Vulnerable bank- API Security Testing Report

Tester: Blessing Isaiah
Report Date: December 05, 2025

Environment

The assessment was performed on a local deployment of the Vulnerable Bank application running inside a docker environment on the host machine.
The web interface was reachable at **127.0.0.1:5000** over HTTP.
BurpSuite to listen on port **8080** for request interception and analysis.

Executive Summary

Objective

In this report, I will be exploiting the APIs in Vulnerable Bank web application.

The goal is to review the API behavior in a controlled environment, identify weaknesses, and document realistic attack paths and their associated risks.

Methodology

The assessment followed a white-box testing method after gaining an understanding of the API structure.

Testing Approach

The workflow included the following stages:

1. **Environment Setup and Proxy Configuration**
   BurpSuite was configured to intercept traffic from the running instance.
2. **Traffic Capture and Reconnaissance**
   Baseline requests and responses were analyzed to understand default behavior.
3. **Manual Specification Enhancement**
   Observed patterns were used to adjust and manipulate URLs to simulate attack scenarios.
4. **Endpoint Testing and Fuzzing**
   Targeted requests were sent to evaluate:
   - Broken Object Level Authorization (BOLA)
   - Broken Authentication
   - Broken Object Property Level Authorization
   - Improper Inventory Management
5. **Documentation and Evidence Collection**
   Every step was recorded, including screenshots, logs, and proof-of-concept details.

Standards Followed

The assessment was aligned with:

- OWASP API Security Top 10 (2023)
- CVSS v4.0 risk scoring
- GDPR, PCI DSS, and NIST 800-53 compliance guidelines

Major Tools Used

- **Recon and scanning tools:** GitHub, Swagger
- **Proxy tools:** BurpSuite Pro, Firefox
- **API testing tools:** Burp Repeater, xJWT.io
- **Environment tools:** Kali Linux, VirtualBox, docker

In Scope

- Local Vulnerable bank application running on the Kali VM
- HTTP traffic captured with BurpSuite
- API documentation

Out of Scope

- Any external or production systems
- Activities outside the controlled lab environment
- Destructive actions or data extraction to external servers

Findings: chronological (OWASP mapping, description, evidence, impact, remediation)

Finding 1: Broken Object Level Authorization (BOLA)

OWASP Mapping

OWASP API3 – Broken Object Level Authorization (2023)
The API exposes object identifiers and fails to validate whether the requester is authorized to access the referenced objects.

Description

The API does not enforce object-level access checks.
A non-privileged user can retrieve sensitive information belonging to another user by modifying object identifiers such as **account numbers** or **card IDs** in the URL.
The backend trusts user-supplied identifiers without verifying ownership.

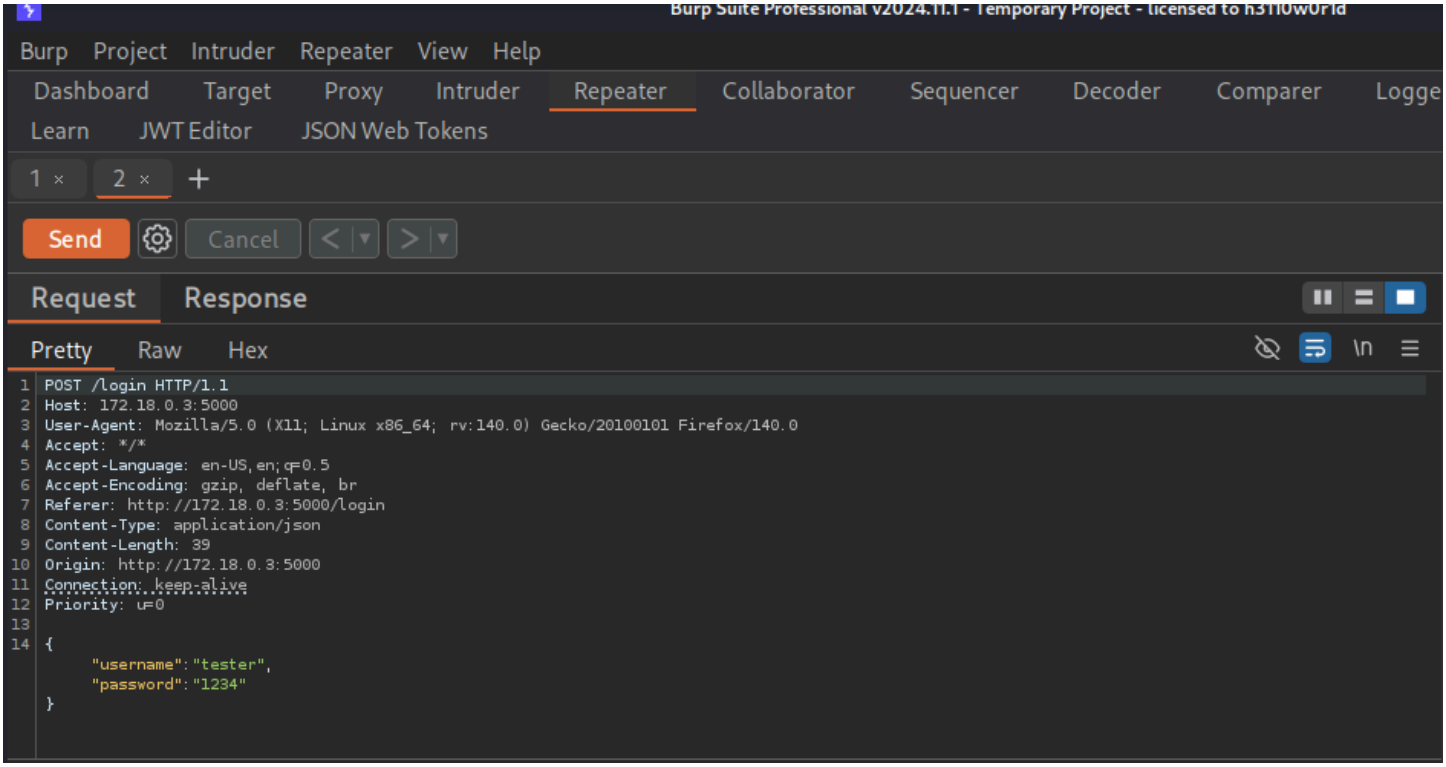
This allows one authenticated user (User A) to access the transaction history, balance information, and account data of another user (User B).

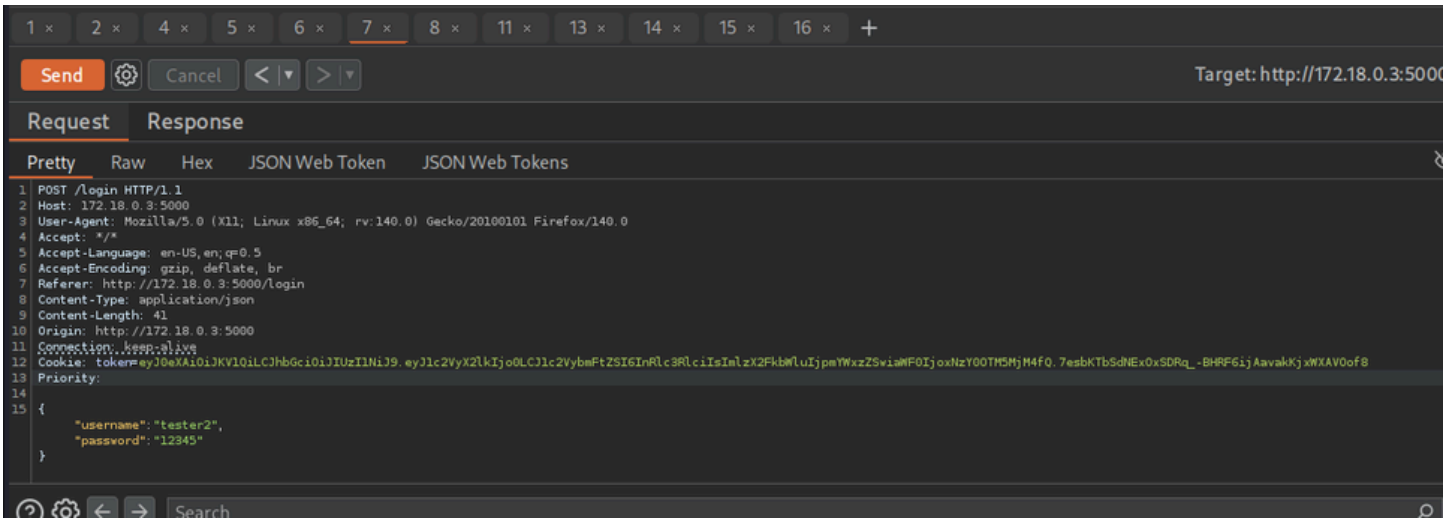
Evidence / Proof of Concept

BOLA Scenario 1 – Unauthorized access to another user's transaction history

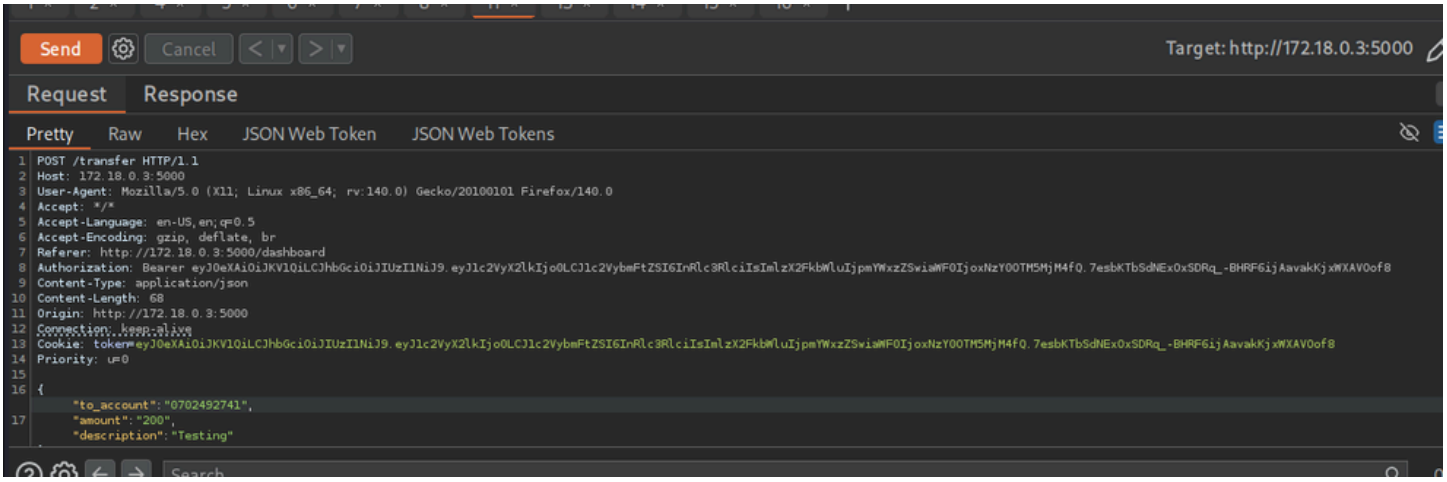
Steps performed:

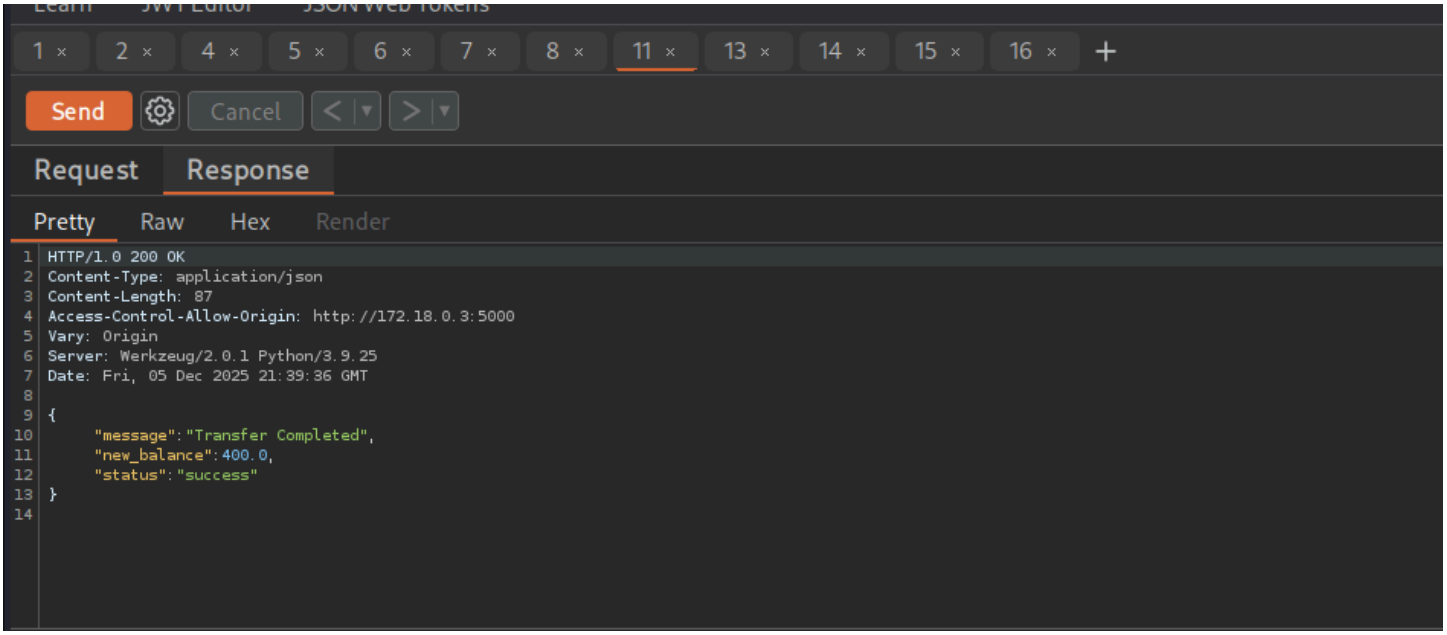
- Register two new accounts (User A and User B) to obtain their account numbers.



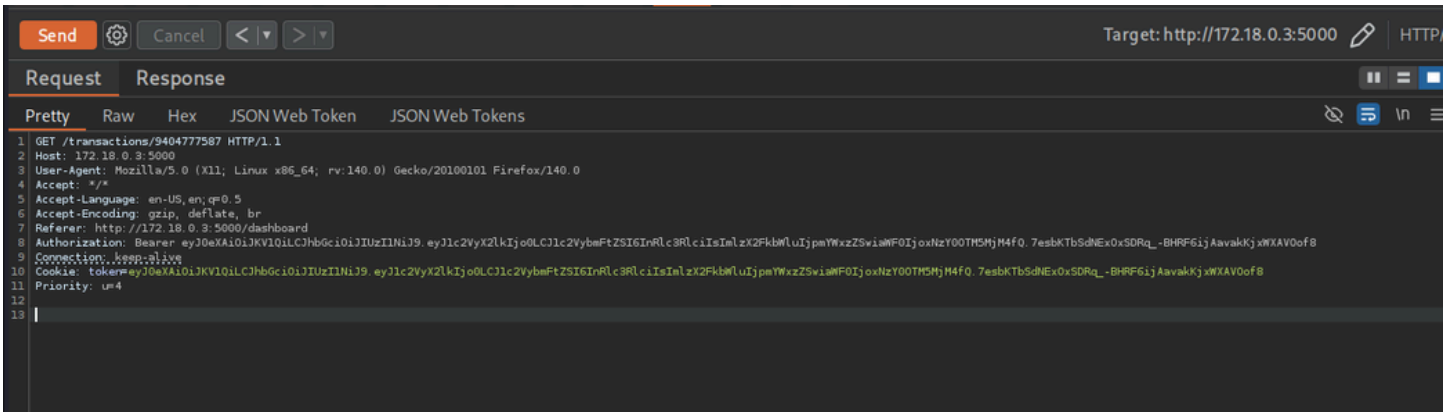


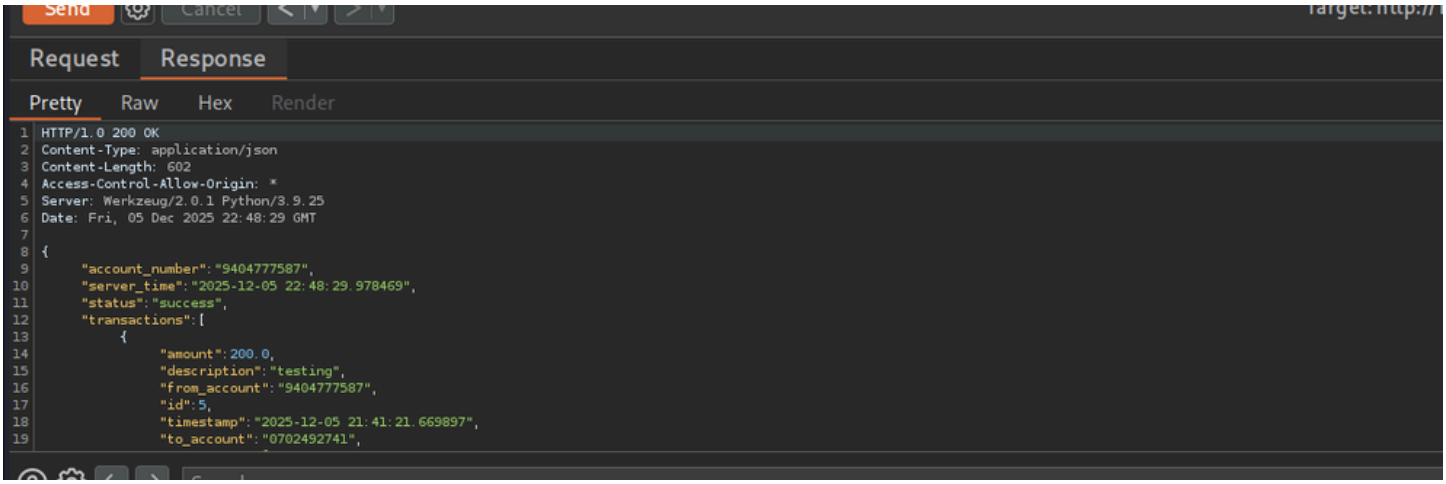
- Log in as User B and make a transaction to User A. Making it easy to get User A's account number. This gives User B both account numbers for testing.





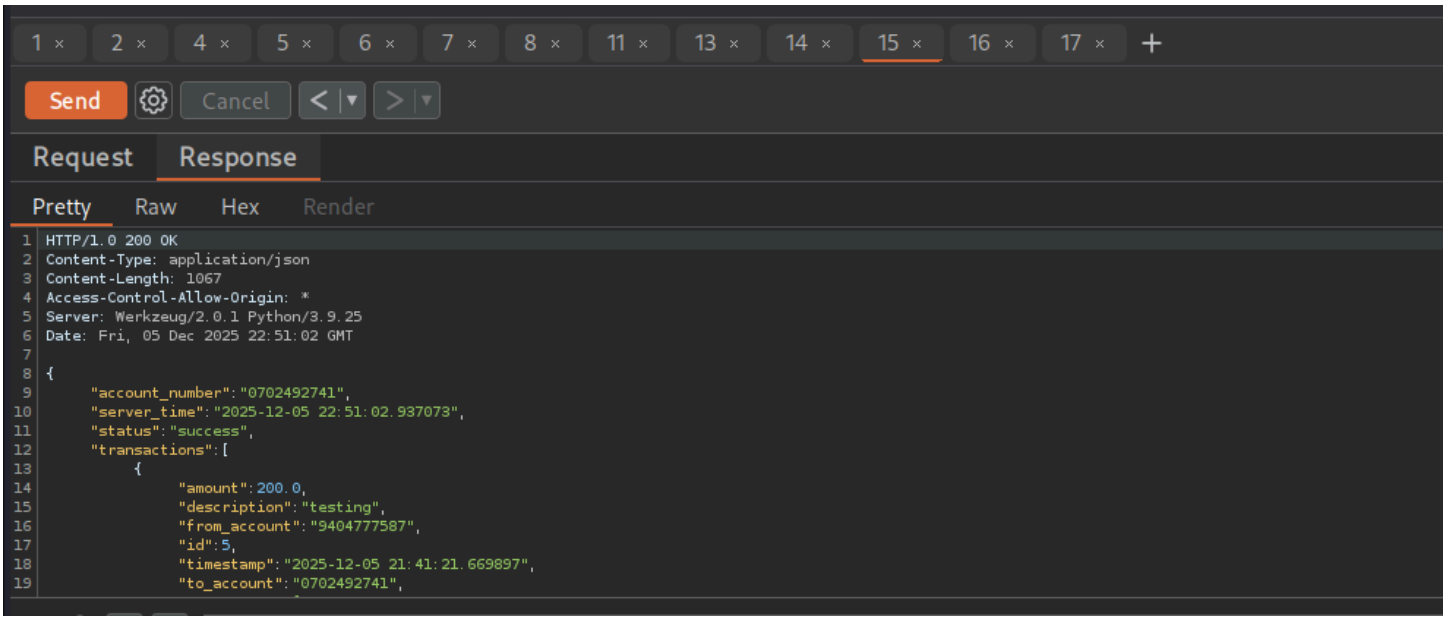
- While logged in as User B, request the transaction history via: **GET /transactions/{account_number}**

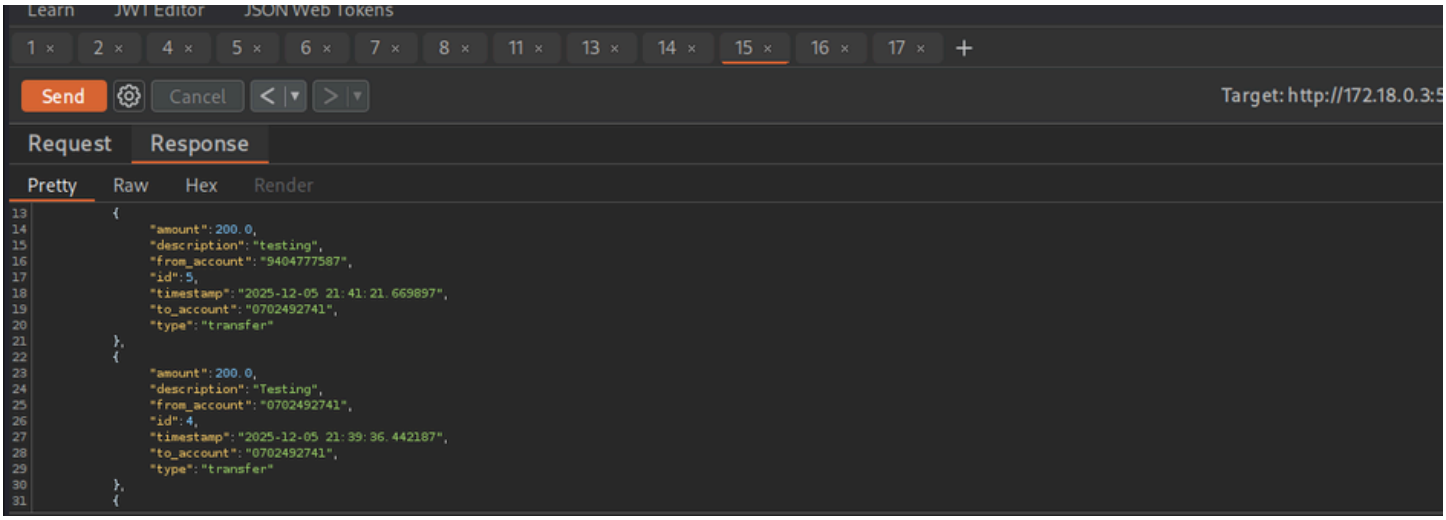




- Replace User B's account number with User A's account number in the URL.

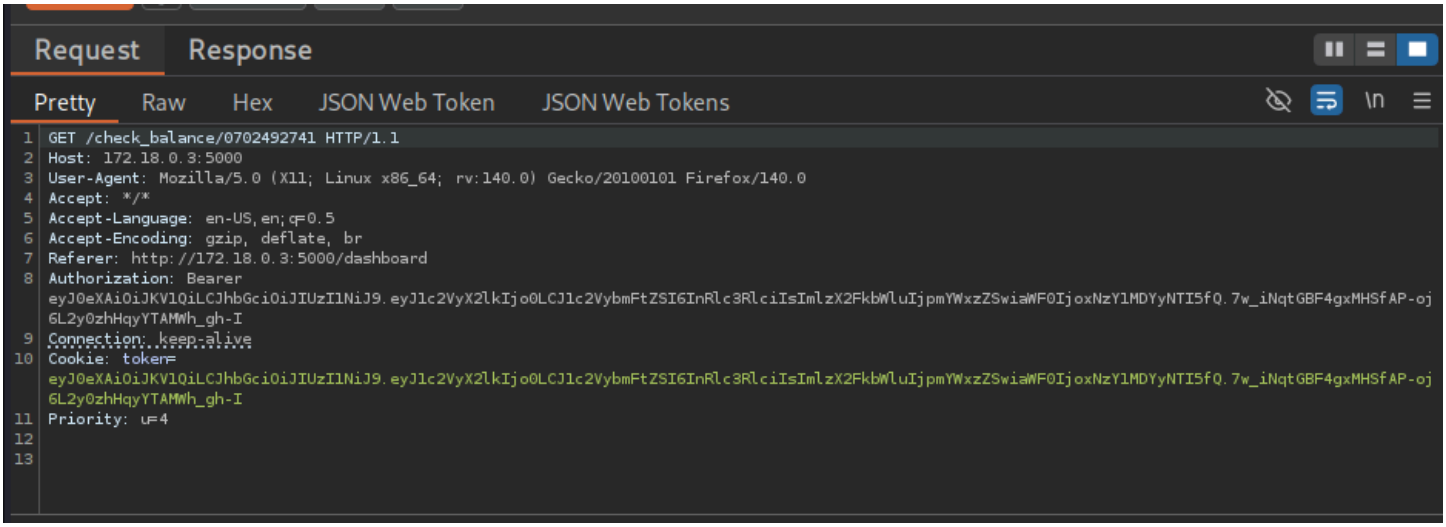
- Here User A can see the transactions history of user B as long as they have account number

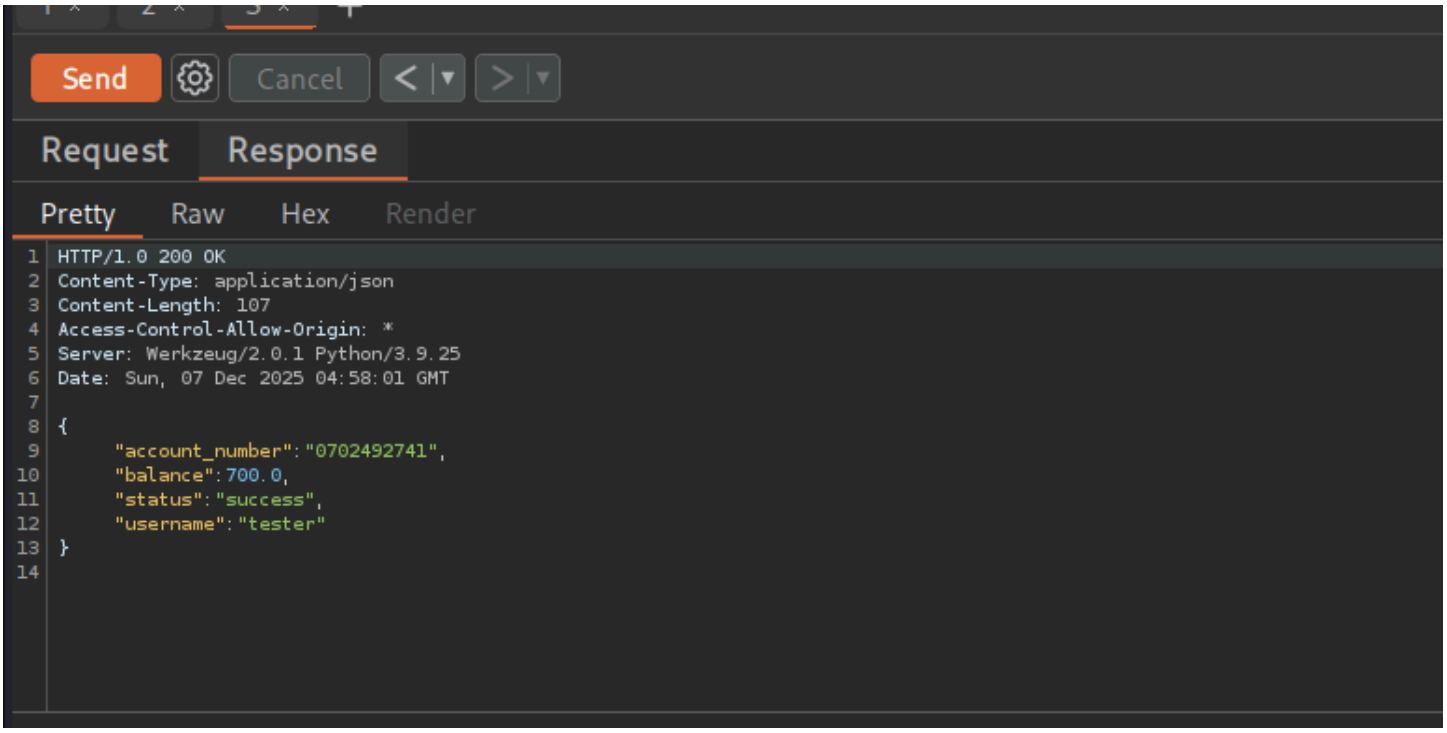




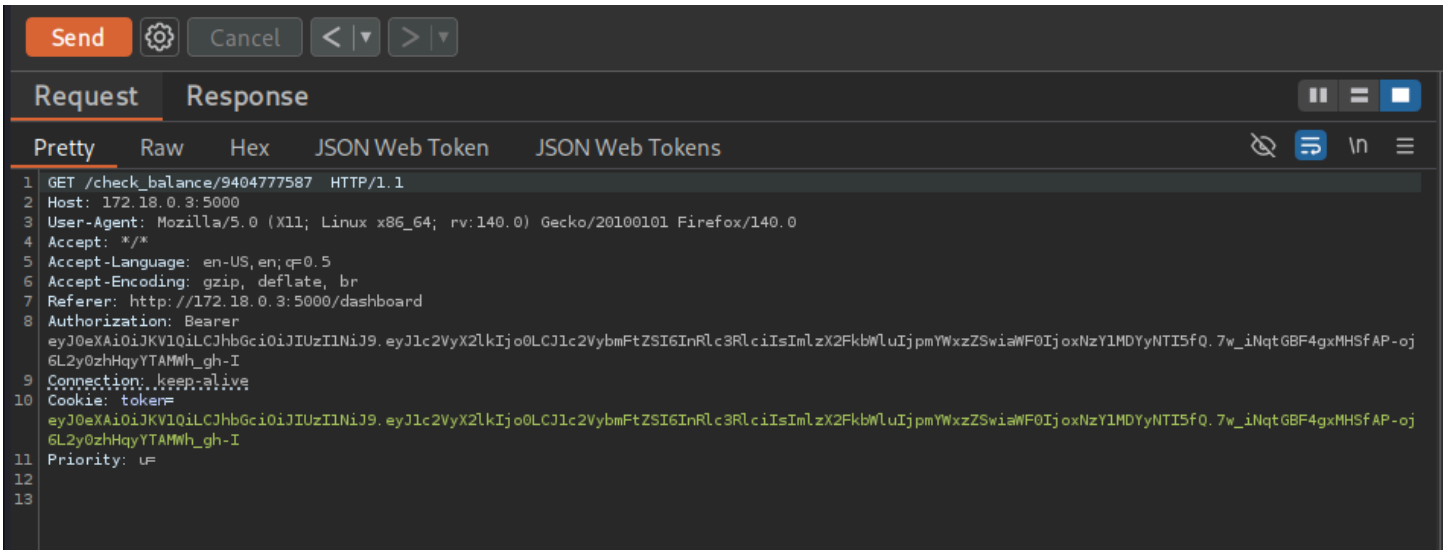
BOLA Scenario 2 – Unauthorized access to another user's account balance

- Stay logged in as User A (token unchanged).
- Request balance information using: **GET /check_balance/{account_number}**



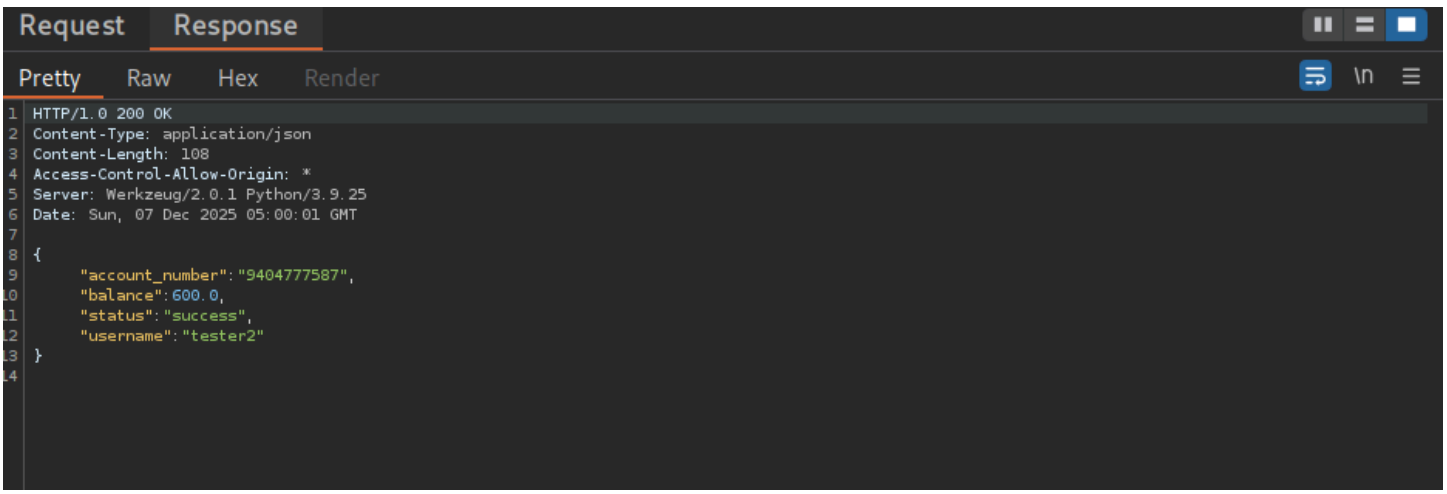


- Replace User A's account number with User B's account number.



Result:

User A receives **User B's account balance**, exposing sensitive financial information.



Impact

This vulnerability allows **full horizontal privilege escalation** across the banking API.
An attacker can:

- Access any user's transaction history
- View any user's balance
- Enumerate valid account numbers
- Build large-scale financial data extraction attacks
- Combine BOLA with automation for mass harvesting
- Assist social engineering attacks (knowing exact balances, transfers, patterns)

In a financial application, BOLA represents a **critical severity** issue with immediate exploitation potential.
It violates basic authorization principles and can lead to severe privacy, financial, and regulatory consequences.

Remediation

To fix BOLA, enforce strict object-level authorization:

1. Validate object ownership server-side

Before returning any account data, ensure that:

- The authenticated user owns the object being requested
- The object belongs to the user ID inside the token

2. Never rely solely on client-submitted identifiers

Object IDs must always be validated against the authenticated user's identity.

3. Use internal identifiers

Map public IDs to internal user-bound IDs to prevent enumeration attacks.

4. Implement centralized access control policies

Use an authorization middleware to enforce consistent checks across endpoints.

5. Add rate limits and anomaly detection

To prevent enumeration of account numbers.

Findings 2 – Multi-Stage Chain Attack (Broken Authentication → BOPLA → BOLA → Improper Inventory Management)

Overview

The API contains a compounded security failure where several vulnerabilities chain together.
The most critical weakness is **BOPLA (Broken Object Property Level Authorization)** because the backend trusts sensitive properties provided by the user in the JSON Web Token (JWT) payload.

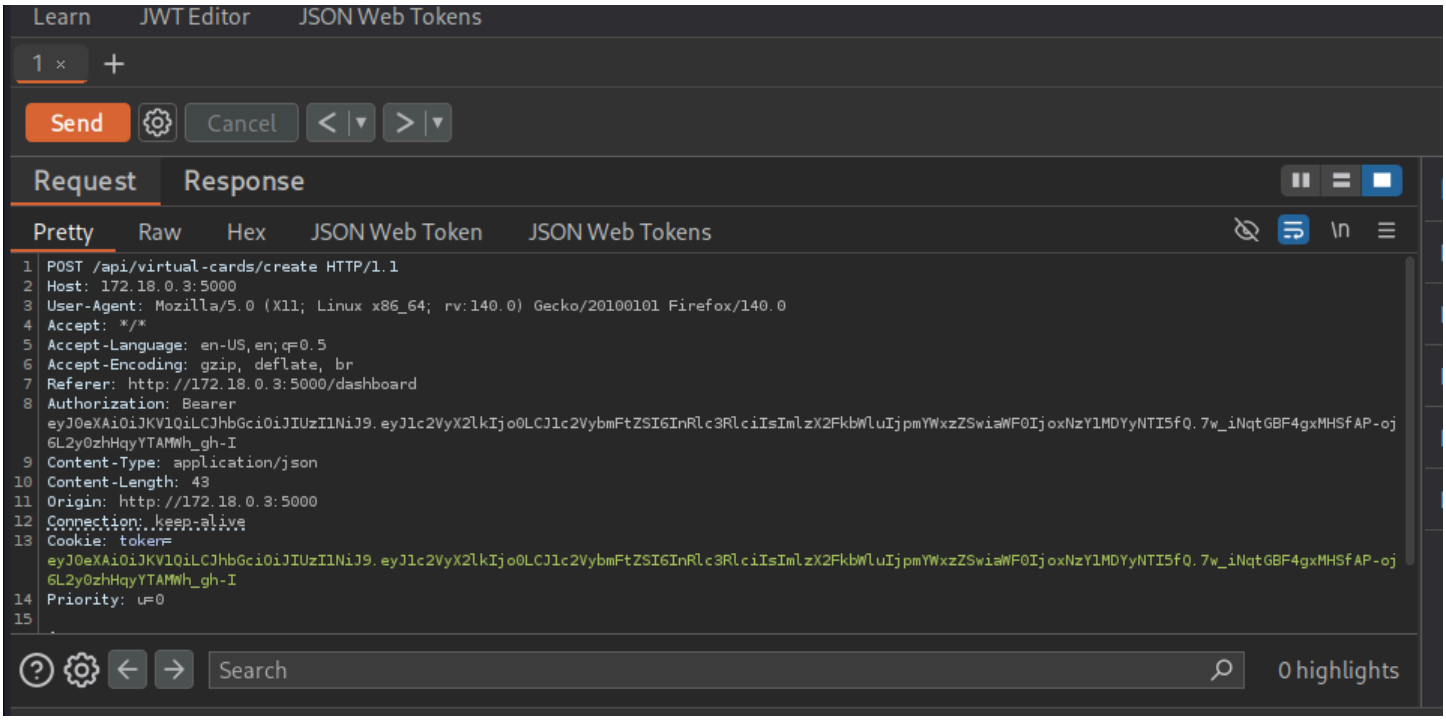
Due to weak JWT protection and missing authorization checks on private object properties, an attacker can obtain admin privileges, generate valid admin tokens, and access sensitive card data that belongs to other users.

Chain Breakdown:

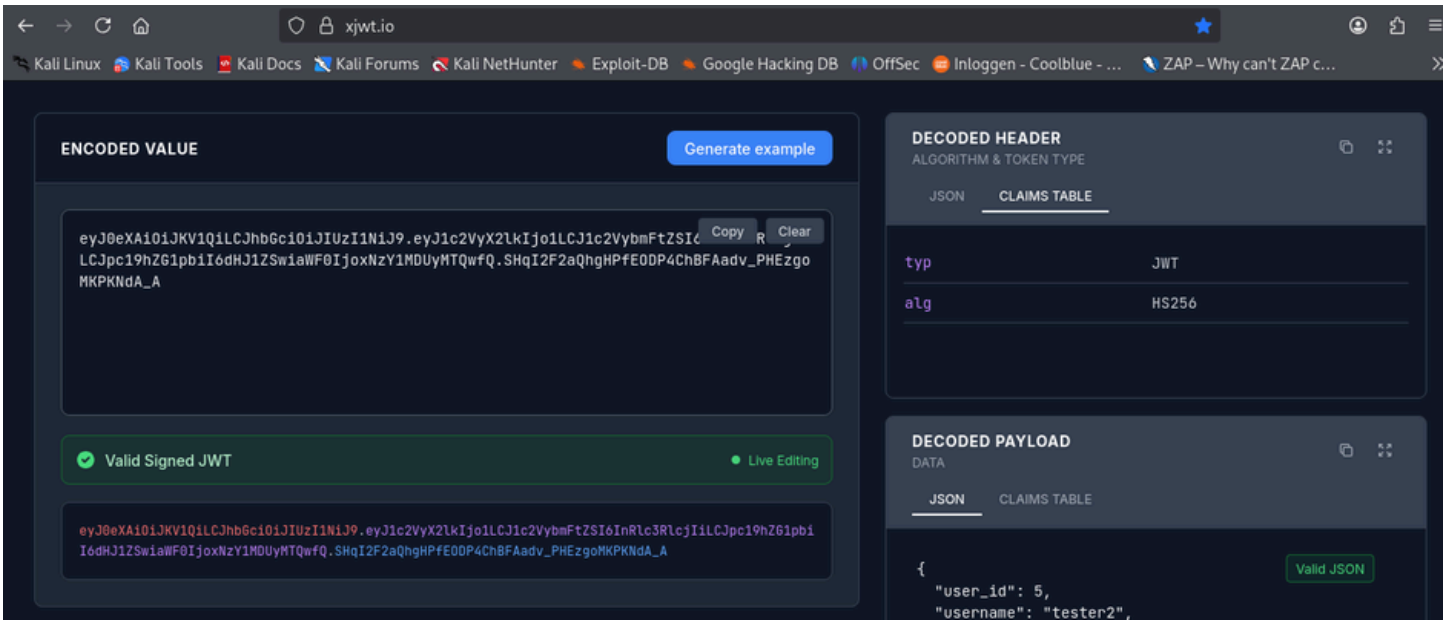
1. Broken Authentication – Weak JWT Signature

The JSON Web Token uses a weak or guessable signing secret.
This allows an attacker to:

- Here the current user create a new virtual card for their self



The user decide to check if the token secret can be cracked using XJWT.io

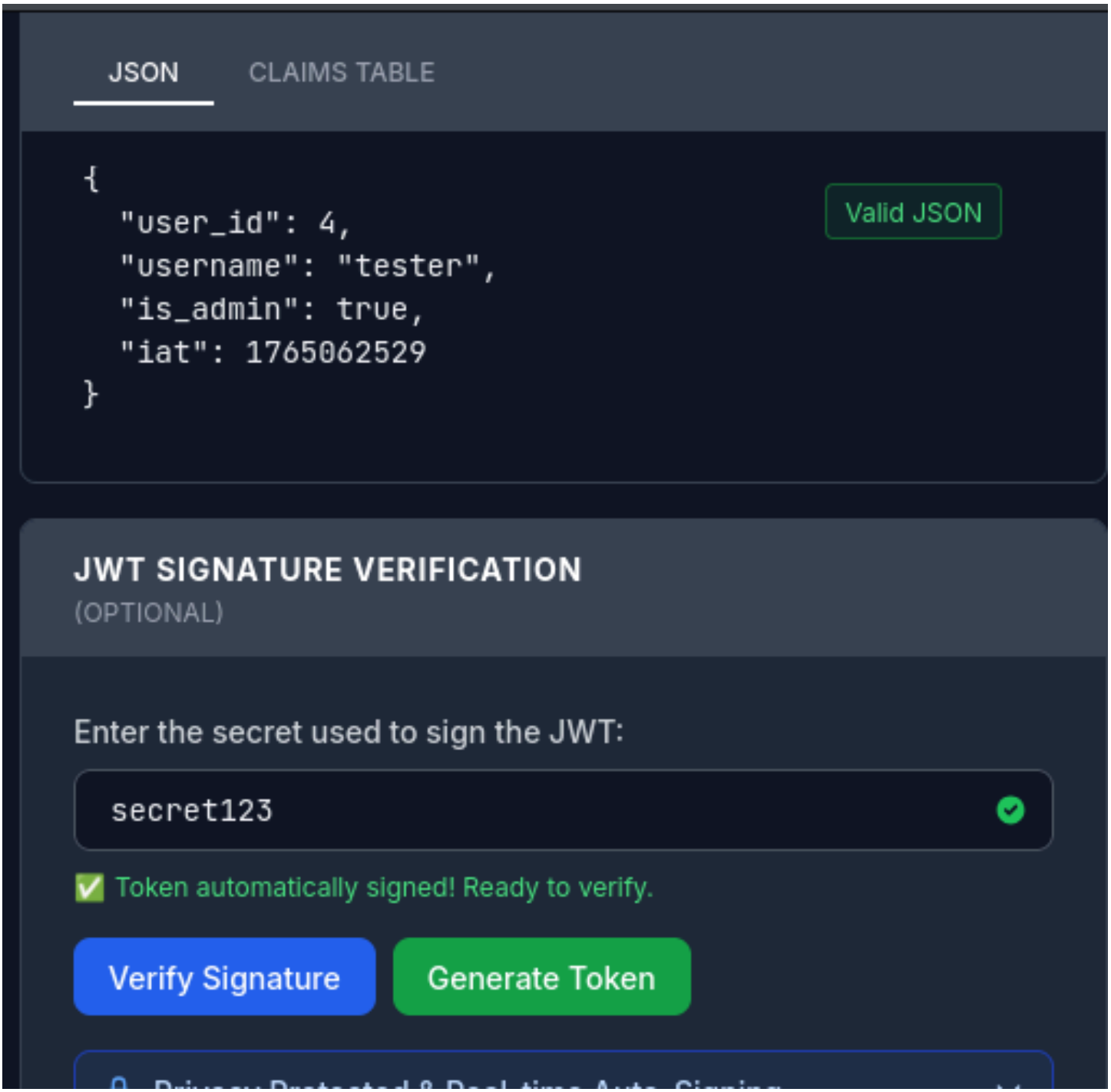


2. BOPLA – Modifying Sensitive Authorization Properties (admin flag)

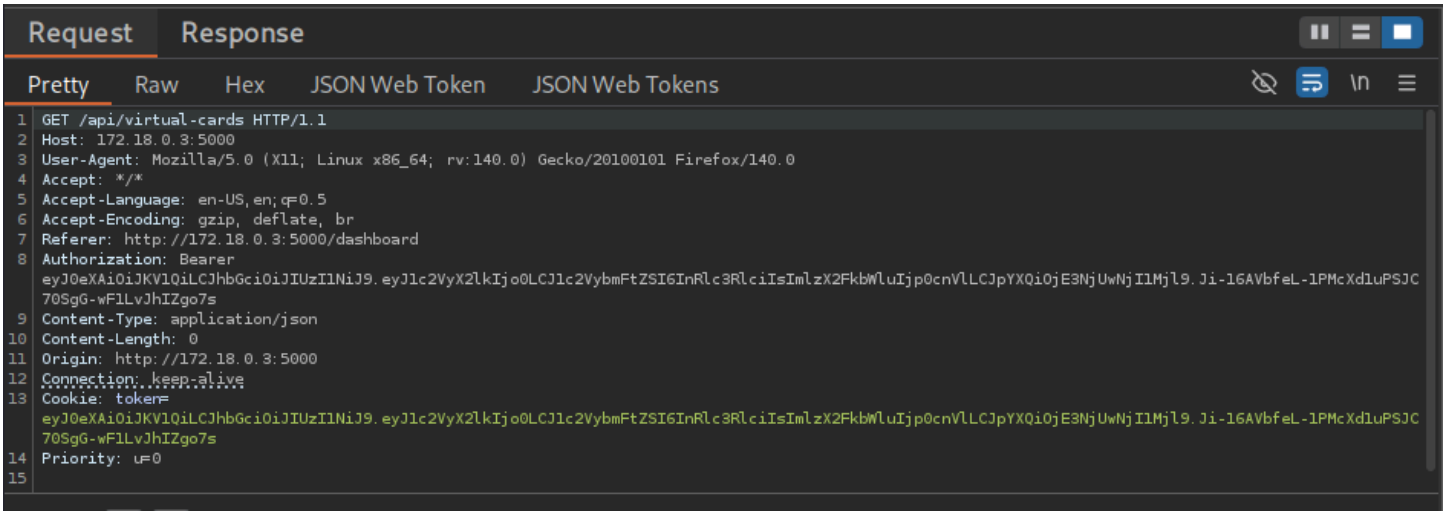
Turns out the JWT is using weak signature algorithm and admin status is available on the payload making it easy for the current user to quickly generate a new token

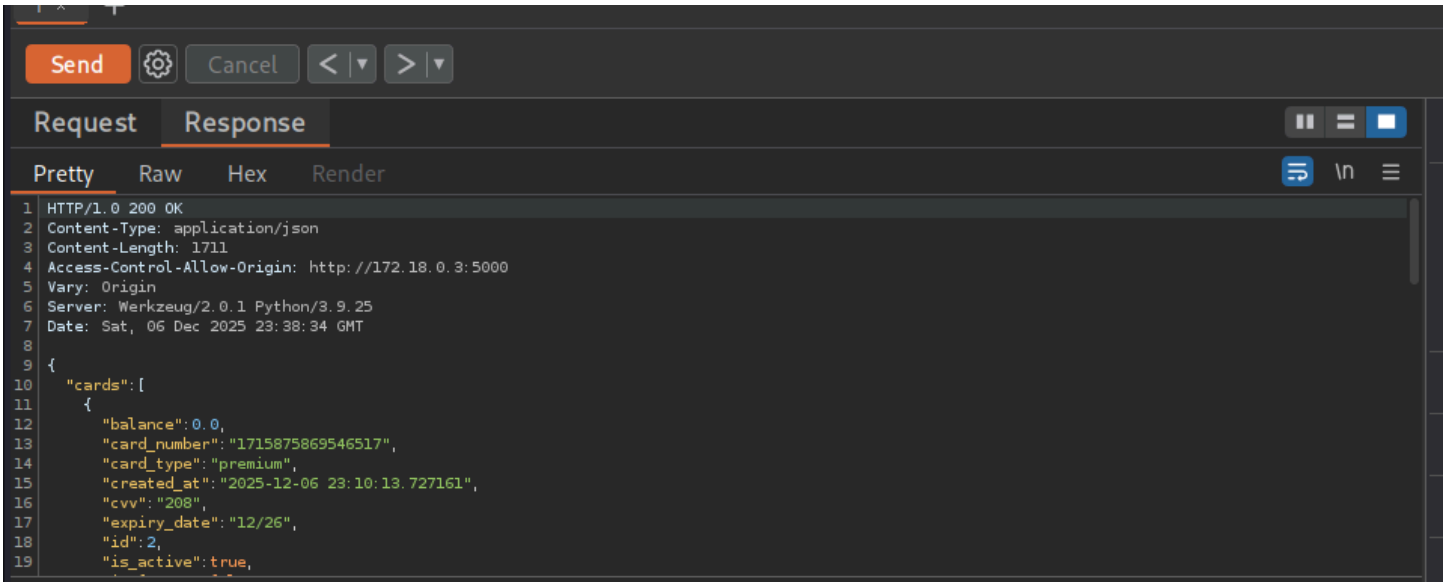
The **is_admin** property is entirely trust-based.
There is no server-side validation or role verification.
This makes it a classic BOPLA issue because the application relies on user-controlled object properties to make authorization decisions.

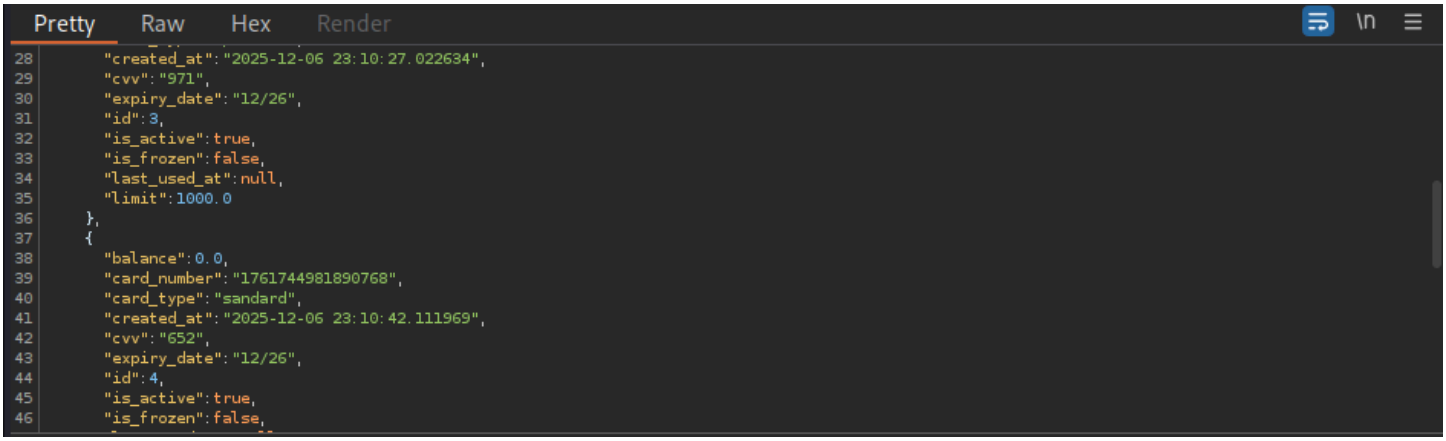
The modified token successfully grants admin access.



The new token is then used to view all virtual all card details which also make the system vulnerable to BOLA (change of token) and BOPLA (Excessive data exposure)

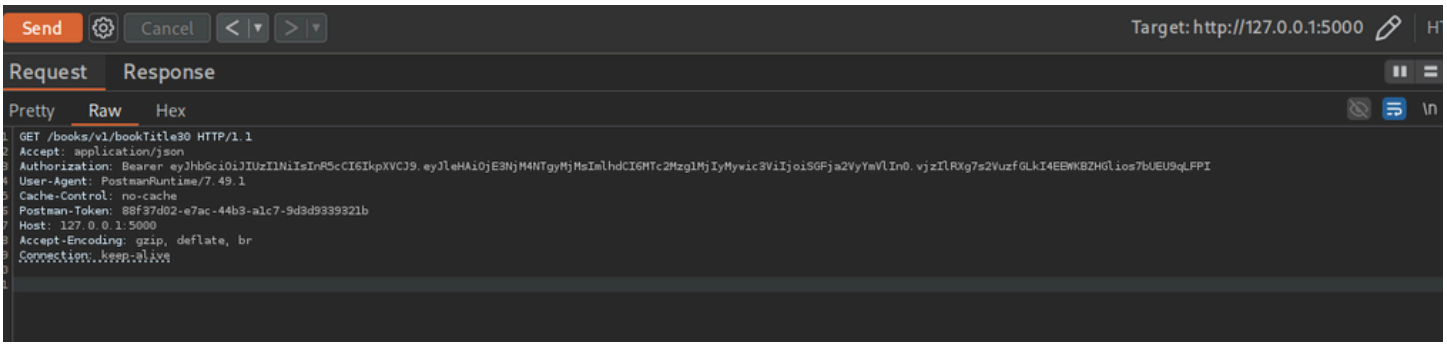


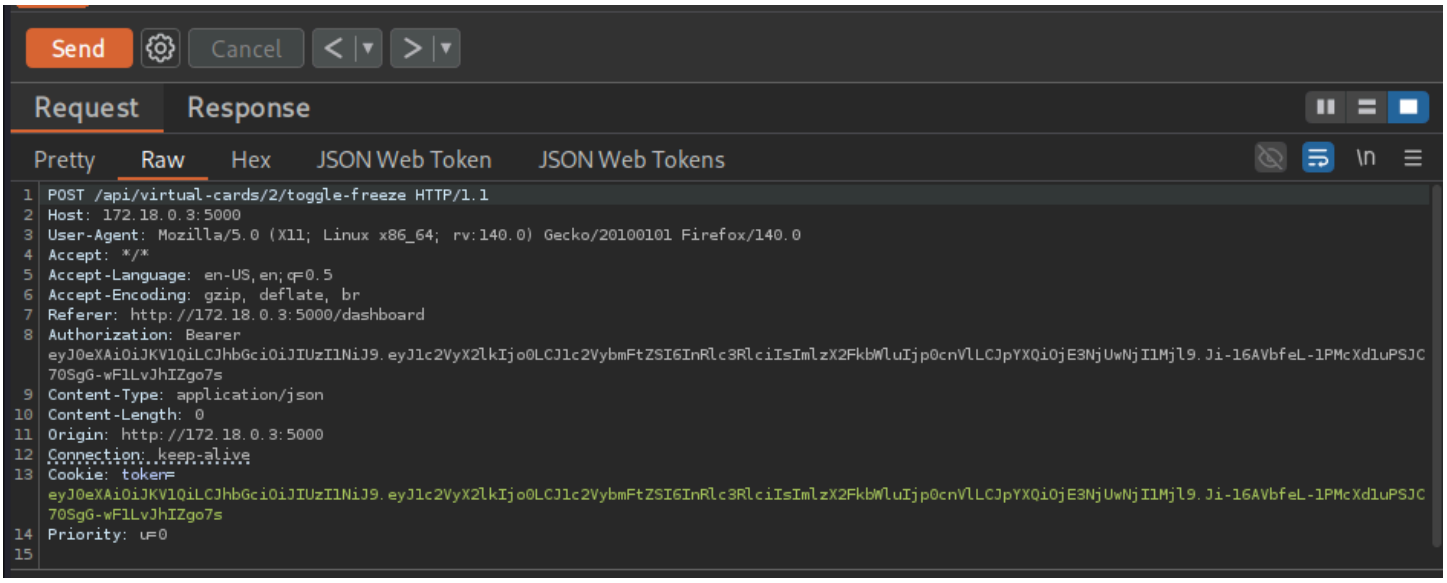


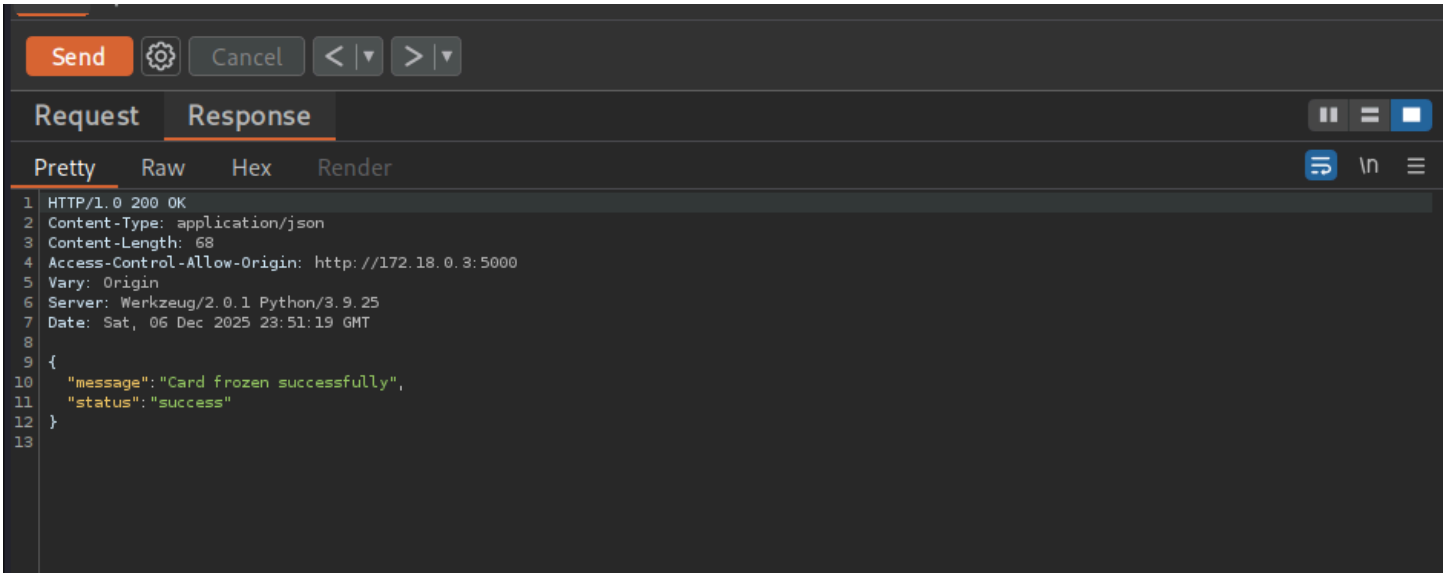


3. BOLA – Accessing Other Users' Virtual Card Details

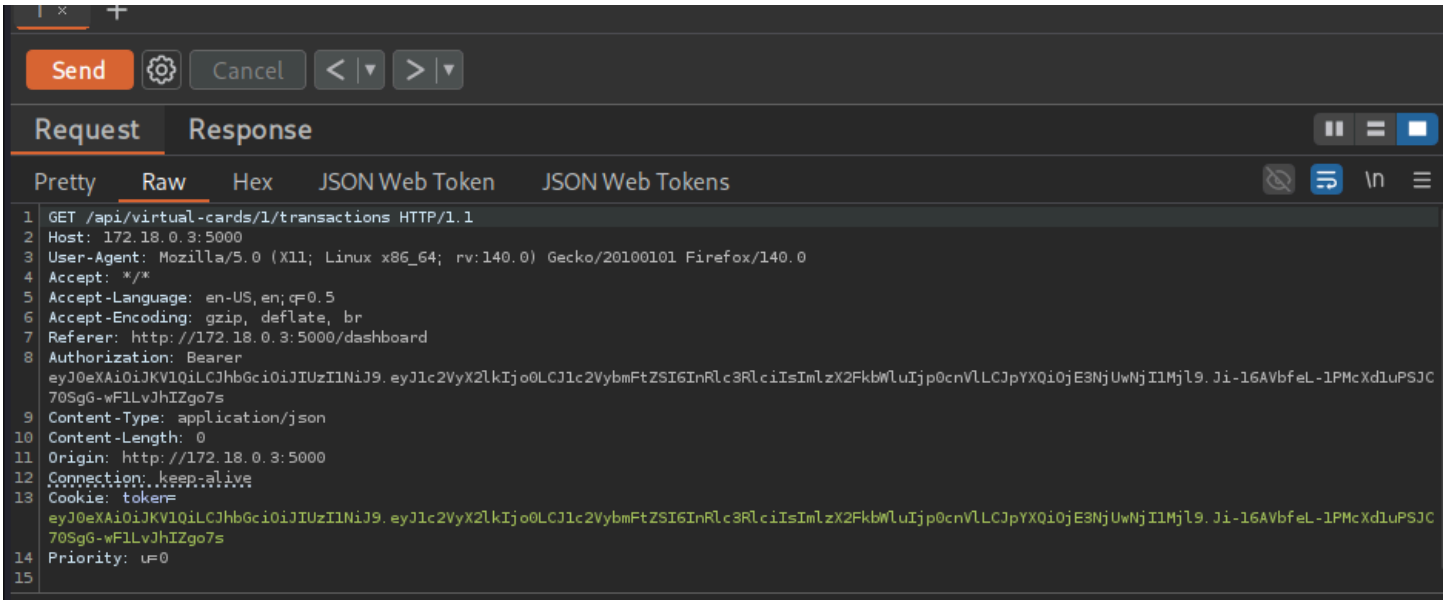
Once all the information about card details is been seen the **card_Id** the attack use it to freeze the cards by changing the ID on the URL: **POST / api/virtual-cards/{card_ID}/toggle-freeze**

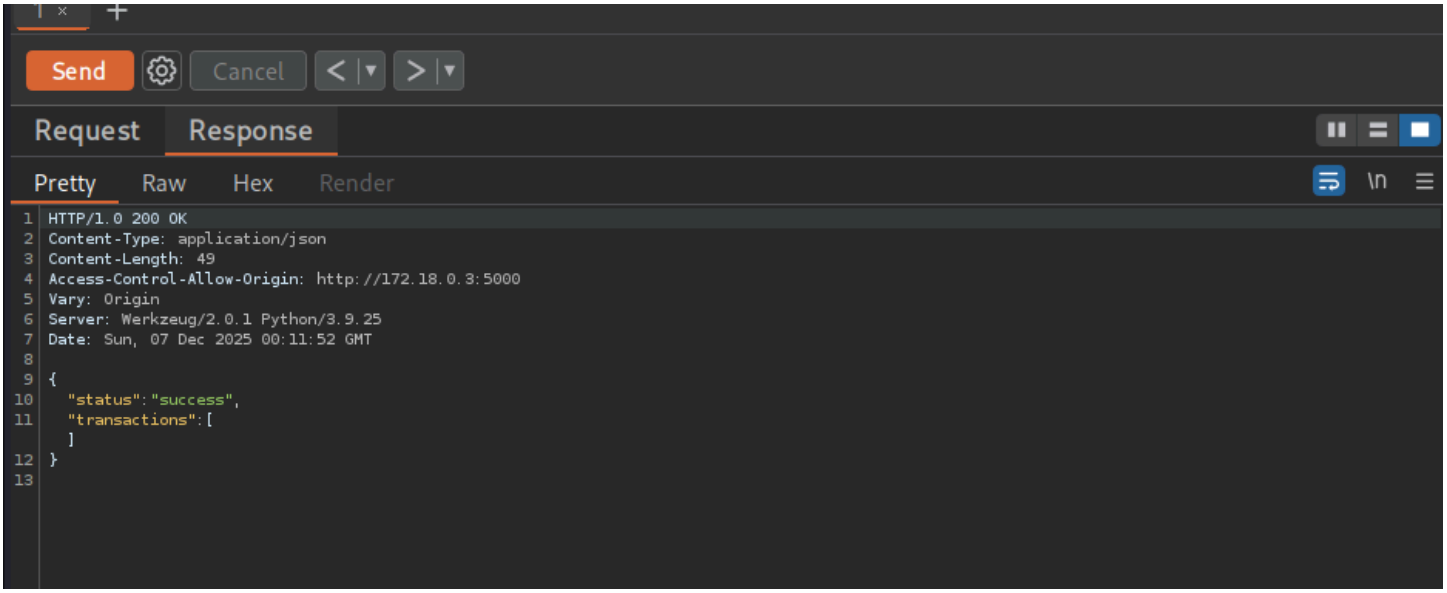


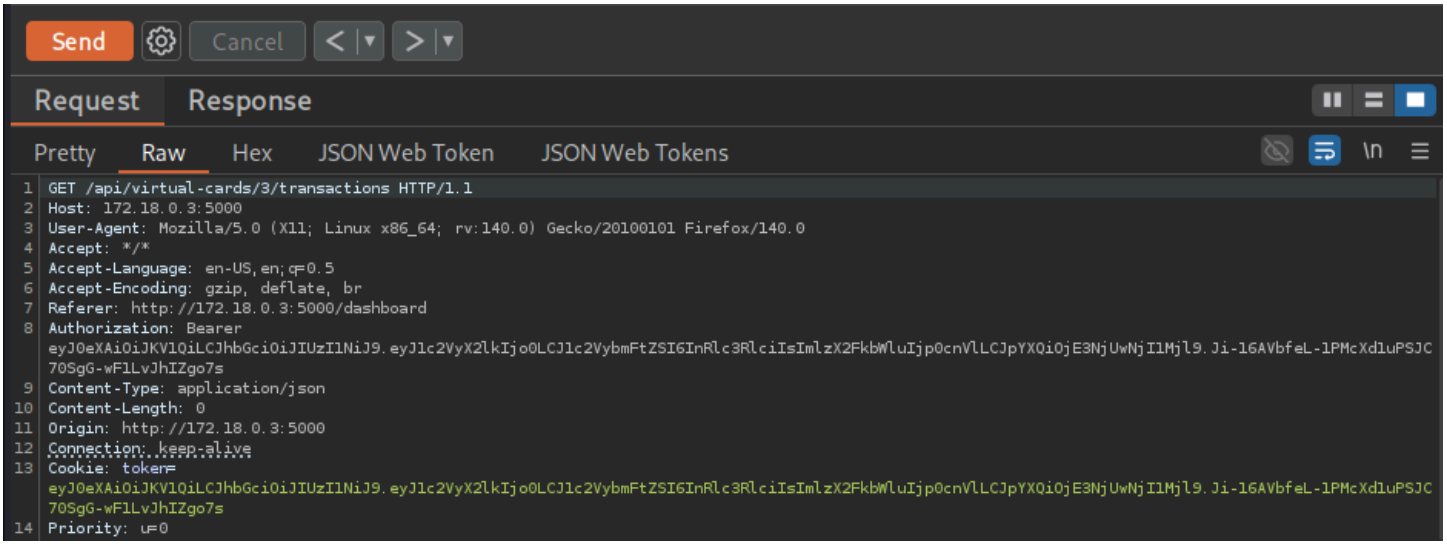




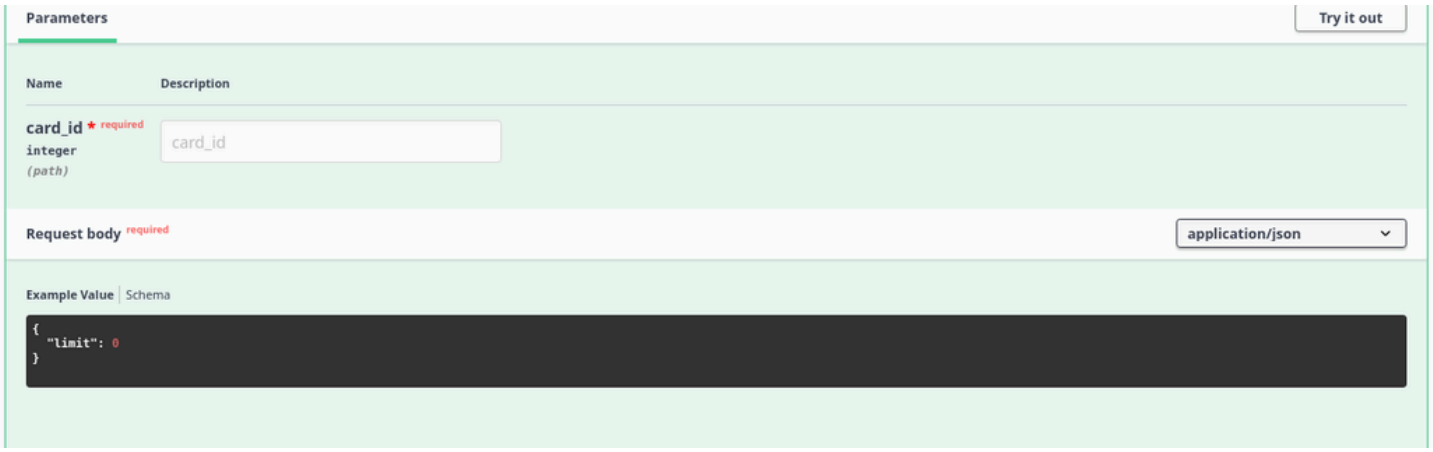
- Attack can also view availbable transactions details on each cards by editing the URL: **GET /api/virtual-cards/{card-Id}/transactions**

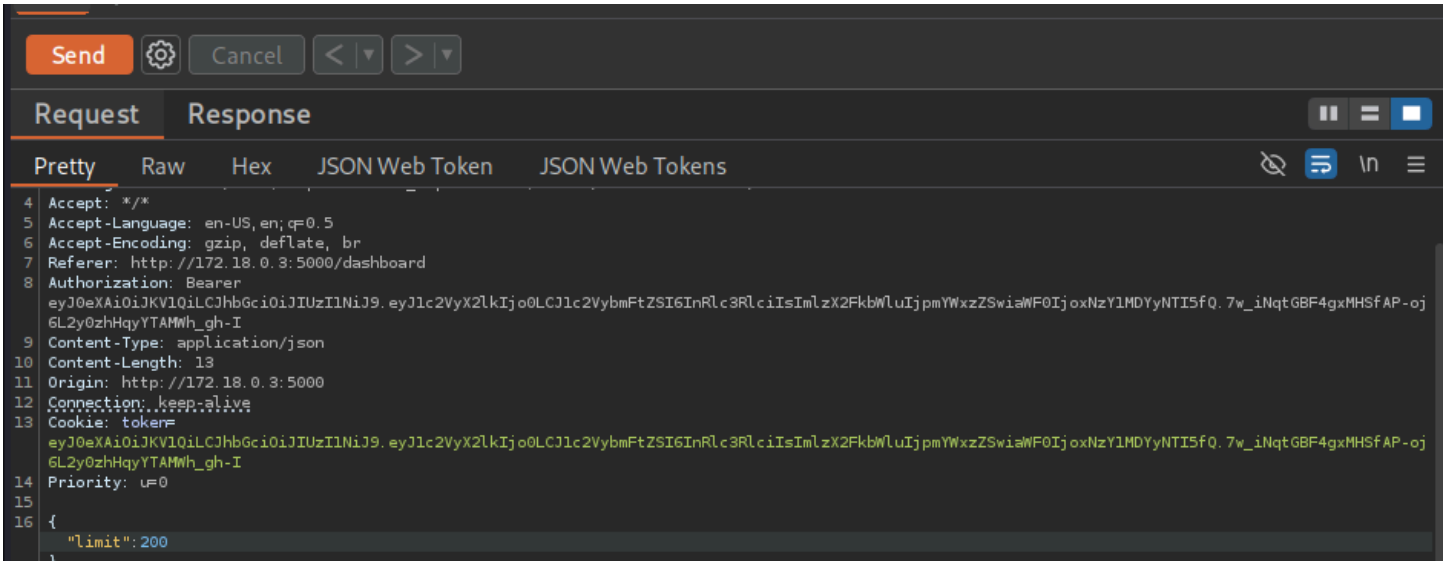


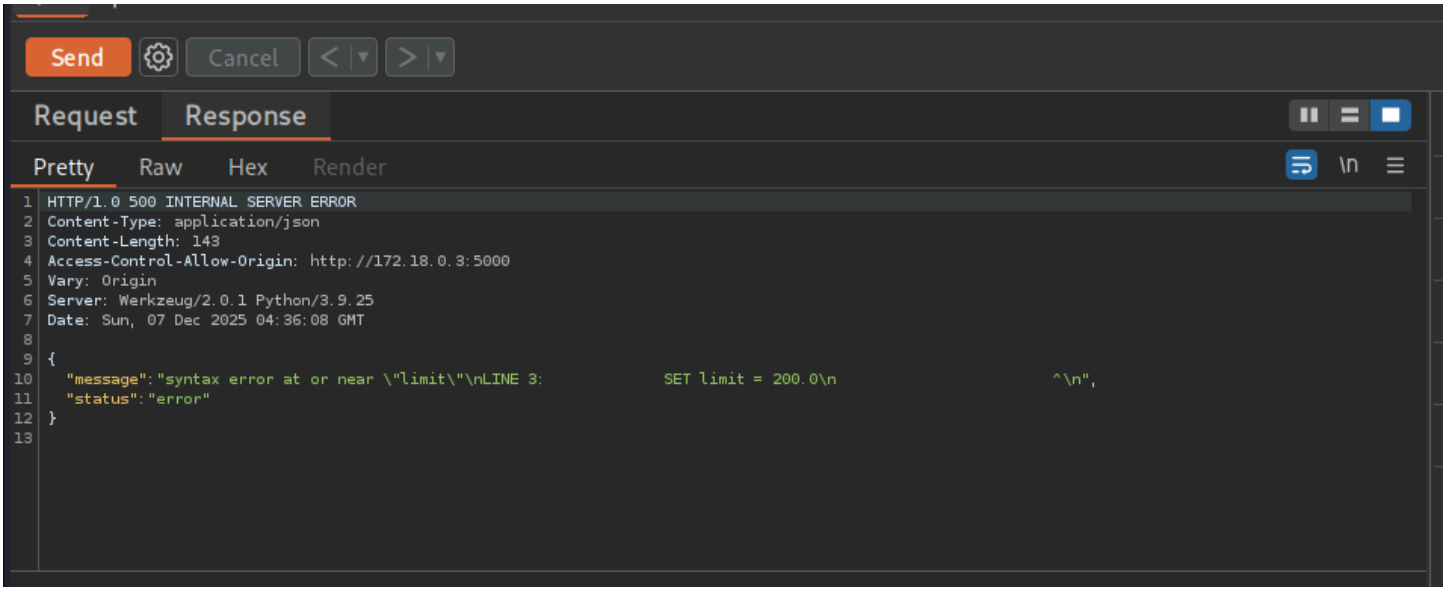




Here I discover improper API documentation when I tried manipulating then **“limit”** payload what the developer documented was not accepted

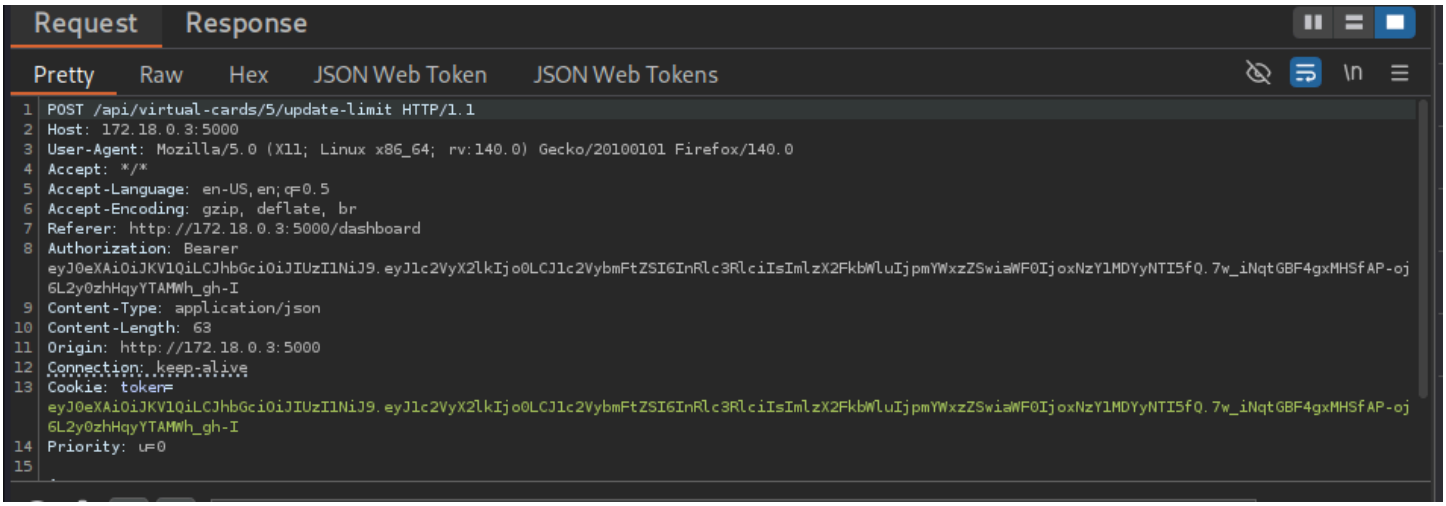


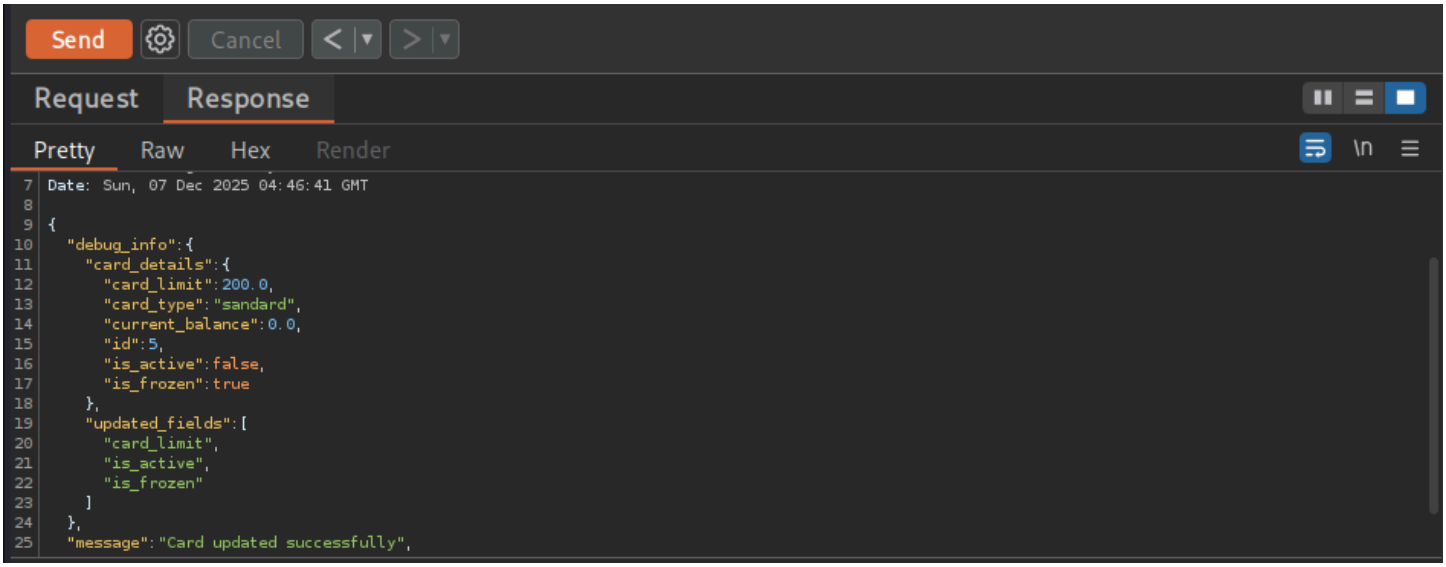




When the right payload was discovered I was able to change the card limit, freeze and deactivate the card that makes the API vulnerable to mass assignment **(BOPLA**) by accepting additional payload via URL :

POST /api/virtual-cards/{card_id}/update-limit





Combined Impact

This chain attack enables:

- Full privilege escalation
- Unauthorized access to admin functionality
- Enumeration of all card identifiers
- Extraction of virtual card details belonging to every user
- Complete compromise of confidentiality and authorization logic

This represents a high-severity, systemic failure in the API design.

Findings 3- OWASP API2 – Broken Authentication

Description

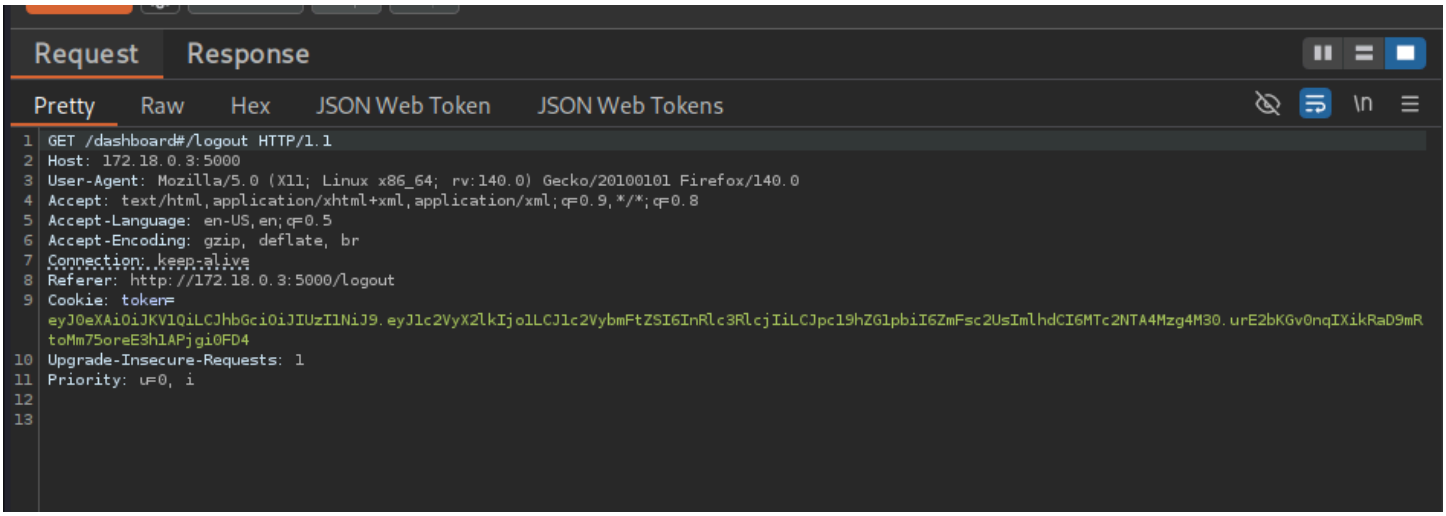
The application uses the same **Bearer token** inside the **cookie** without proper token invalidation or session destruction. Logging out does not revoke or expire the token, and the token remains fully active. The login response contains excessive sensitive information, including user roles and properties that should never be client controlled. A new user can register, modify the sensitive authorization properties in the login response, and escalate privileges through **mass assignment**. After escalation, the attacker can perform administrative actions such as deleting accounts, including the main administrator account.

The logout endpoint is not documented in the API specification, which results in Improper Inventory Management and makes it difficult to understand the session handling design. Even after logout, the attacker can still use the old token to update card limits or perform other privileged actions.

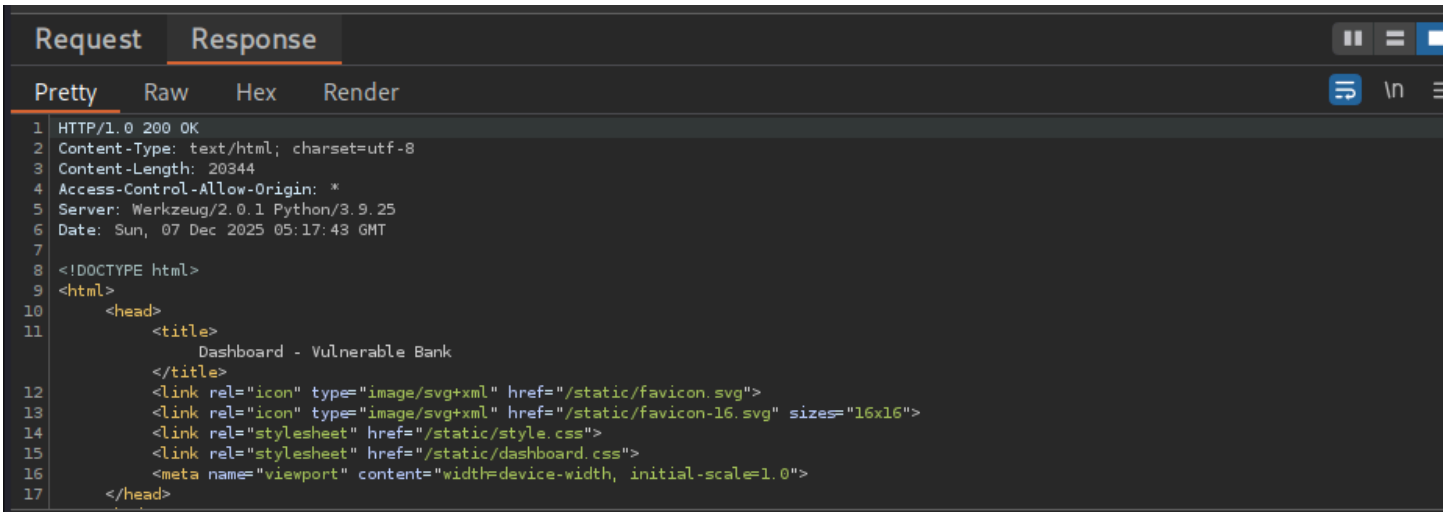
Because the system has BOLA issues, the attacker can change account identifiers in the URL and delete any account in the system.

- **Logout does not invalidate the session**

The user logs out using the undocumented endpoint: **GET /dashboard#/logout**



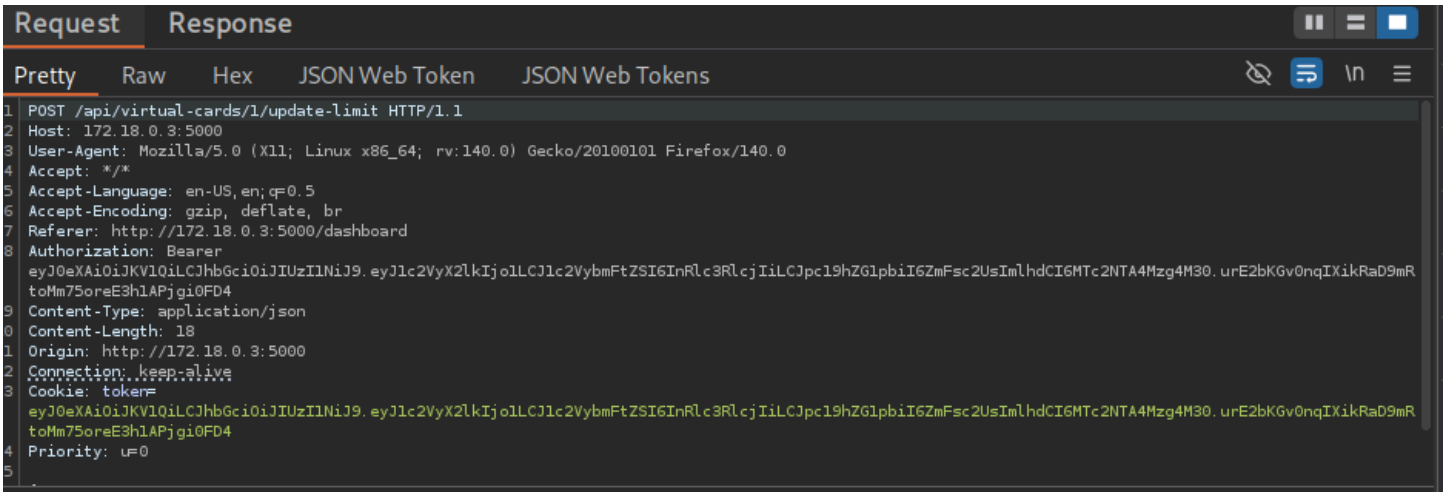
- The endpoint returns a successful logout response, but the previously issued token remains valid.



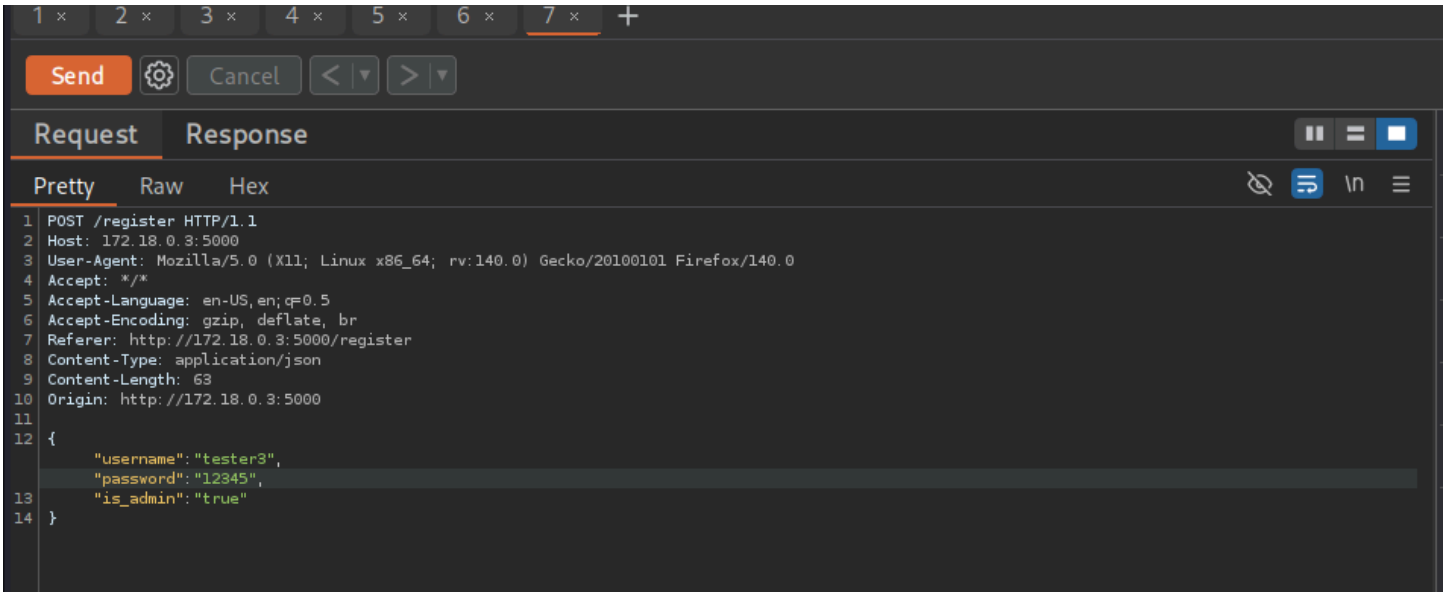
- The attacker uses the same token to update a card limit:

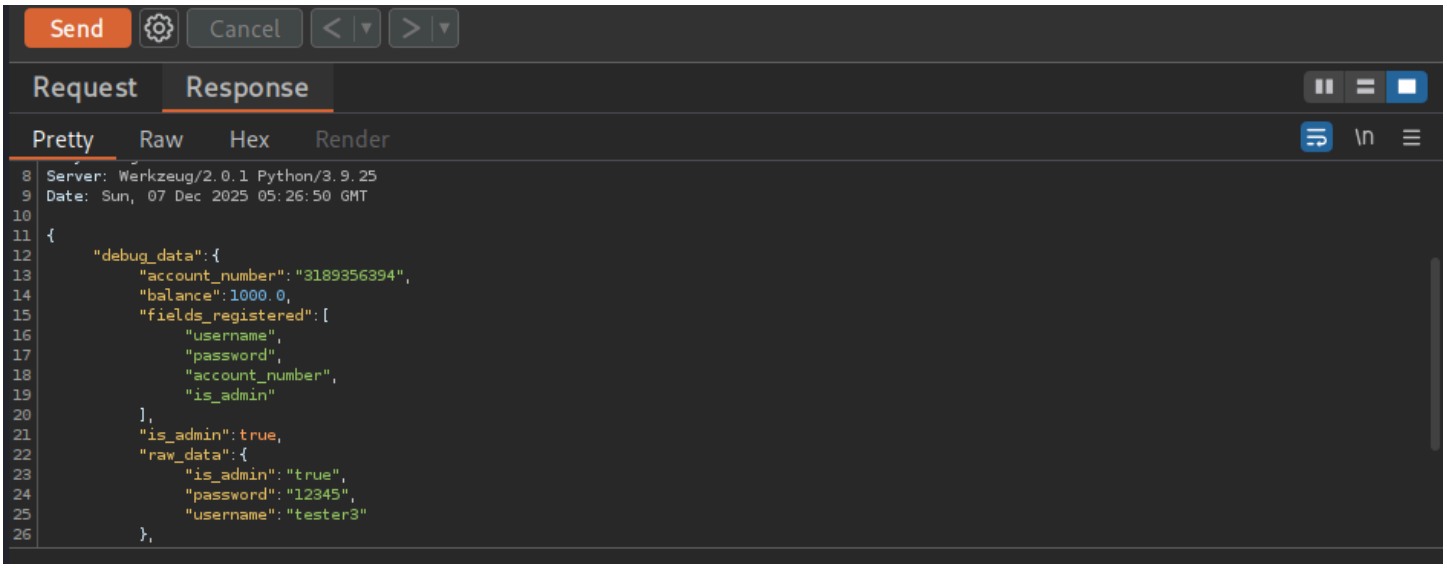
POST /card/update_limit/{card_id}

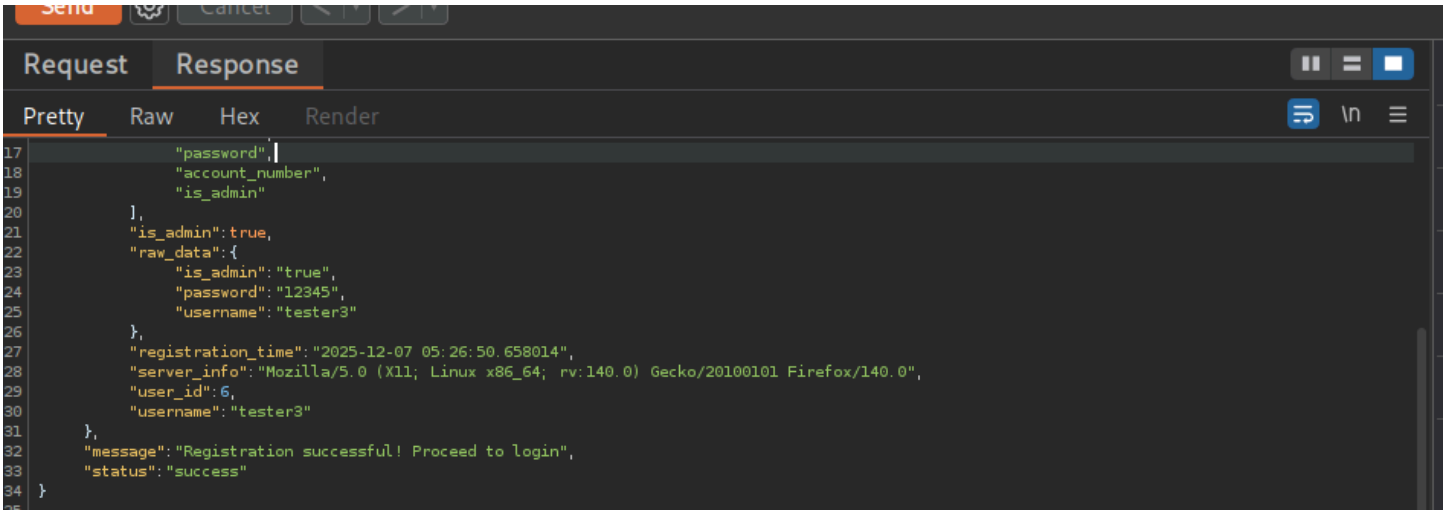
Authorization: Bearer <old_token>



- New user registration endpoint accept sensitive properties







Because the application is vulnerable to BOLA the new admin can easily use the account_id to delete users in the system

Pretty Raw Hex JSON Web Token JSON Web Tokens

```
POST /admin/delete_account/1 HTTP/1.1
Host: 172.18.0.3:5000
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:140.0) Gecko/20100101 Firefox/140.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Referer: http://172.18.0.3:5000/sup3r_s3cr3t_admin
Authorization: Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjo2LCJ1c2VybmFtZSI6InRlc3RlcjMiLCJpc19hZG1pbiI6dHJ1ZSwiaWF0IjoxNzY1MDg1NDcwfQ.4aH1MH4zS3GNurVjYyOnokG8GJR2vsPcVW_INxcFt14
Content-Type: application/json
Origin: http://172.18.0.3:5000
Connection: keep-alive
Cookie: token=
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjo2LCJ1c2VybmFtZSI6InRlc3RlcjMiLCJpc19hZG1pbiI6dHJ1ZSwiaWF0IjoxNzY1MDg1NDcwfQ.4aH1MH4zS3GNurVjYyOnokG8GJR2vsPcVW_INxcFt14
Priority: u=0
Content-Length: 0
```

1 × 2 × +

Send Cancel < >

Request Response

Pretty Raw Hex Render \n

```
HTTP/1.0 200 OK
Content-Type: application/json
Content-Length: 199
Access-Control-Allow-Origin: http://172.18.0.3:5000
Vary: Origin
Server: Werkzeug/2.0.1 Python/3.9.25
Date: Sun, 07 Dec 2025 06:20:24 GMT

{
    "debug_info":{
        "deleted_by":"tester3",
        "deleted_user_id":1,
        "timestamp":"2025-12-07 06:20:24.927692"
    },
    "message":"Account deleted successfully",
    "status":"success"
}
```

Search 0 highlights

Request Response

Pretty Raw Hex JSON Web Token JSON Web Tokens \n

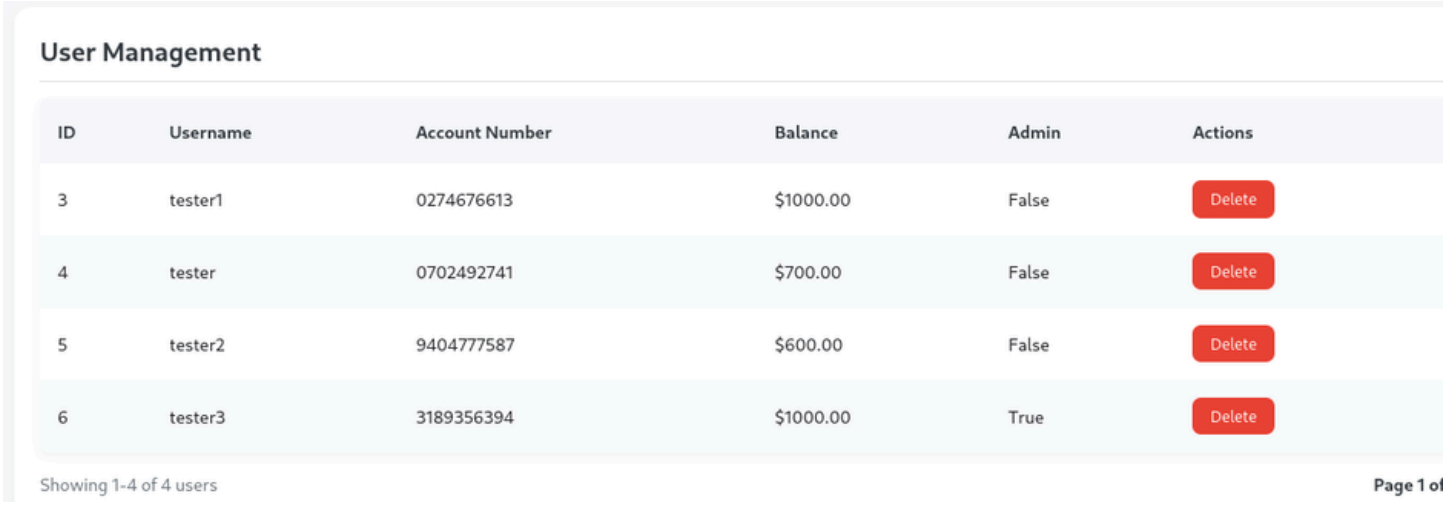
```
POST /admin/delete_account/2 HTTP/1.1
Host: 172.18.0.3:5000
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:140.0) Gecko/20100101 Firefox/140.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Referer: http://172.18.0.3:5000/sup3r_s3cr3t_admin
Authorization: Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjo2LCJ1c2VybmFtZSI6InRlc3RlcjMiLCJpc19hZG1pbiI6dHJ1ZSwiaWF0IjoxNzY1MDg1NDcwfQ.4aH1MH4zS3GNurVjYyOnokG8GJR2vsPcVW_INxcFt14
Content-Type: application/json
Origin: http://172.18.0.3:5000
Connection: keep-alive
Cookie: token=
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjo2LCJ1c2VybmFtZSI6InRlc3RlcjMiLCJpc19hZG1pbiI6dHJ1ZSwiaWF0IjoxNzY1MDg1NDcwfQ.4aH1MH4zS3GNurVjYyOnokG8GJR2vsPcVW_INxcFt14
Priority: u=0
Content-Length: 0
```

Request Response

Pretty Raw Hex Render

```
HTTP/1.0 200 OK
Content-Type: application/json
Content-Length: 199
Access-Control-Allow-Origin: http://172.18.0.3:5000
Vary: Origin
Server: Werkzeug/2.0.1 Python/3.9.25
Date: Sun, 07 Dec 2025 06:21:27 GMT

{
    "debug_info":{
        "deleted_by":"tester3",
        "deleted_user_id":2,
        "timestamp":"2025-12-07 06:21:27.693138"
    },
    "message":"Account deleted successfully",
    "status":"success"
}
```

- remaining users



Impact

- Logout functionality does not destroy sessions, which results in persistent authentication.
- Attackers can modify sensitive authorization fields through mass assignment.
- Full privilege escalation is possible without server validation.
- Any account can be deleted through BOLA.
- The entire user base can be removed by an unauthorized actor.
- The main administrative account can be destroyed, which compromises the integrity of the entire system.

Remediation

- Implement strict server-side token invalidation on logout.
- Remove sensitive authorization fields such as **is_admin** from client controlled responses.

- Enforce server-side validation of all authorization properties.
- Prevent mass assignment by using an explicit **allowlist** for updatable fields.
- Apply object ownership validation for all account and card related actions.
- Restrict and document all endpoints to eliminate Improper Inventory Management.

Final Conclusion

The assessment of the vulnerable bank application revealed multiple high-impact weaknesses across authentication, authorization, and endpoint inventory. The issues identified form a complete attack chain that allows unauthorized data access, privilege escalation, and full compromise of user accounts. These weaknesses significantly reduce the overall security posture of the application.

Using **CVSS v4.0 risk scoring**, the combined severity of the findings reaches the **Critical** level. Broken Authentication, Broken Object Level Authorization, Mass Assignment, and Improper Inventory Management enable an attacker to bypass trust boundaries and perform actions that directly affect confidentiality, integrity, and availability.

The application does not meet several essential requirements from **GDPR**, **PCI DSS**, and **NIST 800-53**. GDPR requires strict protection of personal data, but the system discloses sensitive information through tokens and vulnerable endpoints. PCI DSS requires strong access control, secure authentication, and hardened session management, all of which are missing. NIST 800-53 emphasizes secure design principles, including least privilege, session control, and proper access enforcement. The application fails to meet these controls based on the observed behavior.

The current security gaps introduce significant operational and compliance risks. Immediate remediation is required to strengthen identity handling, enforce access boundaries, improve endpoint visibility, and align the system with the necessary regulatory and security standards.

VAmPI – Vulnerable API Security Testing Report

Tester: Blessing Isaiah
Report Date: November 23, 2025

Environment

Local VAmPI instance deployed directly on the host system inside a virtual environment.
The web interface was available at **127.0.0.1:5000** over HTTP.
BurpSuite and Postman listened on **port 8080** for traffic interception.

Executive Summary

Objective

A company requested a security review of their API before release. The task was to review the Swagger file and look for possible security issues. The goal of the assessment was to test the API in a controlled lab environment, find weaknesses in its design and behavior, and provide clear recommendations.

Methodology

The assessment followed a white-box testing method after gaining an understanding of the API structure.

Testing Approach

Testing steps included:

- Passive and active reconnaissance through Postman
- Automated OpenAPI generation with Swagger tools
- Manual editing of **Openapi.yml** for proper structure and JSON conversion
- Targeted fuzzing to test authentication and authorization weaknesses
- Combining findings to simulate realistic attack paths

Attacker Progression

1. Environment Setup and Proxy Configuration

- Configured BurpSuite and Postman to intercept HTTP traffic between VAmPI and the server.

2. Traffic Capture and Reconnaissance

- Observed user actions to capture baseline requests and responses.

3. Manual Specification Enhancement

- Edited **Openapi.yml** in nano to add missing IP information.

- Moved the file to **~/Downloads** for importing into Postman.

4. Endpoint Testing and Fuzzing

Sent targeted requests to endpoints to test for:

- Broken Object Level Authorization(BOLA)
- Excessive data exposure
- Lack of access controls on sensitive endpoints

5. Documentation and Evidence Collection

- Recorded all actions in order.
- Saved screenshots and proof-of-concept logs for each finding.

Standards Followed

The assessment was aligned with:
- OWASP API Security Top 10 (2023)
- CVSS v4.0 risk scoring
- GDPR, PCI DSS, and NIST 800-53 compliance guidelines

Major Tools Used

- **Recon and scanning tools:** GitHub, Swagger
- **Proxy tools:** BurpSuite Pro, Firefox
- **API testing tools:** Postman, Burp Repeater, Burp Intruder
- **Specification tools:** nano, Swagger Editor
- **Environment tools:** Kali Linux, VirtualBox, pyenv for Python version control

Scope

In Scope

- Local VAmPI application running on the Kali VM
- HTTP traffic captured with BurpSuite and Postman
- OpenAPI specification generation and updates

Out of Scope

- Any external or production systems
- Activities outside the controlled lab environment
- Destructive actions or data extraction to external servers

Findings: chronological (OWASP mapping, description, evidence, impact, remediation)

Finding 1:

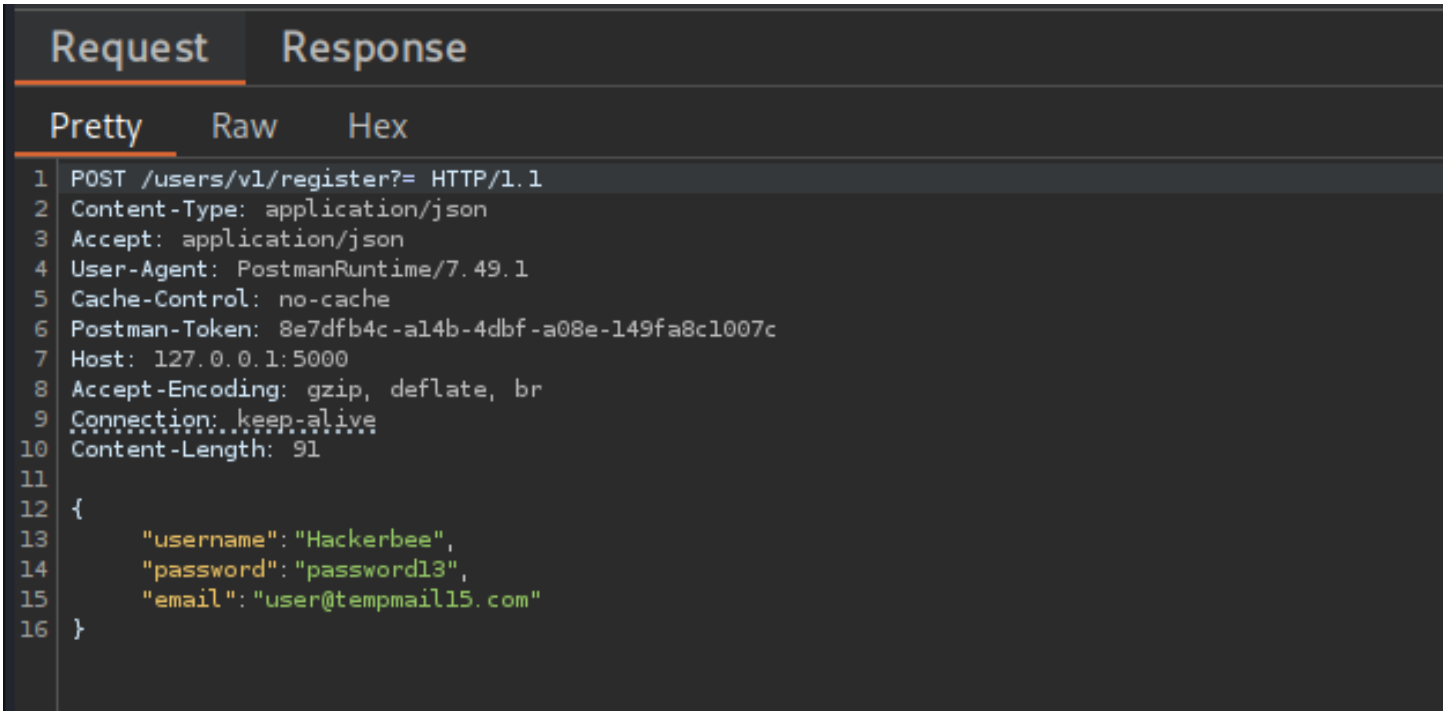
OWASP Mapping

OWASP API3 – Broken Object Level Authorization (2023)
The object-level access checks are missing

Description
The API does not verify that the user requesting a book is the actual owner of that book. The database query does not include the requester's username or user ID. As long as a user has a valid Bearer token, that user can retrieve books created by others, including the associated secret value.

Evidence / Proof of Concept

- Register a new user



- Log in and obtain a valid authentication token.

Request Response

Pretty Raw Hex

```
1  POST /users/v1/login HTTP/1.1
2  Content-Type: application/json
3  Accept: application/json
4  User-Agent: PostmanRuntime/7.49.1
5  Cache-Control: no-cache
6  Postman-Token: b3ce7049-ad1b-47d0-8933-35cfe3035a5a
7  Host: 127.0.0.1:5000
8  Accept-Encoding: gzip, deflate, br
9  Connection: keep-alive
10 Content-Length: 57
11
12 {
13     "username":"Hackerbee",
14     "password":"password13"
15 }
```

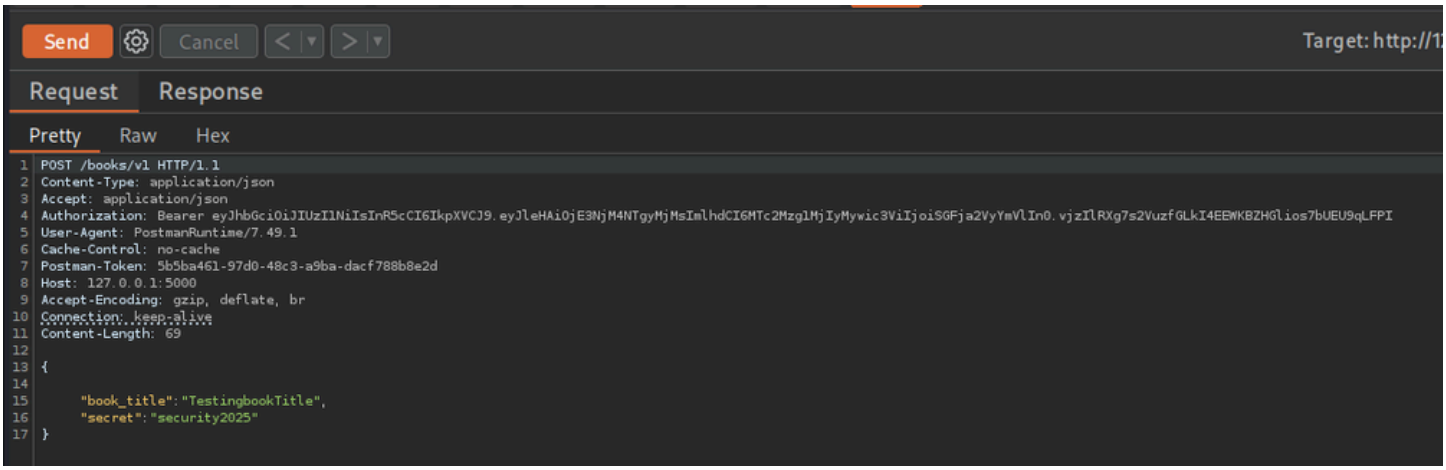
- Copy the Authentication token

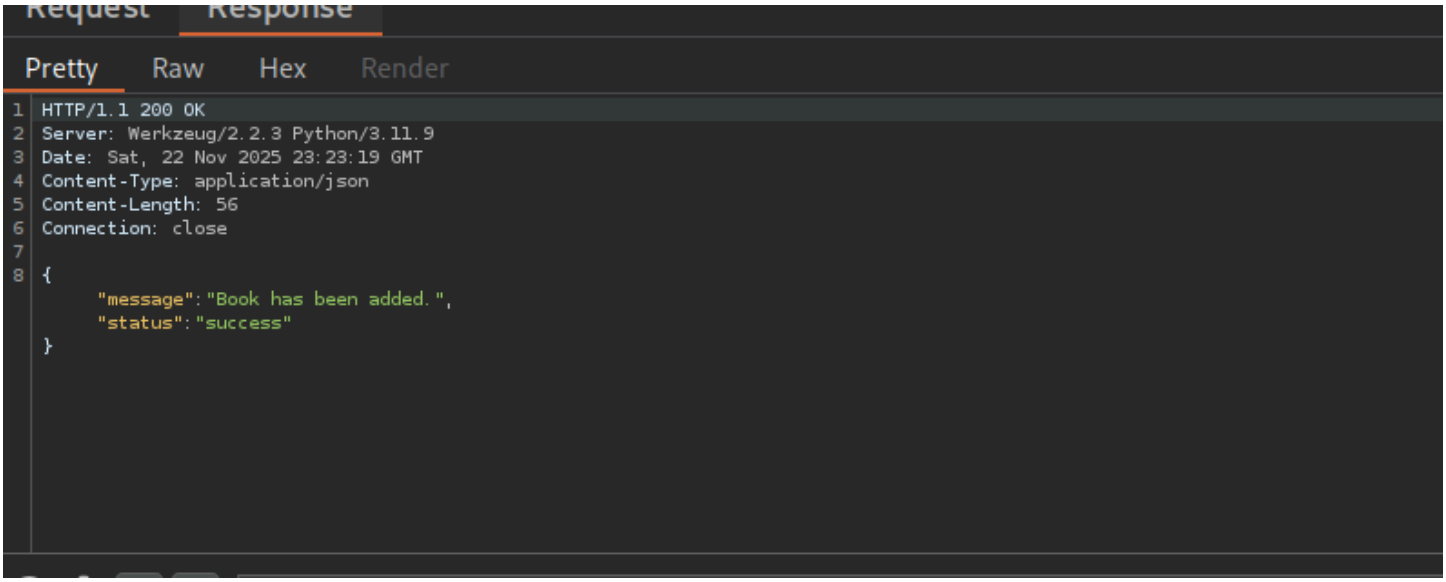
Request Response

Pretty Raw Hex Render

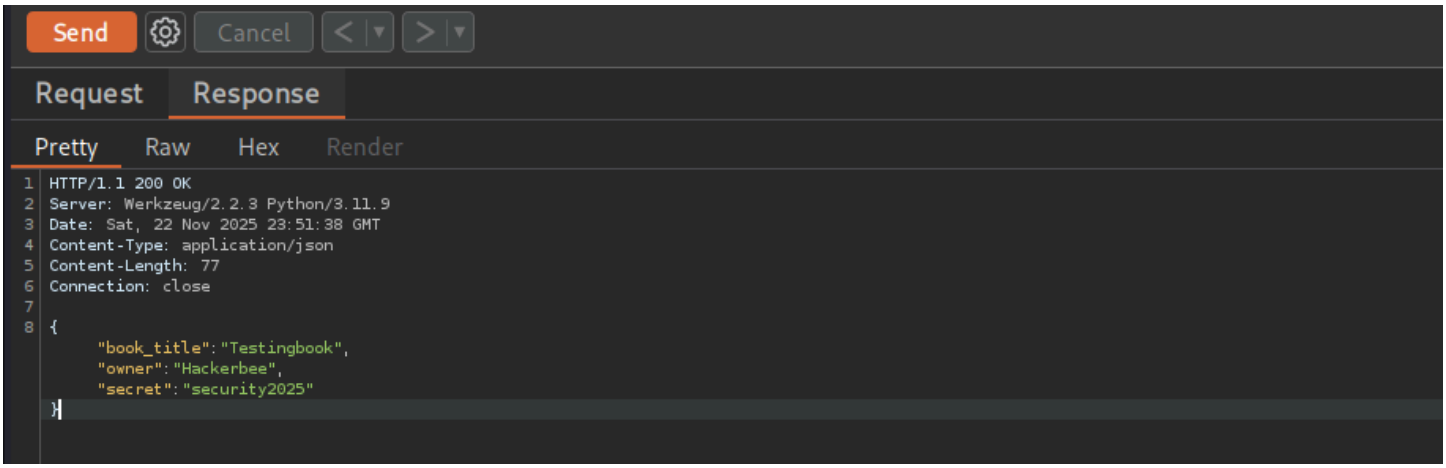
```
1 HTTP/1.1 200 OK
2 Server: Werkzeug/2.2.3 Python/3.11.9
3 Date: Sat, 22 Nov 2025 22:57:03 GMT
4 Content-Type: application/json
5 Content-Length: 229
6 Connection: close
7
8 {
    "auth_token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE3NjM4NTgyMjMsImlhdCI6MTc2MzglMjIyMywic3ViIjoiSGFja2VyYmVlIn0.vjzIlRXg7s2VuzfGLkI4EEWKBZHGlios7bUEU9qLFPI",
    "message":"Successfully logged in.",
    "status":"success"
}
```

- Use the token to create your own book.

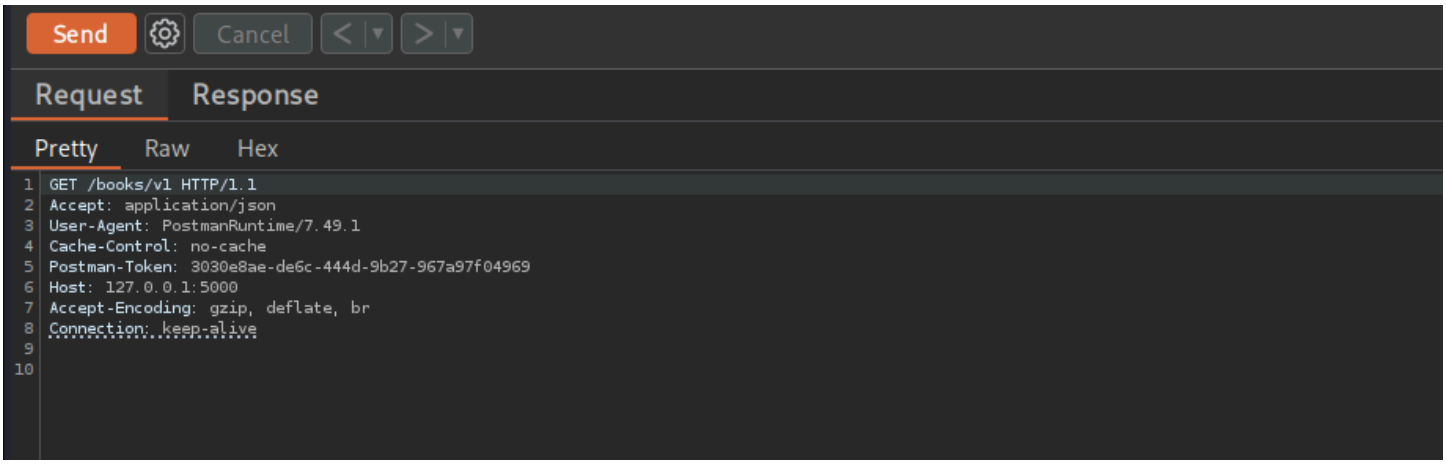




- Retrieved the book through: **GET /books/v1/Testingbook HTTP/1.1**



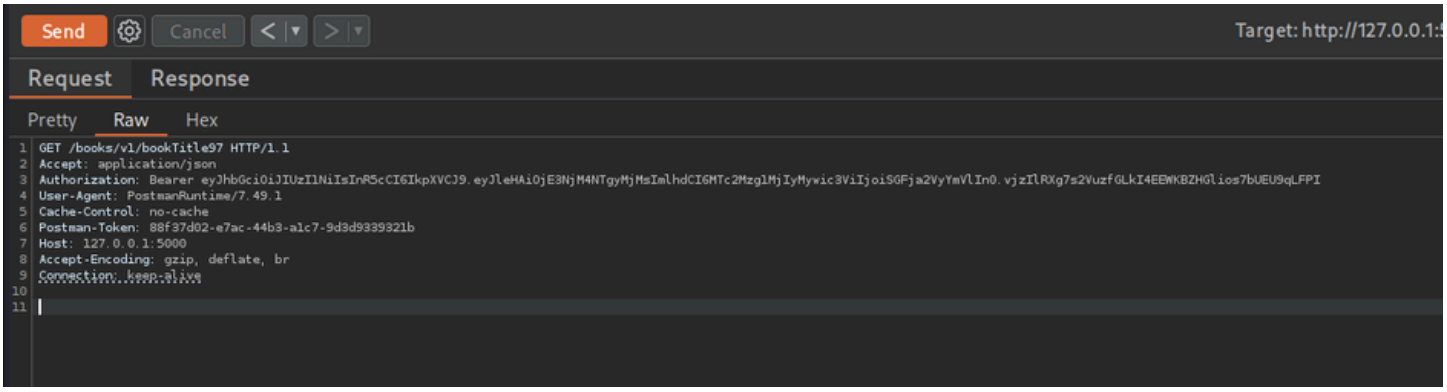
- Remove the book name from the path and attempt to list all books.

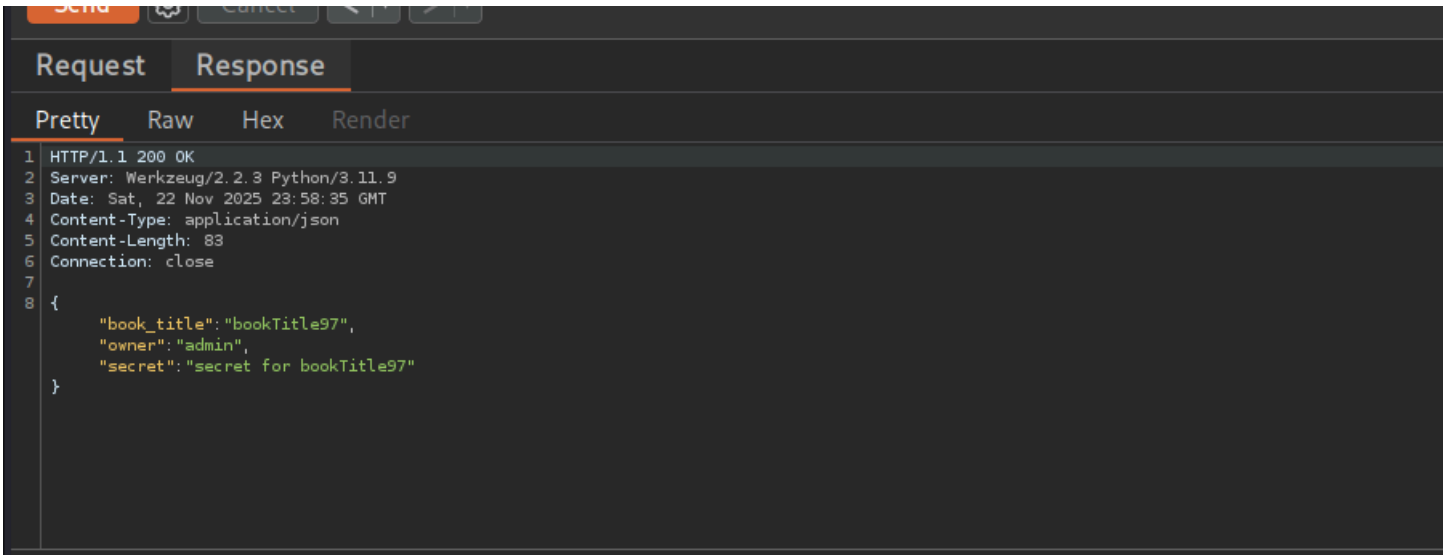


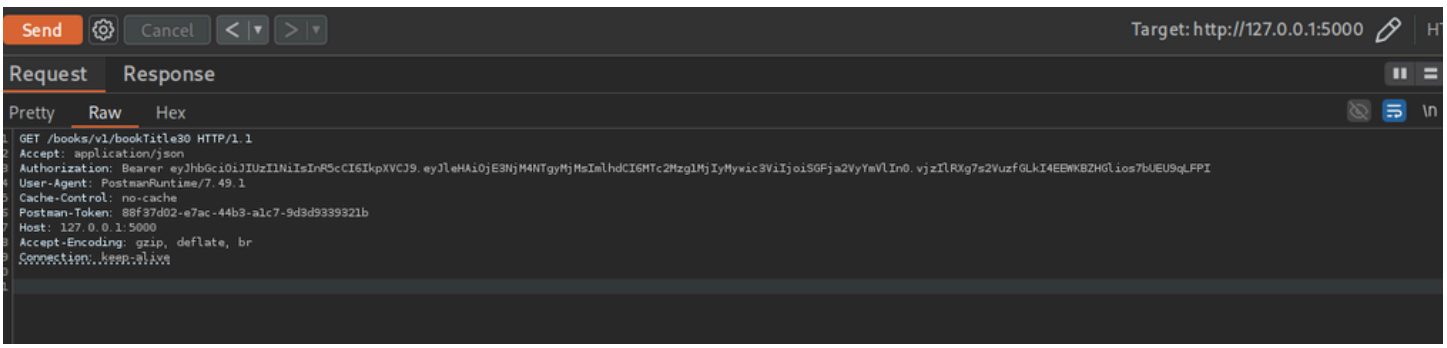
- The API exposes books created by other users

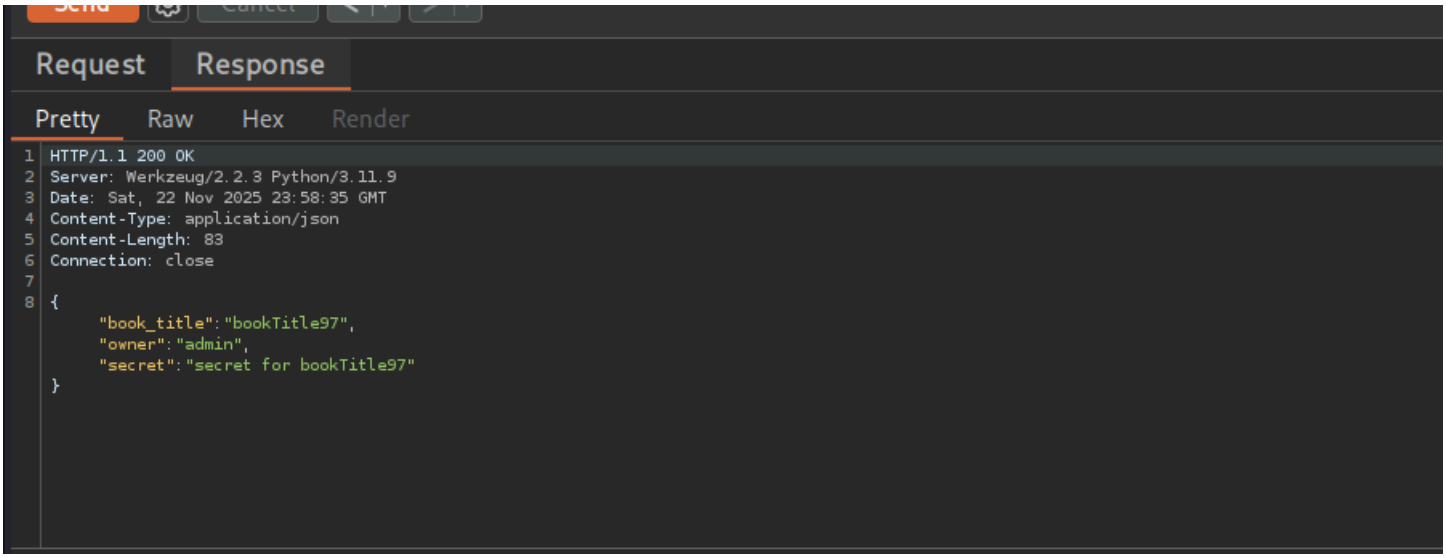
```
        },
        {
            "book_title": "bookTitle30",
            "user": "name2"
        },
        {
            "book_title": "bookTitle97",
            "user": "admin"
        },
        {
            "book_title": "TestingbookTitle",
            "user": "Hackerbee"
        },
        {
            "book_title": "Testingbook",
            "user": "Hackerbee"
        }
    ]
}
```


- Copy any book title belonging to another user and place it in the endpoint.
- The API returns that book and its secret without checking ownership.









Impact

Unauthorized access to other users' book objects exposes sensitive information and breaks data confidentiality.

CVSS v4.0 Risk Rating: High
The vulnerability allows unauthorized access to sensitive data with no special privilege beyond a valid token.

GDPR Impact:
Personal data disclosure violates **GDPR Article 5 (data minimization)** and **Article 32 (security of processing).** Any exposed data linked to identifiable individuals is considered a data breach.

PCI DSS Impact:
If the API ever handles or touches account-related information tied to payment activity, unauthorized data access would violate **PCI DSS Requirement 3 (protect stored data)** and **Requirement 7 (restrict access to cardholder data).**

NIST 800-53 Impact:
This flaw breaks **AC-3 (Access Enforcement)** and **AC-6 (Least Privilege)** because the system does not enforce proper ownership rules and exposes data to unauthorized subjects.

Recommended Fix

- Enforce ownership checks at the object level.
- Before returning any book record, validate that the user ID from the authentication token matches the user ID of the record owner. This prevents unauthorized access to data belonging to other users.

Finding 2: Excessive data exposure

OWASP Mapping

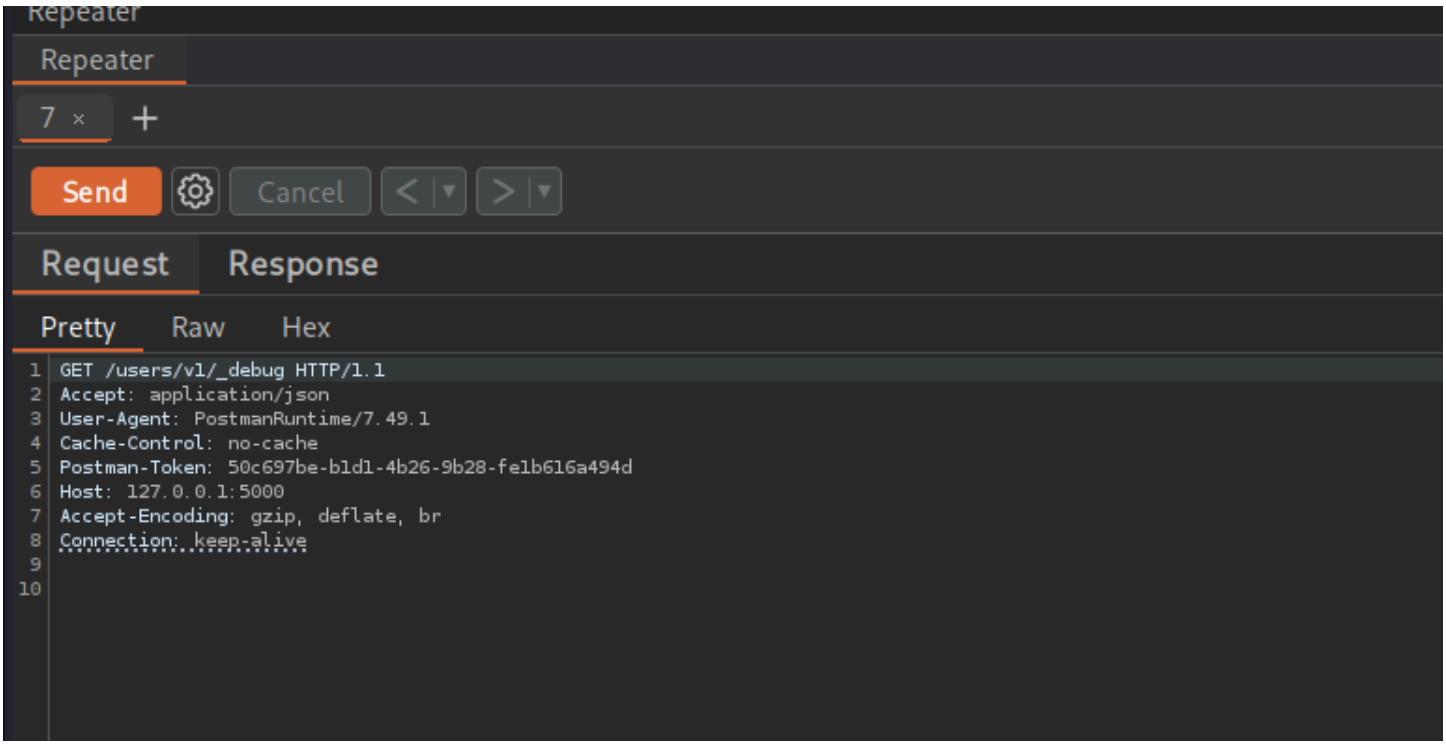
OWASP API3 – Broken Object Property Level Authorization (2023)
The endpoint returns more fields than intended, including confidential properties such as passwords and admin indicators.

Description

A debug endpoint was left active in the application. This type of endpoint is often created during development for troubleshooting and is commonly forgotten during cleanup. Although debug routes usually do not appear in API documentation, this one was still reachable and returned sensitive fields that should never be exposed. The endpoint exposed full user records, including credentials and administrative attributes, that belonged to previously deleted users but were still active in the database.

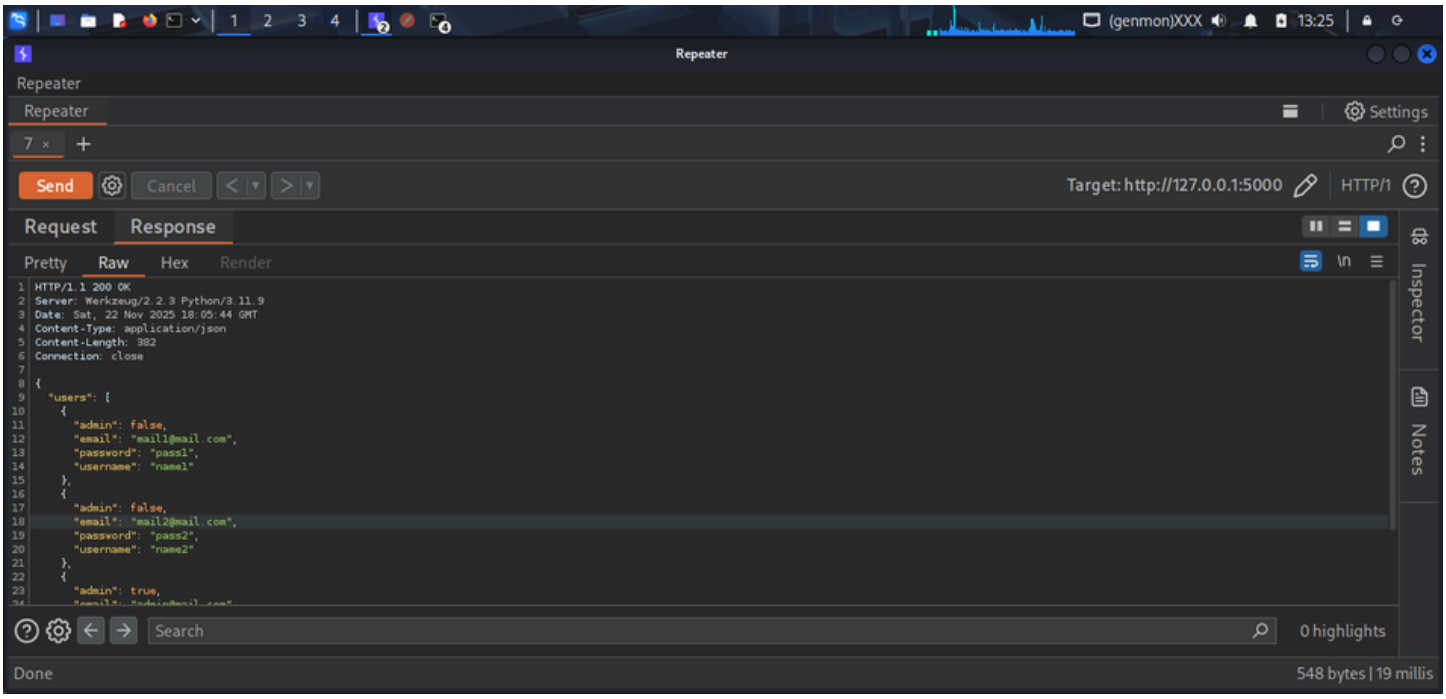
Evidence / Proof of Concept

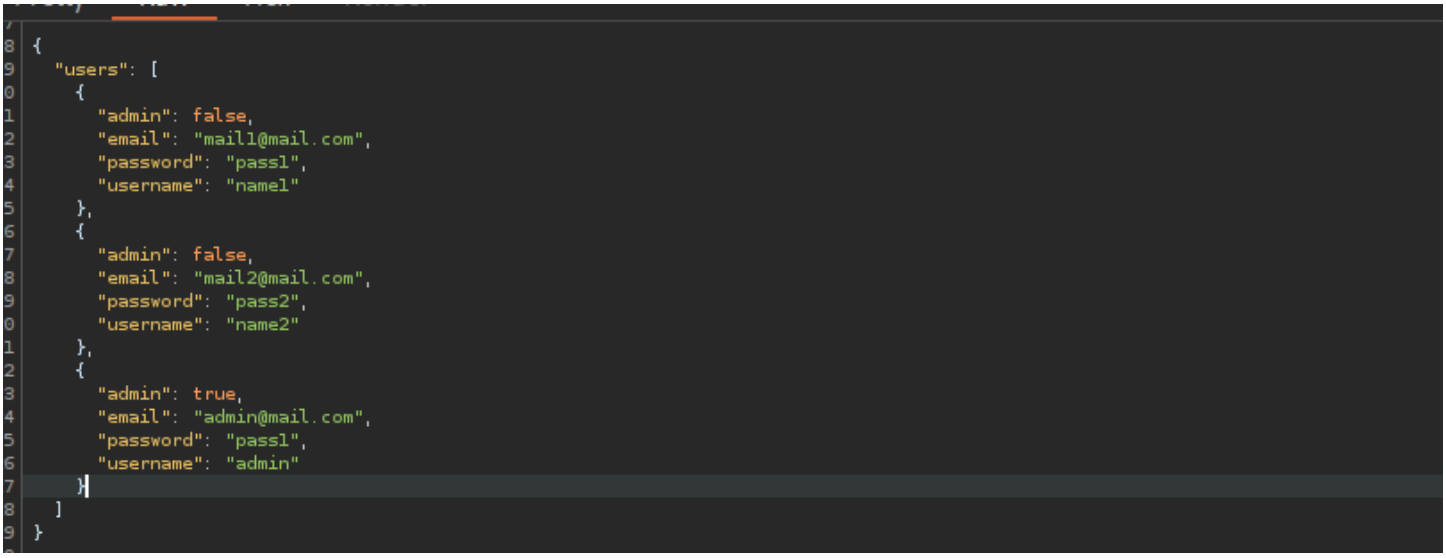
- Access the debug endpoint through the exposed route: **GET/users/v1/_debug HTTP/1.1**



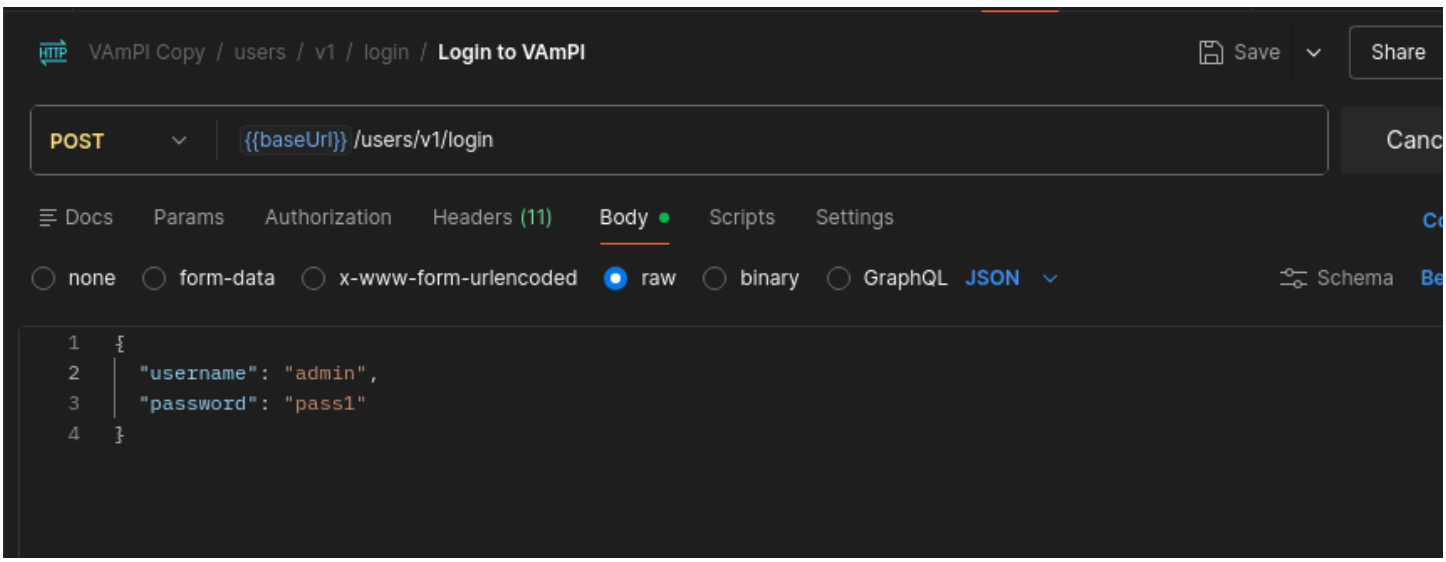
The endpoint responded with full user objects containing:

- Username
- Email
- Password
- Admin status





- Deleted users were still active in the system, allowing login using the leaked credentials.
- The endpoint returned far more data than necessary, confirming excessive data exposure.



Impact

Exposing credentials and admin status allows an attacker to access the system with elevated privileges, potentially compromising all users and administrative functions.

CVSS v4.0 Risk Rating: Critical

GDPR Impact: Violates GDPR **Articles 5 and 32** due to improper handling of personal and sensitive data. Leaked credentials linked to identifiable users constitute a reportable breach.

PCI DSS Impact: Violates requirements 3, 6, and 7 for protecting stored data, secure development, and access control.

NIST 800-53 Impact: Violates:

- AC-3 (Access Enforcement)
- AC-6 (Least Privilege)
- IA-5 (Authenticator Management)
- SI-12 (Information Exposure)

Recommended Fix

- Remove all debug endpoints from production code.
- Implement property-level access control to ensure only necessary fields are returned based on the requesting user's role. Fully deactivate deleted users and enforce secure storage of credentials.

Finding 3: Lack of Access Controls on Sensitive Endpoints

OWASP Mapping

API5 -Security Misconfiguration (2023)

Sensitive operations were exposed without proper authorization checks, allowing unauthorized users to perform privileged actions.

Description

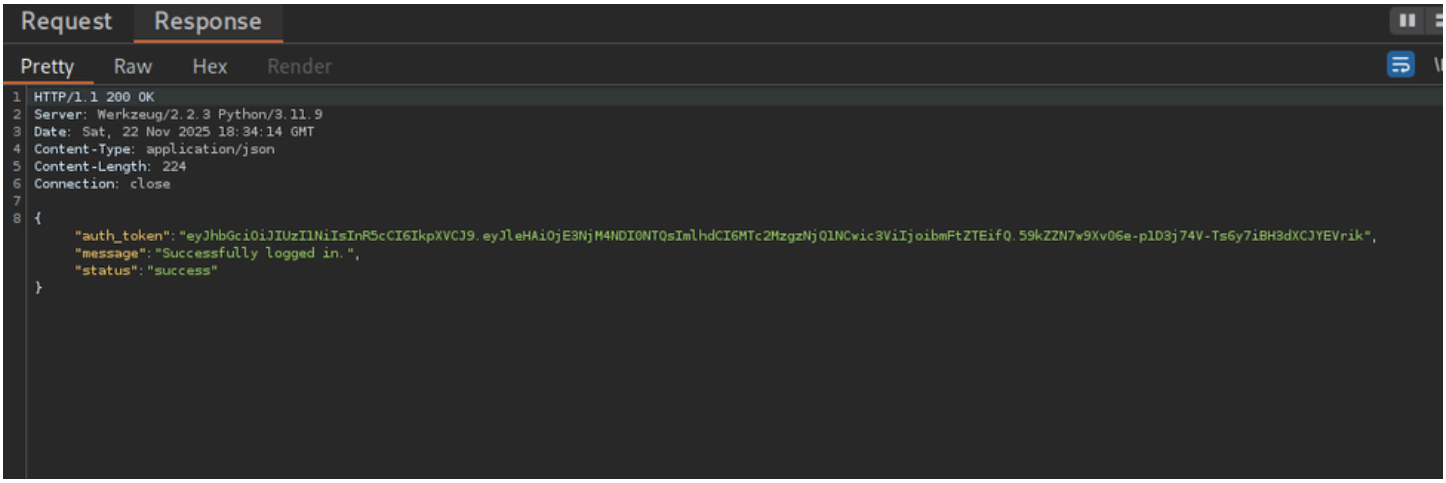
The password update functionality relied on the username supplied in the URL path. The application did not validate the requesting user through the token. This flaw allowed an attacker to update the password of any account.

Evidence / Proof of Concept

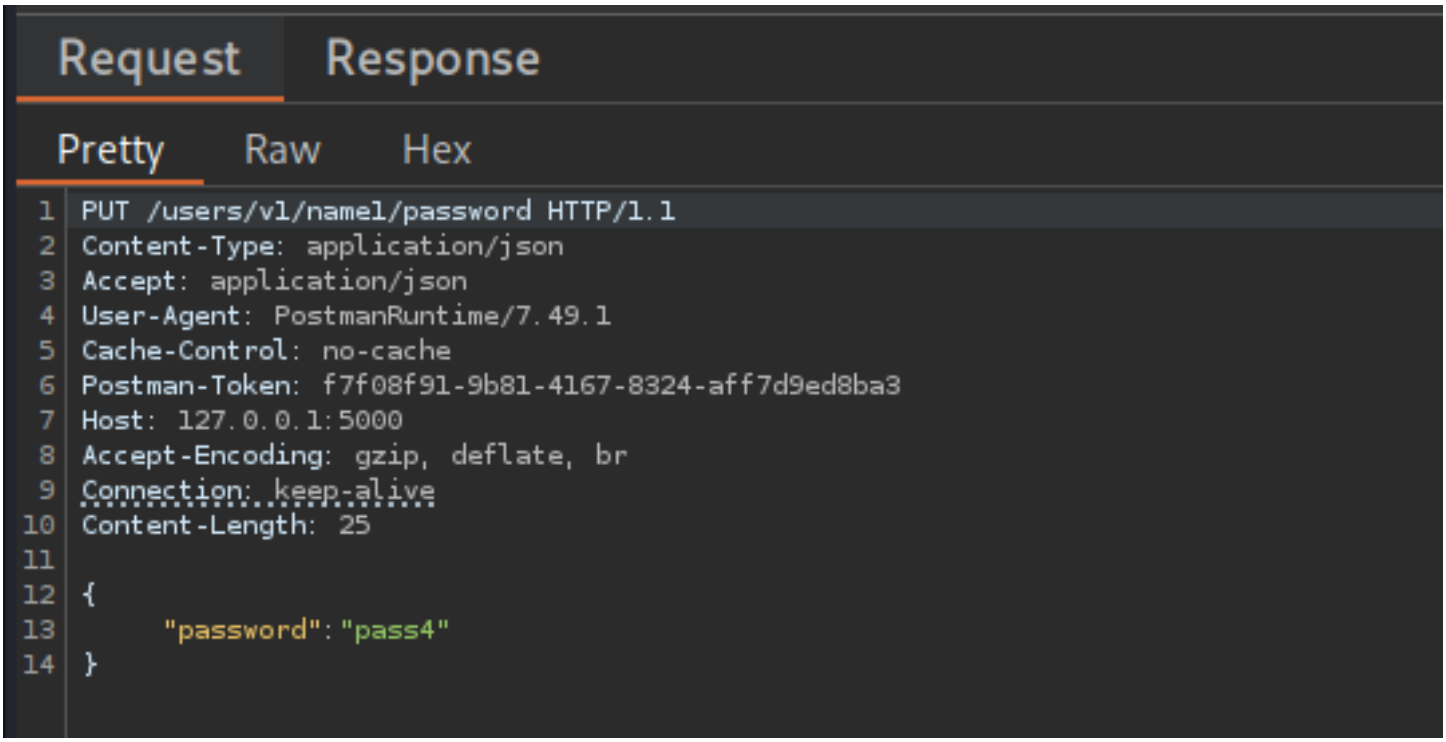
- Login as a regular user named **“name1**”.

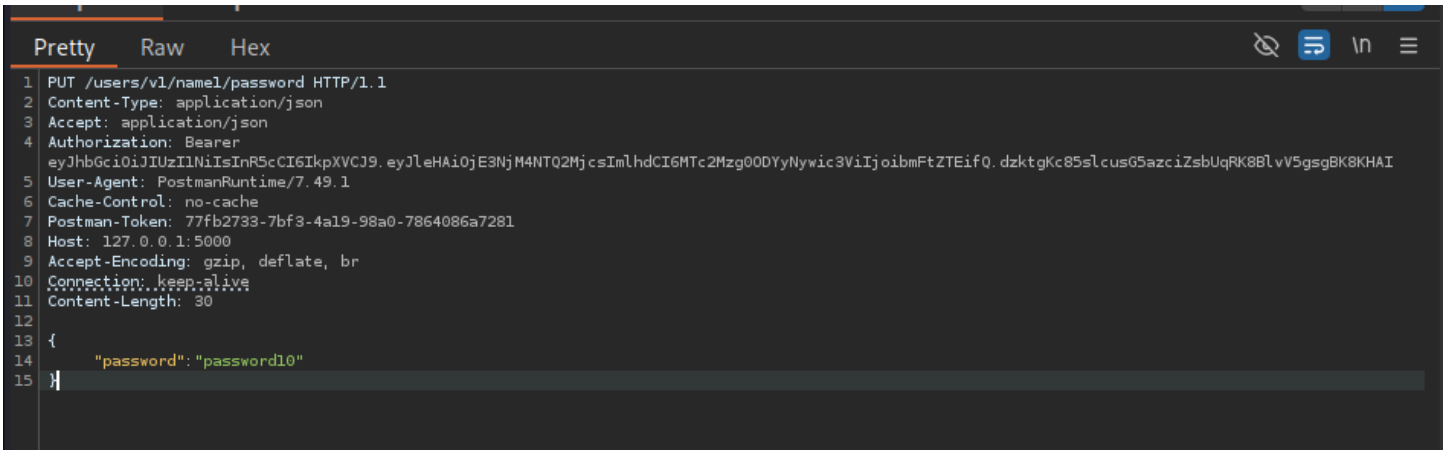
```
POST /users/v1/login HTTP/1.1
Content-Type: application/json
Accept: application/json
User-Agent: PostmanRuntime/7.49.1
Cache-Control: no-cache
Postman-Token: 7c6015ff-cad2-49c0-8bfc-2b0e0ac0e8d7
Host: 127.0.0.1:5000
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Content-Length: 52

{
    "password":"pass1",
    "username":"name1"
}
```

- Copy the valid Bearer token provided at login.



- Use the token to update your own password through: **PUT /users/v1/name1/password HTTP/1.1**





- Confirm the new password works.

Send Cancel

Request Response

Pretty Raw Hex

```
1  POST /users/v1/login HTTP/1.1
2  Content-Type: application/json
3  Accept: application/json
4  User-Agent: PostmanRuntime/7.49.1
5  Cache-Control: no-cache
6  Postman-Token: 87a2456b-abe4-4a7d-86a2-ff93e3b518f4
7  Host: 127.0.0.1:5000
8  Accept-Encoding: gzip, deflate, br
9  Connection: keep-alive
10 Content-Length: 53
11
12 {
13     "username":"name1",
14     "password":"password10"
15 }
```

Pretty Raw Hex Render \n

```
1  HTTP/1.1 200 OK
2  Server: Werkzeug/2.2.3 Python/3.11.9
3  Date: Sat, 22 Nov 2025 22:18:21 GMT
4  Content-Type: application/json
5  Content-Length: 224
6  Connection: close
7
8  {
       "auth_token":
       "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE3NjM4NTU5MDEsImlhdCI6MTc2Mzg0OTkwMSwic3ViIjoibmFtZTEifQ.QkTEP29aZ0YoV0FPc0iklBosqfpLc8yIA_Zjg-21
       phk",
       "message":"Successfully logged in.",
       "status":"success"
   }
```

- Using the same name1 token, modify the URL and update the administrator password by sending: **PUT /users/v1/admin/password HTTP/1.1**

```
PUT /users/v1/admin/password HTTP/1.1
Content-Type: application/json
Accept: application/json
Authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE3NjM4NTQ2MjcsImlhdCI6MTc2Mzg0ODYyNywic3ViIjoibmFtZTEifQ.dzktgKc85slcusG5azciZsbUqRK8BlvV5gsgBK8KHAI
User-Agent: PostmanRuntime/7.49.1
Cache-Control: no-cache
Postman-Token: 0141b367-246f-4268-a213-6f2634c3b8de
Host: 127.0.0.1:5000
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Content-Length: 30

{
    "password":"password11"
}
```

Send Cancel

Request Response

Pretty Raw Hex Render

```
1 HTTP/1.1 204 NO CONTENT
2 Server: Werkzeug/2.2.3 Python/3.11.9
3 Date: Sat, 22 Nov 2025 22:20:51 GMT
4 Content-Type: application/json
5 Connection: close
6
7
```

- Test the previous administrator password to confirm that it no longer works.

Request Response

Pretty Raw Hex

```
1 POST /users/v1/login HTTP/1.1
2 Content-Type: application/json
3 Accept: application/json
4 User-Agent: PostmanRuntime/7.49.1
5 Cache-Control: no-cache
6 Postman-Token: 5cac9d7f-8b5d-4849-b816-5bbeff217994
7 Host: 127.0.0.1:5000
8 Accept-Encoding: gzip, deflate, br
9 Connection: keep-alive
10 Content-Length: 48
11
12 {
13     "username":"admin",
14     "password":"pass1"
15 }
```

Request Response

Pretty Raw Hex Render

```
1 HTTP/1.1 200 OK
2 Server: Werkzeug/2.2.3 Python/3.11.9
3 Date: Sat, 22 Nov 2025 22:22:53 GMT
4 Content-Type: application/json
5 Content-Length: 81
6 Connection: close
7
8 {
      "status":"fail",
      "message":"Password is not correct for the given username."
  }
```

- Login with the newly set administrator password.

Request Response

Pretty Raw Hex

```
1  POST /users/v1/login HTTP/1.1
2  Content-Type: application/json
3  Accept: application/json
4  User-Agent: PostmanRuntime/7.49.1
5  Cache-Control: no-cache
6  Postman-Token: 5cac9d7f-8b5d-4849-b816-5bbeff217994
7  Host: 127.0.0.1:5000
8  Accept-Encoding: gzip, deflate, br
9  Connection: keep-alive
10 Content-Length: 53
11
12 {
13      "username":"admin",
14      "password":"password11"
15 }
```

- Full administrator access is now obtained without any authorization checks.

Pretty Raw Hex Render \n

```
1 HTTP/1.1 200 OK
2 Server: Werkzeug/2.2.3 Python/3.11.9
3 Date: Sat, 22 Nov 2025 22:23:58 GMT
4 Content-Type: application/json
5 Content-Length: 224
6 Connection: close
7
8 {
      "auth_token":
      "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE3NjM4NTYyMzgsImlhdCI6MTc2MzglMDIzOCwic3ViIjoiYWRtaW4ifQ.LmnXUL0BDyTHDNRDk6UWBcmbenxv5J8_0Xoc_HNfrgk",
      "message":"Successfully logged in.",
      "status":"success"
  }
```

Impact

Unauthorized password updates allow complete account takeover. An attacker can lock out legitimate users, gain administrator access to the system, and perform privileged operations.

CVSS v4.0 Risk Rating: Critical

GDPR Impact: Violates GDPR **Articles 5, 24, and 32** due to failure to enforce proper security controls on personal accounts. Unauthorized access through password manipulation qualifies as a significant data protection failure.

PCI DSS Impact: Violates requirements **7, 8, and 10** concerning access control, authentication management, and monitoring of security events.

NIST 800-53 Impact: Violates:

- AC-2 (Account Management)
- AC-3 (Access Enforcement)
- IA-5 (Authenticator Management)
- SC-28 (Protection of Information at Rest)

Recommended Fix

- Validate all sensitive operations using the authenticated user identity from the token.
- Remove all user identifiers from URL based trust.
- Enforce role based access control rules for password changes.
- Add server side verification that only the account owner or an authorized administrator can modify credentials.

Conclusion

This exercise served as an educational training activity. VAmPI is a purposely vulnerable API designed for learning and practicing API security testing. The assessment allowed me to explore real API weaknesses, understand how attackers move through an environment, and apply OWASP API Security Top 10 concepts in a controlled and safe lab setup. The findings highlight common security issues that appear in real applications and show why secure design, careful validation, and proper access control are important.

Top 5 Immediate Actions (Executive Checklist):

1. Fix Broken Object-Level Authorization by enforcing strict ownership checks.
2. Remove or secure all debug endpoints that expose sensitive user data.
3. Correct the password update logic to validate users from tokens.
4. Apply data minimization controls to limit sensitive information in responses.
5. Deploy centralized access control and configuration checks to avoid misconfigurations.

End of report