



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

[Introduction](#)

[Requirements](#)

[Let's set up](#)

[Hello World](#)

[Extra: Full code](#)

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 potato ... 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 separate 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 environment) called GoudAPI-BAC which is marked by a new folder, now we have to switch to it.

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

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

And now we can easily use pip to install flask

```
pip install Flask
```

```
Collecting Flask
  Downloading Flask-2.0.1-py3-none-any.whl (94 kB)
    |-----| 94 kB 3.3 MB/s
Collecting Jinja2>=3.0
  Downloading Jinja2-3.0.1-py3-none-any.whl (133 kB)
    |-----| 133 kB 32.4 MB/s
Collecting Werkzeug>=2.0
  Downloading Werkzeug-2.0.1-py3-none-any.whl (288 kB)
    |-----| 288 kB 26.0 MB/s
Collecting click>=7.1.2
  Downloading click-8.0.1-py3-none-any.whl (97 kB)
    |-----| 97 kB 23.2 MB/s
Collecting itsdangerous>=2.0
  Downloading itsdangerous-2.0.1-py3-none-any.whl (18 kB)
Collecting MarkupSafe>=2.0
  Downloading MarkupSafe-2.0.1-cp39-cp39-macosx_10_9_x86_64.whl (13 kB)
Installing collected packages: MarkupSafe, Werkzeug, Jinja2, itsdangerous, click, Flask
Successfully installed Flask-2.0.1 Jinja2-3.0.1 MarkupSafe-2.0.1 Werkzeug-2.0.1 click-8.0.1 itsdangerous-2.0.1
WARNING: You are using pip version 21.0.1; however, version 21.2.1 is available.
You should consider upgrading via the '/Users/wesleythijs/GoudAPI/GoudAPI/bin/python3.9 -m pip install --upgrade pip' command.
```

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 separately 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 one so we display it and we can move on to 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,firstn
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.

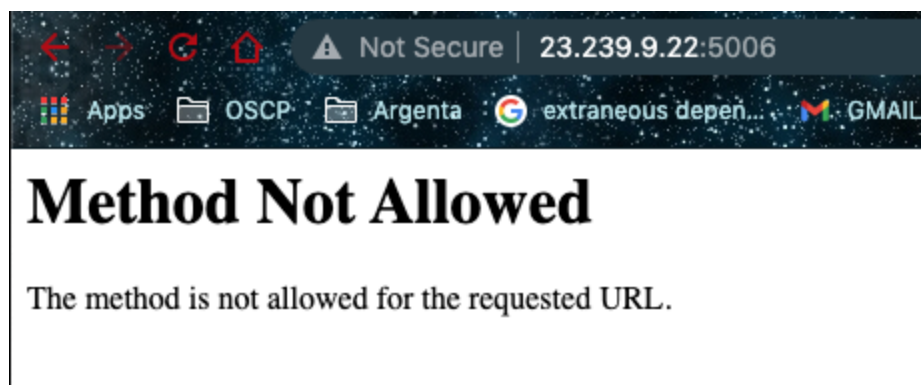
Combine all of that into a file and we are ready to go.

```
python training1.py
```

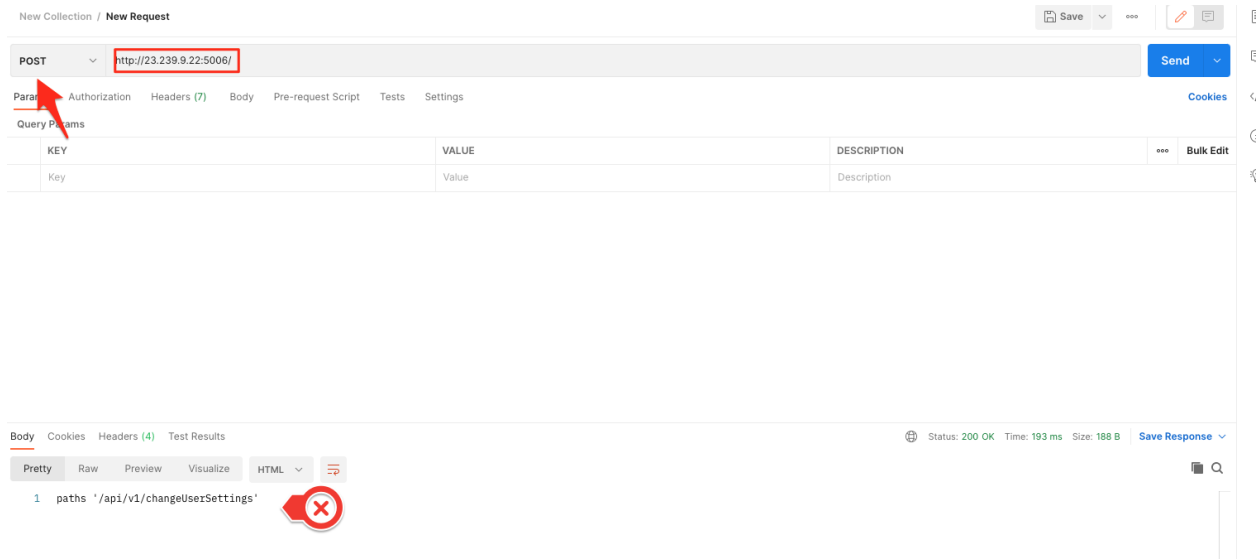
replace training1.py with whatever you named your file.

```
root@localhost:~/projects/api# python api-ids.py
* Serving Flask app "api-ids" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: on
* Running on http://0.0.0.0:5002/ (Press CTRL+C to quit)
* Restarting with stat
* Debugger is active!
* Debugger PIN: 641-322-905
81.242.164.154 - - [07/Aug/2021 22:07:47] "GET /api/v1/resources/books?id=2 HTTP/1.1" 200 -
81.242.164.154 - - [07/Aug/2021 22:07:48] "GET /favicon.ico HTTP/1.1" 404 -
81.242.164.154 - - [07/Aug/2021 22:07:52] "GET /api/v1/resources/books?id=3 HTTP/1.1" 200 -
```

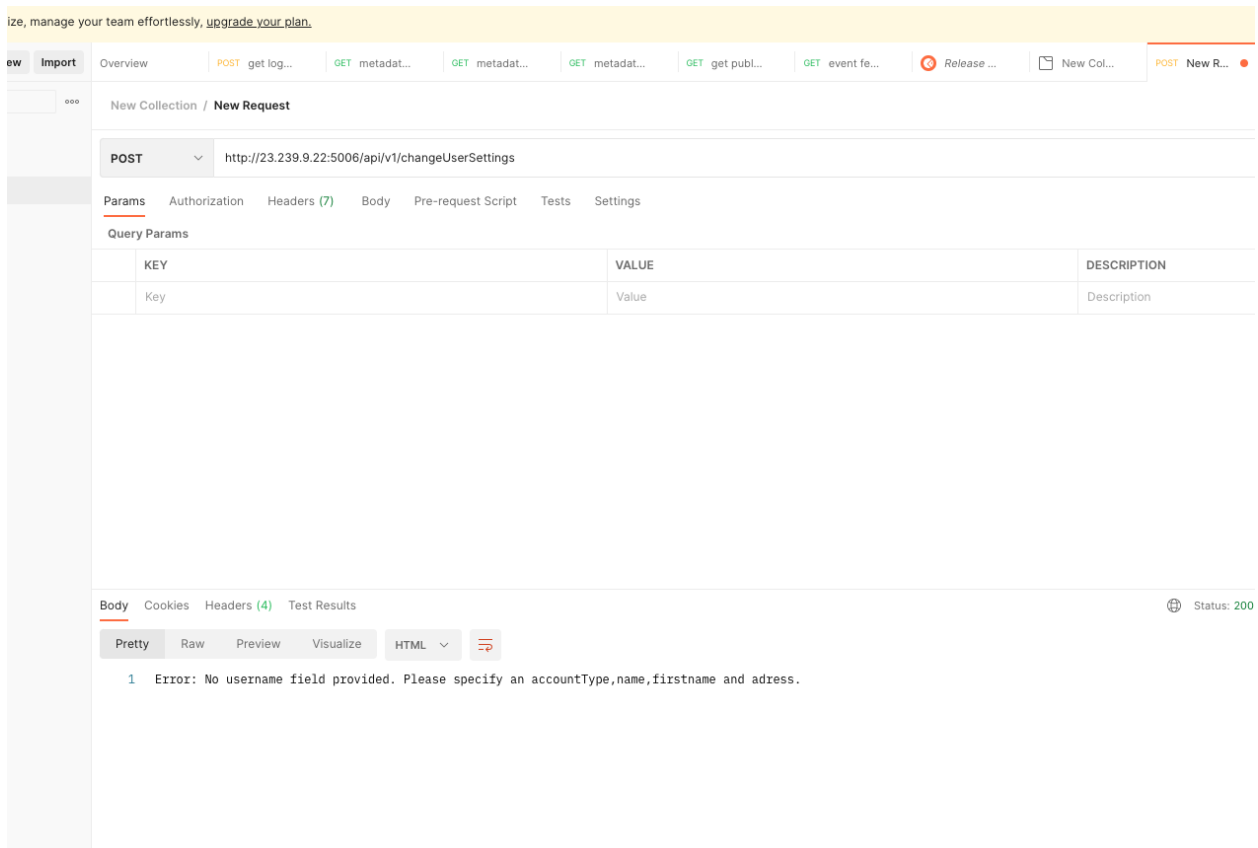
And with that, we should be able to surf to our VPS' IP adres on port 5006 and see what our flask app is saying.



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.

The screenshot shows a REST client interface with a POST request to `http://23.239.9.22:5006/api/v1/changeUserSettings?accountType=1&name=1&firstname=1&adress=1&username=1`. The request is successful, returning a 200 OK status. The response body, viewed in 'Pretty' format, shows a single line: `1 account settings saved`.

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> accountType	1	
<input checked="" type="checkbox"/> name	1	
<input checked="" type="checkbox"/> firstname	1	
<input checked="" type="checkbox"/> adress	1	
<input checked="" type="checkbox"/> username	1	
Key	Value	Description

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

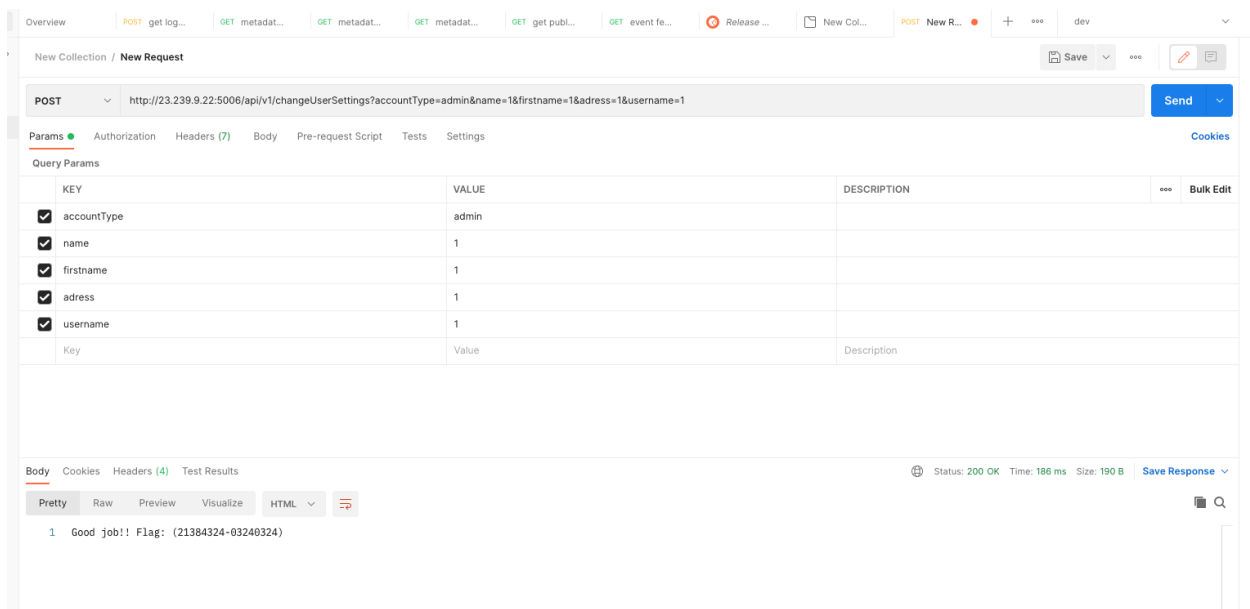
The screenshot shows the same REST client interface, but the `accountType` parameter has been unchecked in the query parameters table. The request is still successful (200 OK), but the response body now contains an error message:

```
1 Error: No type field provided. Please specify an accountType,name,firstname and adress. The type can be either user or  
2 reader
```

A red arrow points to the word `reader` in the error message.

KEY	VALUE	DESCRIPTION
<input type="checkbox"/> accountType	1	
<input checked="" type="checkbox"/> name	1	
<input checked="" type="checkbox"/> firstname	1	
<input checked="" type="checkbox"/> adress	1	
<input checked="" type="checkbox"/> username	1	
Key	Value	Description

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,first
name 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,firstn
        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")

```