



CTF WRITEUP



Author : Tushar (benzo)



Challenge: sasti



Concept: ssti flask

In this particular challenge we are given a site running... and when we get to that site we will see something like this -

```
#!/usr/bin/env python3

from flask import Flask, render_template_string, request, Response

app = Flask(__name__)
app.config.from_pyfile('topsecret.py')
@app.route('/')
def index():
    return Response(open(__file__).read(), mimetype='text/plain')

@app.route('/vuln')
def vuln():
    query = request.args['query'] if 'query' in request.args else 'eh?'
    if len(query) > 21:
        return "nononono"
    return render_template_string(query)

app.run('0.0.0.0', 1337)
```

Ehh?? I know it's irritating, I mean where's the hint ? where is the flag hidden? What can I infer from the above text getting displayed?

Let me get you through this.

So to start with the solution there is some prerequisite knowledge that you must have -

1. Python (ofc becoz the above text getting displayed is a python code in case you don't know)
2. How websites work ? (idea of frontend, backend and web servers)
3. Flask
(sounds fancy, not that chemistry experiment wala flask)
(Basically it's a python framework[library/module] for building web application)
4. Jinja2 and SSTI in flask
(don't worry just fucking easy things, you'll know soon)

Let's briefly discuss each of these things and proceed to find the flag... 🚀

Python

(nam to suna hoga, if not then read this section)

Python is a high-level, interpreted programming language known for its simplicity, readability, and versatility. Created by Guido van Rossum and first released in 1991, Python has become one of the most popular programming languages worldwide, used in various domains including web development, data science, automation, artificial intelligence, scientific computing, and more.

Basic idea of python for this challenge is enough for you. You just need to be able to identify that the code we are seeing is written in python.

Later things and python knowledge that are used in this challenge will be mentioned as we proceed 😊

How websites work?

Website or you may say web app, is....

Wait you know what it is fosho.. 😎

Ever tried hitting google.com or youtube.com or instagram.com on your bloody device?

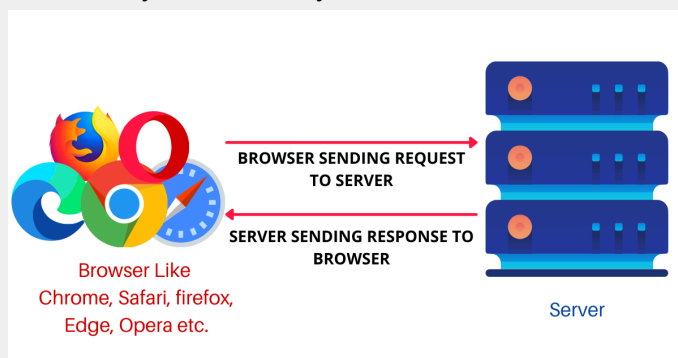
You know when you hit a particular domain name on your browser you get some content getting displayed.

HOW the FUCK that WORKS?

Every website you are visiting is running on another computer (we call it a webserver), which is exposed to this world and anyone can access that using it's IP address (ofc there are some restriction, you can't access everything), an IP address is a unique identifier of a computer over a network which is a shitty code like thing for eg - 192.168. 123.132, this is what an IP address looks like, and a domain name (something.com) is linked to that particular IP address of a web server so that we can access our site using that domain name, without the headache of remembering that loooong IP address.

For simplicity, consider IP address as coordinate (longitude, latitude) of your house and domain name as your house address.

Now what you do when you access a website?



You as a client use your browser to send requests to the web server of a particular website and in response to the request you send you get what's running on that website.

Frontend or backend?

Now the website majorly consist of two parts - frontend and backend.



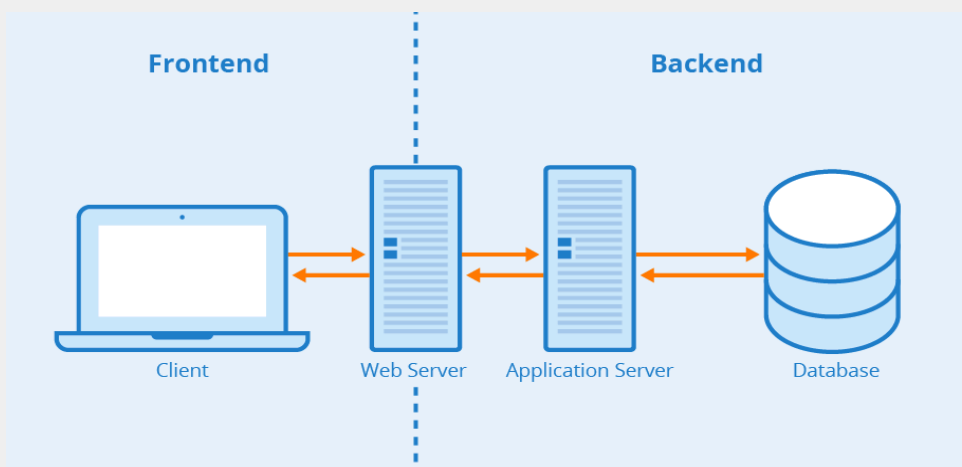
Frontend

In simple words frontend is that part of website that you see, it consist of all the user interface of website that you see, the way you interact and the way website functions from pov of user. Frontend of a website generally made using HTML, CSS and Javascript (Don't worry if you don't know this)

Backend

It is that part of the website that you can't see how a site works, how it stores data on a database, how it keeps files, how it performs logical operations all is part of the backend. Backends are made through different sets of programming languages such as Node.js, Python, Java and PHP. (except python you may don't know others, not an issue)

A nice combination and stable communication between frontend and backend is important for a good website.



Is your website static or dynamic?

On the basis of functionality websites can be static or dynamic -

- Static sites display static content such as plain text, images etc.
 - We use static websites when we only have to show some data or information in our websites, it directly loads website files from the web server without worrying about further communication with the backend, as there is no role of things like backend, file storage, user authentication.
 - Example of static site is - a site showing resume/portfolio of a person
- Dynamic sites are different from static one
 - Dynamic sites contain all the functionalities such as data/file storage, user authentication, maybe payment checkouts and more, it not only displays content only but also functions as per the action taken by client/user.
 - Example - google, facebook or instagram all sites are dynamic

That's more than enough knowledge of websites that you need, we can move forward now!!

>>>>>

Flask

Flask is a framework for web development, it is python based.

Now you know when we go to the challenge link we get some code on our screen -

```
#!/usr/bin/env python3

from flask import Flask, render_template_string, request, Response

app = Flask(__name__)
app.config.from_pyfile('topsecret.py')
@app.route('/')
def index():
    return Response(open(__file__).read(), mimetype='text/plain')

@app.route('/vuln')
def vuln():
    query = request.args['query'] if 'query' in request.args else 'eh?'
    if len(query) > 21:
        return "nononono"
    return render_template_string(query)

app.run('0.0.0.0', 1337)
```

First of all the above code is written in python and if we try to understand it it basically uses flask to create a simple web server.

Let's get line by line understanding of code

```
from flask import Flask, render_template_string, request, Response
```

Firstly it imports our flask library and some basic flask functions that the program is going to use

```
app = Flask(__name__)
app.config.from_pyfile('topsecret.py')
```

Creating our flask app and defining the configuration file.

Configuration file is used to store settings and parameters that configure how the Flask application behaves, here all the configurations are defined in a separate file 'topsecret.py' and it is utilising that file for configurations. Config files generally contain some keys, secret, environment variables and database configurations.

```
@app.route('/')
def index():
    return Response(open(__file__).read(), mimetype='text/plain')
```

Here it is defining what to do when client hits '/' route

In the context of websites, **routes** refer to the paths defined within a web application that determine how URLs are mapped to specific functions or actions within the server. When a user visits a URL, the route determines which part of the code should be executed to handle the request and generate a response.

Now when a user hits '/' route on the challenge website in response it is getting some code.

```
return Response(open(__file__).read(), mimetype='text/plain')
```

Because it is already defined there that when a user hits the '/' URL on challenge site in response it will get the content of the main python file in text format in which our web server is defined.

That means the code we get in return is the same code of the web server itself.

(__file__ is used to refer to the current file)

The function is opening the current file, reading its content and sending the same content of the python file as response in text format.

```
@app.route('/vuln')
def vuln():
    query = request.args['query'] if 'query' in request.args else 'eh?'
    if len(query) > 21:
        return "nononono"
    return render_template_string(query)
```

Now we know that the code we are seeing on hitting '/' route is code of web server itself, so looking down below in the code we could know that there also exist also a '/vuln' route

Let's try to hit '/vuln' route on challenge site -

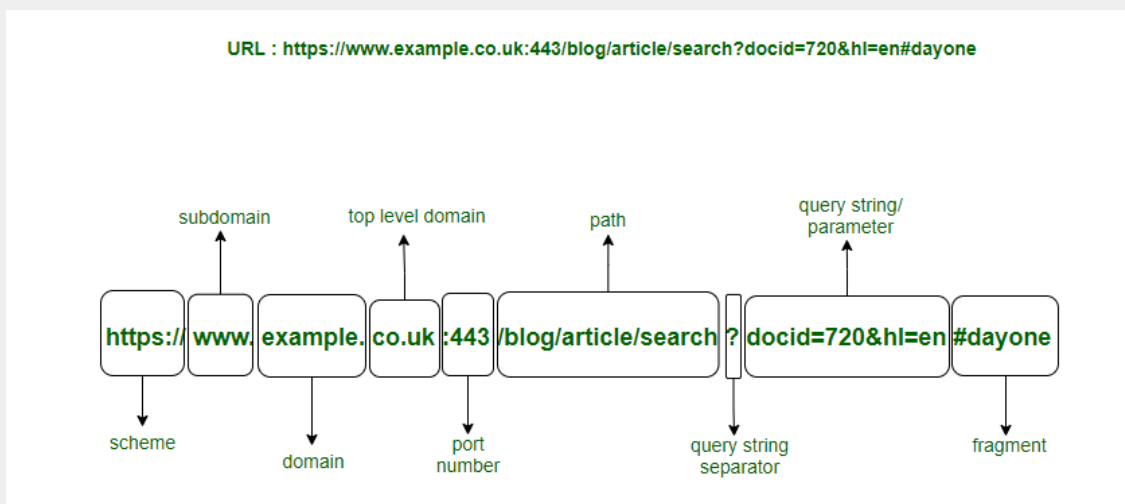
eh?

What's this response.... Eh?

Let's deep dive into the function defined for '/vuln' route...

```
query = request.args['query'] if 'query' in request.args else 'eh?'
if len(query) > 21:
    return "nononono"
return render_template_string(query)
```

query here is storing the text that we are passing as a query parameter in our URL.



Here `request.args` stores all the parameters that we are passing with URL. Let's see how to pass parameters along with URL

`/vuln` - `request.args` is empty

`/vuln?x=y` - `request.args['x']` stores the value 'y'

`/vuln?x=y&n=m` - `request.args['x']` is 'y' and `request.args['n']` is 'm'

Now we have to pass a query as our parameter in the URL.

`/vuln?query=<sometext>` - `request.args['query']` stores value `<sometext>`

```
query = request.args['query'] if 'query' in request.args else 'eh?'
```

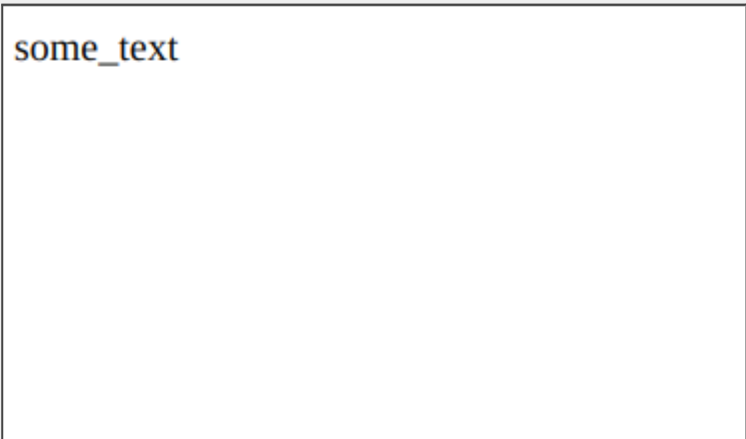
Now the value stored in the query variable is `'eh?'` If no parameter named query is passed along with the URL, otherwise if we pass something as a query parameter in the URL we will see that same text getting displayed.

`/vuln` - No query parameter



eh?

`/vuln?query=some_text` - query parameter with value `some_text`



some_text

There is also a case checking later -

```
if len(query) > 21:  
    return "nononono"  
return render_template_string(query)
```

If the query parameter passed exceeds a max length of 21 character then we will see `'nononono'` in response otherwise the same query parameter will be rendered as a string in response (That's why we are able to see what we are passing as query in response)

Let's see again...

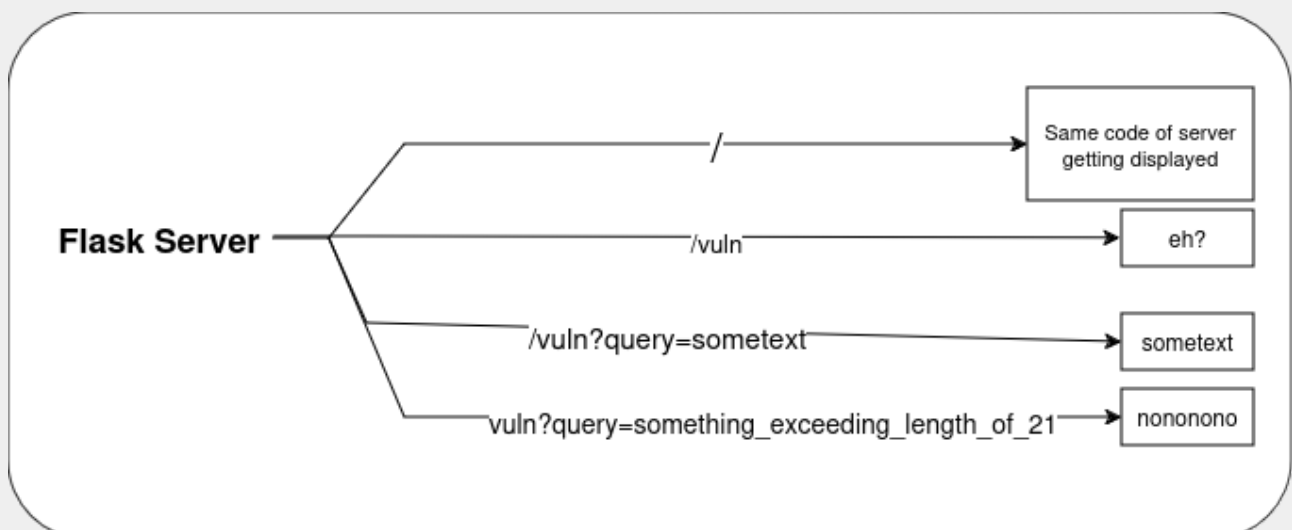
`‘/vuln?query=hello_world’`

hello_world

`‘vuln?query=something_exceeding_length_of_21’` - when query parameter has a length > 21

nononono

Finally we are done with all flask code, given below is a diagram for better understanding of given flask code (checkout for reference)



Jinja2 and SSTI in flask

Let's dive into the main exploitation part, how can we exploit the python code and get our flag.

First of all, where the hell could the flag possibly be???

Closely looking at code (that we get on '/' route on challenge link) we could analyse that there is a high probability of the flag to be hidden inside the configuration files because most secret keys are hidden in the configuration files.

So we need to look at the file which is storing all the configuration to get our flag. Right na ??

Is it topsecret.py??

Yes. But any way of reading it ?? may be in another universe

Without access to the web server in which these files are stored we can't read the config stored in topsecret.py. 😞

What to do now?

Let's play with some other solution to read configuration in a flask app.

Now flask app can render HTML dynamically with the help of template engines (like Jinja2)

Jinja2 is a template engine for flask applications.

A **template engine** is a software component used in web development to combine templates with data to produce documents, typically HTML pages. The template contains placeholders or special syntax that is replaced with actual data when the template is processed by the engine. This allows developers to create dynamic web pages or other text-based documents in a clean and maintainable way.

Given below we have an example of a simple flask app that renders a template with some predefined data passed along with

```
from flask import Flask, render_template

app = Flask(__name__)

@app.route('/')
def home():
    return render_template('home.html', username='Benzo')

app.run(debug=True)
```

Now this above code, if we run start a simple flask server locally and render a template home.html on '/' route, which is as follows -

```
<html>
<head>
  <title>Home</title>
</head>
<body>
  <h1>Welcome, {{ username }}!</h1>
</body>
</html>
```

The above code is a simple page that will show the text 'Hello Benzo!', because when we are rendering our template through flask app the value of parameter *username* that we passed along with *render_template* function will be used by template for showing username.

Here we can say that the value of username is not static, it is dynamically loaded from the server and after that the website is showing the username, in real cases the data that is getting rendered is usually loaded from a database or something else.

In this way template engine works, for this challenge we must know to while rendering template to show dynamically loaded values (such as username in above example) we use syntax like this in Jinja2, in our html template -

`{{ <name_of_parameter_passed_along_with_response> }}`

Use `{{ }}` double curly brackets.

Now let's jump to the final step which is using SSTI for solving this problem.

SSTI stands for Server Side Template Injection, It is a type of security vulnerability that occurs when user input is unsafely injected into a server-side template. Using this we can inject malicious code to the template, and execute some dangerous code on the server.

By far our task is to investigate the flask app config to search for the flag, but we can't view the topsecret.py file directly. The better way of viewing the config is to make use of templating techniques and inject a well designed input in the query parameter of the vuln route that could help us to breach the flask app and give access to the app configs.

If you search hard on google, you will know that by default app config are accessible in template engines like jinja2 by making use of *config* object.

In Jinja2 config object can be accessed in this way -
`{{config}}`

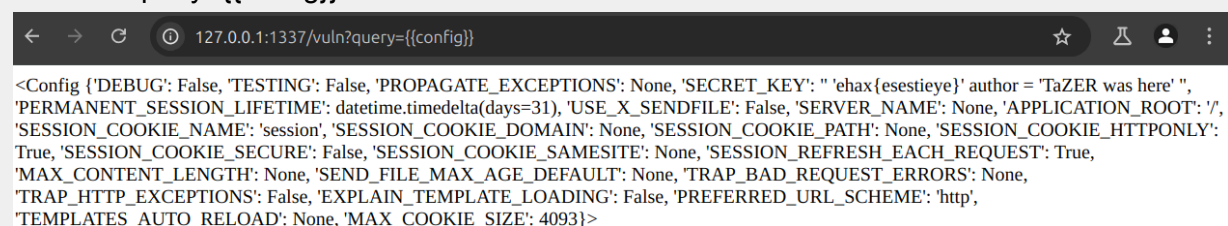
So if we pass the above text in /vuln route as a query parameter to our main challenge flask app it should work fine and display all the app configurations.

```
@app.route('/vuln')
def vuln():
    query = request.args['query'] if 'query' in request.args else 'eh?'
    if len(query) > 21:
        return "nononono"
    return render_template_string(query)
```

From the above code we can say that text passed in the query parameter of /vuln route is rendered as a template string and in a template string we can use Jinja2 syntax, to get our app configuration, where the flag might be stored.

Let's do it...

Hit /vuln?query={{config}}



```
<Config {'DEBUG': False, 'TESTING': False, 'PROPAGATE_EXCEPTIONS': None, 'SECRET_KEY': "'ehax(esestieye)'" author = 'TaZER was here' ", 'PERMANENT_SESSION_LIFETIME': datetime.timedelta(days=31), 'USE_X_SENDFILE': False, 'SERVER_NAME': None, 'APPLICATION_ROOT': '/', 'SESSION_COOKIE_NAME': 'session', 'SESSION_COOKIE_DOMAIN': None, 'SESSION_COOKIE_PATH': None, 'SESSION_COOKIE_HTTPONLY': True, 'SESSION_COOKIE_SECURE': False, 'SESSION_COOKIE_SAMESITE': None, 'SESSION_REFRESH_EACH_REQUEST': True, 'MAX_CONTENT_LENGTH': None, 'SEND_FILE_MAX_AGE_DEFAULT': None, 'TRAP_BAD_REQUEST_ERRORS': None, 'TRAP_HTTP_EXCEPTIONS': False, 'EXPLAIN_TEMPLATE_LOADING': False, 'PREFERRED_URL_SCHEME': 'http', 'TEMPLATES_AUTO_RELOAD': None, 'MAX_COOKIE_SIZE': 4093}>
```

Hurray!!!

We got our app configuration from this random looking shit text if we observe with naked eye we can see a "SECRET_KEY" value is hidden in it with a value of `ehax{esestieye}`.

Yoo, we got our flag for the challenge.
Congratulations to me.

Flag is : `ehax{esestieye}`

Learning: Making use of SSTI in flask to design input such that we can exploit the written code.

Thanks for reading.