BUGFYND · APPLICATION SECURITY

API Security Penetration Testing Report

Prepared for: Sample Client Pvt Ltd
Application: VAmPI – book and user management 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 VAmPI, an open-source, intentionally vulnerable REST API for user and book management, 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 API for user accounts and a book catalogue, with JWT-based authentication 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
VAmPI	REST API (Flask)	[x.y]	/users/v1, /books/v1, OpenAPI specification

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

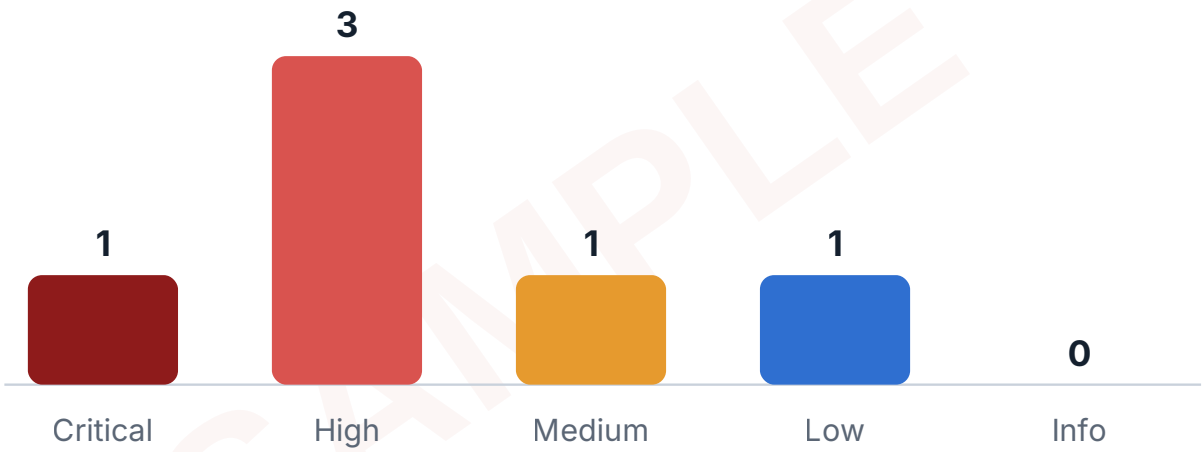
3. Executive summary

Bugfynd identified **6 security findings** in the application: **1 Critical, 3 High, 1 Medium and 1 Low**. The most serious issues let an unauthenticated attacker read the entire user database through SQL injection, and let any user register themselves as an administrator by adding one field to the sign-up request. A debug endpoint also lists every user with their password.

After the client's fixes, a retest confirmed that **5 of 6** findings are remediated. Two Low findings (user enumeration and missing rate limiting on login) 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	0	1	1

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.

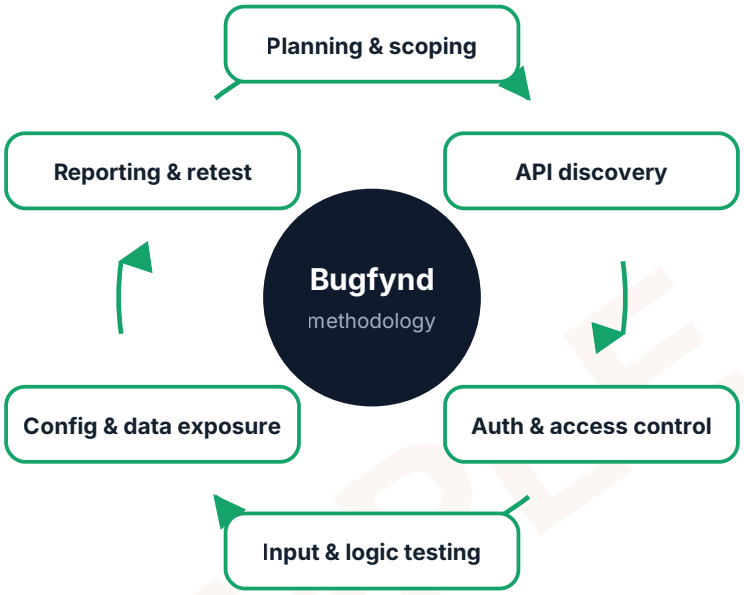
Severity	CVSS score	What it usually means
Critical	9.0 – 10.0	Easy to exploit, often remotely and without credentials; can fully compromise the application or its data. Fix immediately.
High	7.0 – 8.9	Exploitable with little effort; gives access to sensitive data or higher privileges. Fix as a priority.
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	SQL Injection in User Lookup Endpoint	Critical	9.8	Remediated
2	Mass Assignment Allows Admin Self-Registration	High	8.8	Remediated
3	Debug Endpoint Exposes All Users and Passwords	High	7.5	Remediated
4	Broken Function-Level Authorization: Change Any User's Password	High	8.1	Remediated
5	BOLA – Read Other Users' Book Secrets	Medium	6.5	Remediated
6	User Enumeration and No Login Rate Limiting	Low	3.7	Open

High-level recommendations

1. Use parameterized queries in every database call.
2. Allow-list fields on registration and profile updates; never accept admin flags from clients.
3. Remove debug endpoints from production and restrict API inventory to documented endpoints.
4. Check object ownership on every request (books, users).
5. Return identical login errors for unknown users and wrong passwords; rate-limit authentication.

SAMPLE

6.1 SQL Injection in User Lookup Endpoint

Critical

Severity / CVSS	Critical · 9.8
CVSS vector	`CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H`
Affected component	GET /users/v1/{username}
Category	CWE-89 Injection
Retest status	Remediated
Description	The username path parameter is placed directly into a SQL query. A crafted value changes the query and returns data from other tables.
Impact	An unauthenticated attacker can read the whole user table, including password hashes and emails, and may modify data.
Steps to reproduce	1. Send GET /users/v1/name1' to confirm a database error. 2. Send a UNION-based payload in the username parameter. 3. The response returns rows from the users table.

Evidence	`GET /users/v1/name1'`
	`500 ... sqlite3.OperationalError: unrecognized token`
	`GET /users/v1/[UNION payload] -> 200 OK [user rows returned]`
	Screenshots and full request/response logs are attached in the evidence pack.

Recommendation	• Use parameterized queries / ORM bindings. • Validate the username format (allowed characters and length). • Run the API's database account with least privilege.
Reference	https://owasp.org/www-community/attacks/SQL_Injection

SAMPLE

6.2 Mass Assignment Allows Admin Self-Registration

High

Severity / CVSS	High · 8.8
CVSS vector	`CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:N`
Affected component	POST /users/v1/register
Category	API3:2023 Broken Object Property Level Authorization
Retest status	Remediated
Description	The registration endpoint stores every field in the request body, including the admin flag. Adding "admin": true creates an administrator account.
Impact	Anyone can create an admin account and use admin-only functions such as deleting users.
Steps to reproduce	1. Send POST /users/v1/register with username, password, email and "admin": true. 2. Log in with the new account and decode the JWT or call an admin endpoint. 3. Admin actions succeed.

Evidence	`POST /users/v1/register`
	`{"username":"bf_test","password":"[REDACTED]","email":"[test]","admin":true}`
	`DELETE /users/v1/[test user] -> 200 OK (admin action allowed)`
	Screenshots and full request/response logs are attached in the evidence pack.

Recommendation	• Accept only allowed fields on registration (username, email, password). • Assign roles only through an admin-controlled server process. • Add schema validation that rejects unknown fields.
Reference	https://owasp.org/API-Security/editions/2023/en/0xa3-broken-object-property-level-authorization/

SAMPLE

6.3 Debug Endpoint Exposes All Users and Passwords

High

Severity / CVSS	High · 7.5
CVSS vector	`CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N`
Affected component	GET /users/v1/_debug
Category	API9:2023 Improper Inventory Management
Retest status	Remediated
Description	An undocumented debug endpoint returns every user record, including passwords, without authentication.
Impact	Anyone can obtain all usernames, emails and passwords and log in as any user, including admins.
Steps to reproduce	1. Run content discovery against /users/v1/. 2. Send GET /users/v1/_debug without a token.

Evidence	`GET /users/v1/_debug`
	`200 OK {"users":`
	`[{"username":"[REDACTED]","password":"[REDACTED]","admin":true}, ...]}`
	Screenshots and full request/response logs are attached in the evidence pack.

Recommendation	• Remove debug endpoints from production builds. • Keep an API inventory and block undocumented routes at the gateway. • Never store or return plain-text passwords.
Reference	https://owasp.org/API-Security/editions/2023/en/0xa9-improper-inventory-management/

SAMPLE

6.4 Broken Function-Level Authorization: Change Any User's Password

High

Severity / CVSS	High · 8.1
CVSS vector	`CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:N`
Affected component	PUT /users/v1/{username}/password
Category	API5:2023 Broken Function Level Authorization
Retest status	Remediated
Description	The password-change endpoint takes the target username from the URL and does not check that it matches the logged-in user.
Impact	Any logged-in user can change another user's password and take over their account.
Steps to reproduce	1. Log in as user A and obtain A's token. 2. Send PUT /users/v1/[user B]/password with a new password using A's token. 3. Log in as user B with the new password.

Evidence	`PUT /users/v1/[user B]/password  (token of user A)`
	`{"password":"[new]"}  -> 204 No Content`
	Screenshots and full request/response logs are attached in the evidence pack.

Recommendation	• Derive the user from the token, not the URL. • Require the current password for password changes. • Add authorization tests for every user-scoped endpoint.
Reference	https://owasp.org/API-Security/editions/2023/en/0xa5-broken-function-level-authorization/

SAMPLE

6.5 BOLA – Read Other Users' Book Secrets

Medium

Severity / CVSS	Medium · 6.5
CVSS vector	`CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:N/A:N`
Affected component	GET /books/v1/{book_title}
Category	API1:2023 BOLA
Retest status	Remediated
Description	Each book has a private 'secret' visible only to its owner, but the endpoint returns the secret for any title requested by any logged-in user.
Impact	Users can read private data belonging to other users.
Steps to reproduce	1. Log in as user A. 2. List all books and pick a title owned by user B. 3. Request GET /books/v1/{title}: B's secret is returned.

Evidence	`GET /books/v1/[B's title]  (token of user A)`
	`200 OK {"book_title":"[...]","owner":"[user B]","secret":"[REDACTED]"}`
	Screenshots and full request/response logs are attached in the evidence pack.

Recommendation	• Check book ownership before returning private fields. • Return the secret only to the owner.
Reference	https://owasp.org/API-Security/editions/2023/en/0xa1-broken-object-level-authorization/

SAMPLE

6.6 User Enumeration and No Login Rate Limiting

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	POST /users/v1/login
Category	API4:2023 Unrestricted Resource Consumption
Retest status	Open
Description	The login endpoint returns different messages for unknown usernames and wrong passwords, and does not limit attempts.
Impact	Attackers can confirm valid usernames and run password-guessing attacks at high speed.
Steps to reproduce	1. Log in with an unknown username and note the message. 2. Log in with a valid username and wrong password and compare. 3. Send 200 rapid attempts: none are blocked.

Evidence	`unknown user -> "Username does not exist"`
	`wrong password -> "Password is not correct for the given username."`
	Screenshots and full request/response logs are attached in the evidence pack.

Recommendation	• Return one generic message for all login failures. • Rate-limit by IP and account; add progressive delays or lockout.
Reference	https://owasp.org/API-Security/editions/2023/en/0xa4-unrestricted-resource-consumption/

SAMPLE

7. Conclusion

VAmPI's API trusted client input for queries, roles and object access. The most urgent issues (SQL injection, admin self-registration, debug endpoint exposure, BOLA and password change of other users) have been fixed and verified. Bugfynd recommends fixing the remaining authentication hardening items and retesting.

Annexure A: Changes to environment

No permanent changes were made. Test users and books created during testing 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