crAPI

Customer:
crAPI
2025-05-16
v2

Contact:
Søren Johanson
+49 156 79 589764
soeren@soeren.codes

Table of contents

Søren Johanson
Heiligengeiststraße 6-8
26121 Oldenburg (Oldb)
Germany

Executive summary

This sample report is built around **crAPI**, an intentionally vulnerable demo API from OWASP that shows the kind of findings you would see in an **API Security Scorecard**.

The layout is founder-friendly:

- **Executive highlights** open every section - plain-English risk, impact and urgency, so you can decide what truly matters before diving into detail.
- **Technical fixes** follow straight after for your engineers, with clear, step-by-step actions.

The recommendations are written so you can hand the report to your own team or any security partner to close the gaps. You are free to book a Rapid Hardening Sprint with me, but you are not tied to my services - that way you stay in full control of cost and timelines.

This is a sample of some of the vulnerabilities found in crAPI and not exhaustive; it's intended to highlight my methodology, depth and how findings map to OWASP Top 10, ISO 27001 and SOC 2. A full report would cover all endpoints and include additional checks.

Søren Johanson
Heiligengeiststraße 6-8
26121 Oldenburg (Oldb)
Germany

Methodology and scope

Methodology

This audit followed a structured and repeatable process to assess the security posture of publicly exposed API endpoints, combining manual techniques with selective automation. The goal is to simulate what a motivated attacker or security-savvy customer might uncover, without intrusive or disruptive testing.

1. Scope of Assessment

The following API components were reviewed:

- Endpoints under `/identity`, `/community/` and `/workshop/` namespaces
- Public documentation or exposed interface elements linked to these APIs

All testing was conducted in a **black-box** context: only publicly accessible endpoints and documentation were used. No credentials or private infrastructure access were granted.

Excluded from scope:

- Source code or internal service logic
- Infrastructure, CI/CD pipelines, cloud security posture
- Mobile apps, SDKs, or third-party dependencies unless directly tied to the API surface

2. Techniques Used

This audit involved:

- **Manual inspection** of endpoints, response structures, and parameter behaviour
- **OWASP API Security Top 10** as reference for vulnerability classes
- Targeted fuzzing, enumeration, and error observation techniques
- Basic token/ID manipulation to test for BOLA/BFLA weaknesses
- Heuristic analysis for CORS, rate-limiting, error leakage, and misconfiguration

Søren Johanson
Heiligengeiststraße 6-8
26121 Oldenburg (Oldb)
Germany

Where relevant, findings are mapped to:

- ISO/IEC 27001:2022 controls (Annex A)
- SOC 2 Trust Services Criteria
- OWASP API Security Top 10 (2023)

3. Limitations

This audit is not a full penetration test. It is a focused review of API-level security risks and assumes:

- No access to internals
- No brute-force or destructive actions

As with any black-box engagement, some risks may remain undiscovered without access to underlying logic, infrastructure, or configuration files.

Søren Johanson
Heiligengeiststraße 6-8
26121 Oldenburg (Oldb)
Germany

Vulnerability overview

In the course of this audit **1 Critical**, **3 High** and **1 Medium** vulnerabilities were identified:

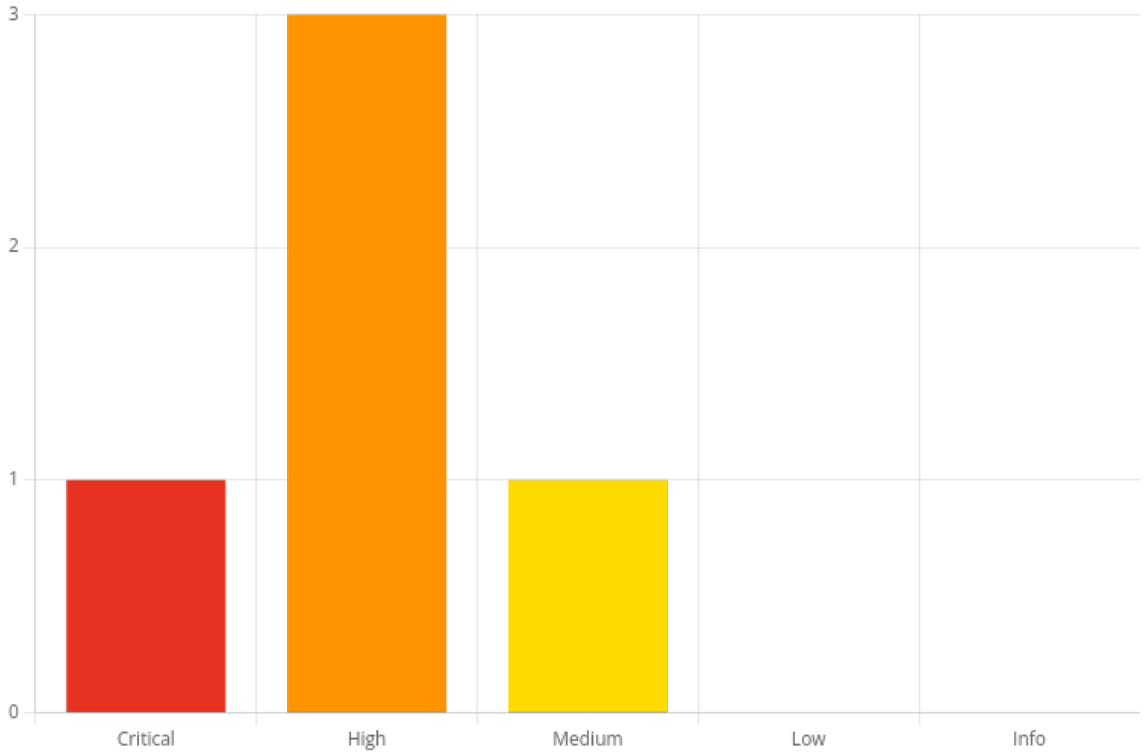

Figure 1 - Distribution of identified vulnerabilities

Vulnerability	Criticality
API2:2023 Weak or missing access controls	Critical
API4:2023 Unrestricted resource consumption	High
API1:2023 Unauthorised access to other users' data	High
API3:2023 Overly broad data returns	High
API3:2023 Unauthorised data modifications	Medium

Søren Johanson
Heiligengeiststraße 6-8
26121 Oldenburg (Oldb)
Germany

1. API2:2023 Weak or missing access controls

Remediation Status:
Criticality: Critical
CVSS-Score: 9.8
Affects: /identity/api/v2/user/reset-password

Executive highlights

The API has a critical logic flaw which allows any user to change another user's password, without verifying if the user has access to that email address. This allows an attacker to perform a full account takeover with ease.

Technical description

Broken authentication arises when the API fails to verify credentials or session tokens correctly.

In this instance, when a user asks to reset their password, there is no secondary verification. A common secondary verification step would be to send a confirmation email to prove that they have access to the given email address. The email always includes a link with a short-lived token to prevent tampering.

Currently, it's possible to stage a full account takeover just by knowing the email address of the user, as the new password is supplied directly in the same API call:

```
curl --request POST \
  --url http://localhost:8888/identity/api/v2/user/reset-password \
  --header 'Authorization: Bearer token' \
  --header 'Content-Type: application/json' \
  --data '{
  "email": "admin@example.com",
  "password": "Admin1234!"
}'
```

Recommendations

- **Implement a secondary verification step**
  To verify that the user has access to the given email address, make sure to send a confirmation email to that address. There should be a link to a separate "Reset password" page in the confirmation email, with a short-

Søren Johanson
Heiligengeiststraße 6-8
26121 Oldenburg (Oldb)
Germany

lived token to verify ownership. Usually, these tokens and thus the confirmation links are set to expire after 15 minutes. Make sure to clearly state how long the confirmation link is valid in the confirmation email.

- **Do not reveal whether a user account exists**
  When the user has entered an email address to receive the confirmation email, be intentionally vague in any notifications on whether a user account exists for the given email address. This is to prevent user enumeration, as you don't want an attacker to definitively know that a person has registered for the service. An example notification would be "If the email address is associated with an account, we've sent a confirmation email to that email address."

Related Standards

- OWASP API Top 10: API2 Broken Authentication
- ISO 27001: A.5.15 - Secure authentication
- ISO 27001: A.5.17 - User registration and de-registration
- ISO 27001: A.8.28 - Secure coding
- SOC 2: CC6.2 - Authentication mechanisms
- SOC 2: CC6.4 - Session controls

Additional information

- https://owasp.org/API-Security/editions/2023/en/0xa2-broken-authentication/
- https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html

Søren Johanson
Heiligengeiststraße 6-8
26121 Oldenburg (Oldb)
Germany

2. API4:2023 Unrestricted resource consumption

Remediation Status:
Criticality: High
CVSS-Score: 8.6
Affects: /workshop/api/merchant/contact_mechanic

Executive highlights

The API includes an endpoint to contact a mechanic, with an attribute that allows repeating the request n number of times if it failed. By setting n to a large number and sending a faulty request, it's possible for an attacker to cause a denial of service attack with ease.

Technical description

Unrestricted resource consumption occurs when the API fails to enforce constraints on resource usage for incoming requests. Common factors include:

- No execution timeouts or low default values, allowing requests to run indefinitely.
- Absence of limits on memory or CPU allocation per request.
- Lack of restrictions on the number of file descriptors, processes or threads spawned.
- No rate limiting or throttling controls on endpoints, enabling high request volumes.
- Missing or misconfigured spending caps for third party integrations, exposing the organisation to inflated bills.

Without these controls in place, crafted requests or automated scripts can exhaust system resources, leading to denial of service or runaway costs.

Here, the `number_of_repeats` and `repeat_request_if_failed` parameters allow an attacker to craft a denial of service attack:

```
curl --request POST \
  --url http://localhost:8888/workshop/api/merchant/contact_mechanic \
  --header 'Authorization: Bearer token' \
  --header 'Content-Type: application/json' \
  --data '{
```

Søren Johanson
Heiligengeiststraße 6-8
26121 Oldenburg (Oldb)
Germany

```
  "number_of_repeats": 9999999,
  "mechanic_api": "string",
  "vin": "string",
  "repeat_request_if_failed": true,
  "problem_details": "string",
  "mechanic_code": "string"
}'
```

Recommendations

- **Remove retry parameters**
  The API includes `repeat_request_if_failed` and `number_of_repeats` parameters. Remove these; do not let the client determine a retry strategy.
- **Define and enforce execution timeouts**
  Set sensible limits for request processing time to prevent runaway operations.
- **Limit resource allocation per request**
  Use containerisation or serverless platforms to constrain CPU, memory and file descriptor usage.
- **Apply rate limiting and throttling**
  Configure per-client or per-endpoint limits to control request rates over defined time windows.
- **Configure spending caps and alerts for third party services**
  Use billing limits or automated notifications to detect and stop excessive consumption.
- **Monitor and log resource usage patterns**
  Collect metrics on request latency, memory and CPU consumption, and alert on anomalous behaviour.

Related Standards

- OWASP API Top 10: API4 Unrestricted Resource Consumption
- ISO 27001: A.5.36 - Technical compliance testing
- ISO 27001: A.8.29 - Security testing
- SOC 2: CC7.2 - Availability protection
- SOC 2: CC6.6 - Resilience mechanisms

Additional information

- https://owasp.org/API-Security/editions/2023/en/0xa4-unrestricted-resource-consumption/

Søren Johanson
Heiligengeiststraße 6-8
26121 Oldenburg (Oldb)
Germany

- https://cheatsheetseries.owasp.org/cheatsheets/Web_Service_Security_Cheat_Sheet.html#availability

Søren Johanson
Heiligengeiststraße 6-8
26121 Oldenburg (Oldb)
Germany

3. API1:2023 Unauthorised access to other users' data

Remediation Status:
Criticality: High
CVSS-Score: 8.1
Affects:

- /identity/api/v2/vehicle/<vehicle_id>/location
- /workshop/api/mechanic/mechanic_report

Recommendation: Implement authorisation checks for all endpoints to verify users can only access resources they own.

Executive highlights

The API does not check that the user is allowed to view or change the particular resource they request. There are multiple endpoints that allow any user to request information without checking that they're authorised to do so, compromising confidential user data and other user's privacy.

Technical description

Broken object-level authorisation occurs when an endpoint accepts a client-supplied identifier without verifying the caller's rights over that exact object.

In this case, the API is missing ownership checks in middleware or controller logic to verify that the requested resource ID belongs to the authenticated user. This allows users to request any vehicle location; if this is updated in real time, this is a severe compromise of user privacy. It's also possible to get any mechanic report just by supplying a different report ID. Each mechanic's report includes confidential user data such as their email address, their vehicle's VIN and more.

To request the real-time location of a specific vehicle, a simple GET request is sufficient:

```
curl --request GET \
  --url http://localhost:8888/identity/api/v2/vehicle/4bae9968-
ec7f-4de3-a3a0-ba1b2ab5e5e5/location \
  --header 'Authorization: Bearer token'
```

Søren Johanson
Heiligengeiststraße 6-8
26121 Oldenburg (Oldb)
Germany

Because there is no ACL (access-control list) or permission check at the object level, simply swapping in another ID in the URL or request body is sufficient to breach confidentiality and integrity.

Recommendations

- **Enforce object-level checks on every endpoint**
  Before returning or modifying a resource, verify that the authenticated user is authorised for that specific ID.
- **Centralise access control logic**
  Implement a shared service or middleware layer that all routes must pass through to validate permissions.
- **Validate identifiers against user entitlements**
  On receipt of an ID parameter, cross-check it against a list of resources owned or permitted for the current user.
- **Harden ID generation and unpredictability**
  Where possible, use unguessable identifiers (for example, cryptographically random UUIDs) rather than simple counters.
- **Log and monitor access failures**
  Record every failed object-access attempt and configure alerts for unusual enumeration patterns.

Related Standards

- OWASP API Top 10: API1 Broken Object Level Authorisation
- ISO 27001: A.5.16 - Access rights
- ISO 27001: A.5.18 - Privileged access
- ISO 27001: A.8.26 - Application security requirements
- SOC 2: CC6.1 - Logical access control
- SOC 2: CC6.3 - Unauthorised access prevention

Additional information

- https://cheatsheetseries.owasp.org/cheatsheets/Authorization_Cheat_Sheet.html
- https://owasp.org/API-Security/editions/2023/en/0xa1-broken-object-level-authorization/

Søren Johanson
Heiligengeiststraße 6-8
26121 Oldenburg (Oldb)
Germany

4. API3:2023 Overly broad data returns

Remediation Status:
Criticality: High
CVSS-Score: 7.5
Affects:

- /workshop/api/mechanic/
- /workshop/api/management/users/all

Executive highlights

The API is exposing internal data without sufficient checks. As a result, an unauthorised user can get detailed information about registered mechanics and all other users, including their email, available credits and internal number.

Technical description

This vulnerability arises when object-level authorisation isn't enforced on individual properties. Whether via REST endpoints that return full JSON objects or GraphQL mutations/queries that let clients specify return fields, the API does not:

- Restrict which properties are serialised or deserialised.
- Validate that the authenticated user is permitted to read or write each specific property.

Attackers can fuzz or enumerate hidden properties - then read fields like fullName or recentLocation, or inject fields such as total_stay_price or blocked - and the server processes them without additional checks.

Recommendations

- **Enforce property-level authorisation**
  Before serialising or applying any client-supplied property, verify the caller is allowed to access or modify that exact field.
- **Use explicit serialisation schemas**
  Avoid blanket methods like to_json() or mass-assignment frameworks. Instead, define DTOs or serialization classes that list only approved fields.

Søren Johanson
Heiligengeiststraße 6-8
26121 Oldenburg (Oldb)
Germany

- **Validate incoming payloads**
  Implement schema-based request validation to reject unexpected or forbidden properties.
- **Review & harden access control logic**
  For every endpoint, codify which roles or users may read or write each property - ideally with a centralised ACL module.
- **Test with property-mapping tools**
  Incorporate fuzzers or automated checks that attempt to inject or request hidden fields as part of your CI-driven API test suite.

Related Standards

- OWASP API Top 10: API3 Broken Object Property Level Authorisation
- ISO 27001: A.5.16 - Access rights
- ISO 27001: A.8.26 - Application security requirements
- SOC 2: CC6.1 - Logical access control
- SOC 2: CC6.3 - Unauthorised access prevention

Additional information

- https://cwe.mitre.org/data/definitions/213.html
- https://cheatsheetseries.owasp.org/cheatsheets/Mass_Assignment_Cheat_Sheet.html

Søren Johanson
Heiligengeiststraße 6-8
26121 Oldenburg (Oldb)
Germany

5. API3:2023 Unauthorised data modifications

Remediation Status:
Criticality: Medium
CVSS-Score: 6.5
Affects: /workshop/api/shop/orders/<order_id>

Executive highlights

Because the API does not restrict which fields a user can change, it is possible to manipulate internal values that should be read-only or protected. Together, these issues allow attackers to obtain free goods, create unauthorised credit and tamper with content without any special privileges or detection.

Technical description

Mass assignment occurs when the API automatically binds all JSON properties in a request to object fields without filtering for authorised or allowed attributes. In this case, the endpoints accept arbitrary fields and apply them directly to internal models. There is no allow-list or schema validation to block protected properties, so an attacker can modify fields they should not control.

In practice:

- A customer can mark an item as “returned” even if they never bought it, and claim it for free.
- A user can inflate their account balance by returning a bogus purchase, crediting themselves with money they never spent.
- Anyone can edit hidden video properties (for example internal flags or metadata) that should only be set by administrators.

Recommendations

- **Enforce an allow-list for updatable fields**
  Only bind and apply properties that are explicitly permitted for each endpoint. Reject any request containing unknown or protected fields.
- **Use explicit data transfer objects (DTOs) or schemas**
  Define precise request and response models (for example with JSON Schema or strong-typing in your framework) so that only intended properties can be serialised or deserialised.

Søren Johanson
Heiligengeiststraße 6-8
26121 Oldenburg (Oldb)
Germany

- **Filter incoming payloads at a central layer**
  Implement middleware or a service that strips out any fields not on the allow-list before reaching business logic.
- **Validate critical business logic on the server**
  For operations such as returns or refunds, confirm that the user has a valid purchase record and that any numeric values fall within expected bounds.
- **Restrict identifier mutability**
  Do not allow clients to alter system-managed properties without passing role-based access controls or workflow approvals.
- **Log and monitor unauthorised field access attempts**
  Record any requests that include removed or forbidden fields and alert on patterns of abuse or enumeration.
- **Include mass-assignment tests in CI**
  Automate tests that attempt to set protected properties and confirm that the API rejects them, ensuring future changes cannot reintroduce the flaw.

Related Standards

- OWASP API Top 10: API3 Broken Object Property Level Authorisation
- ISO 27001: A.5.16 - Access rights
- ISO 27001: A.8.26 - Application security requirements
- SOC 2: CC6.1 - Logical access control
- SOC 2: CC6.3 - Unauthorised access prevention

Additional information

- https://cwe.mitre.org/data/definitions/213.html
- https://cheatsheetseries.owasp.org/cheatsheets/Mass_Assignment_Cheat_Sheet.html

Søren Johanson
Heiligengeiststraße 6-8
26121 Oldenburg (Oldb)
Germany

List of changes

Version	Date	Description	Author
1	2025-05-11	Initial version	Søren Johanson
2	2025-05-13	Added ISO 27001, SOC 2 references	Søren Johanson

Disclaimer

This report was prepared based on a limited, black-box review of publicly accessible API endpoints at the time of testing. No access was provided to source code, infrastructure, or internal documentation. While reasonable effort has been made to identify common API security issues, this audit does not guarantee the absence of vulnerabilities.

Findings in this report are based on observable behaviour and available data during the assessment period. New vulnerabilities may emerge over time as the application evolves or as external threat landscapes change.

This report reflects the application state as of the report date written above. Findings and recommendations may no longer apply if the application has changed since this assessment.

This report is intended solely for use by the client named in the engagement and may not be redistributed, published, or relied upon by third parties without written consent. The recommendations are advisory in nature, and implementation decisions remain the responsibility of the client.

Imprint

Søren Johanson
Heiligengeiststraße 6-8
26121 Oldenburg (Oldb)
Germany