BUGFYND · APPLICATION SECURITY

API Security Penetration Testing Report

Prepared for: Sample Client Pvt Ltd
Application: crAPI – vehicle owner platform (REST API)

Version 1.1 · Final report

SAMPLE REPORT: findings come from a deliberately vulnerable lab application, not from a real client

Confidentiality statement

This document is confidential and intended only for the client named on the cover. It may not be copied, shared or published, in whole or in part, without written permission from the client and Bugfynd. The client may share it with its own auditors, customers and partners to support its security and compliance programs.

About this sample: this report shows the format and level of detail Bugfynd delivers after an API security test. All findings come from OWASP crAPI (completely ridiculous API), an open-source, intentionally vulnerable REST API for a car-owner community and shop, tested in our lab. Client, dates and evidence are illustrative; secrets are redacted.

Contents

#	Section
1	Document details
2	Introduction, scope and duration
3	Executive summary and risk ratings
4	Approach and methodology
5	Results overview and high-level recommendations
6	Detailed findings (6)
7	Conclusion
A–C	Annexures: changes to environment, tools used, tests performed

SAMPLE

1. Document details

Document control

Document title	API Security Testing Report
Classification	Confidential – Final report
Report version	1.1 (includes retest results)

Document history

Version	Date	Prepared by	Status
1.0	[Date]	Bugfynd testing team	Initial report
1.1	[Date]	Bugfynd testing team	Final report after retest

Client information

Company	Sample Client Pvt Ltd
Point of contact	[Name, title]
Email	[contact@client.example]

Bugfynd team

Name	Role	Responsibility
Consultant A	Lead Consultant	Testing and report preparation
Consultant B	Security Consultant	Technical review
Consultant C	Security Consultant	Testing support and retest
[Client contact]	Client representative	Report acceptance

SAMPLE

2. Introduction

Bugfynd performed a manual penetration test of the client's REST APIs behind a car-owner web and mobile platform (identity, community, workshop and shop services) to find security weaknesses an attacker could exploit, measure their business impact and give clear steps to fix them. Testing was carried out under written authorization from the client and within the agreed rules of engagement.

Scope

Application	Type	Version	Components
crAPI	REST APIs (microservices)	[x.y]	/identity/api, /community/api, /workshop/api, mail service

Duration

Activity	Dates
Initial assessment including reporting	[Start date] – [End date]
Retest and report update	[Retest date]

Test approach

- Grey-box testing with two standard user accounts and one administrator account per role provided by the client.
- API documentation (OpenAPI / Postman collection) supplied by the client and supplemented by traffic analysis.
- All requests carried a Bugfynd identification header and stayed within agreed rate limits.
- No denial-of-service or destructive testing was performed.

Limitations

- Results reflect the application version and environment available during the test window.
- Server-side components were tested only where reachable from the client application.

SAMPLE

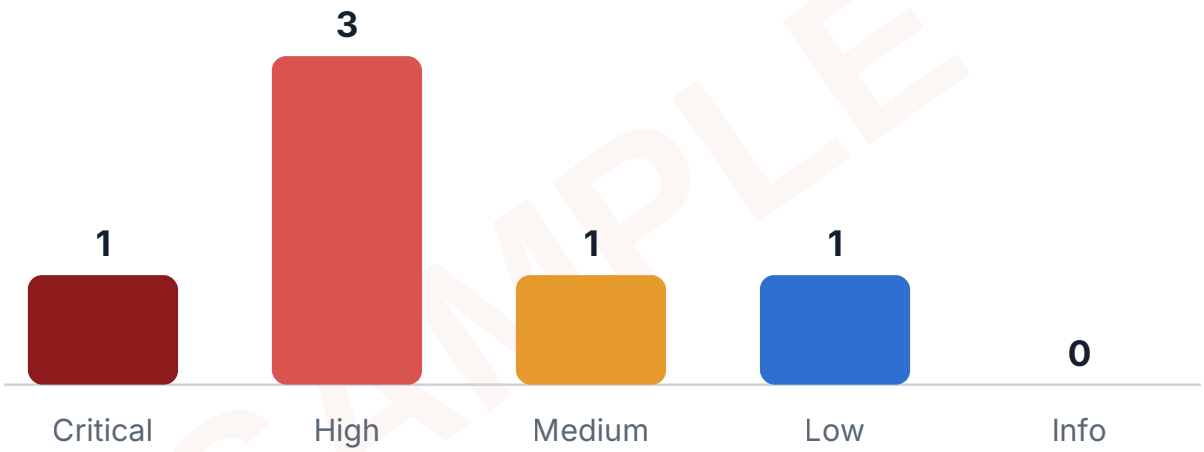
3. Executive summary

Bugfynd identified **6 security findings** in the application: **1 Critical, 3 High, 1 Medium and 1 Low**. The most serious issues let any registered user see the live GPS location of other users' vehicles and take over any account by guessing the password-reset code. A user can also get money credited to their account by changing an order's status through the API.

After the client's fixes, a retest confirmed that **4 of 6** findings are remediated. Two findings (excessive data exposure in community posts and verbose error messages) remain open with an agreed plan.

1	3	1	1	0
Critical	High	Medium	Low	Info

Findings by severity



SAMPLE

Initial test vs retest

Status	Critical	High	Medium	Low	Total
Initial assessment	1	3	1	1	6
Open after retest	0	0	1	1	2

Risk rating

Each finding is scored with **CVSS v3.1** (Common Vulnerability Scoring System), the industry standard for rating how severe a vulnerability is. Scores run from 0 to 10.

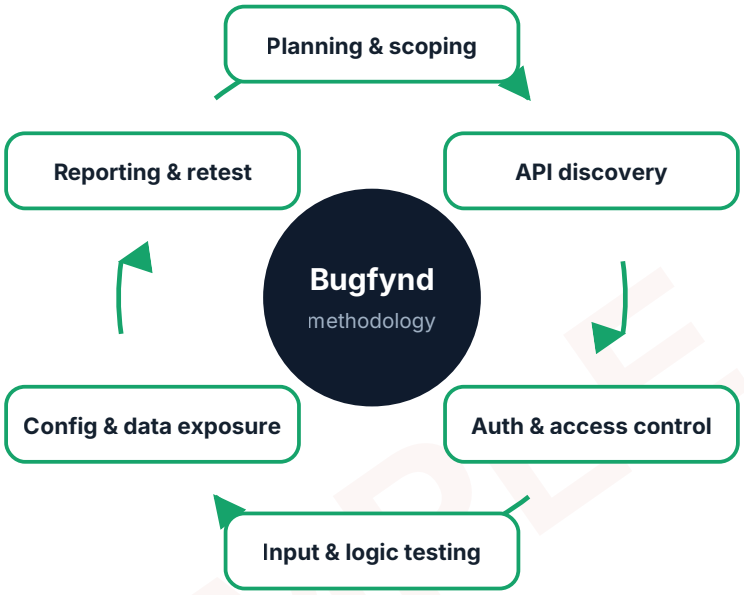
Severity	CVSS score	What it usually means
Critical	9.0 – 10.0	Easy to exploit, often remotely and without credentials; can fully compromise the application or its data. Fix immediately.
High	7.0 – 8.9	Exploitable with little effort; gives access to sensitive data or higher privileges. Fix as a priority.
Medium	4.0 – 6.9	Needs specific conditions or some access; limited impact on its own but can be chained. Fix in the next release.
Low	0.1 – 3.9	Minor impact or hard to exploit; mainly defence-in-depth. Fix as part of normal hardening.
Info	0.0	Best-practice observation with no direct security impact.

Scores are calculated with the FIRST CVSS 3.1 calculator (first.org/cvss/calculator/3.1) and adjusted for the client's environment where relevant.

SAMPLE

4. Approach and methodology

Bugfynd's API testing methodology is manual-first and follows recognised standards: **OWASP API Security Top 10 (2023)**, **OWASP Web Security Testing Guide**, **OWASP ASVS**, **CWE Top 25** and the **Penetration Testing Execution Standard (PTES)**. Every endpoint is tested with at least two user accounts and each role, because the most serious API flaws are authorization flaws that scanners cannot see.



Phase	What we do
Planning & scoping	Agree in-scope APIs, environments, test accounts per role, rate limits and testing windows; obtain written authorization.
API discovery	Map every endpoint from documentation (OpenAPI/Postman), client traffic and content discovery; identify versions, hidden and deprecated endpoints.
Auth & access control	Test authentication, tokens (JWT/OAuth), sessions and object-, property- and function-level authorization using multiple accounts and roles.
Input & logic testing	Test injection, mass assignment, SSRF, file handling, business flows and resource limits (rate limiting, pagination, payload size).
Config & data exposure	Review TLS, CORS, security headers, error handling, excessive data in responses and third-party API usage.
Reporting & retest	Document each finding with impact, reproducible requests, evidence and fixes; report critical issues the same day; retest after fixes.

Communication

- Daily status updates during testing.
- Critical findings reported to the client contact the same day.
- Walkthrough call with the development team after report delivery.

SAMPLE

5. Results overview

#	Finding	Severity	CVSS	Retest status
1	Account Takeover via OTP Brute Force on Older API Version	Critical	9.1	Remediated
2	BOLA – Any User Can Track Other Users' Vehicle Location	High	7.7	Remediated
3	Mass Assignment Allows Fake Refunds	High	7.1	Remediated
4	Server-Side Request Forgery in 'Contact Mechanic'	High	7.5	Remediated
5	Excessive Data Exposure in Community Posts	Medium	5.3	Open
6	Verbose Errors Disclose Internal Details	Low	3.7	Open

High-level recommendations

1. Enforce object-level authorization on every endpoint that takes an object ID (vehicles, reports, orders).
2. Rate-limit and lock out OTP and login attempts on every API version; retire old versions.
3. Accept only explicitly allowed fields on update requests (no mass assignment).
4. Return only the fields each screen needs; remove emails and IDs of other users from public responses.
5. Validate and allow-list outbound URLs to prevent SSRF.

SAMPLE

6.1 Account Takeover via OTP Brute Force on Older API Version

Critical

Severity / CVSS	Critical · 9.1
CVSS vector	`CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:N`
Affected component	POST /identity/api/auth/v2/check-otp
Category	API2:2023 Broken Authentication
Retest status	Remediated
Description	The password-reset flow sends a 4-digit code. The current v3 endpoint limits attempts, but the older v2 endpoint is still live and has no limit, so all 10,000 codes can be tried.
Impact	An attacker who knows a user's email can reset their password and take over the account without any interaction.
Steps to reproduce	1. Request a password reset for test user B's email. 2. Send POST /identity/api/auth/v2/check-otp with B's email, a new password and OTP values 0000–9999 using Burp Intruder. 3. One request returns success; log in as B with the new password.

Evidence	`POST /identity/api/auth/v2/check-otp`
	`{"email":"[user B]","otp":"[0000-9999]","password":"[new]"}`
	`... attempt [n]: 200 OK {"message":"OTP verified"}`
	Screenshots and full request/response logs are attached in the evidence pack.

Recommendation	• Apply the same rate limit and lockout to every API version, or remove v2. • Use longer, expiring codes (6+ digits, 10-minute expiry, single use). • Alert on repeated OTP failures for one account.
Reference	https://owasp.org/API-Security/editions/2023/en/0xa2-broken-authentication/

6.2 BOLA – Any User Can Track Other Users' Vehicle Location

High

Severity / CVSS	High · 7.7
CVSS vector	`CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:C/C:H/I:N/A:N`
Affected component	GET /identity/api/v2/vehicle/{vehicleId}/location
Category	API1:2023 BOLA
Retest status	Remediated
Description	The vehicle location endpoint returns the GPS position of any vehicle ID supplied, without checking that the vehicle belongs to the logged-in user. Other users' vehicle IDs are exposed in community posts (finding 5).
Impact	Any registered user can track the real-time location of other customers' cars, a serious privacy and physical-safety risk.
Steps to reproduce	1. Log in as user A and copy user B's vehicleId from a community post response. 2. Call GET /identity/api/v2/vehicle/{B's vehicleId}/location with A's token. 3. The response returns B's latitude, longitude and name.

Evidence	`GET /identity/api/v2/vehicle/[B vehicleId]/location (token of user A)`
	`200 OK {"carId":"[B]","vehicleLocation":`
	`{"latitude":"[REDACTED]","longitude":"[REDACTED]"},"fullName":"[User B]"}`
	Screenshots and full request/response logs are attached in the evidence pack.

Recommendation	• Check vehicle ownership against the authenticated user on every request. • Never accept a vehicle ID for data the user should only see for their own vehicle; derive it from the session. • Add automated BOLA tests for all ID-based endpoints.
Reference	https://owasp.org/API-Security/editions/2023/en/0xa1-broken-object-level-authorization/

6.3 Mass Assignment Allows Fake Refunds

High

Severity / CVSS	High · 7.1
CVSS vector	`CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:H/A:N`
Affected component	PUT /workshop/api/shop/orders/{orderId}
Category	API3:2023 Broken Object Property Level Authorization
Retest status	Remediated
Description	The order update endpoint accepts a status field from the client. Setting status to returned triggers a refund to the user's balance even though no item was returned.
Impact	Users can credit money to their account repeatedly and buy goods for free, causing direct financial loss.
Steps to reproduce	1. Place a shop order as user A and note the orderId. 2. Send PUT /workshop/api/shop/orders/{orderId} with body {"status":"returned"}. 3. Check A's balance: the order amount has been refunded.

Evidence	`PUT /workshop/api/shop/orders/[orderId]`
	`{"status":"returned"}`
	`200 OK  Balance before: [x]  Balance after: [x + order amount]`
	Screenshots and full request/response logs are attached in the evidence pack.

Recommendation	• Allow-list the fields users may update; reject status and quantity changes from customers. • Process returns only through a server-side workflow with verification. • Reconcile refunds against confirmed returns.
Reference	https://owasp.org/API-Security/editions/2023/en/0xa3-broken-object-property-level-authorization/

Sample

6.4 Server-Side Request Forgery in 'Contact Mechanic'

High

Severity / CVSS	High · 7.5
CVSS vector	`CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:C/C:L/I:L/A:N`
Affected component	POST /workshop/api/merchant/contact_mechanic
Category	API7:2023 SSRF
Retest status	Remediated
Description	The request includes a mechanic_api URL that the server calls on the user's behalf. Any URL is accepted, so the server can be made to request internal or external addresses chosen by the attacker.
Impact	An attacker can use the server to reach internal services and cloud metadata endpoints, potentially exposing credentials and internal data.
Steps to reproduce	1. Submit a 'Contact mechanic' request and intercept it in Burp. 2. Change mechanic_api to a Bugfynd-controlled test URL (Burp Collaborator). 3. The test server receives a request from the crAPI backend.

Evidence	`{"mechanic_api":"http://[collaborator-test-host]/", "mechanic_code":"TRAC_JHN", ...}`
	`Collaborator: HTTP request received from [crAPI server IP]`
	Screenshots and full request/response logs are attached in the evidence pack.

Recommendation	• Do not let clients supply URLs the server will call; store mechanic endpoints server-side. • If needed, allow-list exact hosts and block private IP ranges and metadata addresses. • Disable redirects for server-side requests.
Reference	https://owasp.org/API-Security/editions/2023/en/0xa7-server-side-request-forgery/

SAMPLE

6.5 Excessive Data Exposure in Community Posts

Medium

Severity / CVSS	Medium · 5.3
CVSS vector	`CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:L/I:N/A:N`
Affected component	GET /community/api/v2/community/posts/recent
Category	API3:2023 Broken Object Property Level Authorization
Retest status	Open
Description	The recent-posts API returns each author's email address and vehicle ID, although the app only shows the author's name. The extra data feeds other attacks (findings 1 and 2).
Impact	Any user can harvest other customers' email addresses and vehicle IDs, enabling account takeover and location tracking.
Steps to reproduce	1. Log in as any user and open the community page. 2. Inspect the response of GET /community/api/v2/community/posts/recent.
Evidence	`{"author": {"nickname":"[name]","email":"[REDACTED]","vehicleid":"[REDACTED]"}, ...}` Screenshots and full request/response logs are attached in the evidence pack.
Recommendation	• Return only fields the UI needs (e.g. nickname, avatar). • Use response schemas/DTOs instead of serialising database objects. • Add tests that fail if sensitive fields appear in public responses.
Reference	https://owasp.org/API-Security/editions/2023/en/0xa3-broken-object-property-level-authorization/

6.6 Verbose Errors Disclose Internal Details

Low

Severity / CVSS	Low · 3.7
CVSS vector	`CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:L/I:N/A:N`
Affected component	Multiple endpoints
Category	API8:2023 Security Misconfiguration
Retest status	Open
Description	Malformed requests return framework stack traces and internal service names.
Impact	Attackers learn about internal technologies and service layout, making other attacks easier.
Steps to reproduce	1. Send an invalid JSON body to POST /workshop/api/shop/orders. 2. Review the error response.
Evidence	`500 Internal Server Error  {"detail":"... Traceback ... [internal path] ..."}` Screenshots and full request/response logs are attached in the evidence pack.
Recommendation	• Return generic error messages; log details server-side only. • Disable debug mode in production.
Reference	https://owasp.org/API-Security/editions/2023/en/0xa8-security-misconfiguration/

SAMPLE

7. Conclusion

crAPI's main weakness is missing server-side authorization: the API trusts IDs and fields sent by the client. The most urgent issues (vehicle location exposure, account takeover, refund abuse and SSRF) have been fixed and verified. Bugfynd recommends fixing the remaining data exposure and error handling, retiring old API versions and adding automated authorization tests to the CI pipeline.

Annexure A: Changes to environment

No permanent changes were made. Test orders and posts created during testing used Bugfynd test accounts only and were deleted after testing.

Annexure B: Tools used

Tool	Purpose
Burp Suite Professional	Intercepting, replaying and fuzzing API requests
Postman	Importing API collections, managing environments and auth
Autorize (Burp extension)	Automated authorization checks across user roles
JWT Editor / jwt_tool	Inspecting and testing JSON Web Tokens
ffuf / Kiterunner	Endpoint and parameter discovery
InQL / GraphQL Voyager	GraphQL schema discovery and query testing
sqlmap / NoSQLMap	Confirming injection findings in lab-safe mode
sslscan	TLS configuration review

SAMPLE

Annexure C: Tests performed

Test area	Status
API discovery and documentation review	Tested
API1 – Broken object level authorization (BOLA)	Tested
API2 – Broken authentication	Tested
API3 – Broken object property level authorization (mass assignment, data exposure)	Tested
API4 – Unrestricted resource consumption	Tested
API5 – Broken function level authorization	Tested
API6 – Unrestricted access to sensitive business flows	Tested
API7 – Server-side request forgery	Tested
API8 – Security misconfiguration (CORS, headers, errors, TLS)	Tested
API9 – Improper inventory management (old versions, debug endpoints)	Tested
API10 – Unsafe consumption of third-party APIs	Tested
Injection (SQL, NoSQL, command)	Tested
JWT and session handling	Tested
Business logic	Tested

SAMPLE