Armageddon walkthrough

Index

List of pictures

Disclaimer

I do this box to learn things and challenge myself. I'm not a kind of penetration tester guru who always knows where to look for the right answer. Use it as a guide or support. Remember that it is always better to try it by yourself. All data and information provided on my walkthrough are for informational and educational purpose only. The tutorial and demo provided here is only for those who're willing and curious to know and learn about Ethical Hacking, Security and Penetration Testing.

Reconnaissance

The results of an initial nMap scan are the following:

Picture 1 - nMap scan results

Open ports are 22 and 80. So, this machine has SSH enabled and an application running on port 80. Also, nMap recognized Linux as operative system and probably it is CentOS.
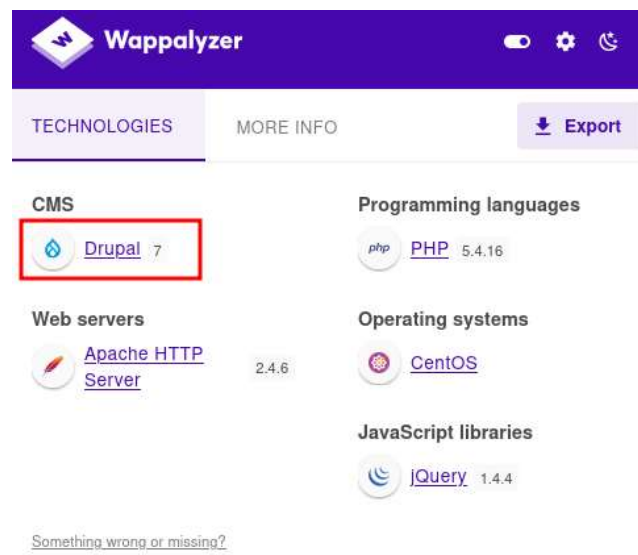
Initial foothold

First of all, I browsed to access to the application. Its index page was the following:
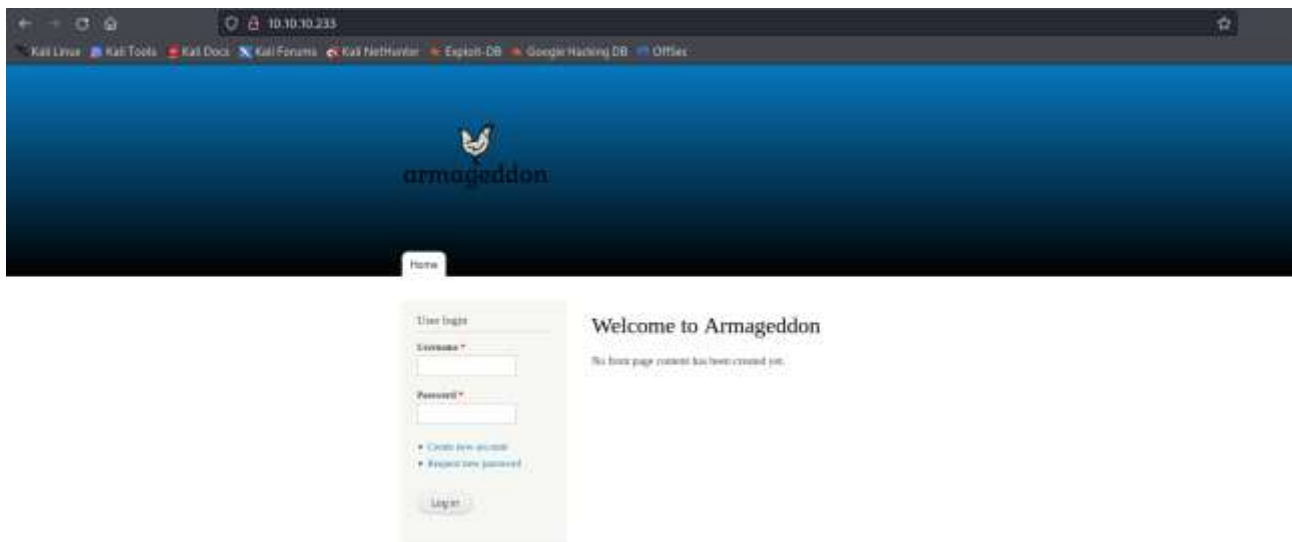
Picture 2 - Application on port 80

Using Wappalyzer extension tool for Firefox, I found that this application was based on Drupal version 7:

Picture 3 - Application details

Based on these information, I searched some useful exploit on Internet. I found the **CVE-2018-7600**. The root cause of this vulnerability is related to the Drupal theme rendering system. To create all of its UI elements, Drupal uses **Form API**, a powerful tool allowing developers to create forms and handle form submissions quickly and easily. To achieve this, the API uses a hierarchical **associative array** (**Render Array**, introduced in Drupal 7) containing the data that will be rendered, as well as some properties which establish how the data should be rendered. The associative array contains two elements (**first** parameter and **second** parameter), both have several parameters. A **parameter key** can be identified as it always starts with the hashtag **#** symbol. The **#type** parameter specifies the type of the HTML element (checkbox, textarea, etc.) and the **#markup** parameter is used to set HTML that will be output on the form. There are many other parameters that can be used with forms. Some of them provide a way to post-process the rendered output by re-parsing it through a user-supplied function. According to Drupal API documentation, this can be used to cache a view and still have some level of dynamic output. If the user-supplied callback function is not properly validated, a potential attacker might be able to insert malicious functions such as exec, system, eval, etc. to execute system commands, and take over the server.

User flag

Since I found a very interesting CVE to try to exploit, I downloaded the relative exploit code (called **Drupalgeddon2**) and run it against the target, as showed in the following picture:

Picture 4 - Exploit

In this way, I obtained a shell with the application user **apache**, but it hadn't the user flag. So, I checked the **/etc/passwd** file and I found the user **brucetherealadmin**:

Picture 5 - /etc/passwd file

Lateral movement

At this point, I tried to brute force his SSH credentials with the following command:

```
hydra -l brucetherealadmin -P /usr/share/seclists/Passwords/Leaked-Databases/rockyou-
75.txt 10.10.10.233 -t 4 ssh -f -V
```

In this way, I found the correct credentials:

Picture 6 - User password brute force

Honestly, I took these screenshots in a second moment and this command didn't work in acceptable time. So, I used a custom password list in this case. Anyway, I assure you that the password is included in **rockyou-75.txt** list file and that the command I inserted in my walkthrough worked. So, I connected in SSH with credentials just found and retrieved the user flag:

Picture 7 - SSH connection and user flag

Privilege escalation

Since I found the user flag, I had to escalate my privileges to root. The useful information to achieve this goal, and retrieve the root flag, was:

Picture 8 - Useful info to privesc

So, the user was able to run **snap install** command as root without password. I searched some exploits on Internet and I used the **dirty_sock** one. In particular, I prepared the following command to exploit this vulnerability:

Picture 9 - Prepare privesc exploit

This command created a **snap** file. I needed to install it to execute its code. In particular, this base64 coded code adds a **dirty_sock** user with password **dirty_sock**. I created this command based on https://github.com/initstring/dirty_sock exploit. All I needed to do was to install this snap file using:

```
sudo /usr/bin/snap install exploit.snap --devmode
```

became **dirty_sock** user:

```
su dirty_sock
```

and to spawn a root shell using the command:

```
sudo -i
```

So, I retrieved the root flag:

Picture 10 - Privilege escalation and root flag