BUGFYND · APPLICATION SECURITY

GraphQL API Security Penetration Testing Report

Prepared for: Sample Client Pvt Ltd
Application: DVGA – pastebin-style GraphQL 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 a GraphQL API security test. All findings come from DVGA (Damn Vulnerable GraphQL Application), an open-source, intentionally vulnerable GraphQL API, 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	GraphQL 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 GraphQL API behind a pastebin-style application 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
DVGA	GraphQL API (single /graphql endpoint)	[x.y]	/graphql queries, mutations and subscriptions; GraphiQL interface

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

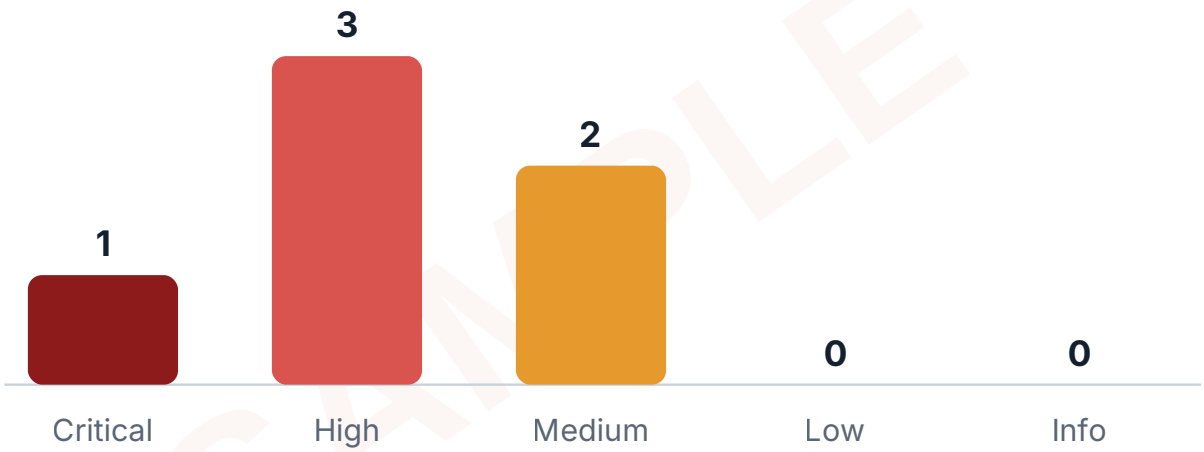
3. Executive summary

Bugfynd identified **6 security findings** in the application: **1 Critical, 3 High, 2 Medium and 0 Low**. The most serious issue lets an attacker run operating-system commands on the server through a GraphQL mutation. Introspection and the GraphiQL console are open to everyone, and deeply nested or batched queries can exhaust server resources.

After the client's fixes, a retest confirmed that **5 of 6** findings are remediated. One Medium finding (field suggestions revealing schema details) remains open with an agreed plan.

1	3	2	0	0
Critical	High	Medium	Low	Info

Findings by severity



SAMPLE

Initial test vs retest

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

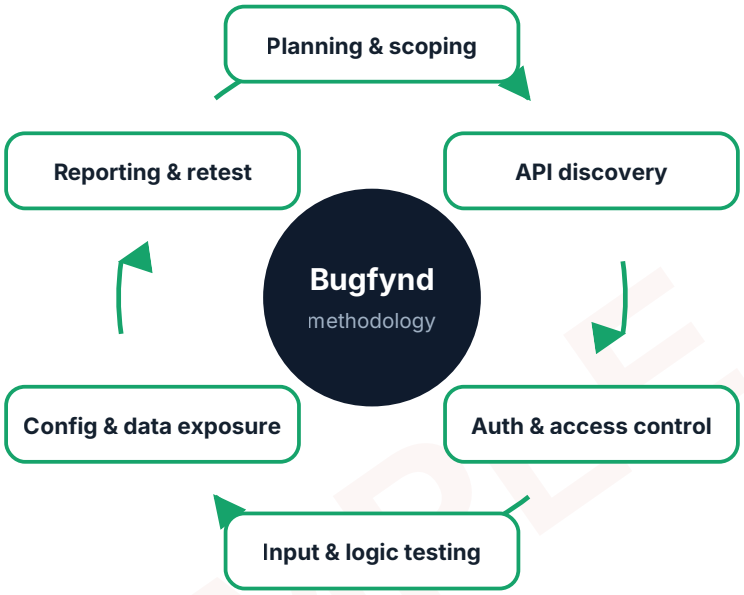
Severity	CVSS score	What it usually means
Critical	9.0 – 10.0	Easy to exploit, often remotely and without credentials; can fully compromise the application or its data. Fix immediately.
High	7.0 – 8.9	Exploitable with little effort; gives access to sensitive data or higher privileges. Fix as a priority.
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	OS Command Injection via GraphQL Mutation	Critical	9.8	Remediated
2	Denial of Service via Deeply Nested Queries	High	7.5	Remediated
3	Unlimited Query Batching	High	7.5	Remediated
4	Missing Authorization on Private Pastes	High	7.1	Remediated
5	Stored XSS via Paste Content	Medium	6.1	Remediated
6	Introspection, Field Suggestions and GraphiQL Enabled	Medium	5.3	Open

High-level recommendations

1. Never pass user input to OS commands; validate and allow-list arguments.
2. Disable introspection, field suggestions and GraphiQL in production.
3. Enforce query depth, complexity and batching limits.
4. Apply authorization in every resolver, not only at the gateway.
5. Sanitize stored content to prevent XSS when pastes are displayed.

SAMPLE

6.1 OS Command Injection via GraphQL Mutation

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	mutation importPaste(host, port, path, scheme)
Category	CWE-78 OS Command Injection
Retest status	Remediated
Description	The importPaste mutation builds a shell command from its arguments to fetch a remote paste. Shell characters in the path argument are not filtered, so extra commands run on the server.
Impact	An attacker can run any command on the API server, leading to full server compromise and data theft.
Steps to reproduce	1. Send the importPaste mutation with a normal host and path. 2. Append `; id` to the path argument. 3. The response contains the output of the id command.

Evidence	`mutation { importPaste(host:"[test-host]", port:80, path:"/; id", scheme:"http") { result } }`
	`-> "result": "... uid=[REDACTED] gid=[REDACTED] ..."`
	Screenshots and full request/response logs are attached in the evidence pack.

Recommendation	• Fetch remote content with an HTTP library, not a shell command. • Validate host, port and path strictly against allow-lists. • Run the service with least privilege in an isolated container.
Reference	https://cwe.mitre.org/data/definitions/78.html

SAMPLE

6.2 Denial of Service via Deeply Nested Queries

High

Severity / CVSS	High · 7.5
CVSS vector	`CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H`
Affected component	/graphql – query depth
Category	API4:2023 Unrestricted Resource Consumption
Retest status	Remediated
Description	Circular relationships in the schema (pastes → owner → pastes) can be nested without limit. A single deeply nested query consumes large amounts of CPU and memory.
Impact	A few requests can make the API slow or unavailable for all users.
Steps to reproduce	1. Build a query nesting pastes { owner { pastes { owner { … } } } } 20 levels deep. 2. Send it once (agreed test window) and measure response time and server load.

Evidence	`Depth 3 query : ~[x] ms`
	`Depth 20 query: > [y] s, CPU spike on API server (monitored with client)`
	Screenshots and full request/response logs are attached in the evidence pack.

Recommendation	• Enforce a maximum query depth (e.g. 5–7). • Apply query cost/complexity analysis and timeouts. • Add rate limiting per client.
Reference	https://cheatsheetseries.owasp.org/cheatsheets/GraphQL_Cheat_Sheet.html

SAMPLE

6.3 Unlimited Query Batching

High

Severity / CVSS	High · 7.5
CVSS vector	`CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H`
Affected component	/graphql – batched requests
Category	API4:2023 Unrestricted Resource Consumption
Retest status	Remediated
Description	The endpoint accepts arrays of queries in one HTTP request with no limit. Rate limits that count HTTP requests are bypassed, and expensive operations can be multiplied.
Impact	Attackers can brute-force values or overload the server while appearing to send only one request.
Steps to reproduce	1. Send a JSON array containing 100 copies of the systemHealth query in one request. 2. All 100 operations execute.

Evidence	`POST /graphql  [ {"query":"query{systemHealth}"}, ... x100 ]`
	`200 OK  -> 100 results returned`
	Screenshots and full request/response logs are attached in the evidence pack.

Recommendation	• Disable batching or limit operations per request. • Count each operation (not each HTTP request) against rate limits.
Reference	https://cheatsheetseries.owasp.org/cheatsheets/GraphQL_Cheat_Sheet.html

SAMPLE

6.4 Missing Authorization on Private Pastes

High

Severity / CVSS	High · 7.1
CVSS vector	`CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:L/A:N`
Affected component	query pastes(public:false) / paste(id)
Category	API1:2023 BOLA
Retest status	Remediated
Description	Private pastes are hidden in the user interface, but the resolver returns them to any caller who asks for public:false or a paste ID.
Impact	Anyone can read other users' private pastes, which may contain confidential information.
Steps to reproduce	1. Query pastes(public:false) { id title content owner { name } } without logging in. 2. Private pastes of other users are returned.

Evidence	`query { pastes(public:false) { id title owner { name } } }`
	`-> [ {"id":"[..]","title":"[private]","owner":{"name":"[user B]"}} ]`
	Screenshots and full request/response logs are attached in the evidence pack.

Recommendation	• Check ownership and visibility inside every resolver. • Never rely on the UI to hide private objects.
Reference	https://owasp.org/API-Security/editions/2023/en/0xa1-broken-object-level-authorization/

SAMPLE

6.5 Stored XSS via Paste Content

Medium

Severity / CVSS	Medium · 6.1
CVSS vector	`CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:C/C:L/I:L/A:N`
Affected component	mutation createPaste(content)
Category	CWE-79
Retest status	Remediated
Description	Paste content is stored without sanitization and displayed as HTML to other users.
Impact	Script runs in the browser of anyone viewing the paste, allowing session theft or actions on their behalf.
Steps to reproduce	1. Create a paste with content `<img src=x onerror=alert(document.domain)>`. 2. Open the paste as another user: the alert appears.

Evidence	`mutation { createPaste(title:"t", content:"[XSS test payload]", public:true){ paste{ id } } }`
	`Viewing paste as user B -> alert(document.domain) executed`
	Screenshots and full request/response logs are attached in the evidence pack.

Recommendation	• Encode output when rendering pastes; treat content as text. • Apply a strict Content Security Policy.
Reference	https://owasp.org/www-community/attacks/xss/

SAMPLE

6.6 Introspection, Field Suggestions and GraphiQL Enabled

Medium

Severity / CVSS	Medium · 5.3
CVSS vector	`CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N`
Affected component	/graphql, /graphiql
Category	API8:2023 Security Misconfiguration
Retest status	Open
Description	Introspection returns the full schema, including hidden admin mutations. When introspection is turned off, error messages still suggest valid field names. The GraphiQL console is public.
Impact	Attackers can map every query and mutation quickly, which made the other findings in this report easy to discover.
Steps to reproduce	1. Send the standard introspection query to /graphql. 2. Send a misspelled field (e.g. pastee) and read the 'Did you mean' suggestion. 3. Open /graphiql in a browser without logging in.

Evidence	`{__schema{types{name}}} -> full schema returned`
	`query { pastee } -> "Cannot query field 'pastee'. Did you mean 'pastes'?"`
	Screenshots and full request/response logs are attached in the evidence pack.

Recommendation	• Disable introspection and GraphiQL in production. • Turn off field suggestions in error messages. • Use persisted (allow-listed) queries for production clients.
Reference	https://cheatsheetseries.owasp.org/cheatsheets/GraphQL_Cheat_Sheet.html

SAMPLE

7. Conclusion

The GraphQL API exposed its full schema and trusted query structure and arguments from clients. The most urgent issues (OS command injection, denial of service through nested and batched queries, missing resolver authorization and stored XSS) have been fixed and verified. Bugfynd recommends disabling field suggestions and adding GraphQL-specific security tests to the release pipeline.

Annexure A: Changes to environment

No permanent changes were made. Test pastes created during testing were deleted. Resource-exhaustion tests were limited to a few requests in an agreed window.

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