Brandon T. Elliott

04-05-2024

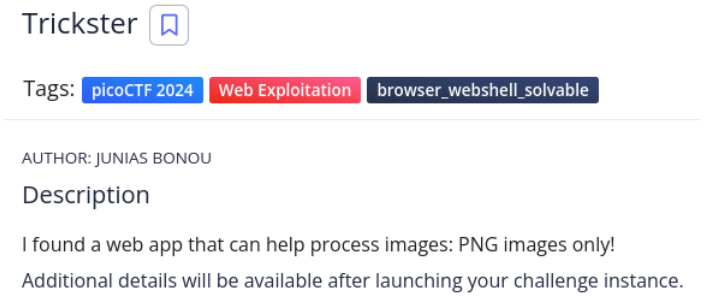
CTF Writeup: picoCTF 2024 - "Trickster"

The CTF

picoCTF 2024 took place from March 12th, 2024 to March 26th, 2024.

The Challenge

This web exploitation challenge began with the following description:



PNG images only, huh?… interesting…

The Web App

After starting the challenge instance, we navigate to the web app and see a simple page with only one functionality, which is to upload a PNG file, specifically.

The First Payload

Wappalyzer indicates that the web server is using PHP.

We can first test the logic of the uploads by writing a small php cmd payload to a php file called `yo.php` and attempt to upload it in order to see how the web app responds.

```
┌──(kali㉿kali)-[~/picoctf2024]
└─$ echo '<?=`$_GET[0]`?>' > yo.php

┌──(kali㉿kali)-[~/picoctf2024]
└─$ cat yo.php
<?=`$_GET[0]`?>
```

I start up Burp Suite and enable FoxyProxy within Firefox to ensure the request will be captured by Burp Suite for further usage later.

With requests going through the Burp Suite proxy, I then try to upload the php file, and receive the following `Error` response from the web server:

Welcome to my PNG processing app

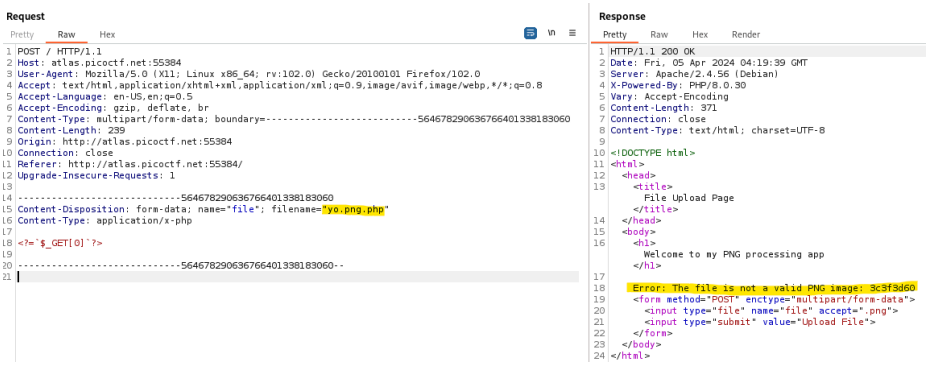
Error: File name does not contain '.png'.

Browse... No file selected. Upload File

Based on the error message, the web server appears to be checking that the file we upload contains `.png` in the file name.

However, `"does not contain"` in this instance implies that the logic may not be sound here, as it likely means that it doesn't require it to be at the *end* of the file name.

We will test this by using Burp Suite to send the previous request to `Repeater` and modify the file name within the request.



In this attempt, we modified the filename to be `yo.png.php` instead.

After sending the modified request, we indeed get a different response from the web server, indicating that it did accept the new file name, but that it still didn't accept the uploaded file due to a new error: `"The file is not a valid PNG image"`.

Magic Bytes

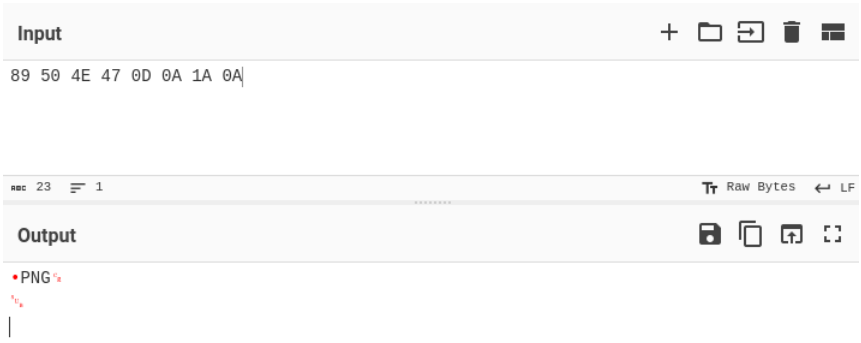
The new error message indicates that the web server is likely checking the file signature, or "magic bytes" of the file to ensure that it matches what it would expect from a PNG file.

Wikipedia has a [useful list of file signatures] that we can use which indicates that the magic bytes (in hex) for a PNG file is `89 50 4E 47 0D 0A 1A 0A`.

Using this along with the previous information, we can now craft a new php payload to attempt to bypass the file restrictions once more.

The Final Payload

First, we need to convert the hex bytes of the PNG file signature using CyberChef in order to download a "template", more or less, for the first part of our final payload.



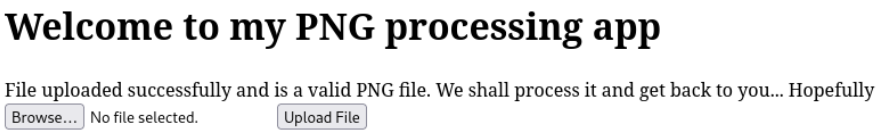
Using the `From Hex` conversion in CyberChef, we can convert the PNG magic bytes from hex to binary data and then click the floppy disk icon to export this to a file.

We then move the downloaded file to a file called `yo.png.php` and use a text editor to add our php payload from earlier `<?=`$_GET[0]`?>` to the end of the file contents.

All in all, our final payload should look something like this, with the PNG magic bytes at the beginning, and our php payload after it:

Success

After uploading our final payload using the web app, we can see that the upload succeeded:



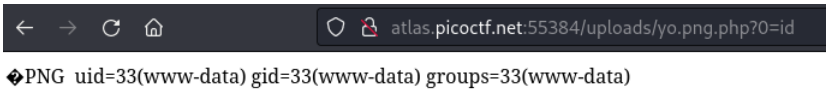
However, it doesn't indicate the directory it uploaded it to, or how it's "processing" it.

Code Execution

Using some educated guesses, we can check the web server for the uploaded file under some common directories such as `/images`, `/files`, `/uploads`, etc.

`/uploads` returns a response of `403 Forbidden` indicating that this is likely the directory, and we indeed find that it did upload the file to `/uploads/yo.png.php`.

Therefore, we now have code execution via our php payload, utilizing `?0` as a parameter for our commands which will be executed by the web server.

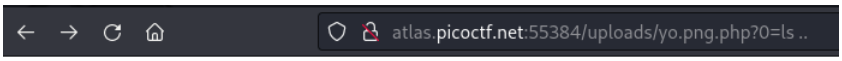


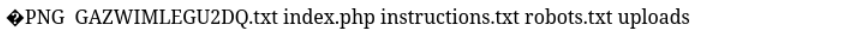
An `id` shows that we are www-data.

We could get a reverse shell here, but this isn't really necessary for the challenge, so let's just explore a bit.

`cat flag.txt` resulted in nothing and `ls` showed that only our php file is in the current directory, so we do have to look a little bit deeper.

An `ls ..` shows that there is a `GAZWIMLEGU2DQ.txt` file in the web server's root directory…





…which seems to not belong there, so we cat this file, and receive the flag.

Editor’s Note

I realized after looking at the `ls ..` screenshot above when doing the writeup that you were actually intended to look at `robots.txt` early on in the challenge (which is common practice for web apps)…

```
User-agent: *
Disallow: /instructions.txt
Disallow: /uploads/
```

which would have clued you in to the `/uploads` directory and the `instructions.txt` file, which contained…

```
Let's create a web app for PNG Images processing.
It needs to:
Allow users to upload PNG images
        look for ".png" extension in the submitted files
        make sure the magic bytes match (not sure what this is exactly but wikipedia says that the first few bytes contain 'PNG' in hexadecimal: "50 4E 47" )
after validation, store the uploaded files so that the admin can retrieve them later and do the necessary processing.
```

… which would have been helpful, but I think the way I solved it without finding these hints is more in line with what you would expect from real life, so I’m going to leave the writeup the way it is.