3 Metasploit Framework [31]

Metasploit es un proyecto open source de seguridad informática, que proporciona información acerca de vulnerabilidades de seguridad, y ayuda en la realización de test de penetración (Pentesting) y el desarrollo de firmas para sistemas de detección de intrusos. Su subproyecto más conocido es el Metasploit Framework, el cual permite la automatización de la explotación de vulnerabilidades en sistemas operativos, aplicaciones y redes. A través de este Framework es posible desarrollar exploits para explotar bugs, conocidos o no, en cualquier sistema informático.

H. D. Moore fue el desarrollador de Metasploit, en 2003. El código fue escrito originalmente en el lenguaje de programación scripting PERL, pero más adelante el software completo de Metasploit fue reescrito en Ruby, uno de los lenguajes de programación más utilizado por los hackers. El 21 de octubre de 2009, el Proyecto Metasploit anunció que había sido adquirido por Rapid7 [32], una empresa de seguridad que ofrece soluciones unificadas de gestión de vulnerabilidades.

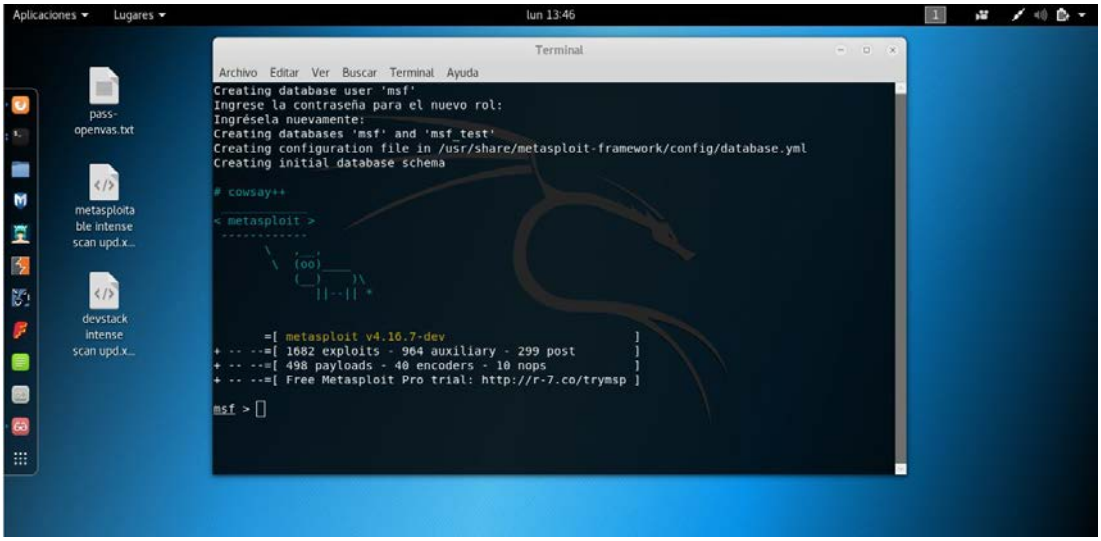

Figura 9: CLI de Metasploit.

La interfaz de administración por defecto de Metasploit es el CLI (Command Line Interface), lo que hace que la curva de aprendizaje sea un poco elevada. Pese a su complejidad, desde esta interfaz es posible acceder a todas las funcionalidades de la herramienta, pudiendo aprovechar todo su potencial.

Por otro lado, un grupo de desarrolladores independientes ha creado Armitage, una interfaz GUI (Graphic User Interface) para la administración de Metasploit con un aspecto muy elegante e intuitivo. En ella, es posible visualizar gráficamente los objetivos, además, el mismo programa recomienda qué exploits usar y expone las opciones además del framework. Aparte de eso, el propio Armitage también permite iniciar un análisis con Nmap, e incluso usar el módulo de Fuerza Bruta para sacar el usuario y la contraseña, entre otras muchas opciones más.

El objetivo principal de Armitage es hacer Metasploit útil para aquellas personas del mundo de la seguridad que saben del hacking, pero no del uso de Metasploit a fondo.

Armitage organiza las capacidades de Metasploit alrededor del proceso de hacking. Hay características para el descubrimiento, acceso, post-explotación y maniobra.

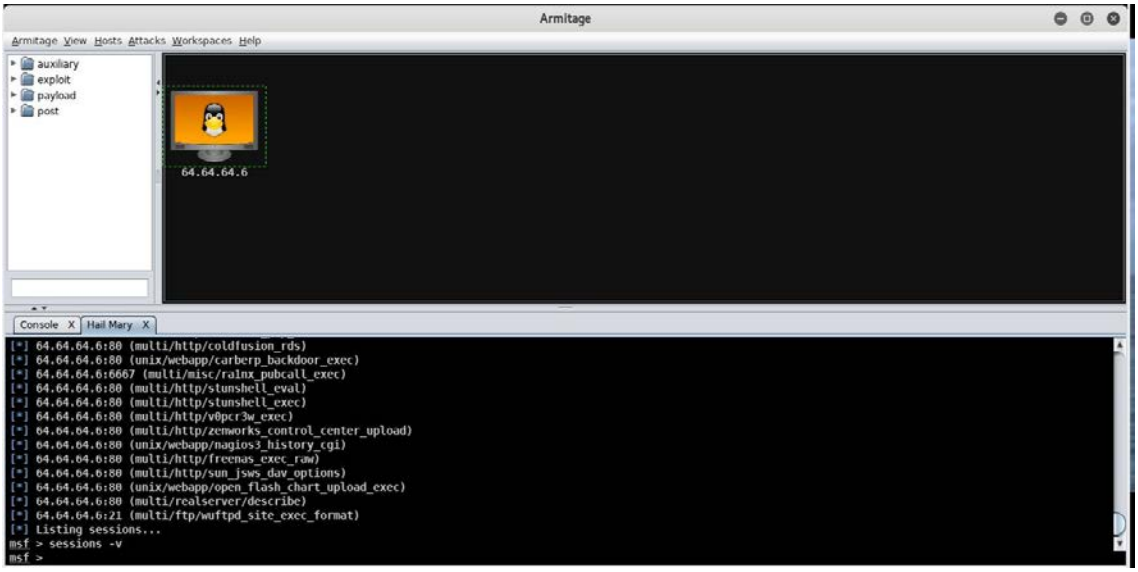

Figura 10: Interfaz GUI de Armitage.

Para el descubrimiento de objetivos, Armitage expone varias de las capacidades de gestión de hosts de Metasploit. Puede importar objetivos o lanzar escaneos para llenar una base de objetivos. Una vez importado o escaneado, el objetivo se mostrará como en la figura X [Imagen de GUI de Armitage].

Armitage es una gran ayuda en la explotación remota, ya que posee características para recomendar exploits automáticamente, o incluso ejecutar comprobaciones que permiten conocer qué exploits funcionarán correctamente. Además, si estas opciones fallan, se dispone del método "Hail Mary", con el que se probarán todos los exploits frente al objetivo automáticamente.

Una vez se consigue entrar, Armitage provee muchas herramientas post-exploración basadas en las capacidades del agente meterpreter.

Con un solo click es posible escalar privilegios, volcar hashes de passwords a una base de datos local de credenciales, navegar por el sistema como si se tuviese acceso directo al mismo, lanzar consolas de comandos, etc.

Finalmente, Armitage ayuda en el proceso de creación de pivotes, una capacidad que le permite usar hosts comprometidos como una plataforma para atacar otros hosts y así seguir investigando la red objetivo. Armitage incluso expone el módulo SOCKS proxy de Metasploit, el cual permite que herramientas externas tomen ventajas de estos pivotes.

3.4.4 OpenStack [33]

OpenStack es un software de código abierto, utilizado para la creación de nubes, tanto públicas como privadas. OpenStack controla grandes grupos de recursos de computación, almacenamiento y redes en un centro de datos, administrados a través de un panel de

control a través de la API de OpenStack. OpenStack trabaja con las tecnologías empresariales y las de código abierto, lo que la hace ideal para infraestructuras heterogéneas. OpenStack es administrado por la OpenStack Foundation [34], una organización sin fines de lucro que supervisa tanto el desarrollo como la construcción de comunidades alrededor del proyecto. OpenStack pertenece a la categoría de Infraestructura como Servicio (IaaS), lo que significa que facilita que los usuarios agreguen rápidamente una nueva instancia, sobre la que pueden ejecutarse otros componentes de la nube. Normalmente, la infraestructura ejecuta una plataforma sobre la que un desarrollador puede crear aplicaciones de software que se entregan a los usuarios finales.

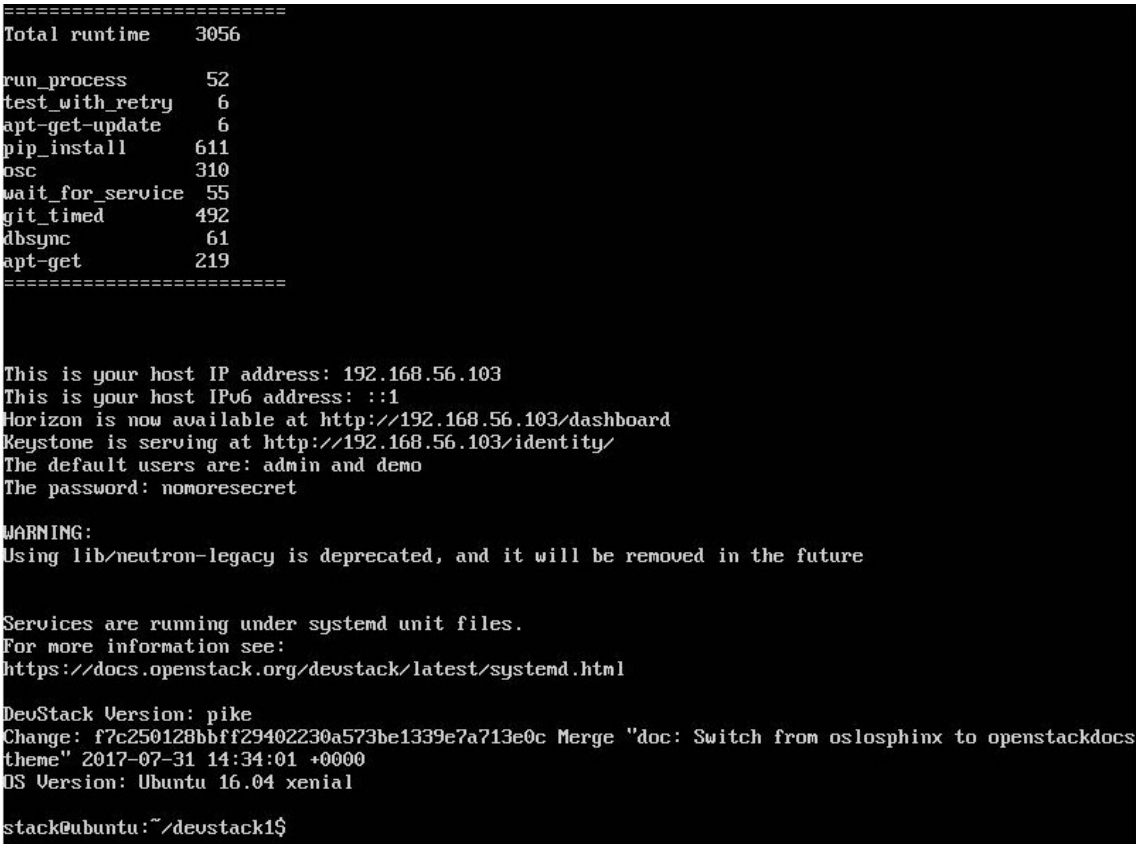

Figura 11: Instalación de OpenStack desde consola de comandos de DevStack.

OpenStack permite a los usuarios implementar máquinas virtuales y otras instancias que manejan diferentes tareas para administrar un entorno Cloud en el momento. Facilita el escalado horizontal, lo que significa que las tareas que se benefician de correr simultáneamente pueden servir fácilmente a más o menos usuarios sobre la marcha, simplemente aumentando el número de instancias.

OpenStack se compone de varias partes diferentes. Debido a su naturaleza abierta, cualquier persona puede agregar componentes adicionales a OpenStack para ayudarlo a satisfacer sus necesidades. Sin embargo, la comunidad de OpenStack ha identificado nueve componentes clave que forman parte del "núcleo" de OpenStack, que se distribuyen como parte de cualquier sistema OpenStack y que son oficialmente mantenidos por la comunidad OpenStack:

- Nova: es el motor principal de computación detrás de OpenStack. Se utiliza para desplegar y administrar grandes cantidades de máquinas virtuales y otras instancias para manejar tareas de computación.
- Swift: es un sistema de almacenamiento de objetos y archivos. En lugar de la idea tradicional de referirse a los archivos por su ubicación en una unidad de disco, los desarrolladores pueden referirse a un identificador único que se refiere al archivo o pieza de información y dejar que OpenStack decida dónde almacenar esta información, lo que facilita el escalado.
- Cinder: es un componente de almacenamiento de bloques, que es más análogo a la noción tradicional de que una computadora pueda acceder a ubicaciones específicas en una unidad de disco. Es importante en escenarios en los que la velocidad de acceso a los datos es la consideración más importante.
- Neutron: proporciona la capacidad de red para OpenStack. Ayuda a garantizar que cada uno de los componentes de un despliegue de OpenStack pueda comunicarse entre sí de una manera eficiente y rápida.
- Horizon: es la única interfaz gráfica para OpenStack, por lo que para los usuarios es el primer componente que realmente ven. Los desarrolladores pueden acceder a todos los componentes de OpenStack individualmente a través de una API (Interfaz de Programación de Aplicaciones), pero la interfaz proporciona a los administradores del sistema de un vistazo rápido lo que está sucediendo en la nube, y gestionarla en caso de ser necesario.
- Keystone: proporciona servicios de identidad para OpenStack. Es esencialmente una lista central de todos los usuarios de la nube de OpenStack, mapeada contra todos los servicios proporcionados por la nube. Proporciona múltiples medios de acceso, lo que significa que los desarrolladores pueden fácilmente asignar sus métodos de acceso de usuario existentes a Keystone.

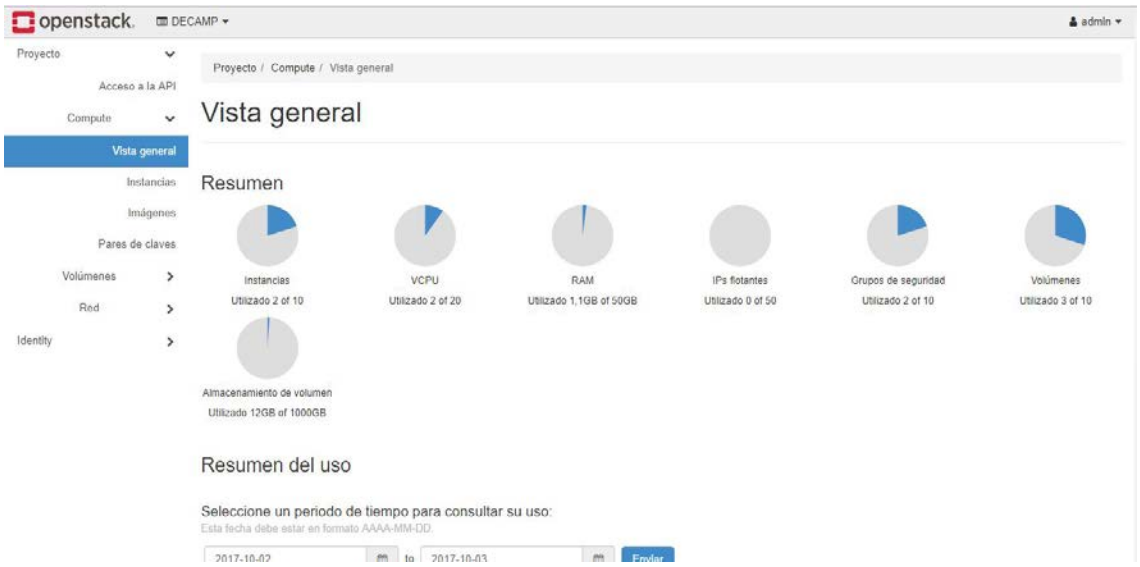

Figura 12: GUI de Horizon para OpenStack.

- Glance: proporciona servicios de imagen a OpenStack, lo que permite que estas imágenes se utilicen como plantillas al implementar instancias de máquinas virtuales.
- Ceilometer: ofrece servicios de telemetría, que permiten a la nube proporcionar servicios de facturación. También mantiene un conteo verificable del uso del sistema de cada usuario de cada uno de los diversos componentes.
- Heat: es el componente de orquestación de OpenStack, que permite a los desarrolladores almacenar los requisitos de una aplicación en la nube en un archivo que define qué recursos son necesarios para dicha aplicación.

3.4.5 DevStack [35]

DevStack es un script opcional para crear de manera rápida un entorno de desarrollo OpenStack. También se puede usar para demostrar el inicio y/o el funcionamiento de los servicios OpenStack y proporcionar ejemplos de cómo usar dichos servicios desde una línea de comandos. Los ejemplos de ejercicios se desarrollaron más allá de simples ejemplos y se hicieron útiles, como una verificación rápida para la instalación de OpenStack.

DevStack no es, y nunca ha sido pensado para ser un instalador general de OpenStack. Ha evolucionado para soportar un gran número de opciones de configuración, plataformas alternativas y servicios de soporte. Sin embargo, esa evolución ha crecido mucho más allá de lo que originalmente se pensaba y, desafortunadamente, muchas de las combinaciones de configuración rara vez, o nunca, son probadas.

La misión principal de DevStack es proporcionar y mantener las herramientas utilizadas para la instalación de los servicios centrales de OpenStack desde el origen (la rama master del repositorio de GIT oficial, o ramas específicas) adecuadas para el desarrollo y pruebas operativas. También muestra y documenta ejemplos de configuración y ejecución de servicios, así como el uso del cliente de la línea de comandos.

3.4.6 Nmap [28]

Nmap es un programa de código abierto, desarrollado originalmente por Gordon Lyon, que se utiliza para rastrear puertos. Generalmente se utiliza para evaluar la seguridad de sistemas informáticos, así como para descubrir servicios o servidores en una red informática. Para realizar esta tarea, lo que hace Nmap es enviar paquetes definidos a otros equipos para después, analizar las respuestas que recibe.

Este software posee varias funciones para sondear redes de computadores, incluyendo la detección de equipos, servicios y sistemas operativos. Estas funciones son extensibles mediante el uso de scripts para proveer servicios de detección avanzados, detección de vulnerabilidades, así como otro tipo de aplicaciones. Este software destaca principalmente por las características que posee para el descubrimiento de servidores, identificación de puertos abiertos, servicios ejecutándose en un determinado computador, y otros datos sobre el mismo, como características del hardware, sistema operativo o versiones de

software. Además, durante un escaneo es capaz de adaptarse a las condiciones de la red, incluyendo la latencia y la congestión de la misma.

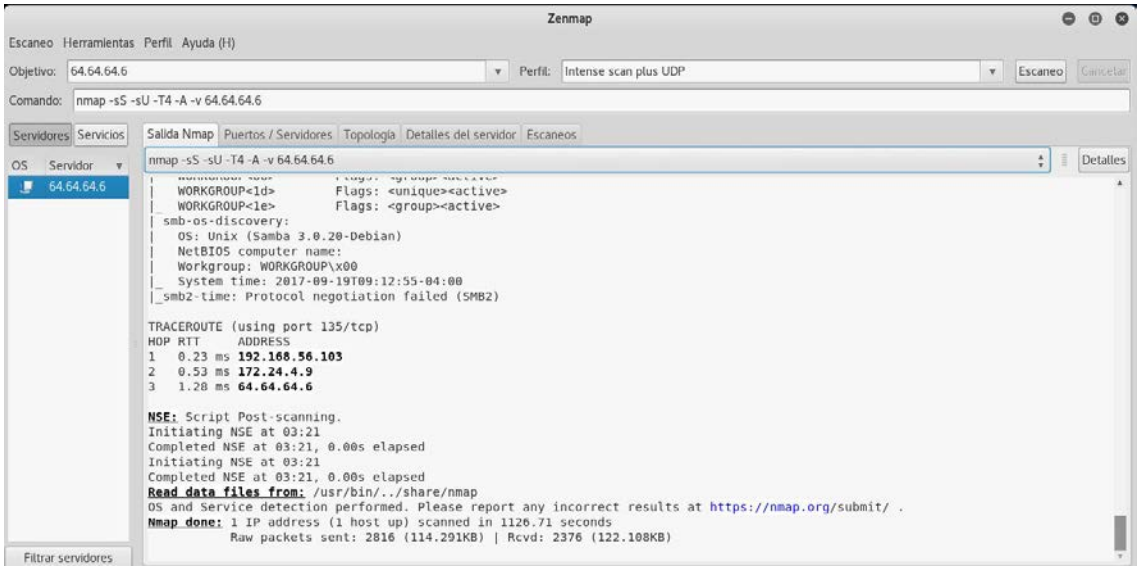

Figura 13: Captura de resultados de Zenmap.

En este proyecto se ha utilizado Zenmap [36]. Zenmap es la GUI oficial de Nmap. Se trata de una aplicación de código abierto que tiene como objetivo hacer que Nmap sea más fácil de usar para los principiantes, a la vez que proporciona funciones avanzadas para usuarios experimentados de Nmap. Los escaneos y exploraciones utilizadas de manera habitual se pueden guardar como perfiles para que sean más accesibles y fáciles de ejecutar repetidamente. Un creador de comandos permite la creación interactiva de líneas de comandos de Nmap. Zenmap permite que los resultados de los análisis se puedan guardar y ver más adelante, además de poderse comparar entre ellos, pare ver cómo difieren.

3.4.7 Metasploitable [37]

La máquina virtual Metasploitable es una versión intencionadamente vulnerable de Ubuntu Linux, diseñada para probar herramientas de seguridad y demostrar vulnerabilidades comunes.

Ya existe una segunda versión de esta máquina virtual disponible para su descarga y uso, y viene con incluso más vulnerabilidades que la imagen original. Esta máquina virtual es compatible con VMWare [5], VirtualBox [7] y otras plataformas de virtualización comunes.

De manera predeterminada, las interfaces de red de Metasploitable están enlazadas a los adaptadores de red NAT y Host-only.

A pesar de que es posible instalar la máquina virtual directamente sobre VirtualBox o VMWare, se ha decidido instalarlo sobre DevStack, para simular aún más el entorno Cloud.

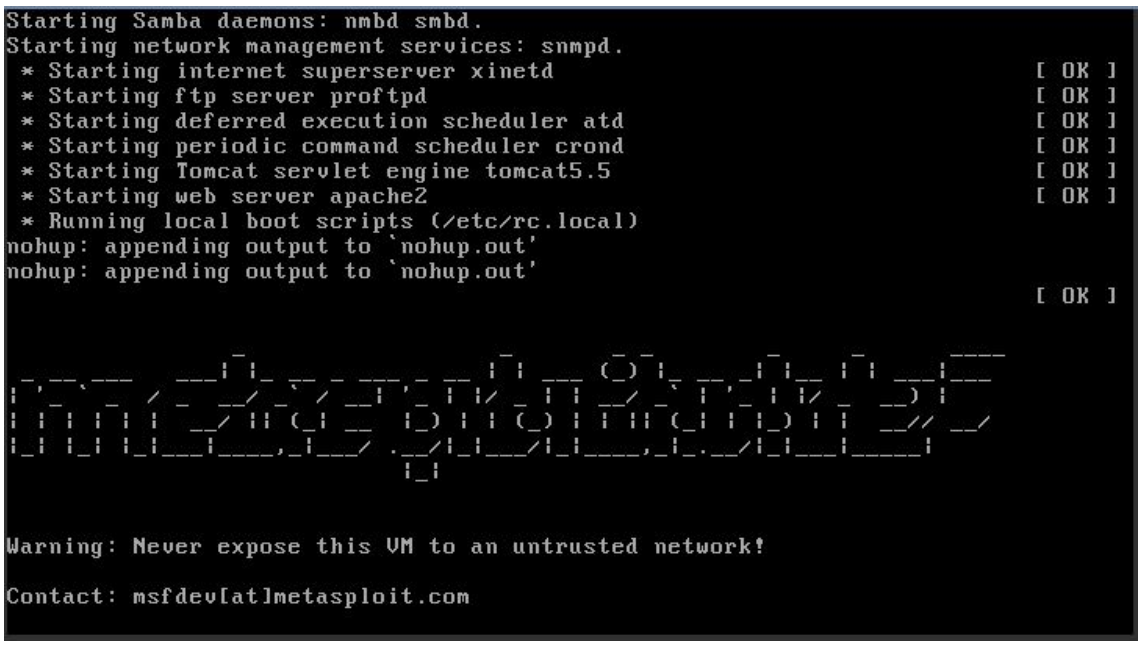

Figura 14: Captura de Metasploitable.

3.4.8 Openvas [38]

OpenVAS (Open Vulnerability Assessment System), denominado inicialmente como GNessUs, una variante del escáner de seguridad Nessus, cuando éste cambió su tipo de licencia. Se propuso como un sistema para pruebas de penetración, pentester en la empresa Portcullis Computer Security, y tras eso fue anunciado como una solución de software libre por Tim Brown.

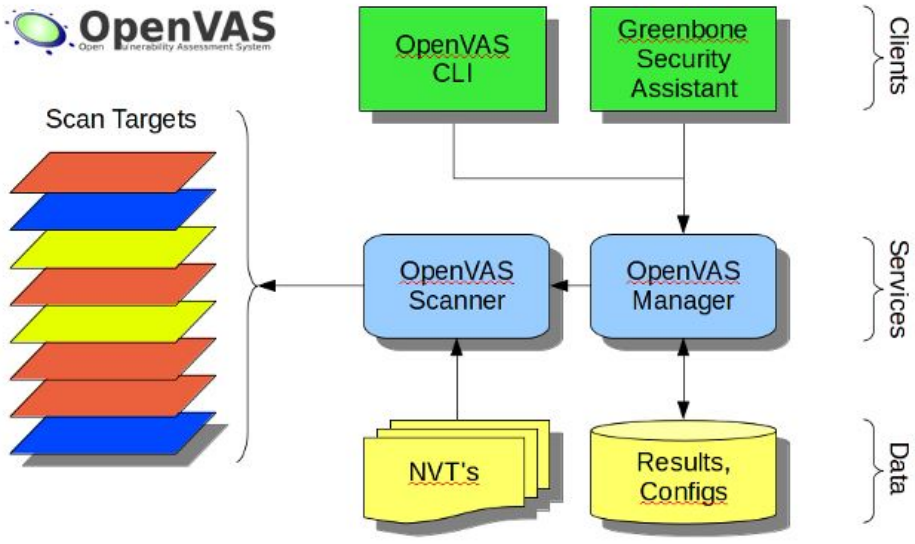

Figura 15: Estructura de funcionamiento de OpenVAS.

OpenVAS es una suite de software, que ofrece un marco de varios servicios y herramientas, que ofrecen una solución completa y potente de gestión de vulnerabilidades y análisis de vulnerabilidades. El marco forma parte de la solución comercial de gestión de vulnerabilidades de GreenBone Networks [39], desde la cual los desarrollos han contribuído a la comunidad Open Source desde 2009. OpenVAS es una de las herramientas de escaneo de vulnerabilidades más conocidas. OpenVAS permite escanear

diferentes tipos de vulnerabilidades, como las que permiten el acceso a datos sensibles del sistema, fallos de configuración, contraseñas por defecto o DoS, entre otras.

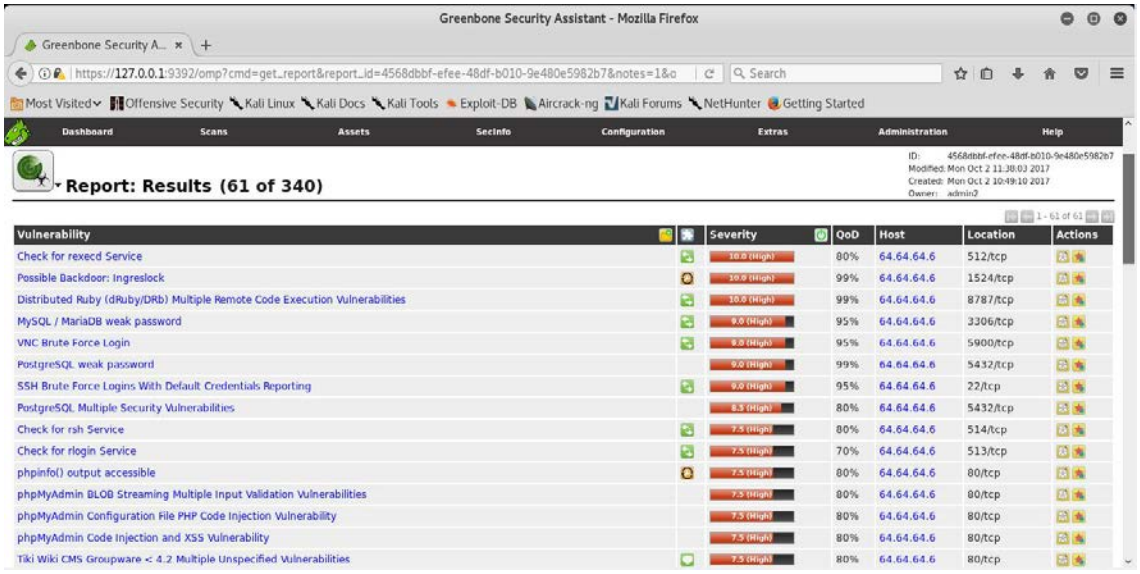

Figura 16: Reporte de análisis de OpenVAS.

En una operación normal, OpenVAS empieza escaneando los puertos con Nmap para buscar qué puertos están abiertos y, después, intentar utilizar varios exploits para atacarlo. Completado el análisis, OpenVAS muestra un informe de vulnerabilidades como el de la Figura 13. En este informe se dará una breve descripción de cada una de ellas, además de indicar el riesgo que suponen para el equipo analizado. En adición, es posible la exportación de los resultados del escaneo como informes en varios formatos, como XML, HTML, NBE y PDF. Además, se pueden guardar los resultados en una base de datos para referencia en futuros escaneos de vulnerabilidades.

El escáner de seguridad real se acompaña con un feed actualizado regularmente de pruebas de vulnerabilidad de red (NVT), más de 50.000 en total. Todos los productos de OpenVAS son software libre, y la mayoría de los componentes están licenciados bajo la Licencia Pública General GNU (GNU GPL).

Capítulo 4.
Desarrollo práctico.

Tras explicar los objetivos del proyecto y las bases teóricas de las que se sustenta, se va a empezar a explicar el desarrollo del mismo. El desarrollo se compone de varias partes en las que se han utilizado diferentes Sistemas Operativos y herramientas para conseguir alcanzar los objetivos.

4.1 Entorno de trabajo.

Antes de empezar con el desarrollo en sí, es conveniente explicar la base sobre la que este proyecto va a ser construido. Este proyecto se ha llevado a cabo utilizando un ordenador portátil Acer Aspire E1-512G con sistema operativo Windows 10. Sobre este sistema anfitrión se han montado, utilizando Oracle VM VirtualBox [7], versión 5.1.26 r117224 tres máquinas virtuales. De forma que una actúe de máquina atacante, con el sistema operativo Kali Linux, la segunda actuará como un servidor virtualizado con una máquina virtual en su interior, Metasploitable dentro de OpenStack, que actuará como víctima, y la tercera también actuará como víctima, con el sistema operativo Windows 7 Professional SP1. Además, para comparar resultados en dos de los ataques realizados, se ha montado una máquina virtual sobre VMWare [5], versión 12.5.5 build-5234757, que actuará también como víctima, la cual también se trata de una máquina virtual con sistema operativo Windows 7 Professional SP1.

Figura 17: Esquema del entorno de trabajo.

En cuanto a la configuración de red, se ha optado por la opción de Adaptador Host-Only en el virtualizador VirtualBox, en el cual se ha definido una red en el rango de la red 192.168.56.0/24, y además, se ha configurado el servidor DHCP para que resuelva direcciones IP empezando por la 192.168.56.101. Esto lo que permite es que las máquinas virtuales, además de tener acceso a través del NAT a la red pública, que se puedan comunicar entre ellas sin tener que salir a Internet. En el virtualizador VMWare no se ha realizado ninguna configuración adicional de las que tiene por defecto.

En la Figura 17 se muestra un esquema del entorno final utilizado en el desarrollo del proyecto.

4.2 Instalación del entorno de trabajo.

Para llevar a cabo este trabajo, primero hay que instalar y configurar cada una de las máquinas virtuales que son necesarias. La instalación de las dos máquinas virtuales de Windows no requiere ningún tipo de configuración adicional. En cambio, tanto la máquina virtual de Kali Linux como la de OpenStack requieren de configuraciones adicionales.

4.2.1 Instalación y configuración de OpenStack.

Como ya se ha visto en el capítulo anterior, DevStack permite a los desarrolladores de software ejecutar una nube OpenStack en una máquina virtual. DevStack se puede instalar en diferentes sistemas operativos, y por lo general se utiliza Ubuntu 14.04, que es también el más recomendado, sobre todo para los usuarios primerizos. En este

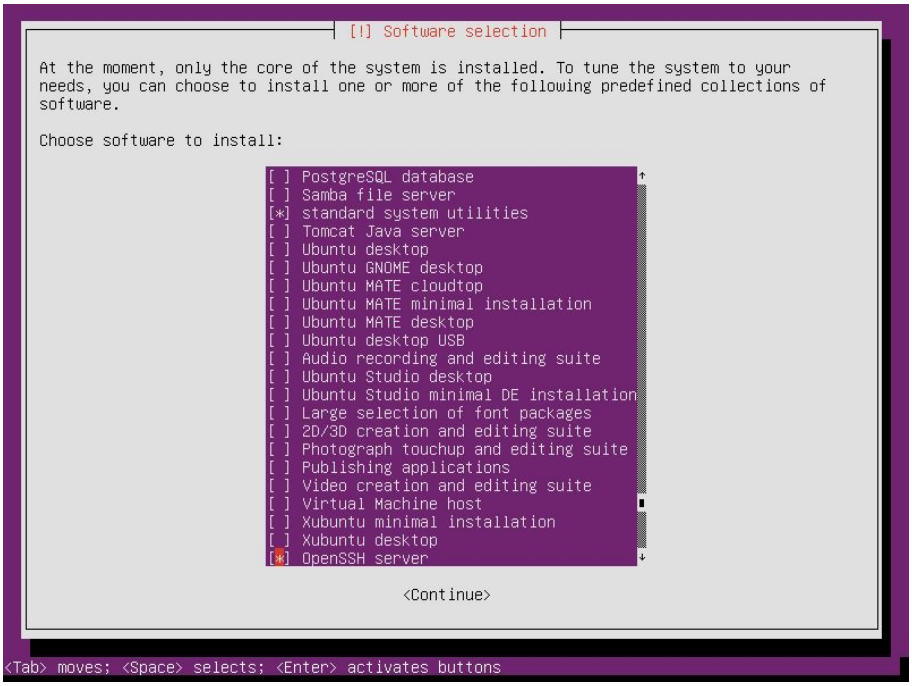

Figura 18: Instalación del servidor Ubuntu.

proyecto se ha optado por usar una imagen ISO mínima de Ubuntu (unos 55MB) como sistema operativo base en la máquina virtual sobre la que se ejecutará OpenStack.

Una vez que se tiene escogido el sistema operativo a utilizar para instalar OpenStack, se procede a instalarlo. Se trata de una instalación normal de una imagen ISO de Ubuntu, salvo que hay que tener cuidado en un detalle, y es que en el software a instalar se tiene que seleccionar la opción de OpenSSH Server, como se muestra en la Figura 18.

Tras instalar el sistema operativo se procede a configurar la máquina e instalar OpenStack. Para ello, primero se tiene que configurar las dos redes existentes de los dos adaptadores, y actualizar el sistema operativo. Teniendo esto, ya se tiene lo necesario para instalar DevStack. Para descargarlo, simplemente hay que descargarlo del repositorio oficial de Git, como se representa en la Figura 19.

```
stack@ubuntu:~$ git clone https://git.openstack.org/openstack-dev/devstack
Cloning into 'devstack'...
remote: Counting objects: 40151, done.
remote: Compressing objects: 100% (19766/19766), done.
remote: Total 40151 (delta 28597), reused 30893 (delta 19789)
Receiving objects: 100% (40151/40151), 8.27 MiB | 1.03 MiB/s, done.
Resolving deltas: 100% (28597/28597), done.
```

Figura 19: Descarga de DevStack.

Cuando se tiene descargado DevStack, se configura como se muestra en la Figura 20. No es más que añadir la dirección IP del sistema host a la configuración, y habilitar el servicio noVNC. noVNC se trata de un cliente VNC[3] escrito completamente en HTML5, de manera que permite conectar a un servidor remoto simplemente utilizando un navegador web.

```
stack@ubuntu:~$ cd devstack
stack@ubuntu:~/devstack$ cp samples/local.conf .
stack@ubuntu:~/devstack$ HOST_IP="192.168.56.103"
stack@ubuntu:~/devstack$ echo "HOST_IP=${HOST_IP}" >> local.conf
stack@ubuntu:~/devstack$ echo "Enable_service n-novnc" >> local.conf
stack@ubuntu:~/devstack$
```

Figura 20: Configuración de DevStack.

Una vez que tenemos configurado todo, se instala OpenStack. Depende del hardware físico tarda más o menos tiempo. Completada la instalación, OpenStack nos muestra los datos de los componentes, como se muestra en la Figura 21:

[3] VNC es una de las herramientas más sencillas y completas para poder controlar un ordenador o máquina virtual de forma remota. Normalmente, para que funcione correctamente esta herramienta, se necesita instalar el servidor en la máquina a controlar, y el cliente en el dispositivo controlador.

```
=========================
Total runtime    3056

run_process       52
test_with_retry    6
apt-get-update     6
pip_install      611
osc              310
wait_for_service  55
git_timed        492
dbsync            61
apt-get          219
=========================



This is your host IP address: 192.168.56.103
This is your host IPv6 address: ::1
Horizon is now available at http://192.168.56.103/dashboard
Keystone is serving at http://192.168.56.103/identity/
The default users are: admin and demo
The password: nomoresecret

WARNING:
Using lib/neutron-legacy is deprecated, and it will be removed in the future


Services are running under systemd unit files.
For more information see:
https://docs.openstack.org/devstack/latest/systemd.html

DevStack Version: pike
Change: f7c250128bbff29402230a573be1339e7a713e0c Merge "doc: Switch from oslosphinx to openstackdocs
theme" 2017-07-31 14:34:01 +0000
OS Version: Ubuntu 16.04 xenial

stack@ubuntu:~/devstack1$
```

Figura 21: Instalación de OpenStack.

Teniendo OpenStack instalado y corriendo, se puede acceder a través de consola de comandos, o a través de la interfaz web proporcionada por Horizon. Por comodidad y rapidez, además de por la facilidad a la hora de realizar las tareas, se accede a través de la interfaz, mostrada en la Figura 22. A esta interfaz se accede introduciendo la dirección IP de OpenStack, en nuestro caso, la dirección 192.168.56.103, en la barra de direcciones. Para entrar al panel de administración hay que introducir el usuario y contraseña proporcionada en el momento de la instalación, al menos la primera vez si no hay más usuarios creados.

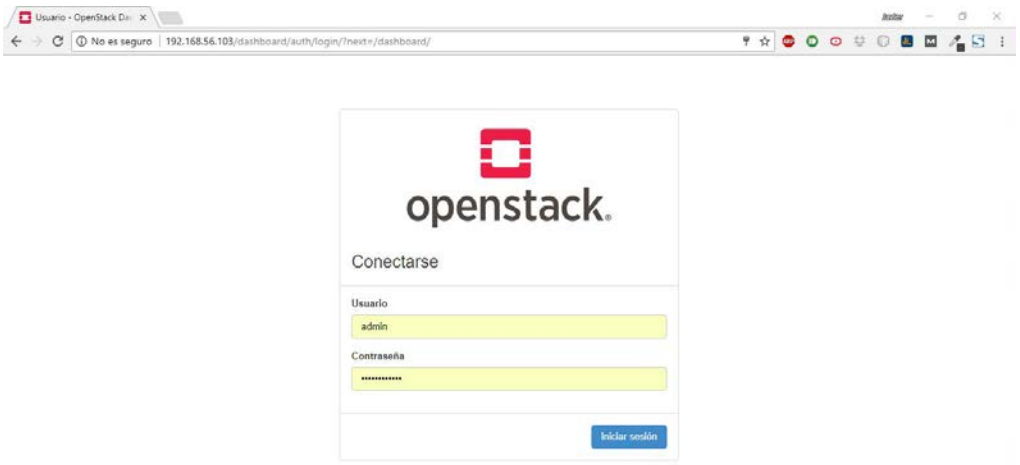

Figura 22: Login de OpenStack.

Para el desarrollo de este trabajo, se ha de crear una nueva red dentro de OpenStack y un router que conecte las redes, tanto la creada, como la que viene por defecto en OpenStack. La nueva red que se creará, en la que más adelante se conectará la máquina virtual Metasploitable, será la red 64.64.64.0 con una máscara de red 255.255.255.0. Además, una vez creada la red, se creará el router virtual, que se conectará a la red "pública" creada por defecto en OpenStack, y se añadirá una interfaz que se conectará a la nueva red creada. En la interfaz, en la sección "Topología de red" se puede ver gráficamente, como se muestra en la Figura 24:

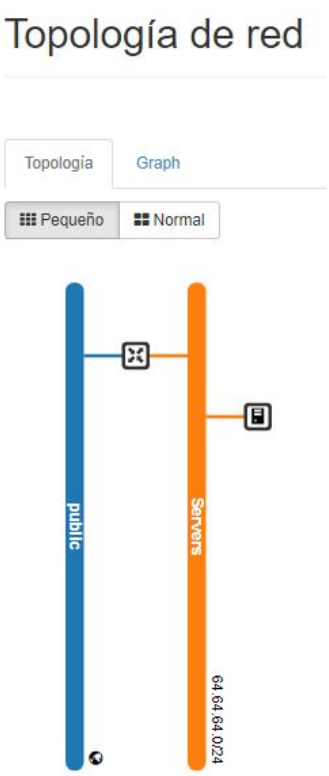

Figura 23: Topología de red en OpenStack.

En adición, las interfaces internas del router creado podrían no estar disponibles aún. Este se debe a que es necesario establecer las reglas correspondientes para la accesibilidad a estas redes. OpenStack define reglas de acceso agrupadas en "Grupos de seguridad". Existe un conjunto de reglas predeterminadas por OpenStack que especifica todos los flujos de entrada/salida, pero sólo especifica completamente la dirección de salida. Para que todo funcione correctamente, se han agregado las reglas de entrada a este grupo de seguridad. Por lo tanto, una vez añadidas, el grupo de seguridad nos quedaría como se muestra en la Figura 24.

Una vez realizado todo lo anterior, ya se tiene OpenStack debidamente configurado para la realización del trabajo. Tras esto, se instala y configura la máquina virtual Metasploitable.

Figura 24: reglas de seguridad.

4.2.2 Instalación y configuración de Metasploitable.

Para la instalación de Metasploitable existen dos maneras de hacerlo dependiendo de los permisos que se tengan. Si se tienen los privilegios adecuados, se puede utilizar la interfaz a través del navegador, aunque la forma más poderosa es utilizar directamente los clientes de línea de comandos OpenStack o Glance, o el servicio Image para administrar imágenes. La segunda opción es la que se ha escogido para la realización de este trabajo.

Lo primero que se ha de hacer es descargar la imagen de la máquina Metasploitable del repositorio oficial en un directorio nuevo, como recomendación. Una vez descargado, se descomprime el archivo descargado en el directorio y se convierte la imagen VMDK a qcow2[4]. Esto último no suele ser necesario, aunque normalmente es bastante útil más adelante. Finalmente, utilizando el comando de glance, se sube a OpenStack la imagen convertida. Cuando se sube la imagen satisfactoriamente, OpenStack devuelve la información de dicha imagen, como se muestra en la Figura 25.

```
stack@ubuntu:~/images/Metasploitable2-Linux$ glance --os-username=admin --os-password=nomoresecret -
-os-tenant-name=DECAMP --os-auth-url=http://192.168.56.103/identity/v2.0 image-create --name Metaspl
oitable2 --disk-format qcow2 --container-format bare < Metasploitable.qcow2
+------------------+--------------------------------------+
| Property         | Value                                |
+------------------+--------------------------------------+
| checksum         | None                                 |
| container_format | bare                                 |
| created_at       | 2017-09-06T22:36:17Z                 |
| disk_format      | qcow2                                |
| id               | 83b5a9c1-4a60-4fc0-b572-3f90842c7576 |
| min_disk         | 0                                    |
| min_ram          | 0                                    |
| name             | Metasploitable2                      |
| owner            | 5ad7146129344352aa58c3353843fcf3     |
| protected        | False                                |
| size             | None                                 |
| status           | queued                               |
| tags             | []                                   |
| updated_at       | 2017-09-06T22:36:17Z                 |
| virtual_size     | None                                 |
| visibility       | shared                               |
+------------------+--------------------------------------+
```

Figura 25: Subida de la imagen de Metasploitable.

[4] Formato de archivo para las imágenes de disco usadas por QEMU.

Tras subir la imagen, queda activarla. En este caso también se ha decantado por utilizar la línea de comandos. Cuando se ejecuta una instancia en OpenStack hay que especificar el ID del flavor[5] que se quiera utilizar para la instancia. Para la ejecución de Metasploitable se recomienda utilizar un flavor que tenga al menos 8GB de disco. En este trabajo se ha optado por escoger dsG1 de la lista de flavors que hay disponibles, que se muestra en la Figura 26, ya que es el flavor que más se adapta a las necesidades del proyecto.

```
stack@ubuntu:~/devstack$ openstack --os-project-name=admin --os-username=admin --os-password=nomores
ecret --os-auth-url=http://192.168.56.103/identity/v2.0 flavor list
+----+-----------+-------+------+-----------+-------+-----------+
| ID | Name      |   RAM | Disk | Ephemeral | VCPUs | Is Public |
+----+-----------+-------+------+-----------+-------+-----------+
| 1  | m1.tiny   |   512 |    1 |         0 |     1 | True      |
| 2  | m1.small  |  2048 |   20 |         0 |     1 | True      |
| 3  | m1.medium |  4096 |   40 |         0 |     2 | True      |
| 4  | m1.large  |  8192 |   80 |         0 |     4 | True      |
| 42 | m1.nano   |    64 |    0 |         0 |     1 | True      |
| 5  | m1.xlarge | 16384 |  160 |         0 |     8 | True      |
| 84 | m1.micro  |   128 |    0 |         0 |     1 | True      |
| c1 | cirros256 |   256 |    0 |         0 |     1 | True      |
| d1 | ds512M    |   512 |    5 |         0 |     1 | True      |
| d2 | ds1G      |  1024 |   10 |         0 |     1 | True      |
| d3 | ds2G      |  2048 |   10 |         0 |     2 | True      |
| d4 | ds4G      |  4096 |   20 |         0 |     4 | True      |
+----+-----------+-------+------+-----------+-------+-----------+
stack@ubuntu:~/devstack$
```

Figura 26: Listado de flavors disponibles.

Además de especificar el flavor, se ha de especificar el ID de la imagen a ejecutar, que se obtiene con el comando de nova image-list, y el ID de la red a la que se va a adjuntar la instancia, que se obtiene con el comando de nova net-list. Ambos comandos se muestran en las Figuras 27 y 28 respectivamente.

Una vez tenemos todo lo necesario, se ejecuta el comando de nova con el nombre que se quiere asignar a la máquina virtual y, tras pasar el tiempo necesario para la creación y ejecución de la instancia, ya se tendrá la máquina virtual Metasploitable completamente configurada.

```
stack@ubuntu:~/devstack$ glance --os-username=admin --os-password=nomoresecret --os-auth-url=http://
192.168.56.103/identity/v2.0 --os-project-name=DECAMP image-list
+--------------------------------------+--------------------------+
| ID                                   | Name                     |
+--------------------------------------+--------------------------+
| 42962499-e9f8-4c2c-8cea-fe9f125b8efe | cirros-0.3.5-x86_64-disk |
| 83b5a9c1-4a60-4fc0-b572-3f90842c7576 | Metasploitable2          |
| 35cfd6fa-29e0-44e5-a1ad-4b11df17b883 | Metasploitable2          |
| 42fac96a-87a3-45ca-8800-1716984e10e9 | Metasploitable2          |
| bc6603ef-f130-43db-a6f6-dc42f2bd2e07 | Metasploitable2          |
| 5ab46cf8-9d73-4f43-a837-2ab565abbc7e | Metasploitable2          |
| c08c59c1-5933-4a66-93ba-db28017bbb0f | Metasploitable2          |
| 94822d19-e03f-4627-b40f-9bab52f27eff | Metasploitable2          |
| 0c2df8b4-a0ba-463e-ad44-e5293c08f027 | Metasploitable2          |
+--------------------------------------+--------------------------+
stack@ubuntu:~/devstack$
```

Figura 27: Listado de imágenes disponibles.

[5] El flavor es un perfil de asignación de recursos. Por ejemplo, especifica cuántas CPUs virtuales y cuánta RAM obtendrá una instancia.

```
stack@ubuntu:~/devstack$ neutron --os-username=admin --os-password=nomoresecret --os-auth-url=http:/
/192.168.56.103/identity/v2.0 --os-project-name=DECAMP net-list
neutron CLI is deprecated and will be removed in the future. Use openstack CLI instead.
+--------------------------------------+---------+------------------------------------------------------
--+
| id                                   | name    | subnets
  |
+--------------------------------------+---------+------------------------------------------------------
--+
| acf8163c-cd19-444d-a94f-25f3e3ab1030 | public  | bd1b48b5-47ce-4468-bef8-89f563446344
  |
|                                      |         | 520d528f-078c-4f52-b13e-0f24b41e75f0
  |
| bfcef9ea-530d-4132-901d-1485cbd9145b | Servers | 7a9cc58d-06ea-40a9-a116-d7da32159f3b 64.64.64.0/2
4 |
+--------------------------------------+---------+------------------------------------------------------
--+
stack@ubuntu:~/devstack$ _
```

Figura 28: Listado de redes disponibles.

4.2.3 Instalación y configuración de Kali Linux.

La instalación de Kali Linux requiere de menos configuración que OpenStack. Primero, para instalar la máquina virtual, se tiene que descargar la imagen ISO que distribuye Offensive Security en su página oficial [26]. Existen cuatro versiones de Kali para VirtualBox, de 32 y de 64 bits, y la versión completa o reducida.

Lo recomendable es instalar la versión completa de 64 bits, pero dependiendo de los recursos de la máquina en la que se instala, puede ser interesante instalar la versión completa de 32 bits. En este trabajo se ha utilizado la versión completa de 64 bits. Tras

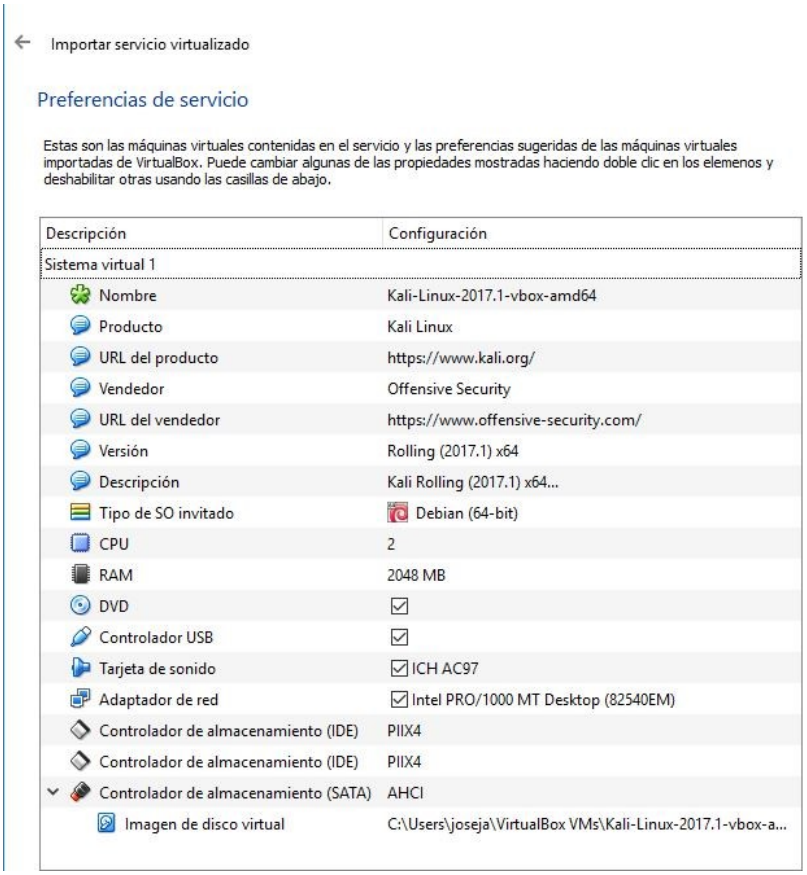

Figura 29: Configuración de Kali Linux al importarse.

descargarlo, hay que importar dicha imagen desde VirtualBox. Automáticamente, la máquina virtual se configurará con 1 GB de memoria RAM y 30 GB de memoria física, como se puede apreciar en la Figura 29.

Una vez que tenemos importada la imagen y, antes de arrancar la máquina virtual, hay que recordar configurar el segundo adaptador de red, asignándole al Adaptador Host-Only mencionado antes. Después de las configuraciones anteriores, se arrancará Kali usando el usuario provisto por defecto (usuario: root, contraseña: toor). Una vez dentro, hay que configurar los adaptadores de red, y activar los dos, ya que por defecto sólo uno estará activo.

A continuación, es importante actualizar la distribución de Kali, ya que se añaden nuevos elementos con mucha frecuencia. Una vez que se tiene Kali actualizado, se añade la ruta a la red de la máquina a atacar, en este caso se trata de la red 64.64.64.0.

4.3 Explotación de vulnerabilidades.

Una vez que tenemos configurado el entorno al completo podemos pasar a explotar las vulnerabilidades del hipervisor. Las vulnerabilidades a explotar son la detección del hipervisor y la captura de tráfico dentro de la máquina explotando una vulnerabilidad remota desde otro equipo. A continuación, se explica el modo de realizarlo, el resultado obtenido y el análisis del mismo.

4.3.1 Detección del hipervisor.

En esta parte del trabajo se va a demostrar cómo es posible detectar la presencia de un hipervisor y, además, cómo identificar de qué hipervisor se trata a través de un ejecutable en una máquina Windows. Para ello, se va a implementar un malware que se ejecutará en el sistema objetivo con la intención de, primero, identificar si se trata de un entorno virtualizado y, segundo, de identificar el hipervisor. Para comparar los resultados dependiendo del entorno en el que se están ejecutando, se ejecutará sobre tres entornos diferentes: dos máquinas virtuales, una alojada sobre VirtualBox y otra alojada sobre VMWare, y una máquina física real.

Este tipo de malware es bastante socorrido por los cibercriminales en sus malwares. Como se ha explicado al principio, una de las funciones de los entonos virtuales es debuggear diferentes programas para comprobar si se trata de malware o no, sin que suponga ningún riesgo para la máquina física real. Por esta razón, los cibercriminales condicionan el funcionamiento de sus programas para que, si se están ejecutando en máquinas virtuales o existe la presencia de programas especializados en depuración de programas, no ejecuten cualquier actividad maliciosa para pasar desapercibida, mientras que si están en una máquina real y sin presencia de depuradores de programas realicen la misión para la que fueron diseñados. O, por el contrario, si se encuentran sobre una máquina virtual que ejecute todas sus funcionalidades maliciosas. Todo depende del objetivo final que tenga el malware.

Este malware se ha escrito en el lenguaje C++, porque se trata de un lenguaje que proporciona mucha versatilidad a la hora de desarrollar. Este malware está orientado a los sistemas operativos Windows, y a los virtualizadores Oracle VM VirtualBox y a VMWare Workstation. Es así porque son los virtualizadores más utilizados, así como el sistema operativo más común. Durante el trabajo se van a observar varias vías más comunes de detección, las diferentes formas de realizarlo según el virtualizador, y cuáles de ellas se han solucionado, ya que como se comentó al principio, aún se está intentando cubrir todas las vulnerabilidades existentes desde el comienzo de los virtualizadores.

Una de ellas, y la primera en analizar es la dirección MAC. El comienzo de cada dirección MAC de dispositivos físicos reales depende del fabricante del adaptador que utilice. En los entornos virtuales son los virtualizadores los que asignan las direcciones MAC de cada máquina virtual, y sus asignaciones están estandarizadas. Por ejemplo, el comienzo de la dirección MAC de una máquina virtualizada por VMWare puede empezar por: "00:05:69:…", "00:0C:29:…", "00:1C:14:…" o "00:50:56:…" mientras que el comienzo de las direcciones MAC de una máquina virtualizada por VirtuaBox es "08:00:27:…" únicamente. Como se muestra en la Figura 30 se trata de comparar inicios de direcciones MAC con la dirección MAC de la máquina en cuestión.

```
12    /* Detección a través de la dirección MAC */
13    void check_mac_vware() {
14        char* iniciomac;
15        const int cont = 4;
16        string MAC_vware[cont];
17        MAC_vware[0] = "000569";
18        MAC_vware[1] = "000C29";
19        MAC_vware[2] = "001C14";
20        MAC_vware[3] = "005056";
21
22        iniciomac = getMAC();
23        for (int i = 0; i < cont; i++) {
24            if (strcmp(iniciomac,MAC_vware[i]) == 0) {
25                printf("Comprobando si la MAC empieza por %s... ", MAC_vware[i]);
26                encontrado();
27            }
28            else {
29                printf("Comprobando si la MAC empieza por %s... ", MAC_vware[i]);
30                no_encontrado();
31            }
32        }
33    }
34    /*-------------------------------------------------------------*/
```

Figura 30: Comprobación de la dirección MAC para VMWare.

Tras la comprobación de la dirección MAC, se va a proceder a comprobar diferentes archivos de sistema existentes en el directorio System32, tanto archivos .DLL como archivos .exe. Estos archivos son necesarios para el correcto funcionamiento de la máquina virtual. En VMWare estos ficheros serían:

- C:\WINDOWS\System32\Drivers\vm3dgl.dll
- C:\WINDOWS\System32\Drivers\vmdum.dll
- C:\WINDOWS\System32\Drivers\vm3dver.dll
- C:\WINDOWS\System32\Drivers\vmtray.dll
- C:\WINDOWS\System32\Drivers\VMToolsHook.dll

- C:\WINDOWS\System32\Drivers\vmmousever.dll
- C:\WINDOWS\System32\Drivers\vmhgfs.dll
- C:\WINDOWS\System32\Drivers\vmGuestLib.dll
- C:\WINDOWS\System32\Drivers\VmGuestLibJava.dll
- C:\WINDOWS\System32\Drivers\vmhgfs.dll

Mientras que en VirtualBox los archivos serían:

- C:\WINDOWS\System32\vboxdisp.dll
- C:\WINDOWS\System32\vboxhook.dll
- C:\WINDOWS\System32\vboxmrxnp.dll
- C:\WINDOWS\System32\vboxogl.dll
- C:\WINDOWS\System32\vboxoglarrayspu.dll
- C:\WINDOWS\System32\vboxoglcrutil.dll
- C:\WINDOWS\System32\vboxoglerrorspu.dll
- C:\WINDOWS\System32\vboxoglfeedbackspu.dll
- C:\WINDOWS\System32\vboxoglpackspu.dll
- C:\WINDOWS\System32\vboxoglpassthroughspu.dll
- C:\WINDOWS\System32\vboxservice.exe
- C:\WINDOWS\System32\vboxtray.exe
- C:\WINDOWS\System32\VBoxControl.exe

Como se muestra en la Figura 31, se ha creado una función que busca el fichero con parámetro de entrada la ruta del archivo en cuestión.

```
92  ** Función para comprobar que exista un fichero determinado.
93  ** Para ello, simplemente que se pueda abrir demuestra que
94  ** existe dicho fichero.
95  */
96  bool existe_archivo(char *ruta) {
97      bool result = false;
98      FILE *archivo;
99      archivo = fopen(ruta, "rb");
100     if (archivo != NULL) {
101         result = true;
102         fclose(archivo);
103     }
104     return result;
105 }
```

Figura 31: Función de búsqueda de un fichero.

Otra vulnerabilidad de los entornos virtuales es la presencia de los archivos de sistema que delatan su presencia. En VirtualBox, estos ficheros son cuatro: VBoxMouse.sys, VBoxGuest.sys, VBoxSF.sys y VBoxVideo.sys. En VMWare se trata de un único archivo, que es Vmmouse.sys. Su presencia delata al entorno virtualizado, que se trata de archivos que en una máquina física son drivers específicos, como el del ratón. En la Figura 32 se muestra la función que busca si existe un archivo de sistema en concreto, utilizando la función que aparece en la Figura 30.

```
void vbox_sysfile() {
    const int count = 4;
    string str[count];
    int res = FALSE, i = 0;
    char message[200];

    str[0] = "C:\\WINDOWS\\system32\\drivers\\VBoxMouse.sys";
    str[1] = "C:\\WINDOWS\\system32\\drivers\\VBoxGuest.sys";
    str[2] = "C:\\WINDOWS\\system32\\drivers\\VBoxSF.sys";
    str[3] = "C:\\WINDOWS\\system32\\drivers\\VBoxVideo.sys";
    for (i = 0; i < count; i++) {
        if (existe_archivo(str[i])) {
            printf("Buscando %s... ", str[i]);
            encontrado();
        } else {
            printf("Buscando %s... ", str[i]);
            no_encontrado();
        }
    }
}
```

Figura 32: Búsqueda de archivos de sistema para VirtualBox.

Otra de las vulnerabilidades reconocidas de los entornos virtualizados es la presencia de las claves de registro y servicios de los entornos virtualizados, así como ciertos valores en las claves de registro. Como en el resto, son diferentes en VirtualBox y VMWare. Esta parte se ha dividido en dos partes. La primera comprueba la existencia de las claves de registros y la segunda comprueba la existencia de ciertos valores dentro de ciertas claves de registro.

Para comprobar la existencia de las clases de registro se ha creado una función que busca si la clave de registro existe en el sistema, como se muestra en la Figura 33. Estas claves de registro a buscar son, en VMWare es únicamente una, "SOFTWARE\\VMware, Inc.\\VMware Tools", mientras que en VirtualBox son más numerosas:

- SOFTWARE\\Oracle\\VirtualBox Guest Additions
- HARDWARE\\ACPI\\DSDT\\VBOX___
- HARDWARE\\ACPI\\FADT\\VBOX___
- HARDWARE\\ACPI\\RSDT\\VBOX___
- SYSTEM\\ControlSet001\\Services\\VBoxGuest
- SYSTEM\\ControlSet001\\Services\\VBoxMouse
- SYSTEM\\ControlSet001\\Services\\VBoxService
- SYSTEM\\ControlSet001\\Services\\VBoxSF
- SYSTEM\\ControlSet001\\Services\\VBoxVideo

```
bool exists_regkey(HKEY hKey, char *regkey_s) {
    HKEY regkey;
    LONG ret;

    if (iswow64()) {
        ret = RegOpenKeyExA(hKey, regkey_s, 0, KEY_READ | KEY_WOW64_64KEY, &regkey);
    }
    else {
        ret = RegOpenKeyExA(hKey, regkey_s, 0, KEY_READ, &regkey);
    }
    if (ret == ERROR_SUCCESS) {
        RegCloseKey(regkey);
        return TRUE;
    }
    else
        return FALSE;
}
```

Figura 33: Búsqueda de archivos de sistema para VirtualBox.

La segunda parte del desarrollo de esta vulnerabilidad es la búsqueda de ciertas cadenas de caracteres dentro de claves de registro determinadas. Es muy similar al anterior, la diferencia reside en que una vez abierto, en vez de cerrarlo y devolver un "true", busca dentro de la clave de registro el valor pasado, y compara el contenido del valor con la cadena de caracteres suministrada. En VirtualBox, como se muestra en la Figura 34, las combinaciones son las siguientes:

- En la clave de registro "*HARDWARE\\DEVICEMAP\\Scsi\\Scsi Port 0\\Scsi Bus 0\\Target Id 0\\Logical Unit Id 0*" y dentro del valor "*Identifier*" se busca que el contenido sea "*VBOX*".
- En la clave de registro "*HARDWARE\\Description\\System*" y dentro del valor "*SystemBiosVersion*" se busca que el contenido sea "*VBOX*".
- En la clave de registro "*HARDWARE\\Description\\System*" y dentro del valor "*VideoBiosVersion*" se busca que el contenido sea "*VIRTUALBOX*".
- En la clave de registro "*HARDWARE\\DESCRIPTION\\System*" y dentro del valor "*SystemBiosDate*" se busca que el contenido sea "*06/23/99*".

Mientras que en VMWare se busca siempre dentro del mismo valor el mismo contenido, pero en diferentes claves de registro. El valor a buscar es "*Identifier*" y el contenido es "*VMWARE*", y las claves de registro en las que se buscan son:

- *HARDWARE\\DEVICEMAP\\Scsi\\Scsi Port 0\\Scsi Bus 0\\Target Id 0\\Logical Unit Id 0.*
- *HARDWARE\\DEVICEMAP\\Scsi\\Scsi Port 1\\Scsi Bus 0\\Target Id 0\\Logical Unit Id 0.*
- *HARDWARE\\DEVICEMAP\\Scsi\\Scsi Port 2\\Scsi Bus 0\\Target Id 0\\Logical Unit Id 0.*

```
void vbox_reg_key2() {
    const int cont = 4;
    string regkeys[cont], values[cont], lookup[cont];

    // SCSI registry key check
    regkeys[0] = "HARDWARE\\DEVICEMAP\\Scsi\\Scsi Port 0\\Scsi Bus 0\\Target Id 0\\Logical Unit Id 0";
    values[0] = "Identifier";
    lookup[0] = "VBOX";

    // SystemBiosVersion registry key check
    regkeys[1] = "HARDWARE\\Description\\System";
    values[1] = "SystemBiosVersion";
    lookup[1] = "VBOX";

    // VirtualBox Guest Additions key check
    regkeys[2] = "HARDWARE\\Description\\System";
    values[2] = "VideoBiosVersion";
    lookup[2] = "VIRTUALBOX";

    // HARDWARE\\DESCRIPTION\\System SystemBiosDate == 06/23/99
    regkeys[3] = "HARDWARE\\DESCRIPTION\\System";
    values[3] = "SystemBiosDate";
    lookup[3] = "06/23/99";

    for (int i = 0; i < cont; i++) {
        if (exists_regkey_str(HKEY_LOCAL_MACHINE, regkeys[i], values[i], lookup[i])) {
            printf("Buscando %s '%s'... ", regkeys[i], values[i]);
            encontrado();
        } else {
            printf("Buscando %s '%s'... ", regkeys[i], values[i]);
            no_encontrado();
        }
    }

}
```

Figura 34: Búsqueda de valores en claves de registro.

La última vulnerabilidad a explotar de manera común es la búsqueda de los procesos activos propios de las máquinas virtuales. Son procesos que se ejecutan en segundo plano para asegurar el correcto funcionamiento de la máquina virtual. Estos procesos son diferentes en VirtualBox y en VMWare. En VirtualBox los procesos son *vboxservice.exe* y *vboxtray.exe*, mientras que en VMWare los procesos son *Vmtoolsd.exe, Vmwaretrat.exe, Vmwareuser.exe* y *Vmacthlp.exe.* Para hacer esta búsqueda se ha implementado una función llamada *comparar_procesos*, que es utilizada por la función utilizada para VMWare, mostrada en la Figura 35, que es muy similar para VirtualBox.

```
/* Detección a través de los procesos */
void vware_procesos() {
    const int contador = 4;
    string proc[contador];
    int j;
    proc[0] = "Vmtoolsd.exe";
    proc[1] = "Vmwaretrat.exe";
    proc[2] = "Vmwareuser.exe";
    proc[3] = "Vmacthlp.exe";
    for (j = 0; j < contador; j++) {
        comparar_procesos(proc[j]);
    }

}
/*-------------------------------------*/
```

Figura 35: Búsqueda de procesos activos.

Otra vulnerabilidad a explotar es la detección de la versión de VMWare. En este caso no se va a implementar para VirtualBox, ya que es bastante más sensible a la hora de desarrollar, y cuando se ejecuta necesita de un archivo .DLL bastante poco común muy pocas distribuciones de Windows poseen, por lo que saltaría una excepción durante la ejecución y se detendría la ejecución del programa. En este caso se ha implementado parte de la función en código ensamblador, ya que se tiene que comprobar el valor de varios campos del procesador, como se muestra en la Figura 36.

```
void check_vware_version()
{
    unsigned int    a, b;

    __try {
        __asm {

            // save register values on the stack
            push eax
            push ebx
            push ecx
            push edx

            // perform fingerprint
            mov eax, 'VMXh' // VMware magic value (0x564D5868)
            mov ecx, 0Ah        // special version cmd (0x0a)
            mov dx, 'VX'        // special VMware I/O port (0x5658)

            in eax, dx          // special I/O cmd

            mov a, ebx          // data
            mov b, ecx          // data (eax gets also modified but will not be evaluated)

                                // restore register values from the stack
                                pop edx
                                pop ecx
                                pop ebx
                                pop eax
        }
    }
    __except (EXCEPTION_EXECUTE_HANDLER) {}
    printf("Buscando la versión de VM Ware... ");
    if (a == 'VMXh') {      // is the value equal to the VMware magic value?
        encontrado();
        printf("\n");
    }
    else {
        no_encontrado();
        printf("\n");
    }

}
```

Figura 36: Detección de la versión de VMWare.

Una vez que tenemos identificado el entorno virtualizado, se procede a implementar la funcionalidad para detectar el hipervisor. Esta funcionalidad se ha implementado en gran parte en código ensamblador, ya que para detectar el hipervisor se ha de llegar al nivel más bajo para llegar a detectarlo. En este caso se han implementado varias funciones que comprueban el contenido de varios registros que delatan la presencia del hipervisor en función de lo que guarden.

Estos registros, además de delatar la presencia de un hipervisor, nos indican qué hipervisor subyace bajo la máquina virtual sobre la que se ejecuta el malware. En la Figura 37 se muestra una función que compara cadenas de caracteres conocidas para comparar con el resultado obtenido del resto de las funciones.

```
int cpu_known_vm_vendors() {
    const int count = 6;
    int i;
    char cpu_hv_vendor[13];
    string strs[count];
    strs[0] = "VMwareVMware"; /* VMware */
    strs[1] = "VBoxVBoxVBox"; /* VirtualBox */
    cpu_write_hv_vendor(cpu_hv_vendor);
    for (i = 0; i < count; i++) {
        if (!memcmp(cpu_hv_vendor, strs[i], 12)) return TRUE;
    }
    return FALSE;
}
```

Figura 37: Detección de un hipervisor.

Una vez que se tiene todo el código implementado, se compila y se prueba en los diferentes entornos planteados para comparar el resultado de la ejecución. Para ejecutar el malware, no hay que hacer más que ejecutar el archivo .exe generado en la compilación. El resultado de la ejecución se muestra en las siguientes figuras.

4.3.2 Resultados de las ejecuciones.

Resultado de la ejecución sobre un entorno real.

Primero se va a ejecutar el malware sobre un entorno real. Como era de esperar, el malware no ha encontrado ningún rastro de las vulnerabilidades buscadas para los virtualizadores, como se muestra en las Figuras 38 y 39. Además, tampoco se ha encontrado rastro de hipervisores, como se observa en la Figura 40, ya que la máquina real no se ejecuta sobre ningún virtualizador.

```
[+]              Metodos de deteccion de VM Ware
[-] Deteccion a traves de la MAC: MAC: F8:A9:63:0C:8E:C0
Comprobando si la MAC empieza por 000569... NO ENCONTRADO
Comprobando si la MAC empieza por 000C29... NO ENCONTRADO
Comprobando si la MAC empieza por 001C14... NO ENCONTRADO
Comprobando si la MAC empieza por 005056... NO ENCONTRADO

[-] Deteccion a traves de Ficheros Conocidos:
Buscando C:\WINDOWS\System32\Drivers\vm3dgl.dll...NO ENCONTRADO
Buscando C:\WINDOWS\System32\Drivers\vmdum.dll...NO ENCONTRADO
Buscando C:\WINDOWS\System32\Drivers\vm3dver.dll...NO ENCONTRADO
Buscando C:\WINDOWS\System32\Drivers\vmtray.dll...NO ENCONTRADO
Buscando C:\WINDOWS\System32\Drivers\VMToolsHook.dll...NO ENCONTRADO
Buscando C:\WINDOWS\System32\Drivers\vmmousever.dll...NO ENCONTRADO
Buscando C:\WINDOWS\System32\Drivers\vmhgfs.dll...NO ENCONTRADO
Buscando C:\WINDOWS\System32\Drivers\vmGuestLib.dll...NO ENCONTRADO
Buscando C:\WINDOWS\System32\Drivers\VmGuestLibJava.dll...NO ENCONTRADO
Buscando C:\WINDOWS\System32\Drivers\vmhgfs.dll...NO ENCONTRADO

[-] Deteccion a traves de Claves de Registro:
Buscando SOFTWARE\VMware, Inc.\VMware Tools... NO ENCONTRADO

[-] Deteccion a traves de Claves de Registro (2):
Buscando HARDWARE\DEVICEMAP\Scsi\Scsi Port 0\Scsi Bus 0\Target Id 0\Logical Unit Id 0
 'Identifier''VMWARE'... NO ENCONTRADO

[-] Deteccion a traves de Drivers:
Buscando C:\windows\System32\Drivers\Vmmouse.sys... NO ENCONTRADO

[-] Deteccion a traves de Procesos:
Buscando Vmtoolsd.exe... NO ENCONTRADO
Buscando Vmwaretrat.exe... NO ENCONTRADO
Buscando Vmwareuser.exe... NO ENCONTRADO
Buscando Vmacthlp.exe... NO ENCONTRADO

[-] Deteccion de la version:
Buscando la versi%n de VM Ware...  NO ENCONTRADO
```

Figura 38: Detección de VMWare en un entorno real.

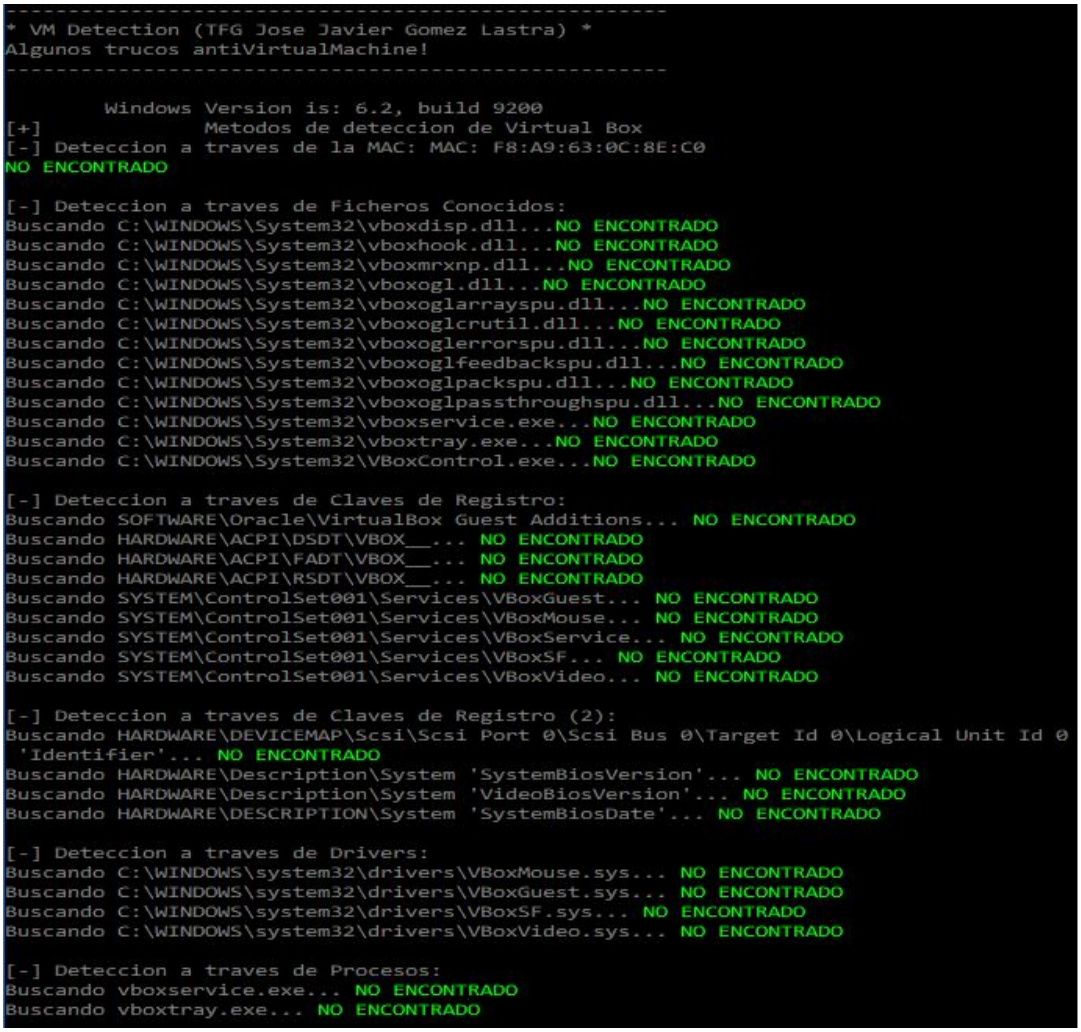

Figura 39: Detección de VirtualBox en un entorno real.

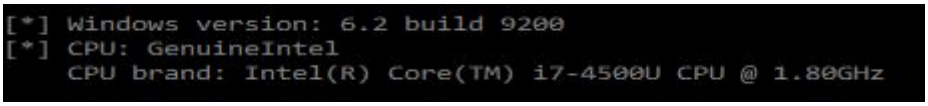

Figura 40: Detección del hipervisor en un entorno real.

Resultado de la ejecución sobre VirtualBox.

Tras analizar los resultados de la ejecución del malware sobre un entorno real, se va a proceder a analizar el comportamiento del malware cuando se ejecuta sobre en una máquina virtual sobre el virtualizador VirtualBox.

Como era de esperar, ninguna de las vulnerabilidades buscadas para VMWare ha sido encontrada en VirtualBox, como se muestra en la Figura 41. En cambio, de las vulnerabilidades buscadas para VirtualBox sí que se han encontrado algunas de ellas. Estas vulnerabilidades encontradas son la detección a través de la dirección MAC, la detección de tres de las claves de registro y la detección a través de la búsqueda de valores en varias claves de registro como refleja la Figura 42.

Esto demuestra que en las últimas versiones se han ido solucionando diversos agujeros de seguridad, pero que aún es posible detectar la presencia de este virtualizador. Además, la detección del hipervisor ha resultado positiva, mostrado en la Figura 43, y ha detectado la presencia del hipervisor de VirtualBox, otro gran agujero de la seguridad.

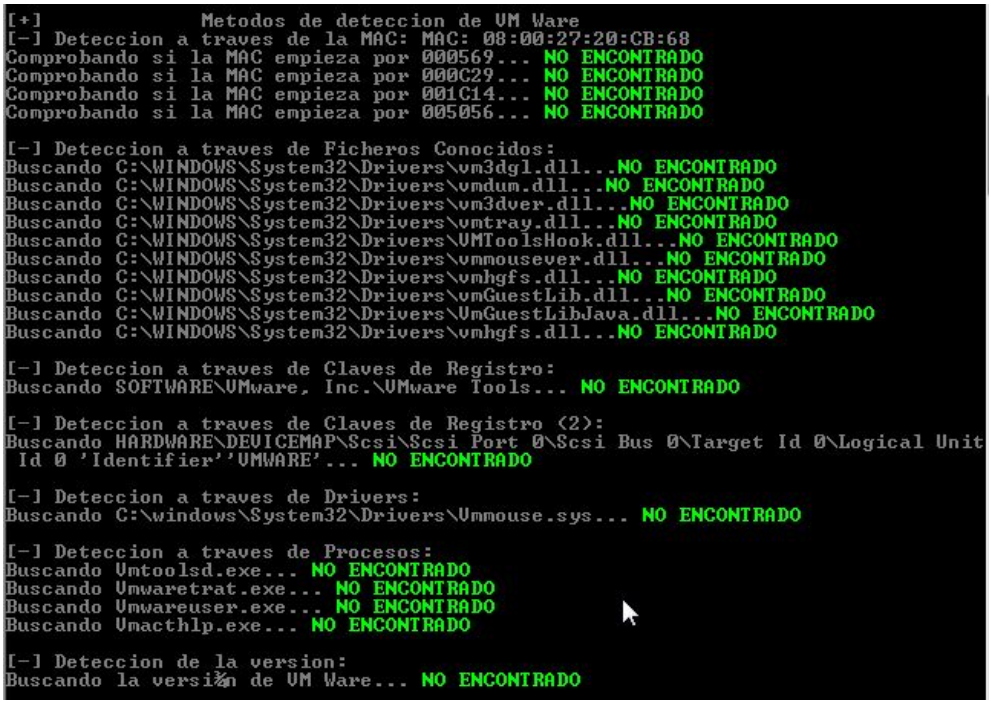

Figura 41: Detección de VMWare sobre VirtualBox.

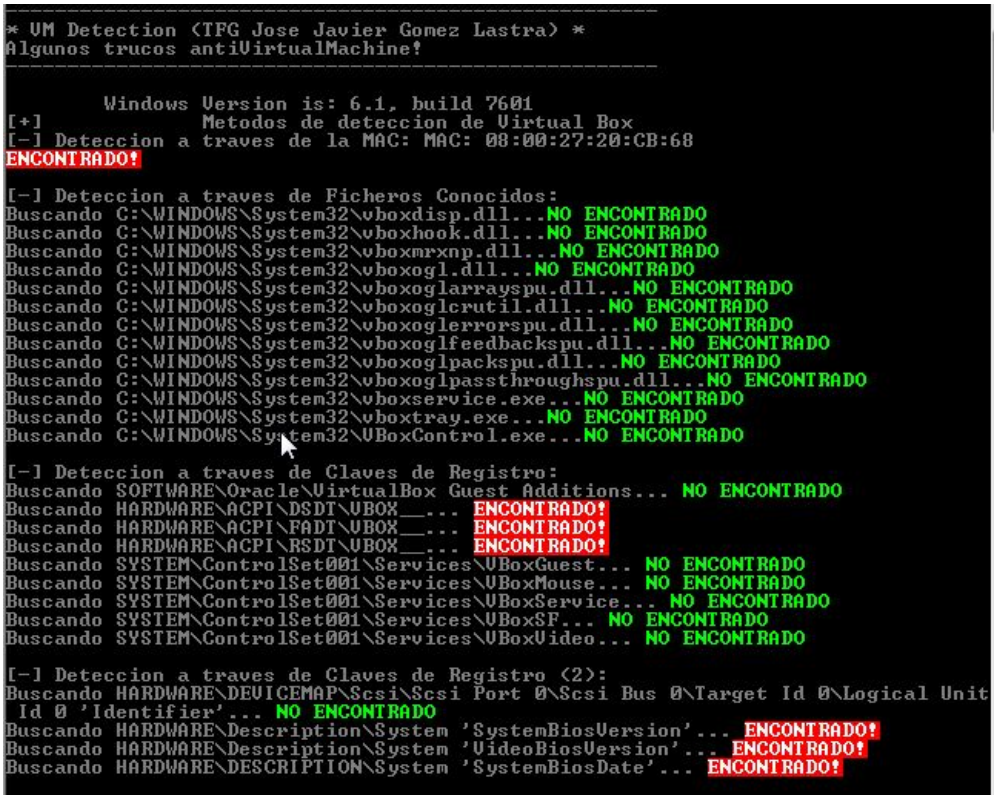

Figura 42: Detección de VirtualBox sobre VirtualBox.

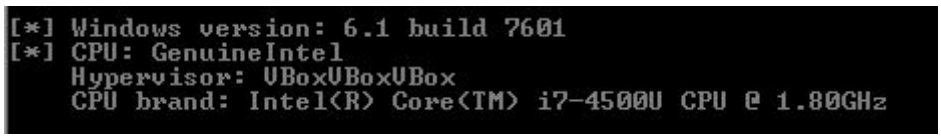

Figura 43: Detección del hipervisor sobre VirtualBox.

Resultado de la ejecución sobre un VMWare.

Por último, después de ver los resultados sobre un entorno real y sobre una máquina montada sobre VirtualBox, se va a ver los resultados de la ejecución sobre una máquina montada sobre VMWare. De la misma manera que en la ejecución sobre VirtualBox, la detección del otro virtualizador no devuelve ningún resultado positivo, como muestra la Figura 45. Por otro lado, como aparece reflejado en la Figura 44, la detección de VMWare sí que devuelve resultados positivos, aunque menos que los que devolvía VirtualBox. Las vulnerabilidades que se han descubierto son la detección a través de la dirección MAC, en concreto el inicio "00:0C:29", la detección a través de valores concretos en claves de registros, y la detección de la versión.

Además, de la misma manera que antes, también se ha detectado la presencia del hipervisor y se ha identificado de qué hipervisor se trata, como refleja la Figura 46. Los resultados muestran que en este virtualizador también se han ido resolviendo problemas de seguridad, pero que aún no han terminado de solventarlos todos.

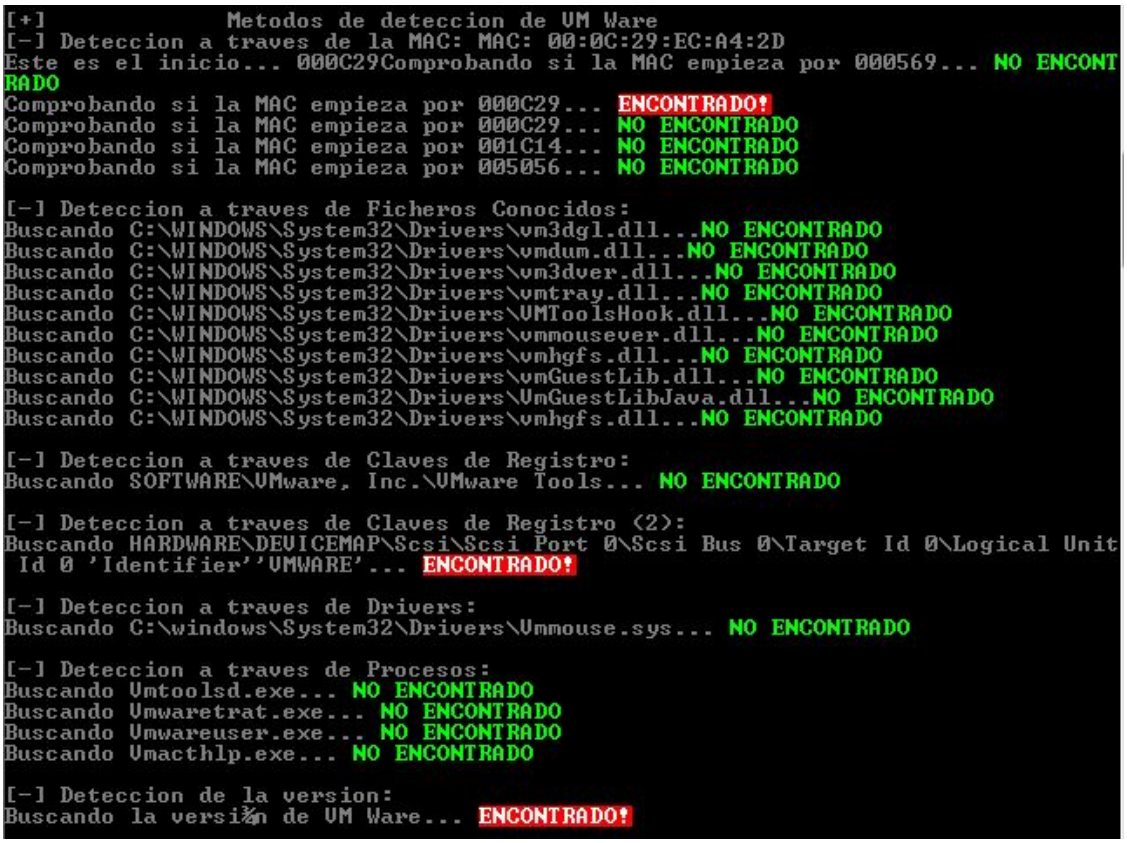

Figura 44: Detección de VMWare sobre VMWare.

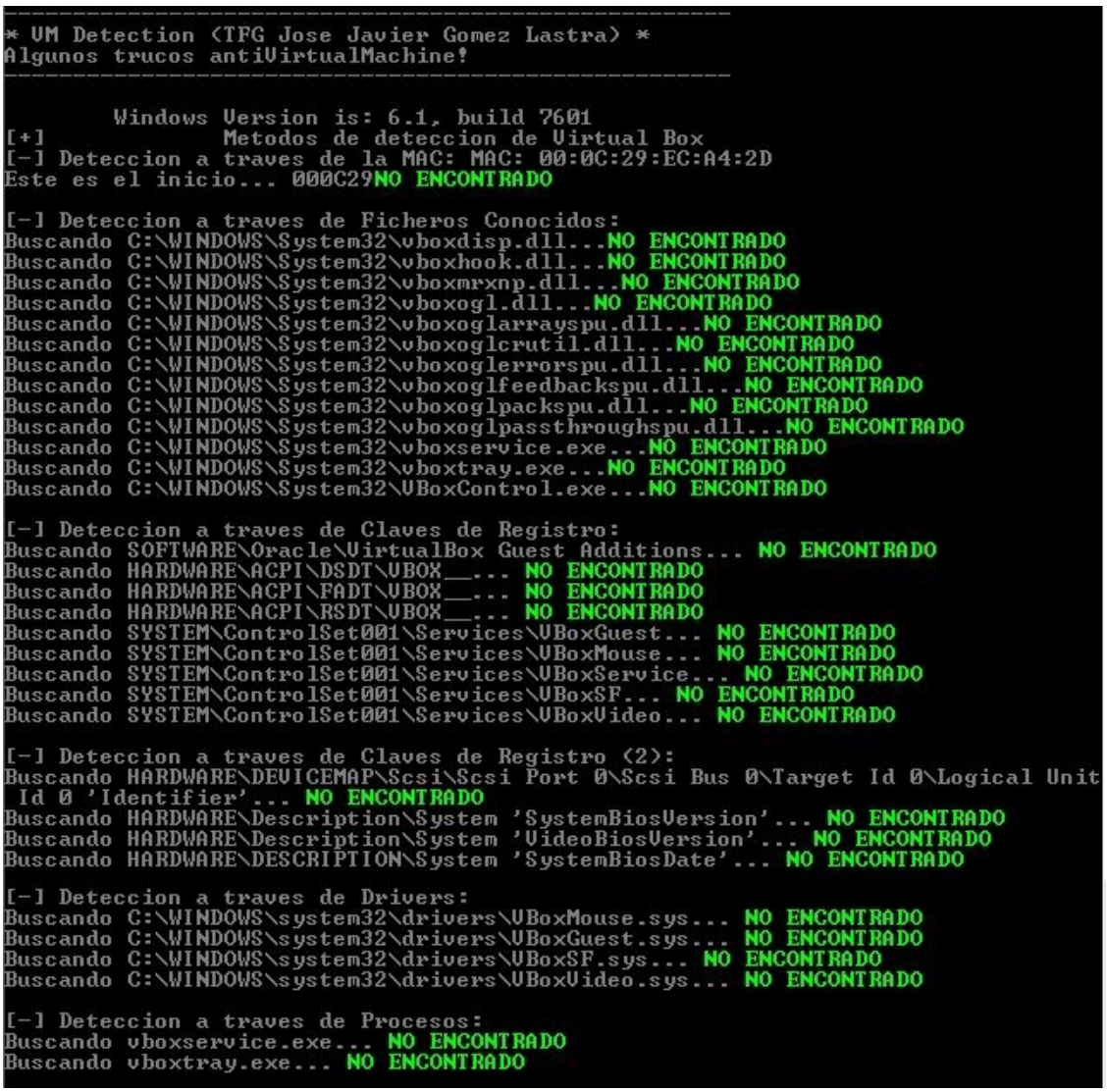

Figura 45: Detección de VirtualBox sobre VMWare.

Figura 46: Detección del hipervisor sobre VMWare.

Analizando los resultados de las ejecuciones del malware sobre los virtualizadores se deduce que la tecnología ha avanzado mucho en términos de ciberseguridad, pero aún no lo suficiente como demuestran los resultados positivos en algunas de las detecciones de vulnerabilidades conocidas.

4.3.3 Sniffing de un hipervisor.

En esta parte del proyecto se va a realizar un ataque remoto a una máquina virtual, y una captura del tráfico de red para intentar encontrar algún rastro del hipervisor subyacente. Para ello, se va a realizar un pentesting, para aprovechar una vulnerabilidad remota para acceder a la máquina y poder lograrlo. Además, se va a realizar

comparaciones en ciertos apartados para comparar cómo se ve una máquina con vulnerabilidades y una máquina que, a priori, no tiene vulnerabilidades.

Para comenzar se va a realizar el Pentesting. Para ello, el mapeo de puertos abiertos es el primer paso del paso Activo. En un proceso completo de Pentesting incluye una fase inicial pasiva, llamada Reconocimiento, donde el atacante obtiene toda la información disponible utilizando Internet y otras fuentes. Para hacer las comparaciones entre dos máquinas, una con vulnerabilidades y otra sin vulnerabilidades a priori, se va a realizar el escaneo sobre Metasploitable y sobre OpenStack.

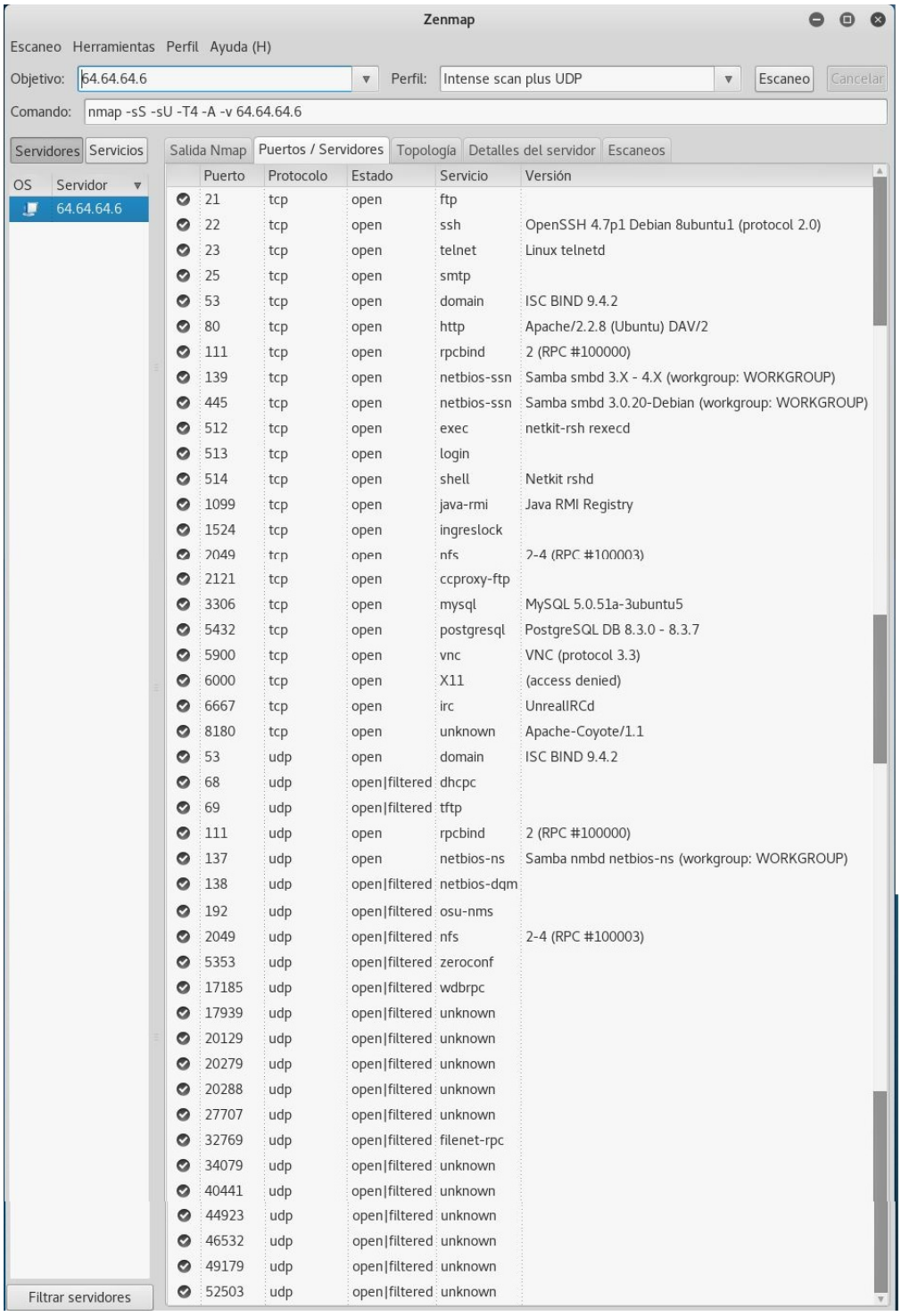

Figura 47: Escaneo de puertos abiertos con Zenmap sobre Metasploitable.

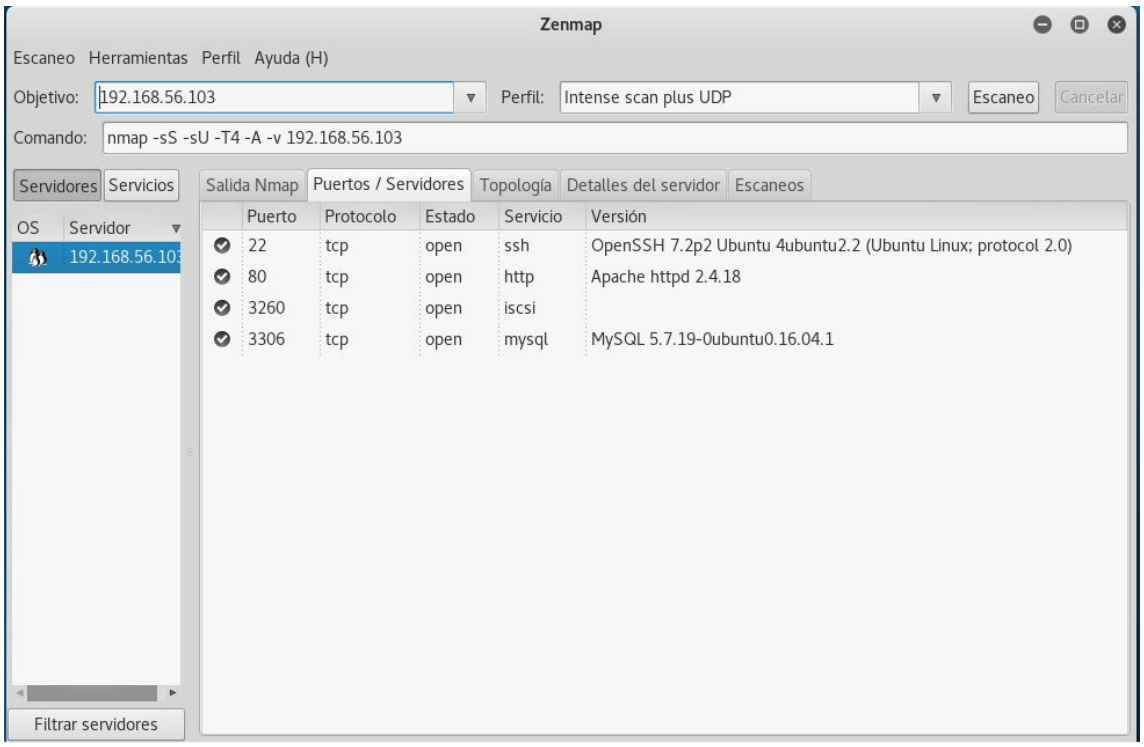

Figura 48: Escaneo de puertos abiertos con Zenmap sobre OpenStack.

Para realizar el mapeo en este caso, se va a utilizar la herramienta Zenmap, que es la interfaz gráfica de Nmap, y se va a realizar un escaneo "Intense scan plus UDP", como se muestra en las Figuras 47 y 48. "Intense scan plus UDP" es una opción de escaneo rápido, que no termina las conexiones que abre para diferenciar entre el estado abierto, cerrado o filtrado de un puerto.

Además, con Zenmap se puede visualizar la topología de la red, los dispositivos que se encuentran en la ruta desde la máquina atacante hasta la máquina objetivo. En la Figura 49 se muestra el número de dispositivos por los que se tiene que pasar para llegar desde la máquina atacante Kali Linux hasta la máquina objetivo Mestasploitable. Esto resulta muy útil, ya que da información sobre cuán lejos (a cuántos saltos de distancia) se encuentra la máquina objetivo de la máquina atacante.

Como muestran las figuras anteriores, OpenStack tiene abiertos únicamente cuatro puertos, los cuales pertenecen a los servicios SSH, HTTP, MySQL y lscsi, mientras que Metasploitable mantiene abiertos más de cuarente puertos (concretamente cuarenta y cinco). Algunos de estos puertos abiertos son de servicios comunes, como los servicios FTP, TELNET o SMTP, pero otros puertos, como por ejemplo los puertos 20129, 20279, 27707 o el 52503 son puertos no asociados a ningún servicio, lo cual supone un problema en la seguridad. Cuantos más puertos abiertos, incluso asociados a servicios, supone un mayor riesgo, ya que hay más oportunidades de explotar vulnerabilidades remotas a través de dichos puertos.

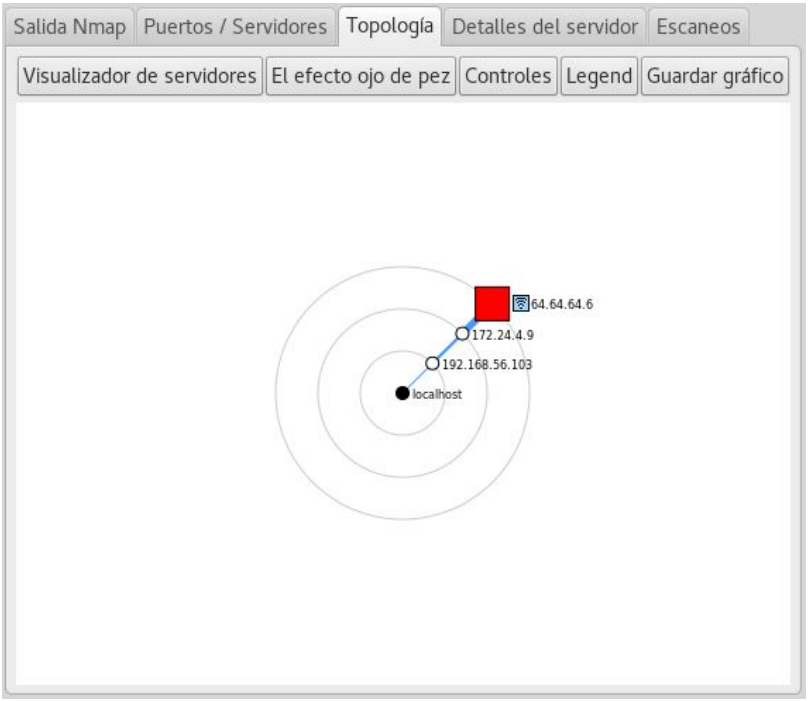

Figura 49: Topología de red.

Una vez descubiertos los puertos abiertos se va a proceder a realizar un análisis de vulnerabilidades completo. Para ello, vamos a utilizar la herramienta de análisis OpenVAS. Pero para poder utilizar OpenVAS, primero hay que descargarla, ya que Kali Linux no la trae por defecto. Una vez que se tiene descargada, se puede crear un nuevo usuario con una contraseña nueva, o utilizar el usuario que viene por defecto. En este caso se ha optado por lo primero, y se ha creado un usuario "admin2" como se muestra en la Figura 5. A pesar de introducir una contraseña cuando se crea un nuevo usuario, la contraseña a utilizar a la hora de logarse en la aplicación es la que se genera en la creación del usuario.

```
root@kali:~# openvasmd --create-user=admin2 --role=Admin openvasmd --user=decamp --new-password=decamp
User created with password 'cc5bab41-01bb-4034-b1c9-c95c7f6aa1b6'.
```

Figura 50: Creación de nuevo usuario en OpenVAS.

Teniendo el usuario nuevo configurado, se arranca la herramienta con el comando *openvas-setup*. Una vez arrancada, se puede acceder a su interfaz gráfica a través del navegador, a través de la dirección: https://127.0.0.1:9392. Tras introducir el usuario y la contraseña creados, se accede al panel de control de OpenVAS, el cual se muestra en la Figura 51. Desde este panel se puede gestionar todo OpenVAS, desde realizar un nuevo análisis, hasta ver los reportes de los análisis.

Realizar un escáner de vulnerabilidades es muy sencillo con OpenVAS. Dentro de "Tasks", perteneciente a la opción "Scans" del menú principal existe un asistente en el cual, introduciendo la dirección IP de la máquina objetivo, se inicia automáticamente el proceso de detección de vulnerabilidades. Una vez finalizado el análisis, OpenVAS nos permite ver los resultados de una manera muy visual y agradable, como se muestra en la Figura 52.

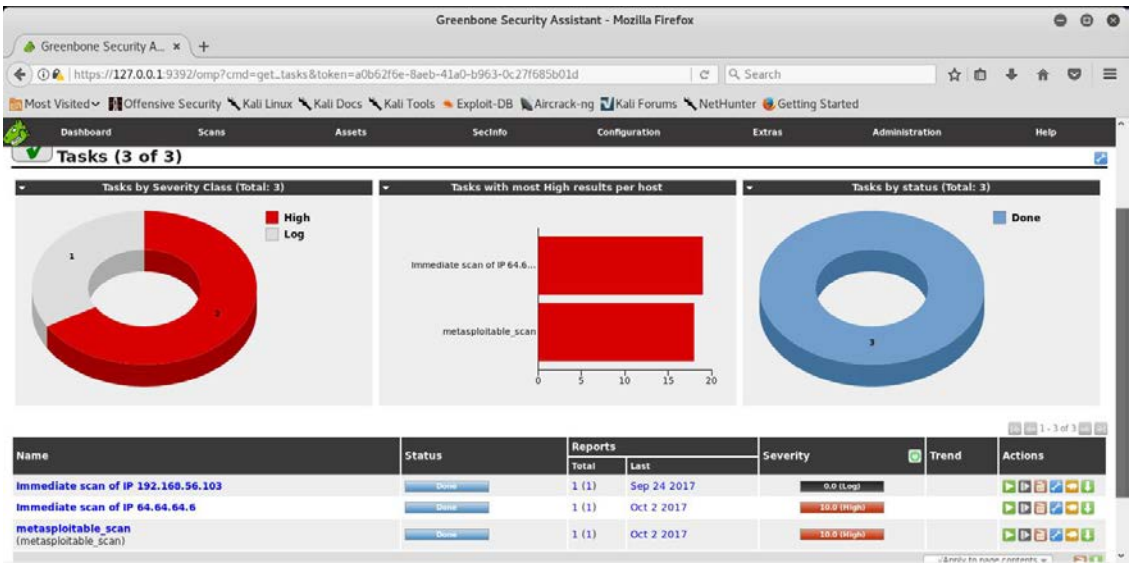

Figura 51: Panel de control de OpenVAS.

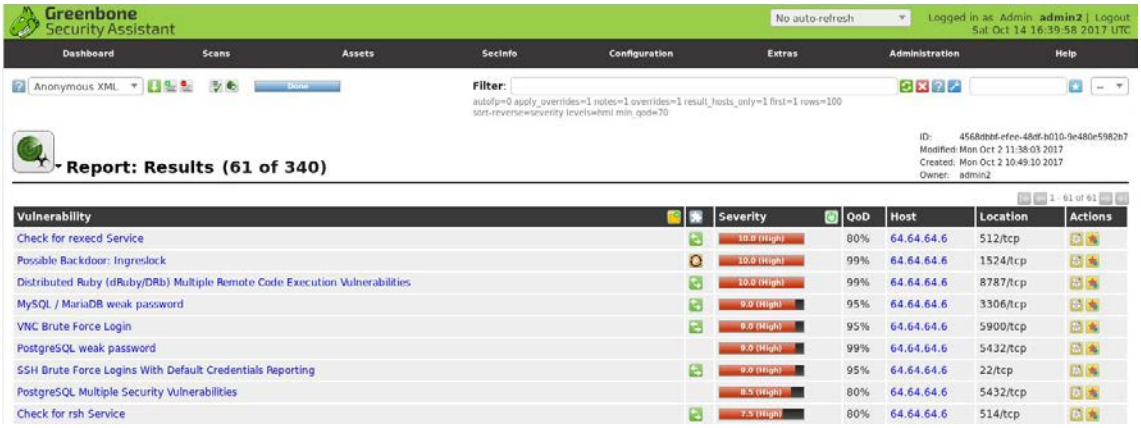

Figura 52: Reporte de análisis de vulnerabilidades de OpenVAS.

Además, como se puede apreciar en la Figura 52, existen hasta 340 vulnerabilidades explotables en la máquina objetivo. Todo lo contrario que cuando se analiza OpenStack, cuyo análisis mostrado en la Figura 53 refleja que no posee vulnerabilidades, al menos de las que tiene OpenVAS en su base de datos.

Figura 53: Reporte de análisis de vulnerabilidades de OpenStack.

Tras analizar las vulnerabilidades, se va a integrar los datos obtenidos con OpenVAS en Metasploit. Para ello, lo primero que hay que hacer es abrir una consola msf, que estará abierta dentro de Metasploit, con el comando *msfconsole*.

Una vez dentro de la consola, para utilizar OpenVAS dentro de Metasploit hay que cargar los módulos de OpenVAS con el comando *load openvas*, mostrado en la Figura 55.

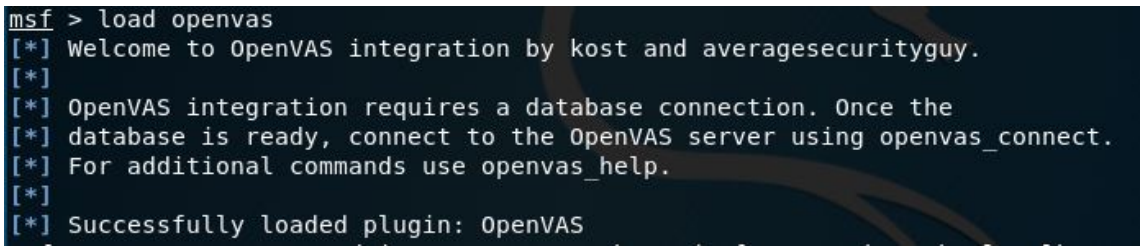

Figura 54: Cargado de OpenVAS en Metasploitable.

Después de cargar los módulos hay que conectar Metasploit con la base de datos de OpenVAS. Para conectarse por la base de datos se utiliza el comando *openvas_connect*, y se pasa en el comando los parámetros del nombre de usuario, la contraseña de dicho usuario, la ubicación de la base de datos y el puerto a través del cual se tiene que conectar, como se muestra en la Figura 55.

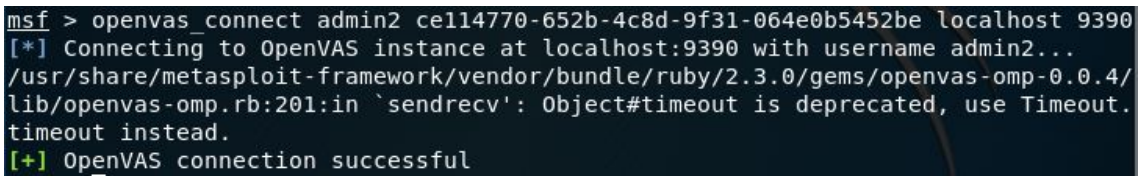

Figura 55: Conexión a la base de datos de OpenVAS desde Metasploitable.

Llegados a este punto se pueden hacer dos cosas, utilizar alguno de los escaneos realizados en la interfaz web, o crear un nuevo objetivo a través de la línea de comandos.

En este caso se va a utilizar el escaneo realizado previamente a través de la interfaz web. Como tenemos más de un escaneo, y el id de dicho escaneo es necesario para continuar, se puede mostrar los escaneos realizados con el comando *openvas_report_list*, como se muestra en la Figura 56.

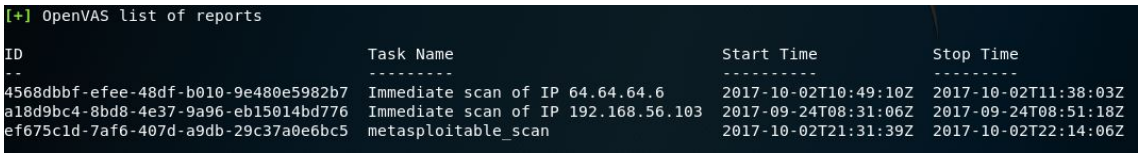

Figura 56: Reporte de escaneos realizados.

De la misma manera que el listado de reportes, también se puede comprobar los formatos que están disponibles para la exportación, como se refleja en la Figura 57.

Figura 57: Reporte de escaneos realizados.

Como se puede apreciar en la imagen, existen diversos formatos para la exportación. Entre ellos se encuentran los formatos HTML, XML o PDF. Aunque los formatos que más interesan para la realización del trabajo son los formatos XML y NBE. NBE es un formato propio de Nessus, desarrollado por Tenable Network Security para realizar el informe de los escaneos.

Este tipo de archivo es bastante importante, ya que para poder trabajar con Armitage es necesario importar los reportes que se quieran utilizar en los formatos XML o NBE. Para ello, se utiliza el comando *openvas_report_import*, pasándole como parámetros el identificador del reporte y el identificador del formato a importar, como se muestra en la Figura 58.

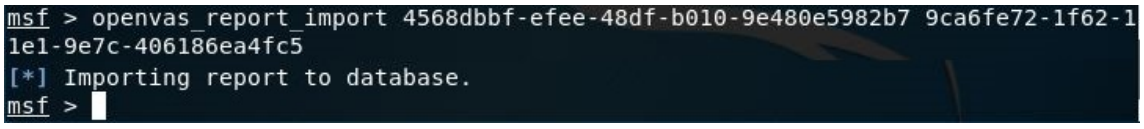

Figura 58: Importación de un escaneo a Metasploitable.

Una vez que se tiene importado a Metasploitable el reporte, las vulnerabilidades asociadas ya estarán disponibles en la base de datos. Teniendo las vulnerabilidades importadas en la base de datos, se puede utilizar esta lista de vulnerabilidades para explotarlas.

Existen varios métodos, comenzando por el uso directo de Metasploit, pero el más simple y potente es utilizar la gestión de ataques Armitage. Es muy útil, ya que representa sus objetivos, recomienda ataques a realizar y expone las capacidades avanzadas del marco Metasploit. Además, esta herramienta organiza todas las funcionalidades, adaptándolas al proceso del hacking.

Los espacios de trabajo dinámicos de Armitage permiten definir y cambiar rápidamente los criterios de destino. Armitage también lanza exploraciones e importa datos de muchos escáneres de seguridad. Además, opcionalmente ejecuta comprobaciones activas para indicar qué exploits funcionarán. Si estas opciones fallan, se puede utilizar el ataque *Hail Mary* para desencadenar la explotación automática de Armitage contra los objetivos.

Para lanzar Armitage, lo primero de todo es comprobar la versión de Java que está presente en Kali Linux, ya que si la versión presente se corresponde con Oracle Java

1.8u131 se tienen problemas. Esto es así porque existe un bug en dicha versión que impide la ejecución de Armitage.

Teniendo la versión adecuada de Java, se procede a ejecutar Armitage, para lo cual, se ha de arrancar la base de datos, con el comando *service postgresql start root*, se inicia la consola de Mestasploit y para finalizar, con el comando *armitage* se arranca Armitage. Por defecto aparece la configuración de la conexión, con la dirección *127.0.0.1:55553*, y aceptándolo, aparece la interfaz de Armitage, la cual se muestra en la Figura 59.

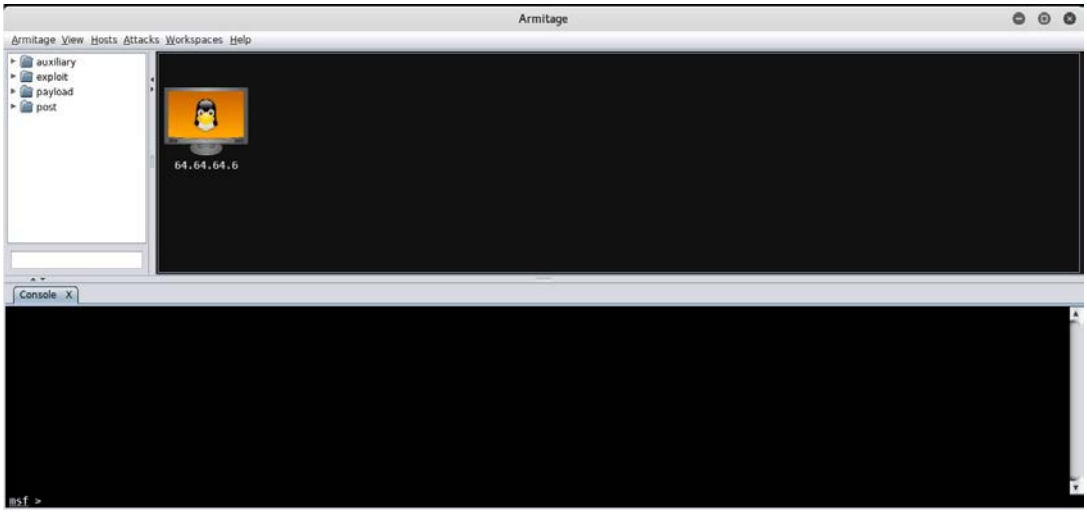

Figura 59: Interfaz gráfica de Armitage.

Una vez dentro de Armitage, éste expone las herramientas de post-explotación integradas en el agente Meterpreter. Con un click dentro de un menú es posible aumentar el nivel de privilegios, registrar las pulsaciones de teclado, deshacer los hashes de contraseñas, navegar por el sistema de archivos y utilizar los comandos de consola. Además, utiliza el módulo proxy SOCKS de Metasploit para poder utilizar las herramientas externas.

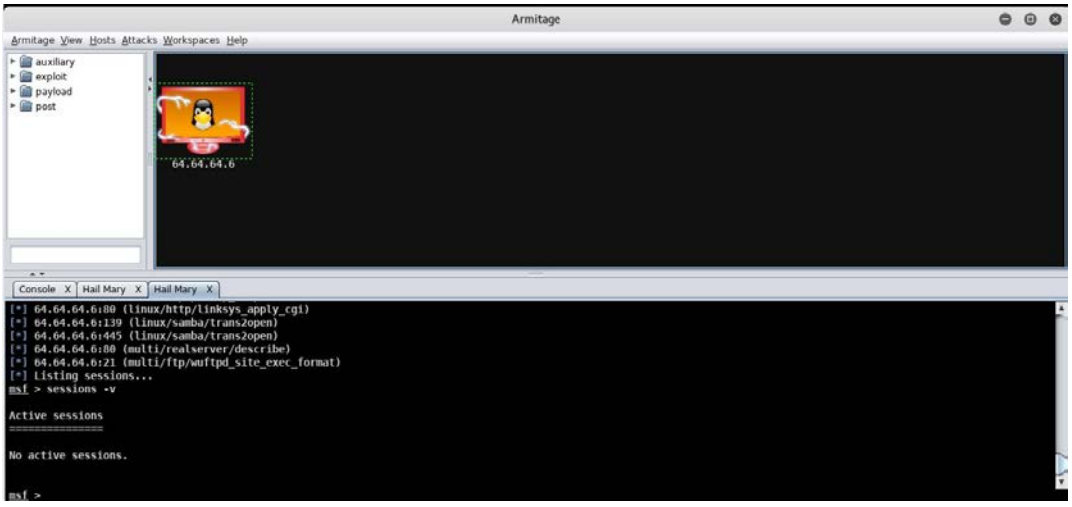

Figura 60: Armitage después de realizar exploit Hail Mary.

Una vez dentro, se va a utilizar el exploit *Hail Mary*, que lo que hace es intentar explotar todas las vulnerabilidades conocidas por Metasploit y ver si la máquina objetivo responde a alguno de ellos. En caso de que la máquina objetivo responda positivamente

a alguno de los ataques realizados, su icono en Armitage cambia, como se puede apreciar en la Figura 60.

Sabiendo que la máquina posee vulnerabilidades, se puede pasar a inicializar el proceso de explotación. En este caso se va a explotar una vulnerabilidad para conseguir el acceso a la consola de comandos de la máquina objetivo, como es el exploit VSFTPD v2.3.4 Backdoor Command Execution y se realizará una captura de tráfico durante el proceso para intentar descubrir al hipervisor subyacente utilizando Wireshark.

El módulo VSFTPD v2.3.4 Backdoor Command Execution explota una puerta trasera maliciosa, que ha sido creada por un intruso, y se configura un listener de la consola en el puerto 6200 de la máquina objetivo. Este exploit se explota desde el puerto 21 de la máquina objetivo, ya que se conecta a través del protocolo ftp. Para saber si la máquina objetivo es vulnerable a este exploit se puede saber a través de Armitage, o a través de la consola de comandos de Metasploit. Aplicando la segunda opción y como muestra la Figura 61, el puerto 21 se encuentra abierto y es vulnerable a VSFTPD v.2.3.4.

```
root@kali:~# nmap -sV -p 21 64.64.64.6

Starting Nmap 7.60 ( https://nmap.org ) at 2017-10-18 20:11 EDT
Nmap scan report for 64.64.64.6
Host is up (0.0021s latency).

PORT   STATE SERVICE VERSION
21/tcp open  ftp     vsftpd 2.3.4
Service Info: OS: Unix

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 13.98 seconds
```

Figura 61: Escaneo de puerto FTP.

Una vez que se sabe que la máquina objetivo es vulnerable al exploit VSFTPD, se procede a ejecutar dicho exploit. Para ello, primero se debe saber la ruta de acceso al exploit, que se obtiene realizando una búsqueda de la cadena de caracteres *'vsftpd'*. Como se muestra en la Figura 62, el exploit se encuentra en la ruta: *'exploit/unix/ftp/vsftpd_234_backdoor'.*

```
msf > search vsftpd

Matching Modules
================

   Name                                  Disclosure Date  Rank       Description
   ----                                  ---------------  ----       -----------
   exploit/unix/ftp/vsftpd_234_backdoor  2011-07-03       excellent  VSFTPD v2.3.4 Backdoor Command Execution
```

Figura 62: Búsqueda del exploit VSFTPD.

Conociendo la ruta del exploit se ejecuta dicho exploit, utilizando el comando use más la ruta al exploit, aunque no es tan fácil como eso. Para saber qué configuraciones necesita el exploit para ejecutarse satisfactoriamente se utiliza el comando 'show options', el cual nos muestra las configuraciones del exploit, de manera que indica qué opciones son obligatorias, cuáles opcionales y cuáles están ya configuradas. Como se muestra en la Figura 63, antes de configurar nada, indica que la dirección IP de la máquina objetivo es obligatoria y que no está indicada. Para ello, se utiliza el comando *'set RHOST IP_Addr'* donde *IP_Addr* en este caso se trata de la dirección IP de Metasploitable, la dirección *64.64.64.6.*

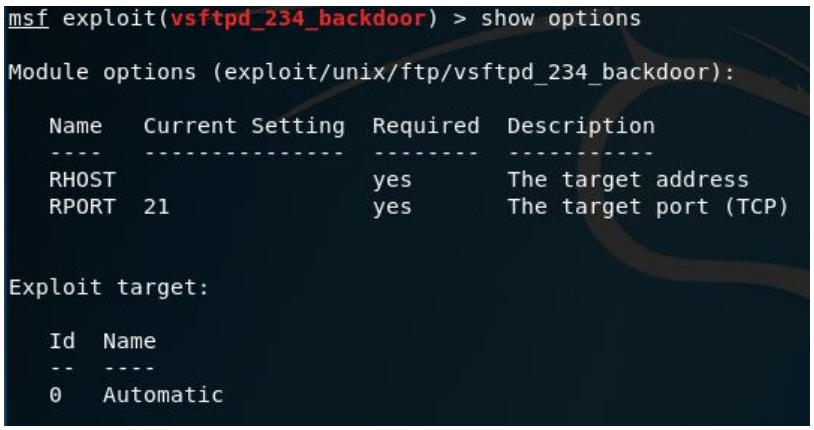

Figura 63: Muestra de opciones del exploit VSFTPD.

Configurada la dirección IP del objetivo del exploit, es necesario indicarle el *payload* que se va a utilizar. Básicamente, el *payload* es la secuencia de instrucciones que se ejecutarán una vez que se haya explotado con éxito la vulnerabilidad. En caso de no indicarle ningún *payload*, el exploit únicamente indicaría que la máquina objetivo es vulnerable a dicho exploit, pero no abriría la consola, que es lo que interesa. De hecho, el Hail Mary realizado previamente es la ejecución de todos los exploits que conoce sin indicarle ningún *payload*, para conocer qué vulnerabilidades están presentes en la máquina objetivo.

Metasploit posee diversos *payloads* con diferentes funcionalidades en función del tipo de arquitectura. Utilizando el comando *'show payloads'* se muestra qué *payloads* son compatibles con el exploit indicado, como se muestra en la Figura 64. Para este caso en concreto se ha escogido el payload *cmd/unix/interact,* ya que se trata de la secuencia de comandos que permite una interacción directa con la consola de la máquina objetivo.

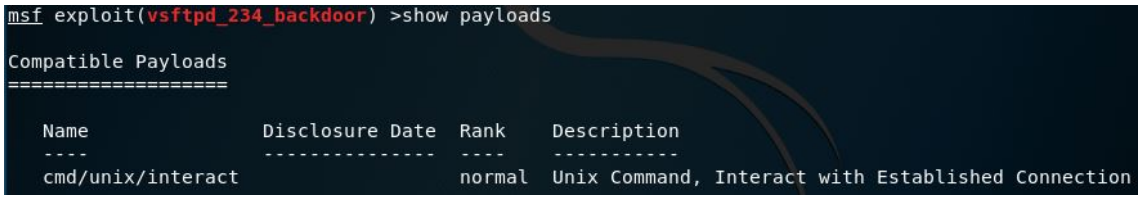

Figura 64: Muestra de payloads disponibles para exploit VSFTPD.

Antes de ejecutar el exploit, para capturar el tráfico e intentar descubrir al hipervisor, se ejecuta Wireshark y se le pone a capturar todo el tráfico de cualquier red. Configurado todo correctamente, se procede a ejecutar el exploit, con el comando *'exploit'*. Una vez ejecutado correctamente, se indica que se tiene acceso a la consola de la máquina objetivo. Para comprobar que realmente se tiene acceso a la consola, se pueden ejecutar los comandos *'whoami'* y *'uname -ar'* y observar lo que devuelve. Como se puede ver en la Figura 65, claramente se trata de la consola de comandos de la máquina objetivo.

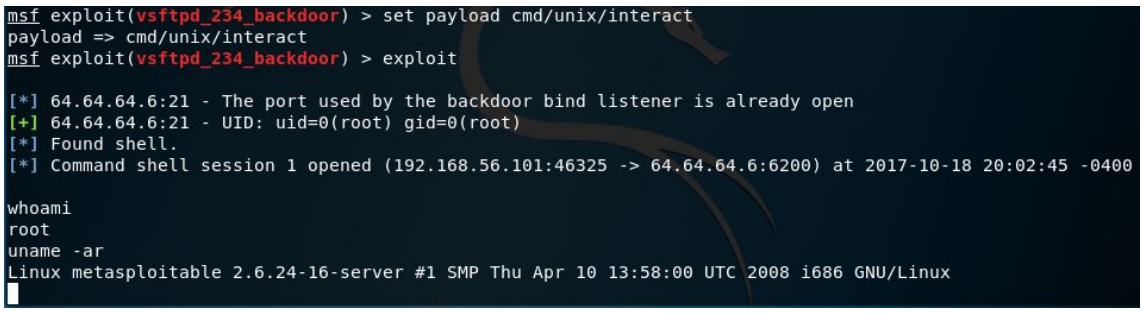

Figura 65: Conexión a Metasploitable.

Una vez que se tiene acceso a la consola, se puede realizar cualquier acción, como mostrar los directorios y moverse entre ellos. Otra acción que se puede realizar es mostrar el fichero *passwd*, fichero en el que están registradas las cuentas de usuarios, así como claves de accesos y privilegios. Este fichero es conocido por ser la primera línea de defensa de un sistema Linux ante accesos no deseados.

Comprobado que se tiene acceso a la consola de la máquina objetivo, se puede terminar dicha conexión, y proceder a analizar la captura realizada con Wireshark, que aparece en la Figura 66. Una vez analizada en profundidad que no existe ni rastro del hipervisor en la captura realizada, excepto en un aspecto. La dirección MAC de la máquina objetivo es 08:00:27:66:cc:53, cuyo inicio concuerda con las direcciones MAC que proporciona VirtualBox., lo que demuestra su presencia.

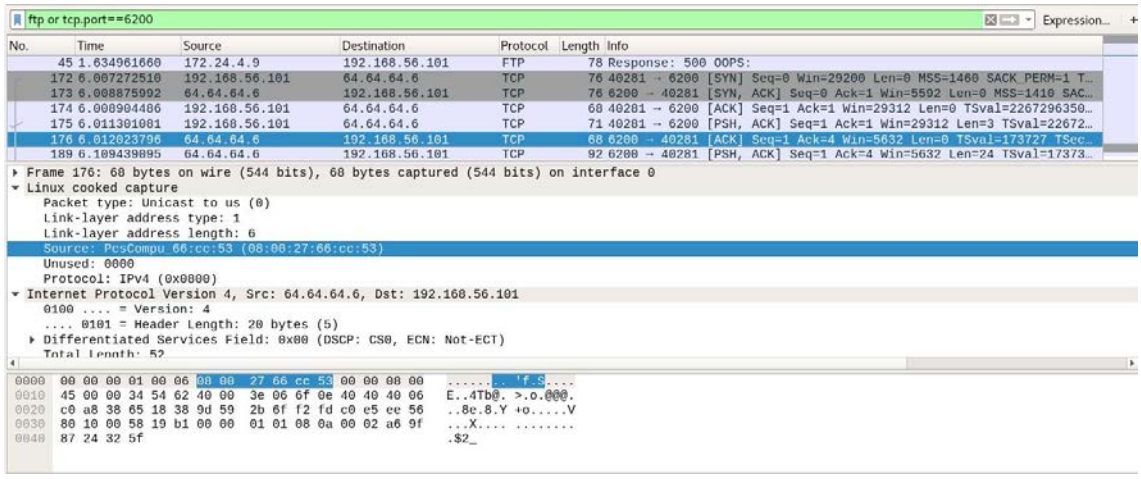

Figura 66: Captura de tráfico de red.

Capítulo 5. Conclusiones.

Finalizado el desarrollo del trabajo, es el momento adecuado de sacar conclusiones a partir de los resultados obtenidos en el mismo. Los puntos más relevantes una vez analizados de todas las valoraciones posibles son:

- Como se ha comentado durante el trabajo, una gran cantidad de agujeros de seguridad han sido solucionados, como se ha visto en el apartado 4.3.1., pero algunos de ellos siguen siendo explotables. Esto indica que a pesar de todos los esfuerzos en securizar los entornos virtualizados aún quedan cosas por hacer.

- La seguridad de un sistema está compuesta de todo el conjunto de medidas de protección utilizados, formando una cadena en la cual su debilidad viene marcada por el eslabón más débil. La parte más vulnerable de un sistema varía; algunas veces se trata de un servicio, otras veces son los protocolos, pero la inmensa mayoría de las veces la parte más vulnerable de un sistema es el software, e incluso el hardware.

- Distintos tipos de ataques, e incluso diferentes vectores de ataque pueden conducir a un atacante lograr un mismo objetivo. A lo largo del trabajo se ha presentado cómo era posible atacar a una máquina montada sobre OpenStack utilizando diferentes tipos de ataques. Un ejemplo de ello sería el buscar y escanear las vulnerabilidades en una máquina si conociendo es máquina se sabe de antemano que presenta diferentes vulnerabilidades potencialmente explotables.

- Los entornos virtualizados y las máquinas virtuales, tanto locales como en un entorno Cloud suelen presentar una robustez ante ataques adecuada, y las vulnerabilidades que presentan residen directamente en el soporte que reciben. La única manera de ir por delante de los atacantes es una constante actualización del sistema operativo y el software de control de los entornos virtualizados. Además del sistema operativo, es importante mantener todos los programas, aplicaciones y servicios actualizados, ya que una única pieza vulnerable conlleva una entrada potencial a todo el entorno.

- En de vital importancia la inversión en seguridad de forma global. Además, es importante que la seguridad de un sistema esté equilibrada. De esta manera, la persona, equipo o empresa destinada a la seguridad de un entorno debe cubrir todas las entradas lo máximo posible. Si se emplean todos los recursos en cubrir

ciertos tipos de ataques, el entorno podría quedar desprotegido ante ataques que provengan de un vector diferente. Además de invertir en personal, es muy importante invertir en formación de dicho personal, ya que la seguridad de un entorno no reside exclusivamente en cerrar todas las posibles entradas potencialmente explotables, sino también en la capacidad de respuesta ante una brecha imprevista en la seguridad y en la rapidez de solucionar dicho problema.

- Además de la formación de personal dedicado a la securización del entorno, es importante la formación de los usuarios que van a utilizar el entorno como de los administradores de los mismos, ya que, a pesar de tener un sistema completamente seguro desde el exterior, un usuario con malas intenciones, o simplemente con desconocimiento es capaz de permitir a un atacante que esté intentando penetrar en el sistema tener acceso completo al mismo. Un claro ejemplo de esto sería la ejecución del malware presentado en el Capítulo 4 por el usuario de la máquina virtual de manera accidentada, o si se encuentra oculta dentro de otra aplicación que a simple vista es inofensivo. Si en vez de ser únicamente para detectar el entorno, estuviese programado para abrir un puerto determinado, el atacante tendría una puerta abierta que podría utilizar para colarse en la máquina y, como consecuencia, en el entorno completo. Mediante el uso de un entorno como el utilizado en la realización del trabajo es posible demostrar a usuarios las consecuencias de determinadas acciones permitiendo un aprendizaje y concienciación interactivos.

- Para finalizar las conclusiones, es fundamental que las labores de la implementación, desarrollo y mantenimiento de las medidas de seguridad de los sistemas esté desarrollado por personal especializado en temas de seguridad. Una manera práctica y que permite una curva de aprendizaje más rápida es la utilización de un laboratorio virtual como el que se utiliza en este trabajo, ya que ofrece la posibilidad de practicar diversas técnicas de ataque, protección y respuesta en un entorno seguro, controlado y completamente legal.

Capítulo 6.
Aplicaciones y Líneas Futuras

Para finalizar el trabajo se puede buscar algunos ejemplos en los que este trabajo pueda resultar útil. A partir del entorno seguro propuesto, es posible realizar una gran cantidad de comprobaciones de seguridad de sistemas operativos, diferentes ataques a entornos virtualizados, posibles protecciones frente a dichos ataques y comprobaciones de las respuestas a los ataques, por lo que una aplicación directa es la realización de pruebas de pentesting a sistemas operativos en Cloud. Para ello, sólo habría que sustituir Metasploitable por el sistema operativo a analizar. Tras ello, se puede realizar todo el proceso de detección, escaneo y explotación de vulnerabilidades.

Otra aplicación inmediata de los resultados es el enorme potencial didáctico, tanto para estudiantes universitarios, como para profesionales que quieren iniciarse en el mundo de la seguridad informática. Esto es así porque se introduce al mundo de los entornos virtualizados, al mundo del pentesting y al mundo del análisis de vulnerabilidades. Todo ello de una manera dinámica y muy explicativa. Una propuesta docente respecto a este tema se compondría de la siguiente estructura:

- Instalación del laboratorio virtual: Se propondrá al alumno la instalación y configuración de todos los componentes necesarios para la realización de la práctica, como son las máquinas virtuales necesarias, Kali Linux y OpenStack.

- Análisis de vulnerabilidades: En esta parte se guiaría al alumno para realizar un análisis del sistema objetivo y así valorar las características del sistema a atacar y su seguridad, con el fin de poder decidir la mejor estrategia para realizar ataques y penetrar en el mismo.

- Explotación de vulnerabilidades: Para finalizar la práctica se propondrá al alumno explotar las vulnerabilidades detectadas en el apartado anterior, para valorar así la importancia de aplicar las medidas de protección asociadas.

Entre las posibles líneas de trabajo asociadas que quedan abiertas tras la realización de este trabajo destacan las siguientes:

- Explotar otro vector de ataque a los entornos virtualizados. En vez de atacar el hipervisor se podría realizar ataques a la red virtualizada, por ejemplo. Para ello sería necesario probar y analizar nuevas herramientas, ya que cada día aparecen nuevas alternativas, así como nuevas vulnerabilidades.

- Desarrollar una amplia colección de máquinas virtuales con distintos tipos de virtualizadores en el entorno propuesto. De esta manera se permitiría a un usuario acceder a una amplia librería de máquinas atacantes, víctimas, ataques y vías de protección con los que realizar configuraciones de red a medida.

- Para finalizar, una posibilidad es realizar este mismo trabajo sobre un entorno Cloud real, y realizar un análisis de vulnerabilidades y una explotación de los mismos para poder comparar cuál es el estado real de la seguridad en entornos Cloud actuales. El problema de este tipo de proyecto reside en que este tipo de pruebas se podría considerar ilegal, por consiguiente este tipo de trabajo se podría realizar si se cumple una de las siguientes condiciones:

    - Crear un entorno Cloud sobre un servidor privado, dando acceso al alumno a dicho servidor.
    - Llegar a un acuerdo, siempre firmado y completamente legal, con algún proveedor de entornos Cloud que proporcione un entorno sobre el que poder practicar.

Capítulo 7. Referencias.

[1]. G. Popek, R. Goldberg: "Formal requirements for virtualizable third generation architectures". Communications of the ACM 17(7), 412–421 (1974).

[2]. J. Ramos: "Security challenges with virtualization" (2009).

[3]. M. Noorafiza, et al. (2013) "Virtual Machine Remote Detection Method Using Network Timestamp in Cloud Computing", Proceedings of the Information Science and Technology (ICIST) on IEEE.

[4]. Xen. Página oficial de Xen. https://www.xenproject.org/

[5]. VMWare. Página oficial de VMWare. https://www.vmware.com/es.html

[6]. QEMU. Página oficial de QEMU. https://www.qemu.org/

[7]. VirtualBox. Página oficial de VirtualBox. https://www.virtualbox.org/

[8]. I. Studnia, E. Alata, Y. Deswarte, M. Kaˆaniche, V. Nicomette. "Survey of Security Problems in Cloud Computing Virtual Machines". Computer and Electronics Security Applications Rendez-vous (C&ESAR 2012). Cloud and security:threat or opportunity, Nov 2012, Rennes, France. p. 61-74, 2012.

[9]. P. Mell, T. Grance: "The NIST Definition of Cloud Computing" (2011).

[10]. M. Alani: "Securing the Cloud: Threats, Attacks and Mitigation Techniques" (2014).

[11]. David S. Linthicum: "Cloud Computing and SOA Convergence in Your Enterprise:A Step-by-Step Guide". (2009).

[12]. Zhang, F., Chen, J., Chen, H., Zang, B.: Cloudvisor: "Retrofitting protection of virtual machines in multi-tenant cloud with nested virtualization". In: Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles. pp. 203–216. ACM (2011).

[13]. T. Garfinkel, M. Rosenblum: When virtual is harder than real: Security challenges in virtual machine based computing environments. In: Proceedings of the 10th conference on Hot Topics in Operating Systems-Volume 10. p. 20. USENIX Association (2005).

[14]. R. Randell, "Virtualization Security and Best Practices", 2006.

[15]. D. Perez-Botero, J. Szefer, B. Lee: “Characterizing Hipervisor Vulnerabilities In Cloud Computing Servers”. In: Proceedings of the Workshop on Security in Cloud Computing (SCC) (2013).

[16]. N. Elhage. Virtunoid: Breaking out of KVM. nelhage.com/talks/kvm-defcon-2011.pdf, August 2011.

[17]. Ferrie, P.: Attacks on more virtual machine emulators. Symantec Technology Exchange (2007)

[18]. RedPill. http://www.securiteam.com/securityreviews/6Z00H20BQS.html

[19]. ScoopyDoo. http://www.trapkit.de/tools/scoopydoo/index.html

[20]. Ormandy, T.: An empirical study into the security exposure to hosts of hostile virtualized environments. In: Proceedings of CanSecWest Applied Security Conference (2007)

[21]. Rutkowska, J.: Subverting vista™ kernel for fun and profit. BlackHat Briefings USA (2006)

[22]. King, S., Chen, P.: Subvirt: Implementing malware with virtual machines. In: Security and Privacy, 2006 IEEE Symposium on. pp. 14–pp. IEEE (2006)

[23]. Rutkowska, J., Tereshkin, A.: Bluepilling the xen hipervisor. Black Hat USA (2008)

[24]. Visual Studio. Página oficial de Microsoft Visual Studio. https://www.visualstudio.com

[25]. KaliLinux. Página oficial de KaliLinux. https://www.kali.org/

[26]. Offensive Security. https://www.offensive-security.com/

[27]. BackTrack. https://www.backtrack-linux.org/

[28]. Nmap. Página oficial Nmap. https://nmap.org/

[29]. WireShark. Página oficial de WireShark. https://www.wireshark.org/

[30]. John de Ripper. http://www.openwall.com/john/

[31]. Metasploit: “Metasploit”, https://www.metasploit.com/

[32]. Rapid7. Página oficial de Rapid7. https://www.rapid7.com/

[33]. OpenStack. Página oficial de OpenStack. https://www.openstack.org/

[34]. OpenStack Foundation. https://www.openstack.org/foundation/

[35]. DevStack. Página oficial de DevStack. http://devstack.org

[36]. Zenmap. Página de Zenmap. https://nmap.org/zenmap/

[37]. Metasploitable. https://www.rapid7.com/resources/test-metasploit-with-metasploitable/

[38]. OpenVAS. Página oficial de OpenVAS. http://www.openvas.org/index.html

[39]. GreenBone Networks. Página oficial de GreenBone Networks. https://www.greenbone.net/

Apéndice A.
Código para detección de entornos virtualizados

A.1. main.cpp

```
// main.cpp: define el punto de entrada de la aplicación de consola.
//

#include "stdafx.h"
#include <windows.h>
#include <stdlib.h>
#include <stdio.h>
#include "intro.h"
#include "utilidades.h"
#include "vbox.h"
#include "vware.h"
#include "cpu.h"



void main(){
	unsigned short original_colors = 0;
	original_colors = init_cmd_colors();
	char winverstr[32], aux[1024];
	char cpu_vendor[13], cpu_hv_vendor[13], cpu_brand[49];
	print_header();
	OSVERSIONINFO winver;

	/*********** DETECCIÓN DE VBOX ***********/
	printf("[+]		Metodos de deteccion de Virtual Box\n");

	/*Intentamos obtener la dirección MAC*/
	printf("[-] Deteccion a traves de la MAC: ");
	check_mac_vbox();
	printf("\n");
	/*------------------------------------------------*/

	/* Detección a través de los ficheros conocidos */
	printf("[-] Deteccion a traves de Ficheros Conocidos: \n");
	check_files1_vbox();
	printf("\n");
	/*------------------------------------------------*/

	/* Detección a través de las claves de registro */
	printf("[-] Deteccion a traves de Claves de Registro: \n");
	vbox_reg_key();
	printf("\n");
	/*------------------------------------------------*/

	/* Detección a través de las claves de registro (2) */
	printf("[-] Deteccion a traves de Claves de Registro (2): \n");
```

```
vbox_reg_key2();
printf("\n");
/*-------------------------------------------------*/

/* Detección a través de los drivers */
printf("[-] Deteccion a traves de Drivers: \n");
vbox_sysfile();
printf("\n");
/*-------------------------------------------------*/


/* Detección a través de los drivers */
printf("[-] Deteccion a traves de Procesos: \n");
vbox_procesos();
printf("\n");
/*-------------------------------------------------*/


printf("\n\n\n");


/*********** DETECCIÓN DE VM WARE ***********/
printf("[+]         Metodos de deteccion de VM Ware\n");

/*Intentamos obtener la dirección MAC*/
printf("[-] Deteccion a traves de la MAC: ");
check_mac_vware();
printf("\n");
/*-------------------------------------------------*/

/* Detección a través de los ficheros conocidos */
printf("[-] Deteccion a traves de Ficheros Conocidos: \n");
check_files_vware();
printf("\n");
/*-------------------------------------------------*/

/* Detección a través de las claves de registro */
printf("[-] Deteccion a traves de Claves de Registro: \n");
vware_reg_key();
printf("\n");
/*-------------------------------------------------*/

/* Detección a través de las claves de registro (2) */
printf("[-] Deteccion a traves de Claves de Registro (2): \n");
vware_reg_key2();
printf("\n");
/*-------------------------------------------------*/

/* Detección a través de los drivers */
printf("[-] Deteccion a traves de Drivers: \n");
vware_sysfile();
printf("\n");
/*-------------------------------------------------*/


/* Detección a través de los drivers */
printf("[-] Deteccion a traves de Procesos: \n");
vware_procesos();
printf("\n");
/*-------------------------------------------------*/

/* Detección de la versión */
printf("[-] Deteccion de la version: \n");
check_vware_version();
printf("\n");
```

```
	/*-----------------------------------------------*/

	system("PAUSE");

	/* Ahora obtenemos la versión del Windows */
	DWORD dwVersion = 0;
	DWORD dwMajorVersion = 0;
	DWORD dwMinorVersion = 0;
	DWORD dwBuild = 0;
	dwVersion = GetVersion();
	// Obtener la versión de Windows.
	dwMajorVersion = (DWORD)(LOBYTE(LOWORD(dwVersion)));
	dwMinorVersion = (DWORD)(HIBYTE(LOWORD(dwVersion)));
	// Obtener el build number.
	if (dwVersion < 0x80000000) {
		dwBuild = (DWORD)(HIWORD(dwVersion));
	}
	printf("	Windows Version is: %d.%d, build %d\n", dwMajorVersion,
dwMinorVersion, dwBuild);
	/*-------- Versión de Windows obtenida ------------*/

	/* Detección del hipervisor y de la CPU */
	cpu_write_vendor(cpu_vendor);
	cpu_write_hv_vendor(cpu_hv_vendor);
	cpu_write_brand(cpu_brand);
	printf("[*] CPU: %s\n", cpu_vendor);
	if (strlen(cpu_hv_vendor)) /* strlen = obtener longitud del argumento; esta
condición es si en cpu_hv_vendor hay algún valor. */
		printf("	Hipervisor: %s\n", cpu_hv_vendor);
	printf("	CPU brand: %s\n", cpu_brand);
	snprintf(aux, sizeof(aux) - sizeof(aux[0]), "Windows version: %s",
		winverstr);

	if (strlen(cpu_hv_vendor))
		snprintf(aux, sizeof(aux) - sizeof(aux[0]), "CPU: %s (HV: %s) %s",
cpu_vendor,
			cpu_hv_vendor, cpu_brand);
	else
		snprintf(aux, sizeof(aux) - sizeof(aux[0]), "CPU: %s %s", cpu_vendor,
			cpu_brand);

	system("PAUSE");

}
```

A.2. cpu.cpp

```
#include <windows.h>
#include <stdio.h>
#include <string.h>
#include <stdint.h>

#include "cpu.h"
#include "stdafx.h"

/* Definición de constantes. */
#define TRUE  1
#define FALSE 0

typedef char * string;
typedef __int32 int32_t;
typedef unsigned __int32 uint32_t;
```

```
static inline unsigned long long rdtsc_diff() {
	unsigned long long ret, ret2;
	unsigned eax, edx;
	__asm { "rdtsc" : "=a" [eax], "=d" [edx] }
	ret = ((unsigned long long)eax) | (((unsigned long long)edx) << 32);
	__asm __volatile("rdtsc" : "=a" (eax), "=d" (edx));
	ret2 = ((unsigned long long)eax) | (((unsigned long long)edx) << 32);
	return ret2 - ret;
}

static inline unsigned long long rdtsc_diff_vmexit() {
	unsigned long long ret, ret2;
	unsigned eax, edx;
	__asm {"rdtsc" : "=a" (eax), "=d" (edx)}
	ret = ((unsigned long long)eax) | (((unsigned long long)edx) << 32);
	/* vm exit forced here. it uses: eax = 0; cpuid; */
	__asm __volatile("cpuid" : /* no output */ : "a"(0x00));
	/**/
	__asm __volatile("rdtsc" : "=a" (eax), "=d" (edx));
	ret2 = ((unsigned long long)eax) | (((unsigned long long)edx) << 32);
	return ret2 - ret;
}

static inline void cpuid_vendor_00(char * vendor) {
	int ebx = 0, ecx = 0, edx = 0;

	__asm __volatile("cpuid" \
		: "=b"(ebx), \
		"=c"(ecx), \
		"=d"(edx) \
		: "a"(0x00));
	sprintf(vendor, "%c%c%c%c", ebx, (ebx >> 8), (ebx >> 16), (ebx >> 24));
	sprintf(vendor + 4, "%c%c%c%c", edx, (edx >> 8), (edx >> 16), (edx >> 24));
	sprintf(vendor + 8, "%c%c%c%c", ecx, (ecx >> 8), (ecx >> 16), (ecx >> 24));
	vendor[12] = 0x00;
}

static inline void cpuid_hv_vendor_00(char * vendor) {
	int ebx = 0, ecx = 0, edx = 0;

	__asm __volatile("cpuid" \
		: "=b"(ebx), \
		"=c"(ecx), \
		"=d"(edx) \
		: "a"(0x40000000));
	sprintf(vendor, "%c%c%c%c", ebx, (ebx >> 8), (ebx >> 16), (ebx >> 24));
	sprintf(vendor + 4, "%c%c%c%c", ecx, (ecx >> 8), (ecx >> 16), (ecx >> 24));
	sprintf(vendor + 8, "%c%c%c%c", edx, (edx >> 8), (edx >> 16), (edx >> 24));
	vendor[12] = 0x00;
}

static inline void cpuid_brand(char * brand, uint32_t eax_value) {
	int eax = 0, ebx = 0, ecx = 0, edx = 0;

	__asm __volatile{ "cpuid" \
		: "=a"(eax), \
		"=b"(ebx), \
		"=c"(ecx), \
		"=d"(edx) \
		: "a"(eax_value) };
	sprintf(brand, "%c%c%c%c", eax, (eax >> 8), (eax >> 16), (eax >> 24));
	sprintf(brand + 4, "%c%c%c%c", ebx, (ebx >> 8), (ebx >> 16), (ebx >> 24));
	sprintf(brand + 8, "%c%c%c%c", ecx, (ecx >> 8), (ecx >> 16), (ecx >> 24));
```

```
	sprintf(brand + 12, "%c%c%c%c", edx, (edx >> 8), (edx >> 16), (edx >> 24));
}

static inline int cpuid_hv_bit() {
	int ecx;
	__asm __volatile("cpuid" \
		: "=c"(ecx) \
		: "a"(0x01));
	return (ecx >> 31) & 0x1;
}

int cpu_rdtsc() {
	int i;
	unsigned long long avg = 0;
	for (i = 0; i < 10; i++) {
		avg = avg + rdtsc_diff();
		Sleep(500);
	}
	avg = avg / 10;
	return (avg < 750 && avg > 0) ? FALSE : TRUE;
}

int cpu_rdtsc_force_vmexit() {
	int i;
	unsigned long long avg = 0;
	for (i = 0; i < 10; i++) {
		avg = avg + rdtsc_diff_vmexit();
		Sleep(500);
	}
	avg = avg / 10;
	return (avg < 1000 && avg > 0) ? FALSE : TRUE;
}

int cpu_hv() {
	return cpuid_hv_bit() ? TRUE : FALSE;
}

void cpu_write_vendor(char * vendor) {
	cpuid_vendor_00(vendor);
}

void cpu_write_hv_vendor(char * vendor) {
	cpuid_hv_vendor_00(vendor);
}

void cpu_write_brand(char * brand) {
	int eax;
	/* Check if Processor Brand String is supported */
	__asm(".intel_syntax noprefix;"
	"mov eax, 0x80000000;"
		"cpuid;"
		"cmp eax, 0x80000004;"
		"xor eax, eax;"
		"setge al;"
		".att_syntax;" : "=a"(eax),
		);
	/* It's supported, so fill char * brand */
	if (eax) {
		cpuid_brand(brand, 0x80000002);
		cpuid_brand(brand + 16, 0x80000003);
		cpuid_brand(brand + 32, 0x80000004);
		brand[48] = 0x00;
	}
}
```

```
int cpu_known_vm_vendors() {
	const int count = 6;
	int i;
	char cpu_hv_vendor[13];
	string strs[count];
	strs[0] = "VMwareVMware"; /* VMware */
	strs[1] = "VBoxVBoxVBox"; /* VirtualBox */
	cpu_write_hv_vendor(cpu_hv_vendor);
	for (i = 0; i < count; i++) {
		if (!memcmp(cpu_hv_vendor, strs[i], 12)) return TRUE;
	}
	return FALSE;
}
```

A.3. intro.cpp

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ws2tcpip.h>
#include <windows.h>
#include "intro.h"
#include "stdafx.h"


void print_header() {
	printf("-------------------------------------------------------\n");
	printf("* VM Detection (TFG Jose Javier Gomez Lastra) *\n");
	printf("Algunos trucos antiVirtualMachine!\n");
	printf("-------------------------------------------------------\n\n");
}
```

A.4. utilidades.cpp

```
#include "stdafx.h"
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <Iphlpapi.h>
#include <Assert.h>
#pragma comment(lib, "iphlpapi.lib")
#include "utilidades.h"
#include "dirent.h"
#include <sys/types.h>
#include <sys/stat.h>

typedef BOOL(WINAPI *IsWow64ProcessProto) (HANDLE, BOOL*);

unsigned short init_cmd_colors() {
	CONSOLE_SCREEN_BUFFER_INFO csbi;
	HANDLE handler = GetStdHandle(STD_OUTPUT_HANDLE);
	// Get original console colors
	GetConsoleScreenBufferInfo(handler, &csbi);
	SetConsoleTextAttribute(handler, FOREGROUND_INTENSITY);
	// Return original console colors
	return csbi.wAttributes;
}

void encontrado() {
```

```
	HANDLE handler = GetStdHandle(STD_OUTPUT_HANDLE);
	SetConsoleTextAttribute(handler, 207);
	printf("ENCONTRADO!\n");
	SetConsoleTextAttribute(handler, FOREGROUND_INTENSITY);
}

void no_encontrado() {
	HANDLE handler = GetStdHandle(STD_OUTPUT_HANDLE);
	SetConsoleTextAttribute(handler, 10);
	printf("NO ENCONTRADO\n");
	SetConsoleTextAttribute(handler, FOREGROUND_INTENSITY);
}

void restore_cmd_colors(unsigned short original_colors) {
	HANDLE handler = GetStdHandle(STD_OUTPUT_HANDLE);
	// Restore original console colors
	SetConsoleTextAttribute(handler, original_colors);
}

/*
**	Función	utilizada	para	obtener	la	dirección	MAC	de
https://social.msdn.microsoft.com/Forums/vstudio/en-US/a96ec13d-a431-4499-b396-
235f33a83721/cant-find-a-way-to-gather-the-mac-address?forum=vcgeneral
** editada un poco para no mostrar toooooda la MAC siempre
*/

char* getMAC() {
	PIP_ADAPTER_INFO AdapterInfo;
	DWORD dwBufLen = sizeof(AdapterInfo);
	char *mac_addr = (char*)malloc(17);
	char *iniciomac = (char*)malloc(6);

	AdapterInfo = (IP_ADAPTER_INFO *)malloc(sizeof(IP_ADAPTER_INFO));
	if (AdapterInfo == NULL) {
		printf("Error	al	asignar	la	memoria	necesaria	para	llamar	a
GetAdaptersinfo\n");

	}

	// Make an initial call to GetAdaptersInfo to get the necessary size into
the dwBufLen	variable
	if (GetAdaptersInfo(AdapterInfo, &dwBufLen) == ERROR_BUFFER_OVERFLOW) {

		AdapterInfo = (IP_ADAPTER_INFO *)malloc(dwBufLen);
		if (AdapterInfo == NULL) {
			printf("Error al asignar la memoria necesaria para llamar a
GetAdaptersinfo\n");
		}
	}

	if (GetAdaptersInfo(AdapterInfo, &dwBufLen) == NO_ERROR) {
		PIP_ADAPTER_INFO pAdapterInfo = AdapterInfo;// Contains pointer to
current adapter info
		do {
			sprintf(mac_addr, "%02X:%02X:%02X:%02X:%02X:%02X",
				pAdapterInfo->Address[0], pAdapterInfo->Address[1],
				pAdapterInfo->Address[2], pAdapterInfo->Address[3],
				pAdapterInfo->Address[4], pAdapterInfo->Address[5]);
			printf("MAC: %s\n", mac_addr);
			sprintf(iniciomac, "%02X%02X%02X", pAdapterInfo->Address[0],
pAdapterInfo->Address[1], pAdapterInfo->Address[2]);
			return iniciomac;

			//printf("\n");
			pAdapterInfo = pAdapterInfo->Next;
```

```
		} while (pAdapterInfo);
	}
	free(AdapterInfo);
}

/*-------------------------------------------------------------*/

/*
** Función para comprobar que exista un fichero determinado.
** Para ello, simplemente que se pueda abrir demuestra que
** existe dicho fichero.
*/
bool existe_archivo(char *ruta) {
	bool result = false;
	FILE *archivo;
	archivo = fopen(ruta, "rb");
	if (archivo != NULL) {
		result = true;
		fclose(archivo);
	}
	return result;
}


/*
** Las siguientes tres funciones son para comprobar que existe una clave de registro, o un string dentro de dicha clave.
** Explicar una por una.
*/

// WOW64 es el emulador x86 que permite a las aplicaciones basadas en 32-bits correr igual en Windows de 64-bits.
//                                                  https://msdn.microsoft.com/es-es/library/windows/desktop/aa384249(v=vs.85).aspx
//                                                  https://msdn.microsoft.com/es-es/library/windows/desktop/aa384274(v=vs.85).aspx
bool iswow64() {
	int result = FALSE;

	IsWow64ProcessProto fniswow = (IsWow64ProcessProto)GetProcAddress(
		GetModuleHandleA("kernel32"), "IsWow64Process");

	return (fniswow) && (fniswow(GetCurrentProcess(), &result) != 0) ? result : FALSE;
}

// Esta es la función que uso para comprobar que exista la clave de registro. Como el uso de la función RegOpenKeyExA
// depende de si estamos en wow64 o no, lo comprobamos primero. Para ello la función anterior.
// Para comprobar si existe, tratamos de abrirlo. Si la función de abrir nos retorna un OK, es que el fichero existe.
bool exists_regkey(HKEY hKey, char *regkey_s) {
	HKEY regkey;
	LONG ret;

	if (iswow64()) {
		ret = RegOpenKeyExA(hKey, regkey_s, 0, KEY_READ | KEY_WOW64_64KEY, &regkey);
	}
	else {
		ret = RegOpenKeyExA(hKey, regkey_s, 0, KEY_READ, &regkey);
	}
	if (ret == ERROR_SUCCESS) {
		RegCloseKey(regkey);
```

```
		return TRUE;
	}
	else
		return FALSE;
}

// Similar a la anterior, solo que una vez que nos retorna OK, lo abrimos para
comprobar si está la cadena de
// caracteres que requiramos.
bool exists_regkey_str(HKEY hKey, char * regkey_s, char * value_s, char * lookup)
{
	HKEY regkey;
	LONG ret, ret1;
	DWORD size;
	char value[1024], *lookup_str;
	size_t lookup_size;

	lookup_size = strlen(lookup);
	lookup_str = (char *)(malloc(lookup_size + sizeof(char)));
	strncpy(lookup_str, lookup, lookup_size + sizeof(char));

	size = sizeof(value);
	if (iswow64()) {
		ret = RegOpenKeyExA(hKey, regkey_s, 0, KEY_READ | KEY_WOW64_64KEY,
&regkey);
	}
	else {
		ret = RegOpenKeyExA(hKey, regkey_s, 0, KEY_READ, &regkey);
	}
	if (ret == ERROR_SUCCESS) {
		//		RegQueryValueEx:		https://msdn.microsoft.com/es-
es/library/windows/desktop/ms724911%28v=vs.85%29.aspx?f=255&MSPPError=-2147217396
		ret1 = RegQueryValueExA(regkey, value_s, NULL, NULL, (LPBYTE)value,
&size);
		RegCloseKey(regkey);
		if (ret1 == ERROR_SUCCESS) {
			size_t i;
			for (i = 0; i < strlen(value); i++) { /* case-insensitive */
				value[i] = toupper(value[i]);
			}
			for (i = 0; i < lookup_size; i++) { /* case-insensitive */
				lookup_str[i] = toupper(lookup_str[i]);
			}
			if (strstr(value, lookup_str) != NULL) {
				free(lookup_str);
				return TRUE;
			}
		}
		else {
		}
	}
	else {
	}
	free(lookup_str);
	return FALSE;
}

/*-------------------------------------------------------------*/

/*
**	Código	obtenido	de:	https://msdn.microsoft.com/en-
us/library/ms686701(v=vs.85).aspx
** Editado para obtener únicamente el nombre de los procesos, no toda la información
adicional que venía con ello.
*/
```

```
#include <tlhelp32.h>

int comparar_procesos(char * compare)
{
	int e = 0;
	size_t newsize = strlen(compare) + 1;
	wchar_t * wcstring = new wchar_t[newsize];
	size_t convertedChars = 0;
	mbstowcs_s(&convertedChars, wcstring, newsize, compare, _TRUNCATE);

	HANDLE hProcessSnap;
	HANDLE hProcess;
	PROCESSENTRY32 pe32;
	DWORD dwPriorityClass;


	// Take a snapshot of all processes in the system.
	hProcessSnap = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);
	if (hProcessSnap == INVALID_HANDLE_VALUE)
	{
		return(FALSE);
	}

	// Set the size of the structure before using it.
	pe32.dwSize = sizeof(PROCESSENTRY32);

	// Retrieve information about the first process,
	// and exit if unsuccessful
	if (!Process32First(hProcessSnap, &pe32))
	{
		CloseHandle(hProcessSnap);          // clean the snapshot object
		return(FALSE);
	}

	// Now walk the snapshot of processes, and
	// display information about each process in turn
	do
	{

		if (wcscmp(pe32.szExeFile, wcstring) == 0) {
			printf("Buscando %s... ", compare);
			encontrado();
			e = 1;
		}

	} while (Process32Next(hProcessSnap, &pe32));

	if (e == 0) {
		printf("Buscando %s... ", compare);
		no_encontrado();
	}

	CloseHandle(hProcessSnap);
	return 0;
}

/*-------------------------------------------------------------*/
```

A.5. vbox.cpp

```
#include <stdio.h>
```

```
#include <stdlib.h>
#include <string.h>
#include <ws2tcpip.h>
#include <conio.h>
#include <windows.h>
#include "intro.h"
#include "stdafx.h"
#include "utilidades.h"
#include <wbemidl.h>
#include <winreg.h>
#include <cstdlib>

typedef char * string;

/* Detección a través de la dirección MAC */
void check_mac_vbox() {
	char *MAC_vbox;
	char *iniciomac;
	MAC_vbox = "080027";
	iniciomac = getMAC();

	if (strcmp(iniciomac, MAC_vbox) == 0) {
		encontrado();
	}
	else {
		no_encontrado();
	}

}
/*-----------------------------------------------------*/

/* Detección a través de los ficheros */
void check_files1_vbox() {
	const int cont = 13;
	string files[cont];


	files[0] = "C:\\WINDOWS\\System32\\vboxdisp.dll";
	files[1] = "C:\\WINDOWS\\System32\\vboxhook.dll";
	files[2] = "C:\\WINDOWS\\System32\\vboxmrxnp.dll";
	files[3] = "C:\\WINDOWS\\System32\\vboxogl.dll";
	files[4] = "C:\\WINDOWS\\System32\\vboxoglarrayspu.dll";
	files[5] = "C:\\WINDOWS\\System32\\vboxoglcrutil.dll";
	files[6] = "C:\\WINDOWS\\System32\\vboxoglerrorspu.dll";
	files[7] = "C:\\WINDOWS\\System32\\vboxoglfeedbackspu.dll";
	files[8] = "C:\\WINDOWS\\System32\\vboxoglpackspu.dll";
	files[9] = "C:\\WINDOWS\\System32\\vboxoglpassthroughspu.dll";
	files[10] = "C:\\WINDOWS\\System32\\vboxservice.exe";
	files[11] = "C:\\WINDOWS\\System32\\vboxtray.exe";
	files[12] = "C:\\WINDOWS\\System32\\VBoxControl.exe";

	for (int i = 0; i < cont; i++) {
		/* Intento leer los ficheros que indican una VM*/
		if (existe_archivo(files[i])) {
			printf("Buscando %s...", files[i]);
			encontrado();
		}
		else {
			printf("Buscando %s...", files[i]);
			no_encontrado();
		}
	}

}
/*-------------------------------------------------------------*/
```

```
/* Detección a través de las claves de registro */
void vbox_reg_key() {
	const int cont = 9;
	string regkeys[cont];
	regkeys[0] = "SOFTWARE\\Oracle\\VirtualBox Guest Additions";
	regkeys[1] = "HARDWARE\\ACPI\\DSDT\\VBOX__";
	regkeys[2] = "HARDWARE\\ACPI\\FADT\\VBOX__";
	regkeys[3] = "HARDWARE\\ACPI\\RSDT\\VBOX__";
	//
	regkeys[4] = "SYSTEM\\ControlSet001\\Services\\VBoxGuest";
	regkeys[5] = "SYSTEM\\ControlSet001\\Services\\VBoxMouse";
	regkeys[6] = "SYSTEM\\ControlSet001\\Services\\VBoxService";
	regkeys[7] = "SYSTEM\\ControlSet001\\Services\\VBoxSF";
	regkeys[8] = "SYSTEM\\ControlSet001\\Services\\VBoxVideo";

	for (int i = 0; i < cont; i++) {
		/* Intento leer los ficheros que indican una VM */
		if (exists_regkey(HKEY_LOCAL_MACHINE, regkeys[i])) {
			printf("Buscando %s... ", regkeys[i]);
			encontrado();
		}
		else {
			printf("Buscando %s... ", regkeys[i]);
			no_encontrado();
		}
	}
}
/*-------------------------------------------------------------*/

/* Función para buscar valores en las claves de registro */

void vbox_reg_key2() {
	const int cont = 4;
	string regkeys[cont], values[cont], lookup[cont];

	// SCSI registry key check
	regkeys[0] = "HARDWARE\\DEVICEMAP\\Scsi\\Scsi Port 0\\Scsi Bus 0\\Target Id
0\\Logical Unit Id 0";
	values[0] = "Identifier";
	lookup[0] = "VBOX";

	// SystemBiosVersion registry key check
	regkeys[1] = "HARDWARE\\Description\\System";
	values[1] = "SystemBiosVersion";
	lookup[1] = "VBOX";

	// VirtualBox Guest Additions key check
	regkeys[2] = "HARDWARE\\Description\\System";
	values[2] = "VideoBiosVersion";
	lookup[2] = "VIRTUALBOX";

	// HARDWARE\\DESCRIPTION\\System SystemBiosDate == 06/23/99
	regkeys[3] = "HARDWARE\\DESCRIPTION\\System";
	values[3] = "SystemBiosDate";
	lookup[3] = "06/23/99";

	for (int i = 0; i < cont; i++) {
		if  (exists_regkey_str(HKEY_LOCAL_MACHINE,  regkeys[i],  values[i],
lookup[i])) {
			printf("Buscando %s '%s'... ", regkeys[i], values[i]);
			encontrado();
		} else {
			printf("Buscando %s '%s'... ", regkeys[i], values[i]);
			no_encontrado();
```

```
		}
	}

}

void vbox_sysfile() {
	const int count = 4;
	string str[count];
	int res = FALSE, i = 0;
	char message[200];

	str[0] = "C:\\WINDOWS\\system32\\drivers\\VBoxMouse.sys";
	str[1] = "C:\\WINDOWS\\system32\\drivers\\VBoxGuest.sys";
	str[2] = "C:\\WINDOWS\\system32\\drivers\\VBoxSF.sys";
	str[3] = "C:\\WINDOWS\\system32\\drivers\\VBoxVideo.sys";
	for (i = 0; i < count; i++) {
		if (existe_archivo(str[i])) {
			printf("Buscando %s... ", str[i]);
			encontrado();
		} else {
			printf("Buscando %s... ", str[i]);
			no_encontrado();
		}
	}
}


/* Detección a través de los procesos */
void vbox_procesos() {
	const int contador = 2;
	string proc[contador];
	int j;
	proc[0] = "vboxservice.exe";
	proc[1] = "vboxtray.exe";
	for (j = 0; j < contador; j++) {
		comparar_procesos(proc[j]);
	}

}
/*--------------------------------------*/
```

A.6. utilidades.cpp

```
#include <windows.h>
#include "intro.h"
#include "stdafx.h"
#include "utilidades.h"
#include <wbemidl.h>
#include <winreg.h>
#include <cstdlib>
#include <string.h>

typedef char * string;

/* Detección a través de la dirección MAC */
void check_mac_vware() {
	char* iniciomac;
	const int cont = 4;
	string MAC_vware[cont];
	MAC_vware[0] = "000569";
```

```
	MAC_vware[1] = "000C29";
	MAC_vware[2] = "001C14";
	MAC_vware[3] = "005056";

	iniciomac = getMAC();
	for (int i = 0; i < cont; i++) {
		if (strcmp(iniciomac,MAC_vware[i]) == 0) {
			printf("Comprobando si la MAC empieza por %s... ",
MAC_vware[i]);
			encontrado();
		}
		else {
			printf("Comprobando si la MAC empieza por %s... ",
MAC_vware[i]);
			no_encontrado();
		}
	}
}
/*-------------------------------------------------------------*/

/**/

void check_files_vware() {
	const int conta = 10;
	string file[conta];

	file[0] = "C:\\WINDOWS\\System32\\Drivers\\vm3dgl.dll";
	file[1] = "C:\\WINDOWS\\System32\\Drivers\\vmdum.dll";
	file[2] = "C:\\WINDOWS\\System32\\Drivers\\vm3dver.dll";
	file[3] = "C:\\WINDOWS\\System32\\Drivers\\vmtray.dll";
	file[4] = "C:\\WINDOWS\\System32\\Drivers\\VMToolsHook.dll";
	file[5] = "C:\\WINDOWS\\System32\\Drivers\\vmmousever.dll";
	file[6] = "C:\\WINDOWS\\System32\\Drivers\\vmhgfs.dll";
	file[7] = "C:\\WINDOWS\\System32\\Drivers\\vmGuestLib.dll";
	file[8] = "C:\\WINDOWS\\System32\\Drivers\\VmGuestLibJava.dll";
	file[9] = "C:\\WINDOWS\\System32\\Drivers\\vmhgfs.dll";

	for (int i = 0; i < conta; i++) {
		/* Intento leer los ficheros que indican una VM*/
		if (existe_archivo(file[i])) {
			printf("Buscando %s...", file[i]);
			encontrado();
		}
		else {
			printf("Buscando %s...", file[i]);
			no_encontrado();
		}
	}
}
/*-------------------------------------------------------------*/

void vware_sysfile() {
	char *str;

	str = "C:\\windows\\System32\\Drivers\\Vmmouse.sys";
	if (existe_archivo(str)) {
		printf("Buscando %s... ", str);
		encontrado();
	}
	else {
		printf("Buscando %s... ", str);
		no_encontrado();
	}

}
```

```
/* Detección a través de los procesos */
void vware_procesos() {
	const int contador = 4;
	string proc[contador];
	int j;
	proc[0] = "Vmtoolsd.exe";
	proc[1] = "Vmwaretrat.exe";
	proc[2] = "Vmwareuser.exe";
	proc[3] = "Vmacthlp.exe";
	for (j = 0; j < contador; j++) {
		comparar_procesos(proc[j]);
	}

}
/*-------------------------------------*/

/* Detección a través de las claves de registro */
void vware_reg_key() {
	char *regkeys;
	regkeys = "SOFTWARE\\VMware, Inc.\\VMware Tools";


	/* Intento leer los ficheros que indican una VM */
	if (exists_regkey(HKEY_LOCAL_MACHINE, regkeys)) {
		printf("Buscando %s... ", regkeys);
		encontrado();
	}
	else {
		printf("Buscando %s... ", regkeys);
		no_encontrado();
	}

}
/*-----------------------------------------------------------------*/

/* Función para buscar valores en las claves de registro */

void vware_reg_key2() {
	const int cont = 3;
	string regkeys[cont];
	char *value;
	char *lookup;

	// SCSI registry key check
	regkeys[0] = "HARDWARE\\DEVICEMAP\\Scsi\\Scsi Port 0\\Scsi Bus 0\\Target Id
0\\Logical Unit Id 0";
	regkeys[1] = "HARDWARE\\DEVICEMAP\\Scsi\\Scsi Port 1\\Scsi Bus 0\\Target Id
0\\Logical Unit Id 0";
	regkeys[2] = "HARDWARE\\DEVICEMAP\\Scsi\\Scsi Port 2\\Scsi Bus 0\\Target Id
0\\Logical Unit Id 0";
	value = "Identifier";
	lookup = "VMWARE";

	if (exists_regkey_str(HKEY_LOCAL_MACHINE, regkeys[0], value, lookup) ||
		exists_regkey_str(HKEY_LOCAL_MACHINE, regkeys[1], value, lookup) ||
		exists_regkey_str(HKEY_LOCAL_MACHINE, regkeys[2], value, lookup)) {
		printf("Buscando %s '%s''%s'... ", regkeys[0], value, lookup);
		encontrado();
	}
	else {
		printf("Buscando %s '%s''%s'... ", regkeys[0], value, lookup);
		no_encontrado();
	}
```

```
}

void check_vware_version()
{
	unsigned int  a, b;

	__try {
		__asm {

			// save register values on the stack
			push eax
			push ebx
			push ecx
			push edx

			// perform fingerprint
			mov eax, 'VMXh'      // VMware magic value (0x564D5868)
			mov ecx, 0Ah         // special version cmd (0x0a)
			mov dx, 'VX'         // special VMware I/O port (0x5658)

			in eax, dx                  // special I/O cmd

			mov a, ebx                  // data
			mov b, ecx                  // data      (eax    gets    also
modified but will not be evaluated)

                                                    // restore register values
from the stack
                                                    pop edx
                                                    pop ecx
                                                    pop ebx
                                                    pop eax

		}
	}
	__except (EXCEPTION_EXECUTE_HANDLER) {}
	printf("Buscando la versión de VM Ware... ");
	if (a == 'VMXh') {          // is the value equal to the VMware magic value?
		encontrado();
		printf("\n");
	}
	else {
		no_encontrado();
		printf("\n");
	}

}
```

A.7. cpu.h

```
#ifndef CPU_H
#define CPU_H

int cpu_rdtsc();

int cpu_rdtsc_force_vmexit();

int cpu_hv();

void cpu_write_vendor(char *);
void cpu_write_hv_vendor(char *);
```

```
void cpu_write_brand(char *);

int cpu_known_vm_vendors();

#endif
```

A.8. intro.h

```
#pragma once
#ifndef COMM_H
#define COMM_H

void print_header();
#endif
```

A.9. utilidades.h

```
#pragma once
#include <wbemidl.h>

unsigned short init_cmd_colors();

void encontrado();

void no_encontrado();

void restore_cmd_colors(unsigned short original_colors);

char* getMAC();

bool existe_archivo(char *ruta);

bool iswow64();

bool exists_regkey(HKEY hKey, char *regkey_s);

bool exists_regkey_str(HKEY hKey, char * regkey_s, char * value_s, char * lookup);

int comparar_procesos(char * compare);
```

A.10. vbox.h

```
#pragma once

void check_mac_vbox();

void check_files1_vbox();

void vbox_reg_key();

void vbox_reg_key2();

void vbox_sysfile();

void vbox_procesos();
```

A.10. vmware.h

```
#pragma once

void check_mac_vware();

void check_files_vware();

void vware_sysfile();

void vware_procesos();

void vware_reg_key();

void vware_reg_key2();

void check_vware_version();
```