



Let's build an API to hack - Part 3: Information disclosure

[Introduction](#)

[Requirements](#)

[Let's set up](#)

[Hello World](#)

[Extra: Full code](#)

[Extra: Another example full code](#)

Introduction

For the issue type "Information disclosure" i wanted to give you guys an example since it does not seem to be very well understood what data counts as sensitive and what should be disclosed as public information.

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, jsonify

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 and jsonify for later since our first API will only return static data so let's build up that static data with the following code.

```
# Create some test data for our catalog in the form of a list of dictionaries.
CreditCards = [
    {'id': 0,
     'creditCard': '1234567901234',
     'user': 'API',
     'CVV': '677',
     'validUntil': '1992'},
    {'id': 1,
     'creditCard': '1234567901235',
     'user': 'API',
     'CVV': '677',
     'validUntil': '2022'},
    {'id': 2,
     'creditCard': '1234567901239',
     'user': 'API',
     'CVV': '677',
     'validUntil': '2023'}
]
```

So now that we have some data, we need to define a way to get that data. In the next step we will define the path `/api/v1/resources/creditCards/all` and the possible methods. When we make a GET request on `/api/v1/resources/creditCards/all` we will return the data from the previous step.

```
# A route to return all of the available entries in our catalog.
@app.route('/api/v1/resources/creditCards/all', methods=['GET'])
def api_all():
    return jsonify(CreditCards)
```

all that is left is for us to start up our application but as of now flask does not know how to start yet so let's tell it. We will also be using another port in this case because port 5000 (the default port) will already be taken from our previous script.

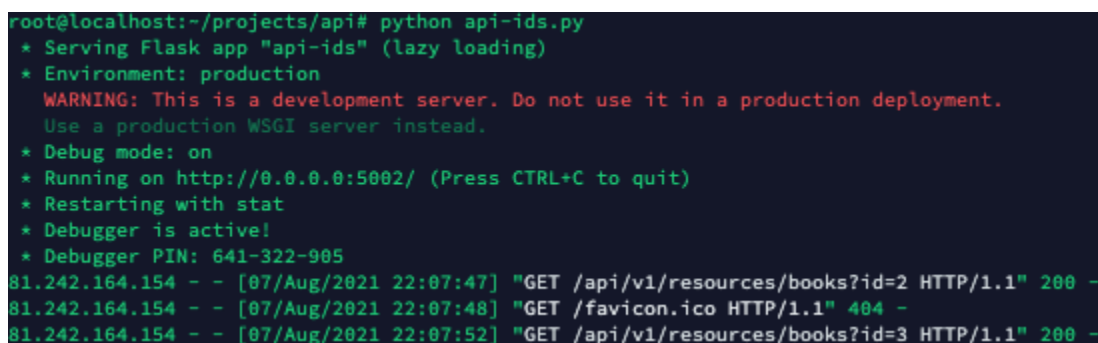
```
app.run(host="0.0.0.0", port="5002")
```

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 5002 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.

A terminal window with a dark background and green text. It shows the output of running a Flask application. The output includes status messages like 'Serving Flask app', 'Environment: production', 'WARNING: This is a development server...', 'Debug mode: on', 'Running on http://0.0.0.0:5002/', 'Restarting with stat', 'Debugger is active!', and 'Debugger PIN: 641-322-905'. It also shows three HTTP requests from 81.242.164.154: a GET request for /api/v1/resources/books?id=2 returning 200, a GET request for /favicon.ico returning 404, and a GET request for /api/v1/resources/books?id=3 returning 200.

```
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 5002 and see what our flask app is saying.



Not Found

The requested URL was not found on the server. If you entered the URL manually please check your spelling and try again.

To get some data, we need to surf to the path we defined earlier.

`"/api/v1/resources/cheese/all"`

```
{
  {
    "author": "The XSS Rat",
    "description": "Dutch: Goudse kaas, \"cheese from Gouda\" is a sweet, creamy, yellow cows milk cheese originating from the Netherlands. ...",
    "id": 0,
    "title": "Gouda",
    "year": "1992"
  },
  {
    "author": "The XSS Rat",
    "description": "Casu martzu[1] (Sardinian pronunciation; literally rotten/putrid cheese) is a traditional Sardinian sheep milk cheese that contains live insect larvae (maggots)",
    "id": 1,
    "title": "Casu marzu",
    "year": "2020"
  },
  {
    "author": "The XSS Rat",
    "description": "Roquefort is a popular French cheese, reported to be a favourite of Emperor Charlemagne. In France, it is called the cheese of kings and popes.",
    "id": 2,
    "title": "Roquefort",
    "year": "2020"
  }
}
```

Make sure this works as this will be our basefile for the rest of the exercises.

We can also surf to `"/api/v1/creditCards/cheese/all"` to find we are stringifying our whole credit cards array with all of its data, including CVV and expiration data. If a MiTM attack would occur this would lead to a much more severe attack than should happen.

Extra: Full code

```
import flask
from flask import request, jsonify

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

CreditCards = [
    {'id': 0,
     'creditCard': '1234567901234',
     'user': 'API',
     'CVV': '677',
     'validUntil': '1992'},
    {'id': 1,
     'creditCard': '1234567901235',
     'user': 'API',
     'CVV': '677',
```

```

        'validUntil': '2022'},
{'id': 2,
 'creditCard': '1234567901239',
 'user': 'API',
 'CVV': '677',
 'validUntil': '2023'}
]

# A route to return all of the available entries in our catalog.
@app.route('/api/v1/resources/creditCards/all', methods=['GET'])
def api_all():
    return jsonify(CreditCards)

app.run(host="0.0.0.0", port="5002")

```

Extra: Another example full code

This script returns the commenters full address and while it might not be displayed in the UI persé, it is vulnerable if the API returns the information like this.

```

import flask
from flask import request, jsonify

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

# Create some test data for our catalog in the form of a list of dictionaries.
comments = [
    {'id': 0,
     'comment': '1234567901234',
     'user': 'testUser',
     'user email': 'test@bla.com',
     'user adress': 'testlane, testing - 340043 testing in testland'}
]

@app.route('/', methods=['GET'])
def home():
    return '''<h1>Distant Reading Archive</h1>
<p>A prototype API for distant reading of science fiction novels.</p>'''

# A route to return all of the available entries in our admin log.
@app.route('/api/v1/comments', methods=['GET'])
def api_cards():
    if 'id' in request.args:
        id = int(request.args['id'])
    else:

```

```
        return "Error: No id field provided. Please specify an id."

    # Create an empty list for our results
    results = []

    # Loop through the data and match results that fit the requested ID.
    # IDs are unique, but other fields might return many results
    for comment in comments:
        if comment['id'] == id:
            results.append(comment)

    # Use the jsonify function from Flask to convert our list of
    # Python dictionaries to the JSON format.
    return jsonify(results)

app.run(host="0.0.0.0",port="5005")
```