

GRAPHQL SECURITY.



A single GraphQL request can nest, alias, and multiply its way to the computational cost of thousands — **the danger isn't how many requests arrive, it's how much any one of them is actually allowed to ask for.**

- INTROSPECTION CONTROL
- QUERY DEPTH & COST ANALYSIS
- ALIAS & BATCHING LIMITS
- FIELD-LEVEL AUTHORIZATION

01 THE MECHANISM



Introspection-Driven Reconnaissance

GraphQL's introspection system lets a client query the entire schema, every type, field, and mutation, in a single request — **an attacker maps the full API surface before sending a single exploit**, far cheaper than REST's one-endpoint-at-a-time discovery.



Nested Query Depth & Complexity

A single, valid GraphQL query can nest relationships dozens of levels deep, multiplying backend work exponentially with each layer — **one HTTP request can do the damage of thousands**, invisible to any control that only counts requests.



Alias-Based Batching Abuse

GraphQL aliases let a client request the same mutation or query many times, with different arguments, inside one request — **a login mutation aliased a thousand times becomes a thousand password guesses** behind a single rate-limit check.

02 THE FLOW

The Hardened GraphQL API

CONTAINED

- Introspection is disabled in production, or restricted to authenticated internal tooling, so the schema isn't handed to every anonymous caller.
- Every query is scored for depth and computed cost before execution, so **an expensive nested query is rejected before it ever touches a resolver**.
- Aliased operations per request are capped, so **a single request can't smuggle thousands of effective operations past request-based rate limits**.

The Wide-Open GraphQL API

EXPOSED

- Introspection stays enabled by default, so any anonymous caller can enumerate the entire schema, including internal or deprecated fields, in one request.
- No depth or cost limit exists, so **a single deeply nested query can exhaust database and compute resources** while looking like one ordinary request.
- Aliasing is unrestricted, so **an attacker batches thousands of login attempts or lookups into one HTTP call**, past any limit that counts requests instead of operations.

03 CORE PILLARS & COVERAGE

PILLAR	WHAT IT COVERS
Introspection Control	Disables or restricts schema introspection in production, closing off the single-request reconnaissance that hands an attacker the entire API surface.
Query Depth & Cost Analysis	Scores every incoming query for nesting depth and computed execution cost, rejecting expensive queries before they reach a resolver, not after they've done the damage.
Alias & Batching Limits	Caps the number of aliased operations permitted in a single request, closing the gap that lets one call smuggle thousands of effective operations past request-count defenses.
Field-Level Authorization	Enforces authorization checks on every field a query can traverse to, not just the top-level query or mutation, since the graph offers more than one path to the same data.

04 DETECTION APPROACHES COMPARED

APPROACH	SUFFICIENT ALONE?	WHAT IT MISSES
Introspection Left Enabled in Production	No	Hands any anonymous caller a complete map of the schema, including fields never meant to be discovered, in a single request.
Request-Count Rate Limiting Alone	No	Counts HTTP requests, not the operations inside them — a single request with a thousand aliased mutations passes as one call.
Depth Limiting Without Cost Analysis	No	Blocks obviously deep queries but misses a wide, shallow query that fans out into thousands of expensive field resolutions.
Authorization Checked Only on Top-Level Queries	No	Misses the same sensitive field reachable through a different, unprotected path deeper in the graph.
Introspection Control + Cost Analysis + Alias Limits + Field-Level Authorization	Yes	Closes reconnaissance, single-request resource exhaustion, batching abuse, and graph-traversal authorization gaps together.

GRAPHQL SECURITY.

LIMIT · SCORE · AUTHORIZE

05 PROGRAM OPERATIONS

A GraphQL API is one endpoint with unlimited shapes of request, not one shape with unlimited endpoints. Every defense built for REST's many-endpoints model has to be rethought for a single, arbitrarily-shaped endpoint.

Cost analysis has to run before execution, not after. Once a resolver has started fetching nested data, the backend work is already spent regardless of whether the response is ultimately blocked.

The graph has more than one path to the same data. A field protected at one entry point in the schema can still be reachable, unprotected, through a different relationship elsewhere in the graph.

Persisted queries trade flexibility for a drastically smaller attack surface. Restricting production traffic to a pre-approved set of query hashes removes arbitrary client-constructed queries as a threat.

06 CONTROL MAPPING

OWASP GRAPHQL CHEAT SHEET

Consolidated guidance on introspection, query depth/cost limiting, batching, and authorization specific to GraphQL APIs.

GRAPHQL SPECIFICATION (INTROSPECTION)

The spec-defined introspection system itself, a feature by design and a reconnaissance risk by default configuration.

PERSISTED QUERIES (APQ)

A production-hardening pattern that restricts accepted queries to a pre-registered, hashed allowlist.

OWASP API SECURITY TOP 10 (2023)

Names unrestricted resource consumption as a top API risk, directly applicable to GraphQL's depth and complexity exposure.

07 WHAT THIS DOES NOT SOLVE

Underlying Resolver Vulnerabilities

Query-level controls don't fix an injection or logic flaw inside a specific resolver's own implementation.

Business Logic Abuse Through Legitimate Queries

Depth and cost limits don't catch a well-authorized query pattern being used for scraping or enumeration at a pace under any single threshold.

Authentication Weaknesses

Hardening the query layer doesn't fix a weak or missing authentication scheme sitting in front of the GraphQL endpoint itself.

Client-Side Data Exposure

Limiting server-side query cost doesn't control what a legitimately authorized client does with the data once it's been returned.

08 MYTHS VS. REALITY

MYTH "Disabling introspection makes a GraphQL API secure."	REALITY It closes one reconnaissance path, but an attacker can still enumerate the schema through error messages or brute-forcing field names — Introspection control is one layer, not the whole defense.
MYTH "Rate limiting works the same way for GraphQL as it does for REST."	REALITY A GraphQL rate limit counting requests misses that one request can contain thousands of effective operations through nesting or aliasing.
MYTH "Limiting query depth is enough to prevent resource exhaustion."	REALITY A wide, shallow query can fan out into as much backend work as a deep one — depth alone doesn't capture actual computational cost.
MYTH "Authorizing the top-level query field secures all the data behind it."	REALITY The same sensitive field can often be reached through a different relationship elsewhere in the graph that never passed through that check.
MYTH "GraphQL is inherently less secure than REST."	REALITY GraphQL isn't less secure by design — it concentrates a different set of risks behind one flexible endpoint, which requires GraphQL-specific controls rather than reused REST assumptions.

09 SPEAK THE PROGRAM — KEY TERMS

INTROSPECTION

A built-in GraphQL feature letting a client query the schema itself, revealing every type, field, and mutation in one request.

QUERY DEPTH ATTACK

A single query nesting relationships many levels deep to multiply backend work exponentially from one request.

QUERY COST ANALYSIS

Scoring a query's expected execution cost before running it, based on field weights and expected result-set sizes, to reject expensive requests up front.

ALIAS-BASED BATCHING

Using GraphQL's alias syntax to request the same operation many times with different arguments inside a single HTTP call.

PERSISTED QUERIES

Restricting a production endpoint to a pre-registered allowlist of query hashes, rejecting arbitrary client-constructed queries.

FIELD-LEVEL AUTHORIZATION

Enforcing access control on every individual field a query can reach, not just the top-level query or mutation it started from.