

Penetration Test Report

CONFIDENTIAL

Report: crAPI Websecurity Test — Executive Findings

Client: AfricaHackon

Prepared by: Squad 9

Date: 20th October 2025

Test window: 10th October 2025 — 20th October 2025

Distribution: Authorized personnel only

EXECUTIVE SUMMARY

We tested the crAPI web App and API to find security problems a real attacker could use. We found several serious issues that let one logged-in user view or change another user's data, manipulate orders/payments, bypass authentication, and execute database attacks. The most urgent problems let attackers see other users' personal data and impersonate accounts. Evidence (request/response captures and screenshots) is attached on the appendices.

SCOPE AND RULES OF ENGAGEMENT

In-scope: All crAPI web app pages, all crAPI API endpoints, and actions performed with all accounts (Jux, Baba). Given it is a test environment you are allowed to affect all users in your local environment setup(crAPI local setup guideline).

Rules: non-destructive PoCs only, tests during agreed window, immediate stop if production impact risk identified.

METHODOLOGY & TOOLS

Recon → automated discovery → manual verification → safe Proof-of-Concepts (PoCs) → reporting.

Main tools: Burp Suite (intercept/repeat/intruder), curl, browser dev tools, non-destructive sql/no-sql checks. All PoCs were performed against test accounts and documented.

FINDINGS

1) Broken Object Level Authorization (BOLA / IDOR)

Severity: Critical

Affected: `GET /vehicles/{vehicle_id}` and community pages exposing vehicle IDs

PoC: Change your request's `vehicle_id` to another user's ID (e.g., Jux → Baba) and the server returns that other user's vehicle details (VIN, location, owner email). Captures show successful access.

Impact: Any logged-in user can read other customers' personal data and live locations.

Remediation: Enforce server-side ownership checks: verify the authenticated user is allowed to access the requested resource. Never trust client-supplied IDs.

2) JWT token tampering / unsigned token acceptance

Severity: Critical / High

Affected: JWT-authenticated endpoints (dashboard/admin data)

PoC: Modify JWT header `alg` to `none`, remove signature, and change `sub` to another user's email — the server accepted the token and returned higher-privilege data.

Impact: Full account impersonation and potential admin takeover.

Remediation: Reject unsigned tokens, accept only approved algorithms, validate signatures and token claims (issuer, audience), and rotate keys.

3) Mass assignment / Business-flow manipulation

Severity: High

Affected: Purchase/order endpoints (quantity, price, status, refund fields)

PoC: Modified JSON fields in order requests (quantity, status) and observed changed balances, refunds, and order states; created orders with manipulated totals. Evidence shows balance increases from manipulated values.

Impact: Financial fraud (free items, inflated balances, unauthorized refunds).

Remediation: Server must compute and enforce prices and totals; ignore or block client-supplied protected fields (status, price, balance). Validate and recalculate server-side.

4) NoSQL / SQL Injection (database injection)

Severity: High

Affected: Coupon input, shop fields, other user inputs

PoC: Submitted NoSQL payload (`{"$ne":1}`) and SQL payloads (`' OR 1=1--`) and observed database-driven changes; `select version()` indicates PostgreSQL 14.19.

Impact: Data disclosure, tampering, or full database compromise.

Remediation: Use parameterized queries/ORM-safe APIs, validate and sanitize inputs, apply least-privilege DB credentials, and add query logging/alerting.

5) Broken Authentication / OTP and account enumeration

Severity: High

Affected: Password reset / OTP flow

PoC: Forgot-password flow reveals whether an email is registered; OTP attempts allow limited guessing but leak account existence.

Impact: Attackers can confirm accounts and attempt targeted attacks (OTP guessing / phishing).

Remediation: Do not reveal account existence in responses, enforce strict rate limits, shorten OTP lifetime, and add monitoring for abnormal attempts.

6) Broken Function-Level Authorization / HTTP verb misuse

Severity: High

Affected: Video management endpoints and admin features (PUT/DELETE/GET)

PoC: Changed HTTP method and ID values; server accepted unexpected verbs and returned admin-related responses or allowed deletion when ID changed.

Impact: Users may perform actions reserved for admins or access restricted information.

Remediation: Enforce role-based checks server-side for every function and restrict allowed HTTP methods per endpoint.

7) Server-Side Request Forgery (SSRF) via mechanic callback URL

Severity: Medium

Affected: `mechanic_api` callback / contact-mechanic form parameter

PoC: Replaced callback URL with <https://google.com> and server followed it — server made outbound request to attacker-controlled URL.

Impact: Attackers can make server contact internal services or external endpoints, potentially exposing internal resources.

Remediation: Allowlist outgoing domains, block internal IP ranges, and validate/sanitize submitted URLs.

8) Unrestricted resource consumption/rate limiting gaps

Severity: Medium

Affected: contact-mechanic repeat count/endpoints with repeat parameter

PoC: Repeat parameter accepts up to 100; above that service becomes unavailable — indicates insufficient server-side limits.

Impact: Resource exhaustion leading to partial denial of service.

Remediation: Implement server-side rate limits, per-account and per-IP quotas, and input bounds checking.

9) Excessive data exposure/security misconfiguration (metadata)

Severity: Low

Affected: Community comments, uploaded video metadata

PoC: Public pages display user emails and internal device paths.

Impact: Extra information helps attackers craft targeted attacks.

Remediation: Remove or redact sensitive metadata from public views; sanitize uploads.

RISK/PRIORITISATION

Immediate (0–7 days) — Fix BOLA/IDOR and JWT verification issues (Findings 1 & 2).

High (7–30 days) — Fix mass-assignment, injection points, authentication/OTP issues, and function-level auth (Findings 3–6).

Medium (30–60 days) — Implement SSRF protections, rate limits, and server-side bounds (Findings 7–8).

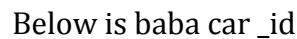
Low (60–90 days) — Remove excessive data exposure, harden config and monitoring (Finding 9).

CONCLUSION

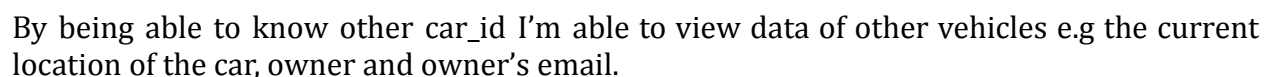
1. **Emergency (today):** Disable endpoints that return other users' data by ID or enforce ownership checks; reject unsigned JWTs; add temporary WAF rules blocking obvious attack patterns.
2. **Fixes (7–30 days):** Patch injection points, rework order/payment logic so server computes totals and handles refunds, harden OTP and authentication flows.
3. **Hardening (30–90 days):** Add rate limiting, SSRF allowlists, strict role checks, input validation, and DB least-privilege.
4. **Long-term:** Integrate secure development practices (parameterized queries, server-side validation), automated security tests, and plan a re-test after fixes.

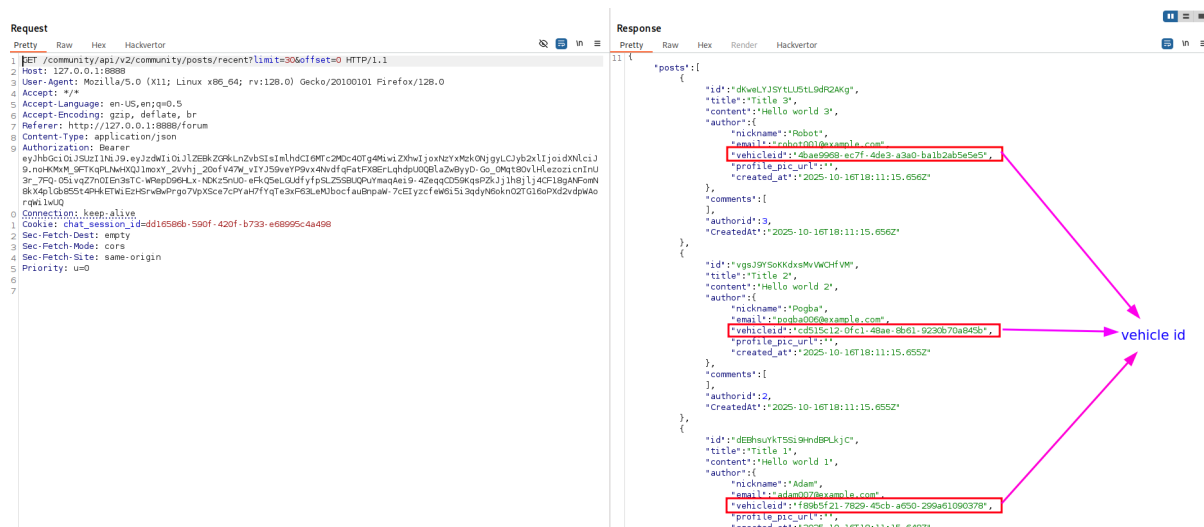
1. Broken Object Level Authorization

So below is jux request and response as captured in burpsuite

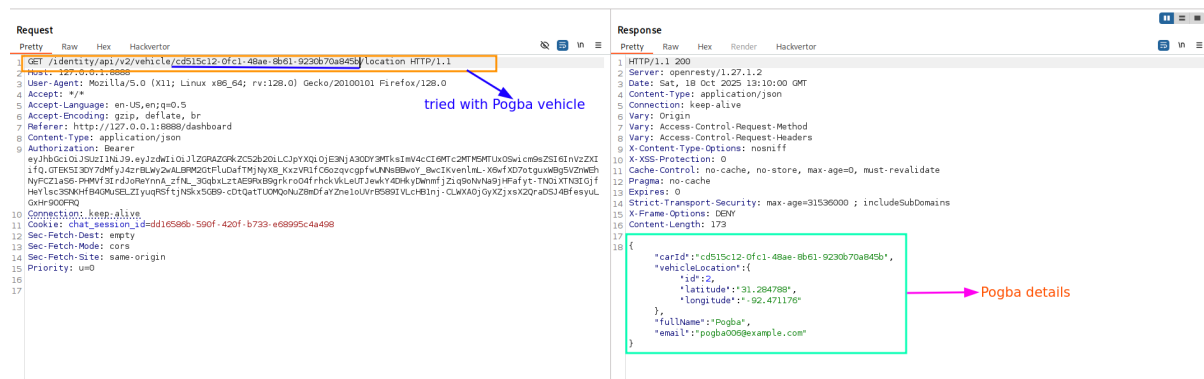


By changing the request of Jux request to Baba's request that is his car_id we get. this



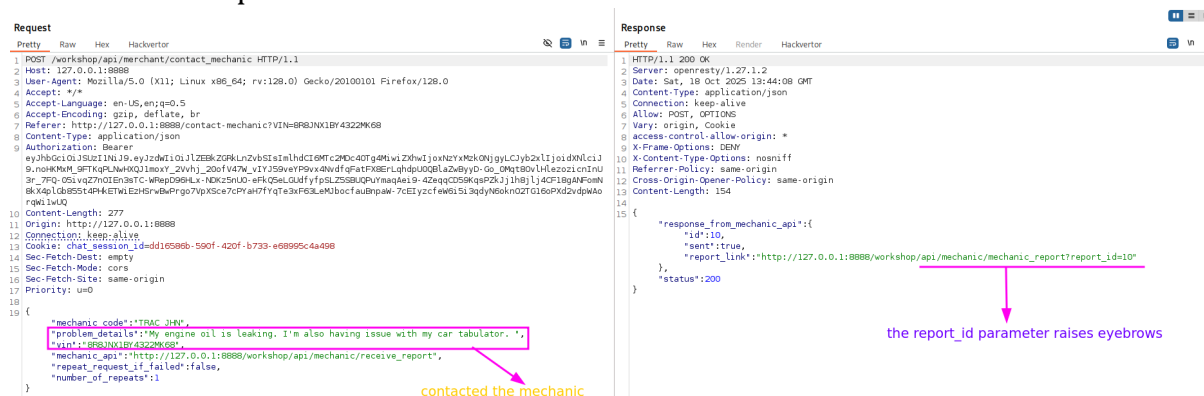


By navigating to the community tab we can get to see various vehicle id and we can use it to see other users data.



Using other vehicle id we can still achieve this.

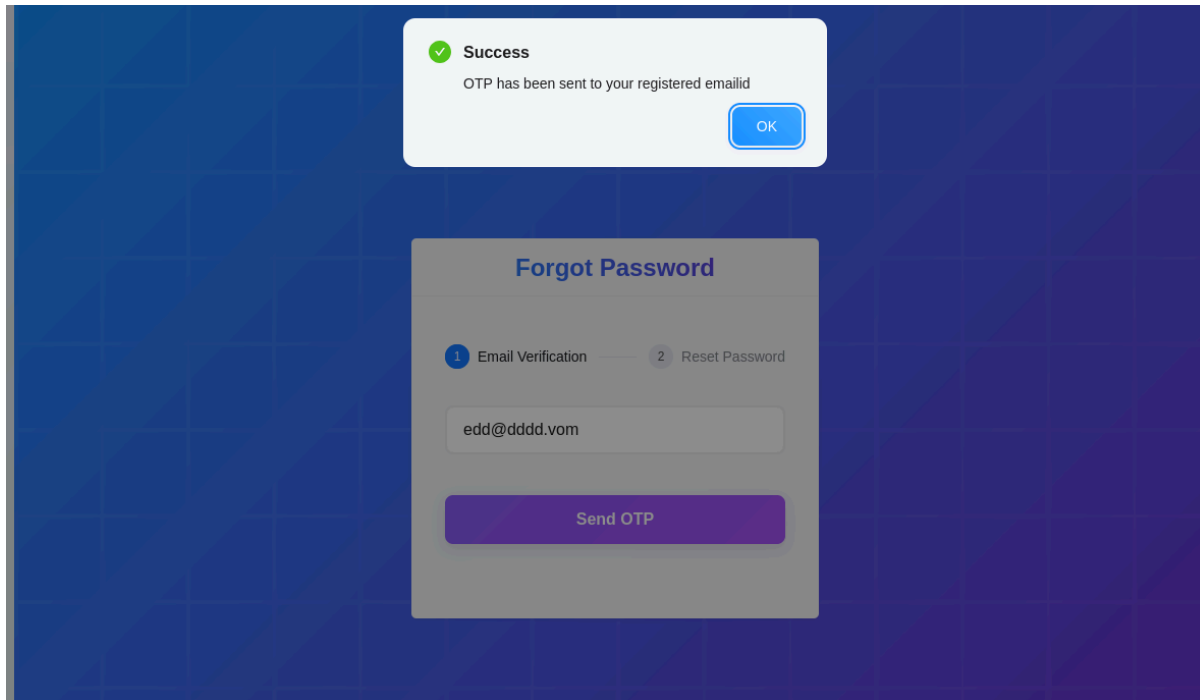
We can contact a mechanic from the dashboard. So we decided to report a car issue and got this from the burpsuite



[illegible][illegible]

Results	Positions						
▽ Capture filter: Capturing all items							
▽ View filter: Showing all items							
Request ^	Payload	Status code	Response received	Error	Timeout	Length	Comment
0		200	1059			804	
1	1	200	681			813	
2	2	200	737			818	
3	3	200	676			824	
4	4	200	715			804	
5	5	200	647			816	
6	6	200	670			726	
7	7	200	723			726	
8	8	200	647			726	
9	9	200	716			726	
10	10	200	668			727	
11	11	400	129			405	
12	12	400	131			405	
13	13	400	133			405	
14	14	400	137			405	
15	15	400	88			405	
16	16	400	83			405	
17	17	400	135			405	
18	18	400	106			405	
19	19	400	101			405	
20	20	400	83			405	

This shows that report_id 1-10 exists, for POC i'll use ID 4 and here is the data.



Sniper attack

Target: ☒ Update Host header to match target

Positions:

```
1 POST /identity/api/auth/v3/check-otp HTTP/1.1
2 Host: 127.0.0.1:8888
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
4 Accept: */*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Referer: http://127.0.0.1:8888/forgot-password
8 Content-Type: application/json
9 Content-Length: 63
10 Origin: http://127.0.0.1:8888
11 Connection: keep-alive
12 Cookie: chat_session_id=dd16586b-590f-420f-b733-e68995c4a498
13 Sec-Fetch-Dest: empty
14 Sec-Fetch-Mode: cors
15 Sec-Fetch-Site: same-origin
16 Priority: u=0
17 {
18   "email": "edd@dddd.vom", "otp": 812599, "password": "(E41234567)"
19 }
```

the start and finish of the values to try out

attack vector where the payload is to try out

number format

Payloads

Payload position: All payload positions

Payload type: Numbers

Payload count: 10,000

Request count: 10,000

Payload configuration

This payload type generates numeric payloads within a given range and in a specified format.

Number range

Type: ☒ Sequential ☐ Random

From: 0000

To: 9999

Step: 1

How many:

Number format

Base: ☒ Decimal ☐ Hex

Min integer digits: 4

Max integer digits: 4

Min fraction digits: 0

Max fraction digits: 0

Examples

0001

4321

Forgot Password

✓ Email Verification — 2 Reset Password

1245

.....



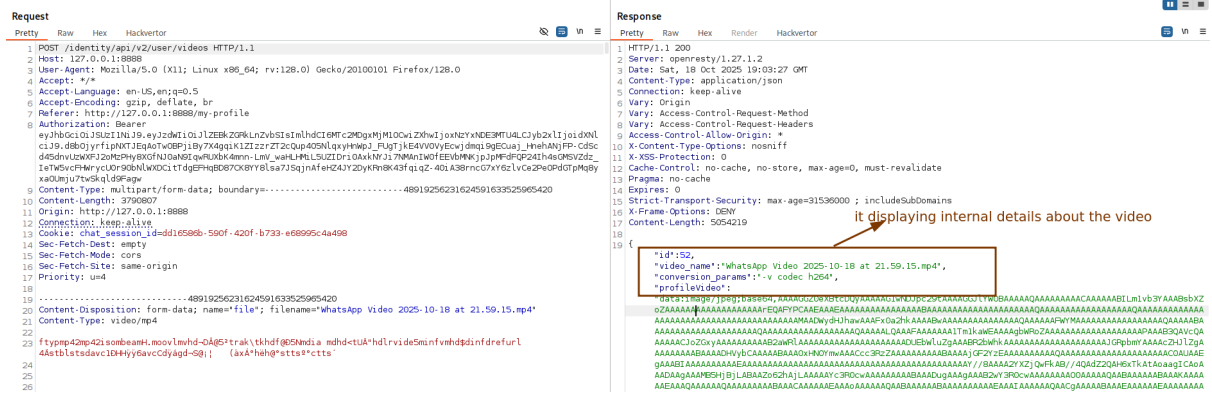
.....



Invalid OTP! Please try again..

Set Password

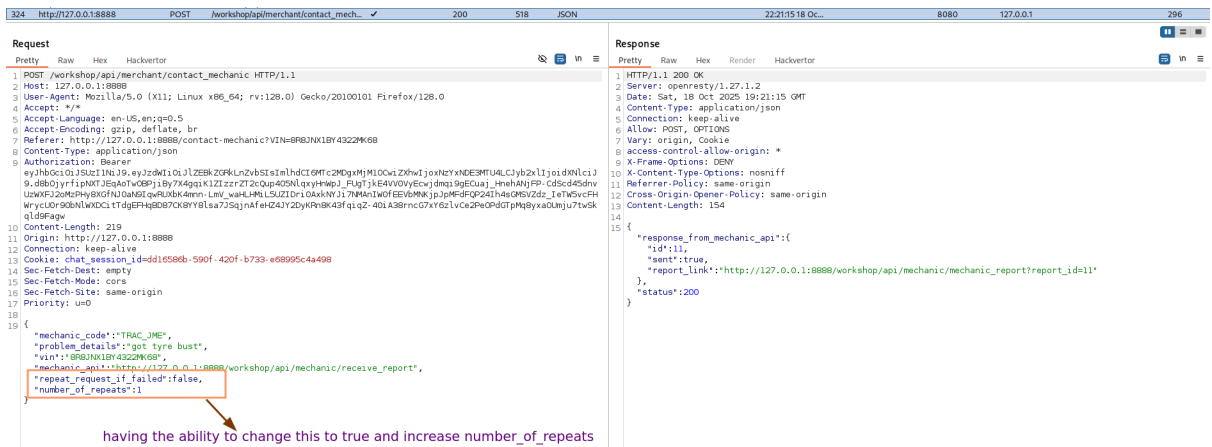
invalid otp/ credential



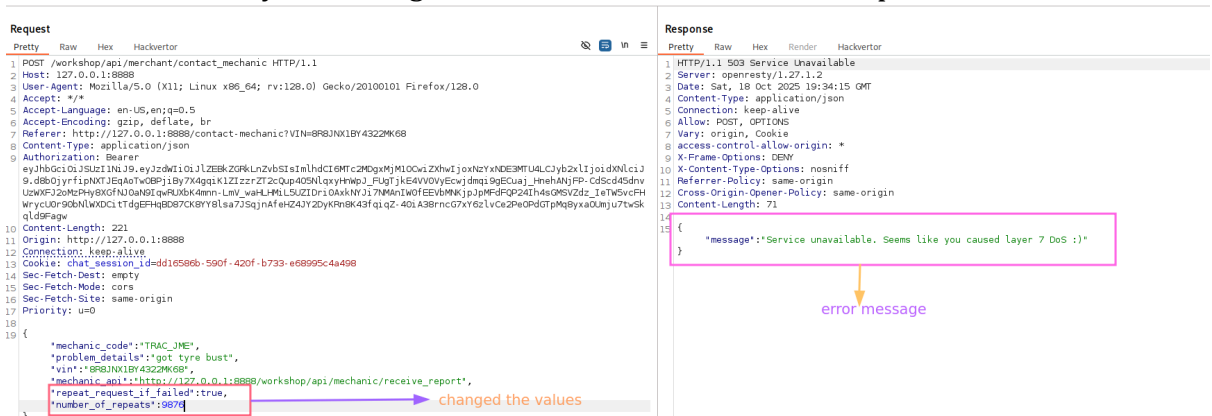
4. Unrestricted Resource Consumption

Lack of Resources & Rate Limiting: APIs may not enforce limits on the number of requests, leading to abuse.

In the contact mechanic option there is the ability to change the 2 options highlighted below



So we decided to try and change the 2 values to see what the set parameter does:



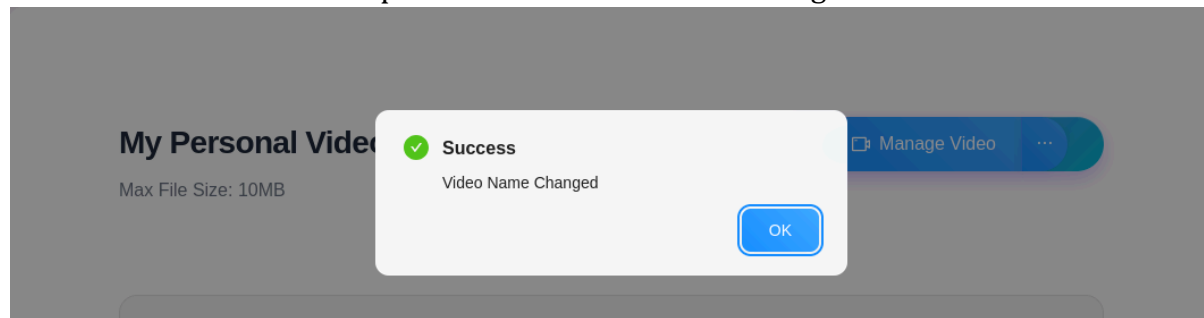
We dug a little dip to see what is the set number of repeats that one can do without taking the site/services offline and got to the conclusion that it is 100. The rate limit is set from 1-100. Above this the service goes offline rendering availability inaccessible.

5. Broken Function Level Authorization

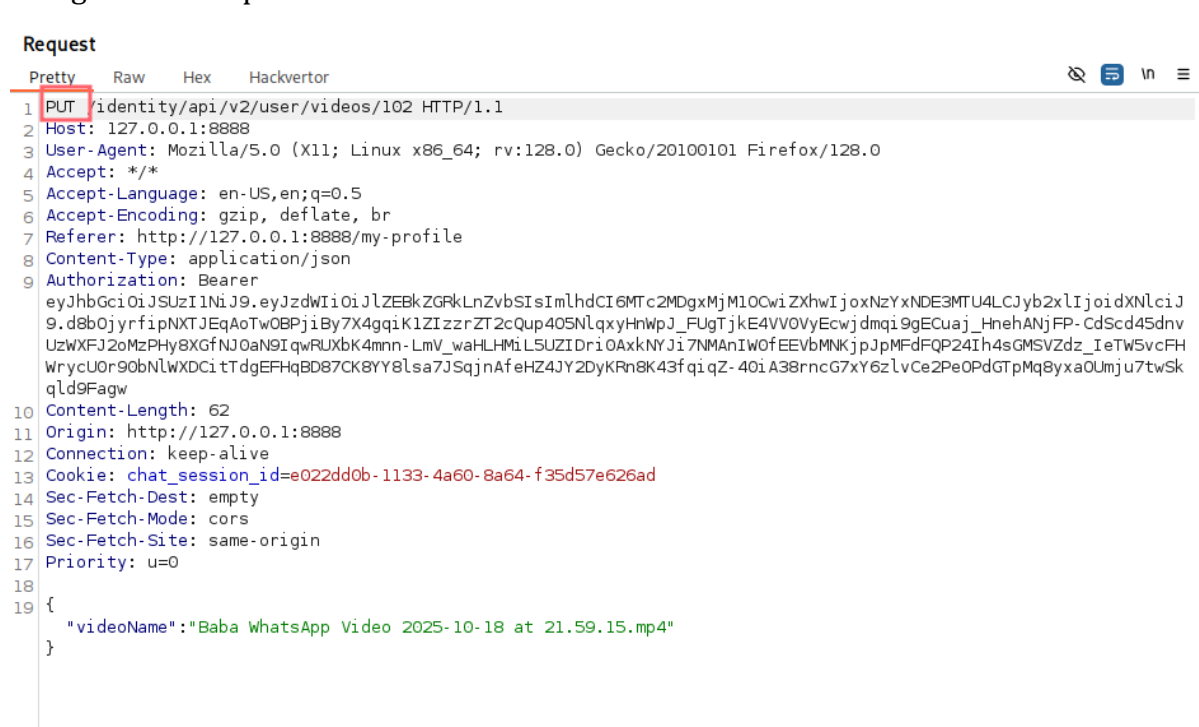
Insufficient checks on user permissions for specific functions can lead to unauthorized access.

We changed the video name from our profile page.

So we went and updated the name and got this success message.



We went to burp and saw the HTTP option in use is PUT and got curious to see if possible to change to other options.



Tried to change it to get, delete and to my surprise it was accepted and got some valuable information that we need to be logged in as an admin to perform the task.

6	105	200	117	573
7	106	404	14	595
8	107	404	14	595
9	108	404	16	595

Request

Response

Pretty

Raw

Hex

Render

Hackvector

```

1 HTTP/1.1 200
2 Server: openresty/1.27.1.2
3 Date: Sun, 19 Oct 2025 17:35:06 GMT
4 Content-Type: application/json
5 Connection: keep-alive
6 Vary: Origin
7 Vary: Access-Control-Request-Method
8 Vary: Access-Control-Request-Headers
9 Access-Control-Allow-Origin: *
10 X-Content-Type-Options: nosniff
11 X-XSS-Protection: 0
12 Cache-Control: no-cache, no-store, max-age=0, must-revalidate
13 Pragma: no-cache
14 Expires: 0
15 Strict-Transport-Security: max-age=31536000 ; includeSubDomains
16 X-Frame-Options: DENY
17 Content-Length: 59
18
19 {
20   "message": "User video deleted successfully.",
21   "status": 200
22 }

```

Successfully Deleted the video

6. Unrestricted Access to Sensitive Business Flows

Mass Assignment: APIs may allow users to modify object properties that they should not have access to.

We went to the shop page to try out and see if we could get an item for free.

Shop

Success

Order sent successfully.

OK

+

Add Coupons

≡

Past Orders

AVAILABLE BALANCE: \$90



Wheel

\$10.00

Buy



Seat

\$10.00

Buy

Previous

Next

Purchased the wheel to observe request

We got to see the http request allowed and tried them out.

← Past Orders



Wheel

\$10.00

Sun Oct 19 2025

Order Details

returned



Seat

\$10.00

Sun Oct 19 2025

Order Details

returned

← Past Orders



Seat

\$5000000.00

Mon Oct 20 2025

Order Details

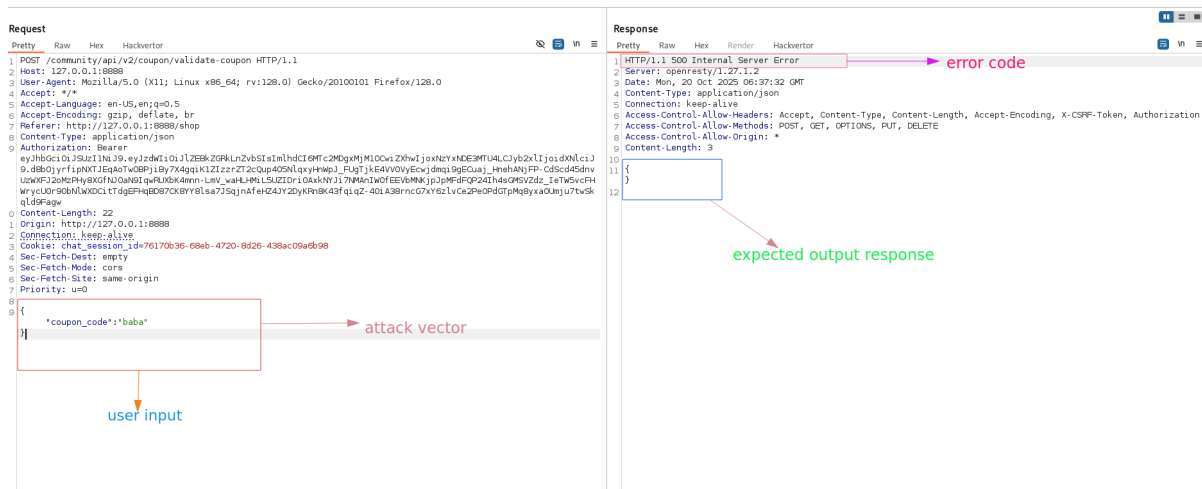
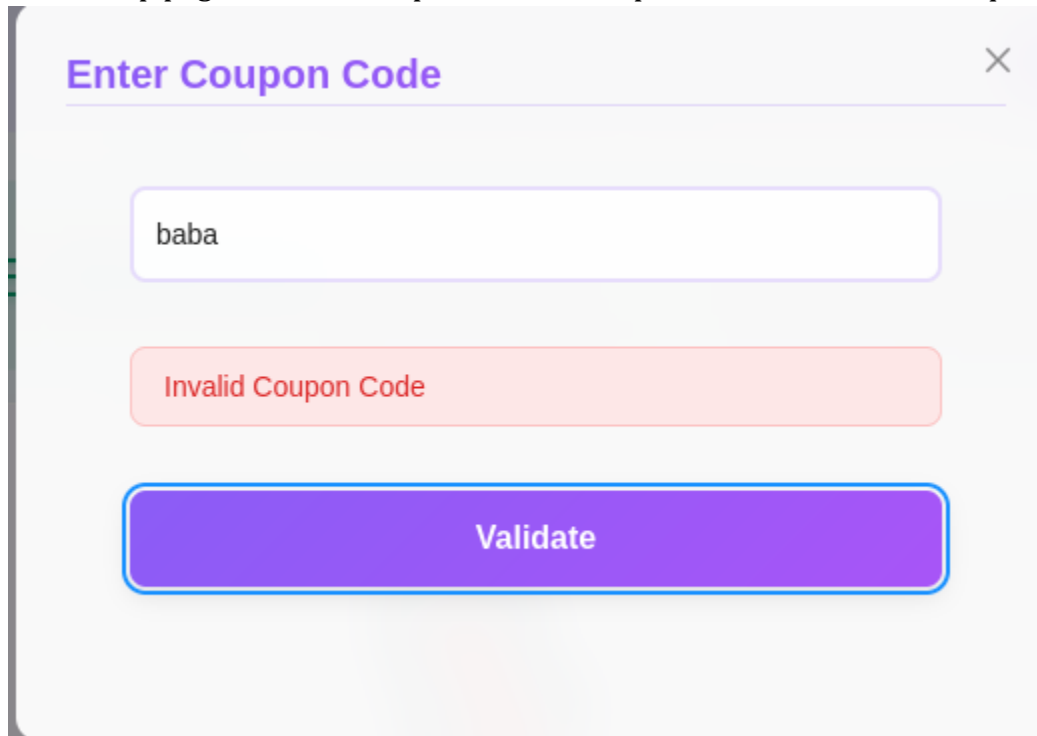
returned

[illegible][illegible]

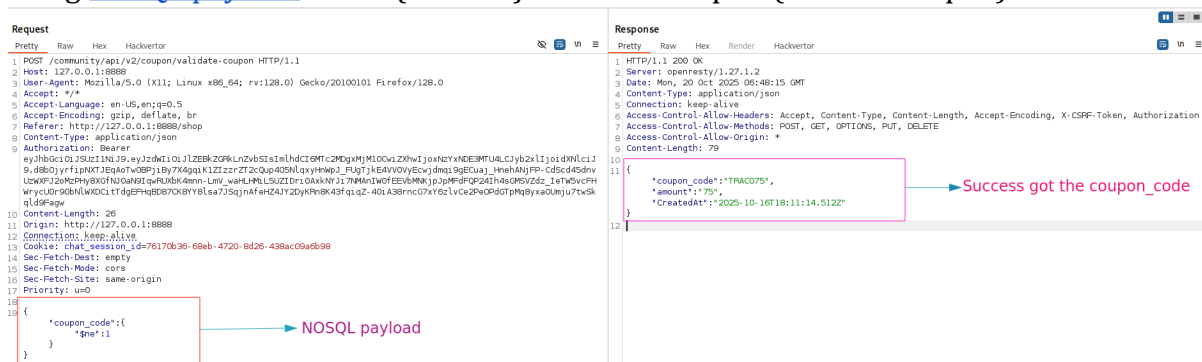
8. Security Misconfiguration

NOSQL

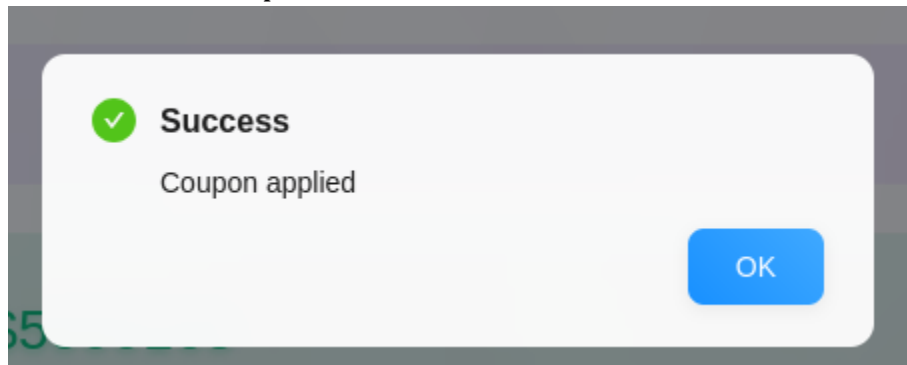
In the shop page there is the option to add coupon code and this is its request captured.



Using [NOSQL payload](#) i used { "\$ne":1} in the user input (\$ne not equal)

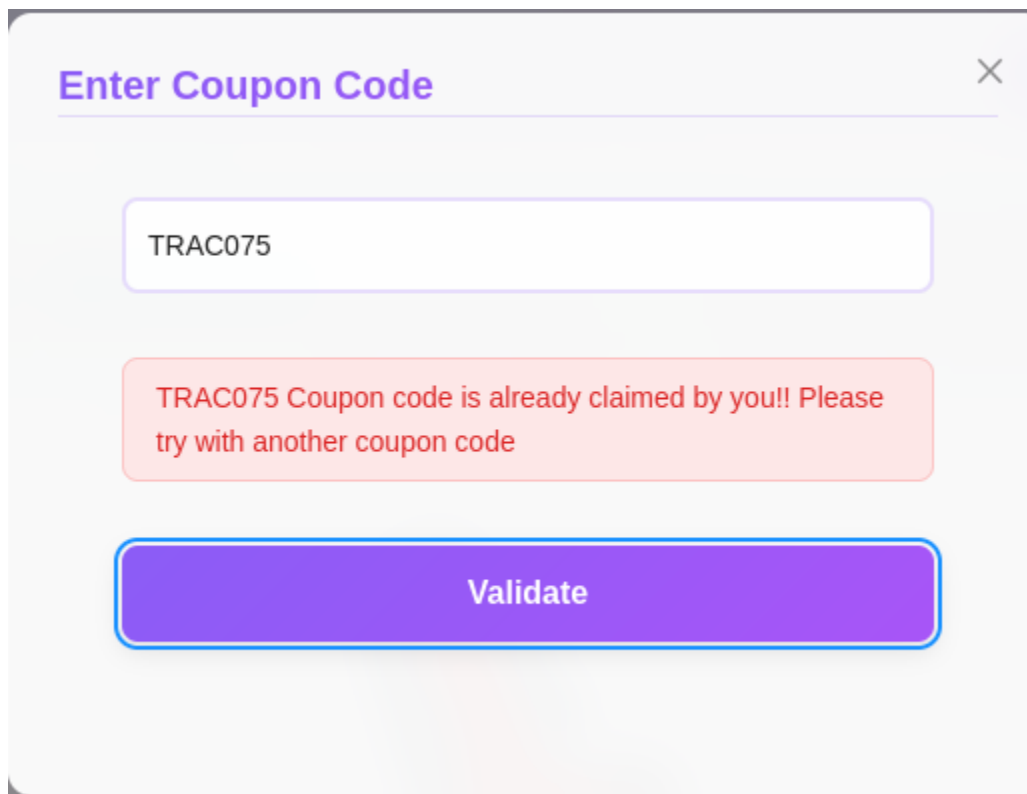


We can use the coupon and see if it works, the initial Available Balance was \$5000090



The Available Balance now is \$5000165

Sql injection

A screenshot of a web form titled "Enter Coupon Code". It has a text input field containing "TRAC075". Below the input field is a red error message: "TRAC075 Coupon code is already claimed by you!! Please try with another coupon code". At the bottom of the form is a large blue button labeled "Validate".

Trying to repeat the process we get the above error saying the coupon is used. It led us to try and see if it is liable to SQL injection. So we used the payload ' or 1=1-- and to our surprise it accepted, meaning its vulnerable to SQL injection.

