Copy of Let's build an API to hack - Part 5: Emulating wonky login systems to get broken authentication issues and injection flaws

Introduction

For the issue type "Broken authentication" There are many things that can go wrong but i wanted to show you that broken authentication

Requirements

- A patato ... Seriously though, a small VPS or spare computer with the minimal amount of RAM and disk space will do. The APIs we will be building do not require much.
- Python 3.x (https://www.python.org/downloads/)
- Flask (pip install Flask but hold off if you did not do it yet, we will be creating a python virtual env)

Let's set up

To start with, we will need to set up a virtual environment first. This is a place we can install our dependencies of a certain project on and keep them seperate from the other projects. This is very useful to keep oversight but also if you have one project that requires a certain version of an import while another project might need a much older and non-compatible version of that library.

```
mkdir "GoudAPI-infodisclosure"
cd GoudAPI-infodisclosure
python3 -m venv GoudAPI-infodisclosure
```

```
mkdir GoudAPI-infodisclosure
cd GoudAPI-infodisclosure
py -3 -m venv GoudAPI-infodisclosure
```

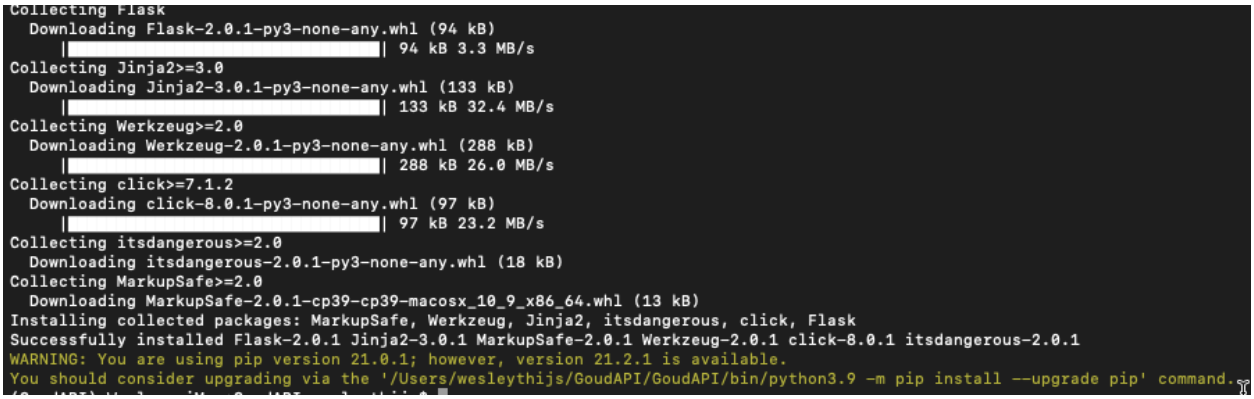
With these commands we are creating a venv (virtual enviornment) called GoudAPI-BAC which is marked by a new folder, now we have to swith to it.

```
. GoudAPI-infodisclosure/bin/activate
```

```
GoudAPI-infodisclosure\Scripts\activate
```

And now we can easily use pip to install flask

```
pip install Flask
```



Now that flask is installed, we can easily create our first vulnerable API.

Hello World

We will first need to write a few lines of code to tell python it should start a flask web application.

```
import flask
from flask import request

app = flask.Flask(__name__)
app.config["DEBUG"] = True
```

These lines will tell python to import flask and to set it up with debugging enabled. app.config["DEBUG"] = True refers to a variable from the flask config. We could set up a flask config file seperatly but let's keep it all in one place for now.

We also want to import request for later since our first API will only return static data so let's build up that static data with the following code.

```
@app.route('/', methods=['POST'])
def api_home():
     return "paths '/api/v1/changeUserSettings'"
```

By adding this basepath, we will allow the user to see the endpoints, there will only be only so we display it and we can move on the next path.

```
# A route to return all of the available entries in our admin log.
@app.route('/api/v1/changeUserSettings', methods=['POST'])
def api_cards():
    if 'username' in request.args:
        name = request.args['name']
    else:
        return "Error: No username field provided. Please specify an accountType,name,firs
tname and adress."

    if 'name' in request.args:
        name = request.args['name']
    else:
        return "Error: No name field provided. Please specify an accountType,name,firstnam
e and adress."

    if 'firstname' in request.args:
        name = request.args['name']
    else:
        return "Error: No firstname field provided. Please specify an accountType,name,fir
stname and adress."

    if 'adress' in request.args:
        name = request.args['name']
    else:
        return "Error: No adress field provided. Please specify an accountType,name,firstn
ame and adress."

    if 'accountType' in request.args:
        type = request.args['accountType']
    else:
        return "Error: No type field provided. Please specify an accountType,name,firstnam
e and adress. The type can be either user or reader"

    if type == 'admin':
        return "Good job!! Flag: (21384324-03240324)"
    else:
        return "account settings saved"
```

This is where the cheating really begins because as you can the user never really gets updated. The server also returns some arguments and actually shows you exactly what parameters are required which you would need a swagger for IRL but this is just the beginning of our easy APIs.

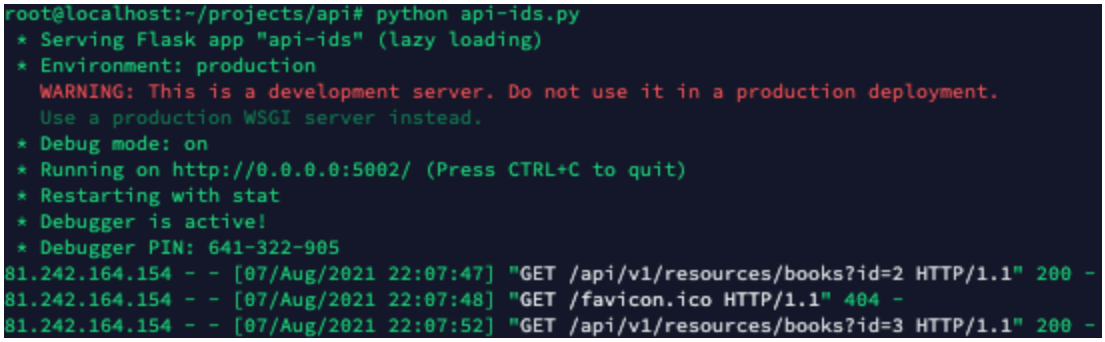
```
app.run(host="0.0.0.0",port="5006")
```

Combine all of this into one script and it will start up when we ask it, one last thing i want to highlight here is the host="0.0.0.0" argument. We do not need this but if we do not give it this argument, the API will only be accesible from localhost. The default port for flask will be 5000 but we told our script to run on port 5006 so make sure nothing else is running on that port.

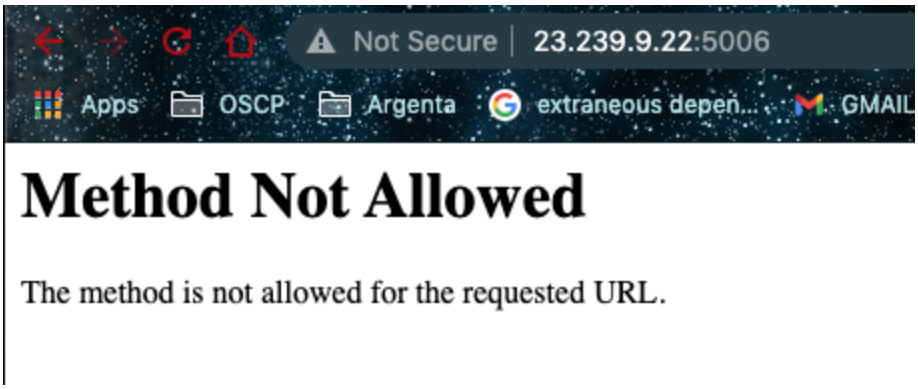
Combine all of that into a file and we are ready to go.

```
python training1.py
```

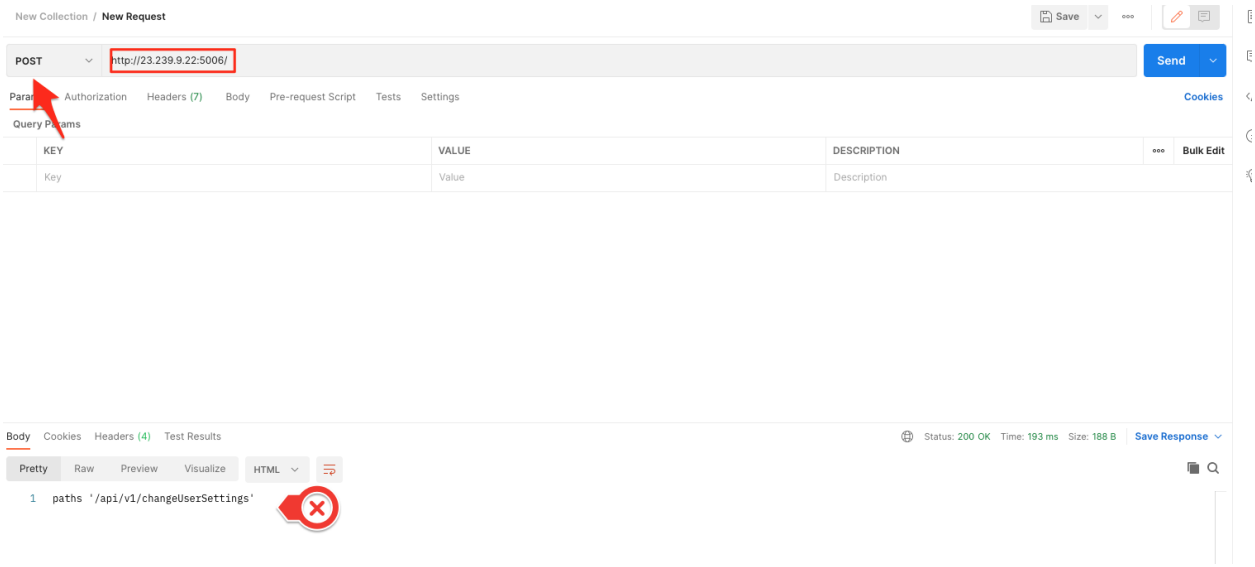
replace training1.py with whatever you named your file.



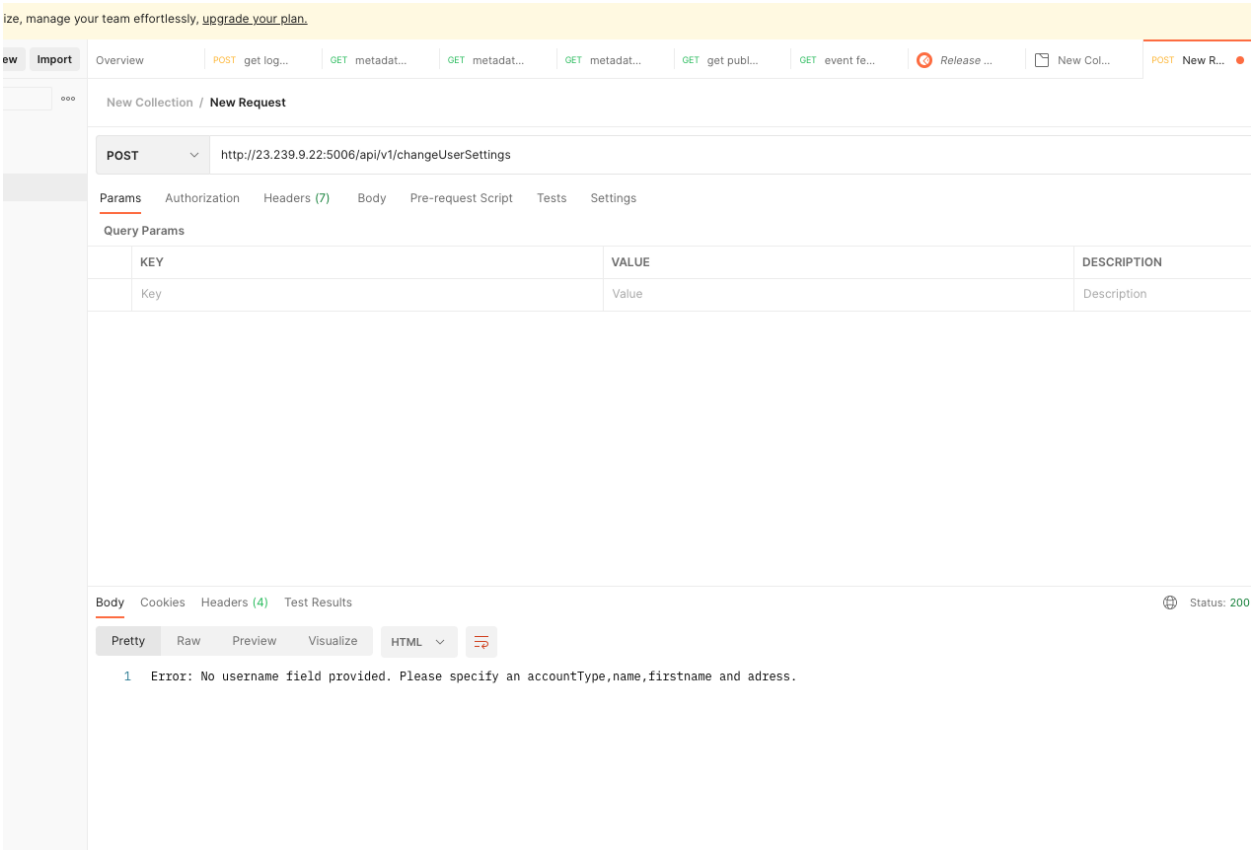
And with that, we should be able to surf to our VPS' IP adres on port 5006 and see what our flask app is saying.



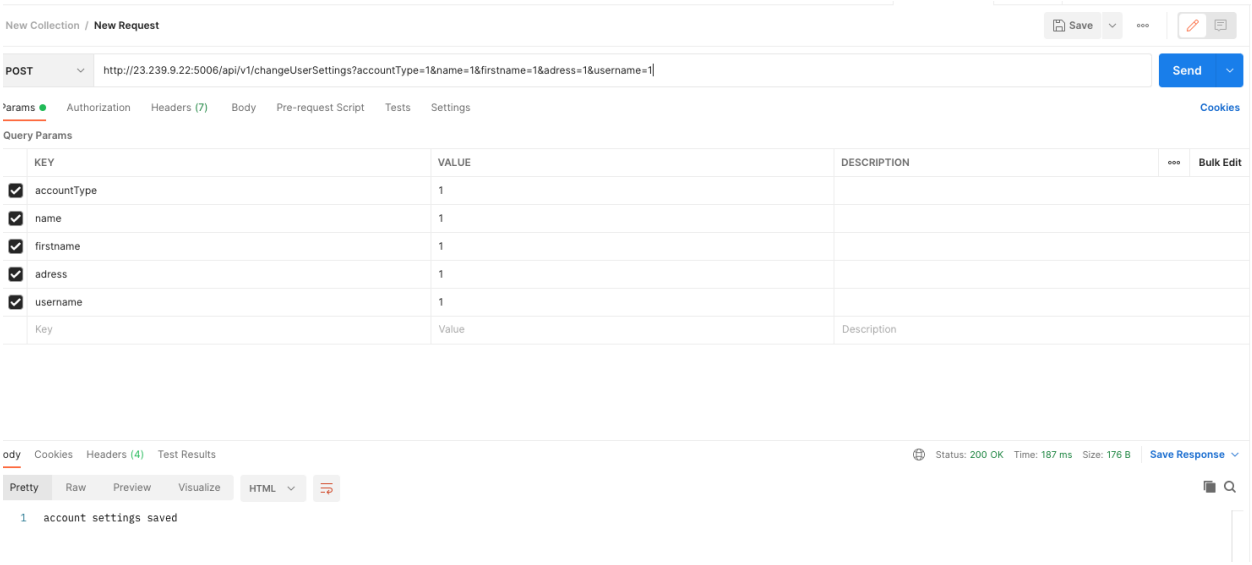
To get some data, we need to surf to the path in a POST method. This we will do with a tool called postman, you should know it by now because there is a chapter in the course about it.



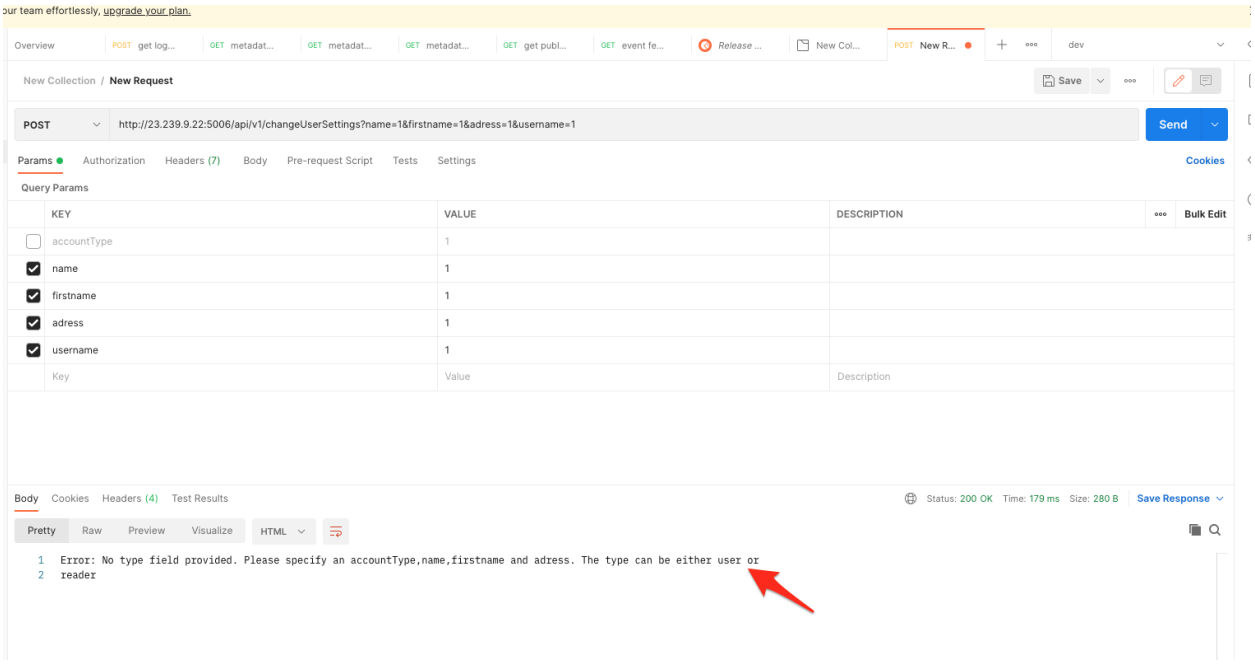
As you can see the server is returning the path we programmed in earlier. To get this result, simply fill in the URL and make sure you set the method to POST.



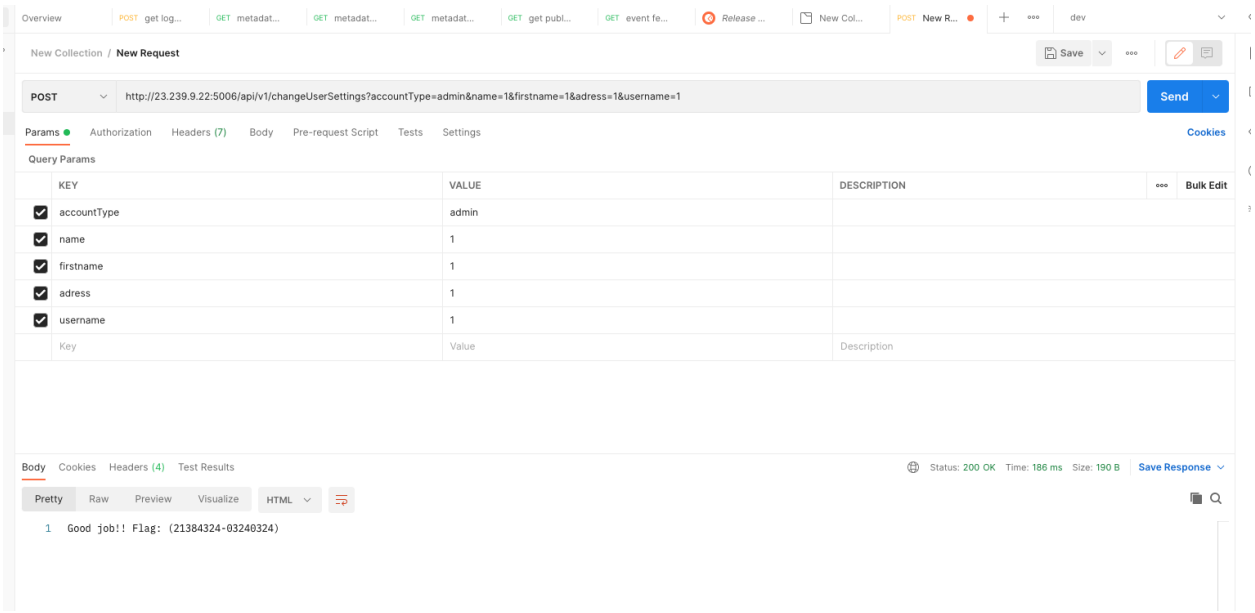
Surfing to this URL will give us the result that the server needs some parameters, most might be inclined to fill in all of them so let's do that.



It might seem like we are stuck here but let's try to remove some things like the accountType.



Now we get a different message that is saying that the account type can only be certain types but what if we try admin?



Extra: Full code

```
import flask
from flask import request, jsonify

app = flask.Flask(__name__)
app.config["DEBUG"] = True

@app.route('/', methods=['POST'])
def api_home():
     return "paths '/api/v1/changeUserSettings'"

# A route to return all of the available entries in our admin log.
@app.route('/api/v1/changeUserSettings', methods=['POST'])
def api_cards():
    if 'username' in request.args:
        name = request.args['name']
    else:
        return "Error: No username field provided. Please specify an accountType,name,firs
tname and adress."

    if 'name' in request.args:
        name = request.args['name']
    else:
        return "Error: No name field provided. Please specify an accountType,name,firstnam
e and adress."

    if 'firstname' in request.args:
        name = request.args['name']
    else:
```

```
        return "Error: No firstname field provided. Please specify an accountType,name,fir
stname and adress."

    if 'adress' in request.args:
        name = request.args['name']
    else:
        return "Error: No adress field provided. Please specify an accountType,name,firstn
ame and adress."

    if 'accountType' in request.args:
        type = request.args['accountType']
    else:
        return "Error: No type field provided. Please specify an accountType,name,firstnam
e and adress. The type can be either user or reader"

    if type == 'admin':
        return "Good job!! Flag: (21384324-03240324)"
    else:
        return "account settings saved"


app.run(host="0.0.0.0",port="5006")
```