The State of GraphQL Security 2024

Insights from 13,000 GraphQL API issues:
A deep dive into the current state of GraphQL security

escape

13K

issues of varying severity

1/3

of API services have **highly-critical** issues

4.4K

secrets exposed in public GraphQL APIs

x3

more issues per API service
compared to last year due to the improved quality of scanning and **more in-depth coverage**

Key Takeaways

Recent findings from Gartner underscore a significant trend: the rapid adoption of GraphQL - by 2027, more than 60% of enterprises will use GraphQL in production, up from less than 30% in 2024. Against this backdrop, GraphQL security becomes increasingly crucial. Our study focused on a sample of public GraphQL APIs and revealed that these services still face vulnerabilities, with thousands of issues identified. Thanks to enhancements in our scanning tools, we now detect more issues per API service than last year. The range and severity of these vulnerabilities highlight the widespread challenge of securing GraphQL APIs and underscore their critical importance.

Tristan Kalos, CEO of Escape

Criticality: **33% of API services have highly critical** GraphQL vulnerabilities, and over 72% are vulnerable to medium-level issues. This indicates potential widespread weaknesses in GraphQL security practices.

Main attack vectors:

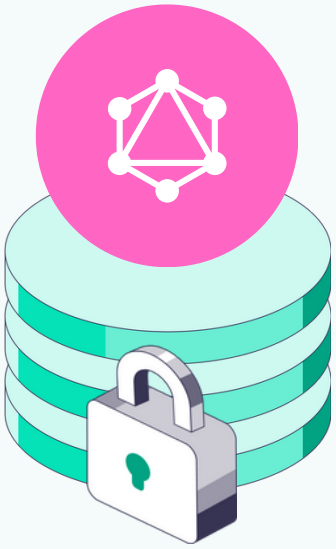
- Nearly **69%** of the API services we scanned had issues related to **Unrestricted Resource Consumption**, making them susceptible to Denial of Service (DoS) attacks.
- Approximately **11.1%** of the services experienced issues associated with **Security Misconfiguration**. Improper customization and configuration options in GraphQL can create security gaps if not properly managed.

Exposed secrets: More than **4,000 exposed secrets** have been uncovered in GraphQL API responses, including access tokens, passwords, and credit card numbers. Preventing public access to these details is crucial.

GraphQL-specificity remains a general concern: 13.4% of the vulnerabilities found are specific to the GraphQL language and its frameworks - showing that organizations do not correctly manage the new risks associated with the technology yet.

Call to action: 80% of issues could have been resolved by implementing best practices such as access control with authorization and authentication, input validation, and rate limiting to block brute-force attacks.

Contents



Introduction

60%

of enterprises will use GraphQL in production by 2027, according to Gartner*

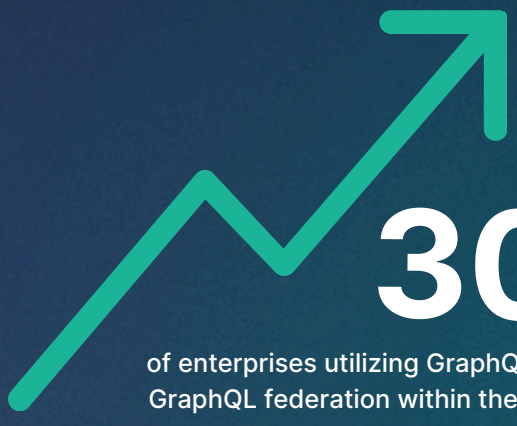
RISING ADOPTION OF GRAPHQL DEMANDS INCREASED SECURITY ATTENTION

GraphQL's flexibility and efficiency have made it increasingly popular among companies, with Gartner projecting its enterprise usage to rise from 30% in 2024 to 60% by 2027*. Furthermore, Gartner predicts that, "By 2027, 30% of enterprises utilizing GraphQL will employ GraphQL federation, up from less than 5% in 2024."

However, this growing reliance on GraphQL underscores the need to address its security risks. Protecting GraphQL APIs is essential to protect sensitive data and ensure application integrity. Security vulnerabilities such as unauthorized access, data exposure, and denial-of-service attacks can lead to data breaches and system downtime.

This report aims to provide insights into the current state of GraphQL API security, highlighting common vulnerabilities and the potential risks they pose. By understanding these security challenges, organizations can proactively protect their GraphQL APIs and prevent breaches.

This report also emphasizes the necessity of securing GraphQL APIs before their release in production. It has been written with CISOs, CTOs, VP Application and Product Security, and their security and GraphQL development teams in mind to help them address these security concerns effectively.



30%

of enterprises utilizing GraphQL will employ GraphQL federation within the next 3 years

escape

*according to Gartner® Report: When to Use GraphQL to Accelerate API Delivery

EXAMPLES OF GRAPHQL VULNERABILITIES FOUND IN ENTERPRISES

Escape discovered a significant vulnerability in Philips' GraphQL API, where it was processing requests over HTTP instead of the more secure HTTPS. This misconfiguration exposed sensitive data to potential man-in-the-middle (MITM) attacks. The issue was reported through Philips' Coordinated Vulnerability Disclosure system, and Philips promptly addressed it, acknowledging the Escape's report in their Hall of Honors.

Varonis Threat Labs discovered two significant vulnerabilities in Zendesk Explore. The first was a SQL injection vulnerability in a GraphQL API endpoint, which could have allowed attackers to access sensitive data from any table in the Zendesk account's RDS. The second was an access control flaw that permitted unauthorized access to data without proper credentials. These vulnerabilities were reported to Zendesk and were patched promptly.

Methodology

160

scanned public GraphQL services

98 hours

Total scanning time

Our research team enumerated 160 full public GraphQL services online.

These services amounted to a total of 3,835 GraphQL operations.

This year, we've decided to limit our scope of research to improve the quality of the results. To discover and analyze these APIs, we used Escape's proprietary inventory and scanning tools, ensuring that our findings are of the utmost quality.

Escape's tool can automatically perform 116 security tests, covering many potential vulnerabilities, with 100 of them supported for GraphQL. These tests are designed to identify weaknesses that attackers could exploit, and the full list is available in our public documentation.

To avoid interfering with the applications we audited, all scans were run in read-only mode, unauthenticated, and with conservative rate limits.

Consequently, most of the findings presented in this report are exploitable remotely without any specific privileged authorization token.

Key definitions

The following key definitions are essential for understanding the context and findings in this report:

Services: The server running GraphQL, typically accessible via a base URL, e.g., https://api.myorg.com/graphql.

Operations: Specific GraphQL operations, such as queries or mutations, e.g., getUsers or createUser.

Findings

13,720

Total issues found

4,527

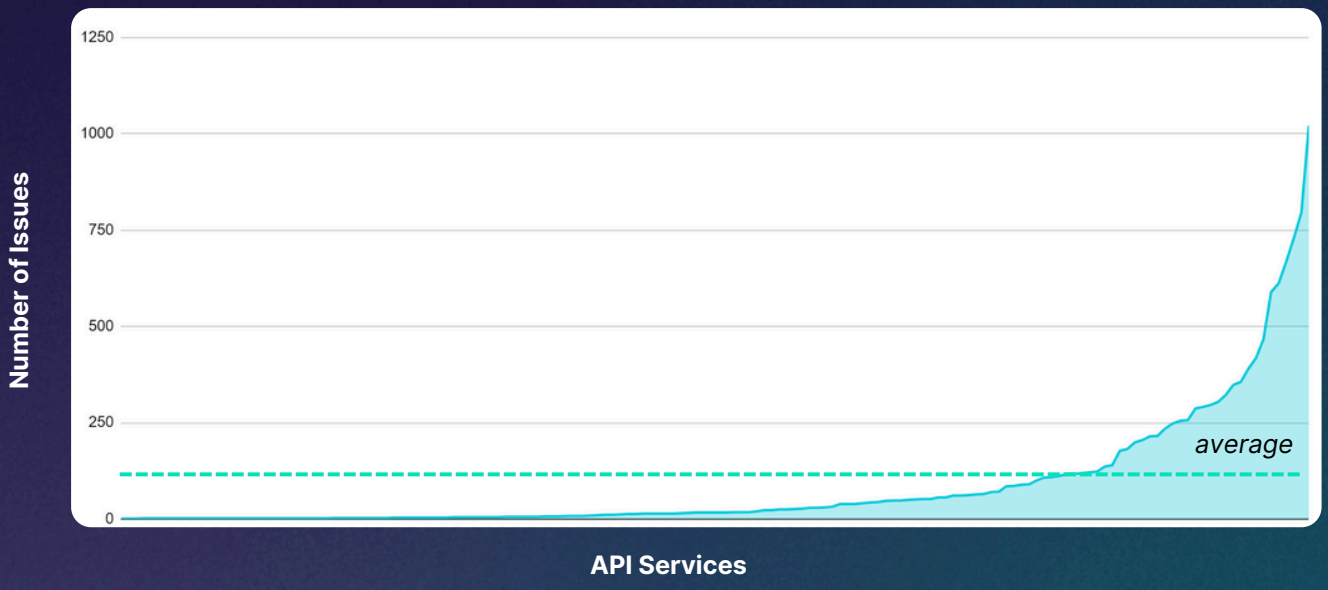
Highly critical

87

average issue per GraphQL service

x3

more issues per API service
compared to last year due to the improved
quality of scanning and **more in-depth coverage**



The histogram of the number of issues per API service shows that most services average around 87 issues. Very few services had fewer than 10 issues, while a few outliers had significantly more, with one service reaching up to 1,021 issues.

Severity of the Issues

33%
with High severity

72%
with Medium severity

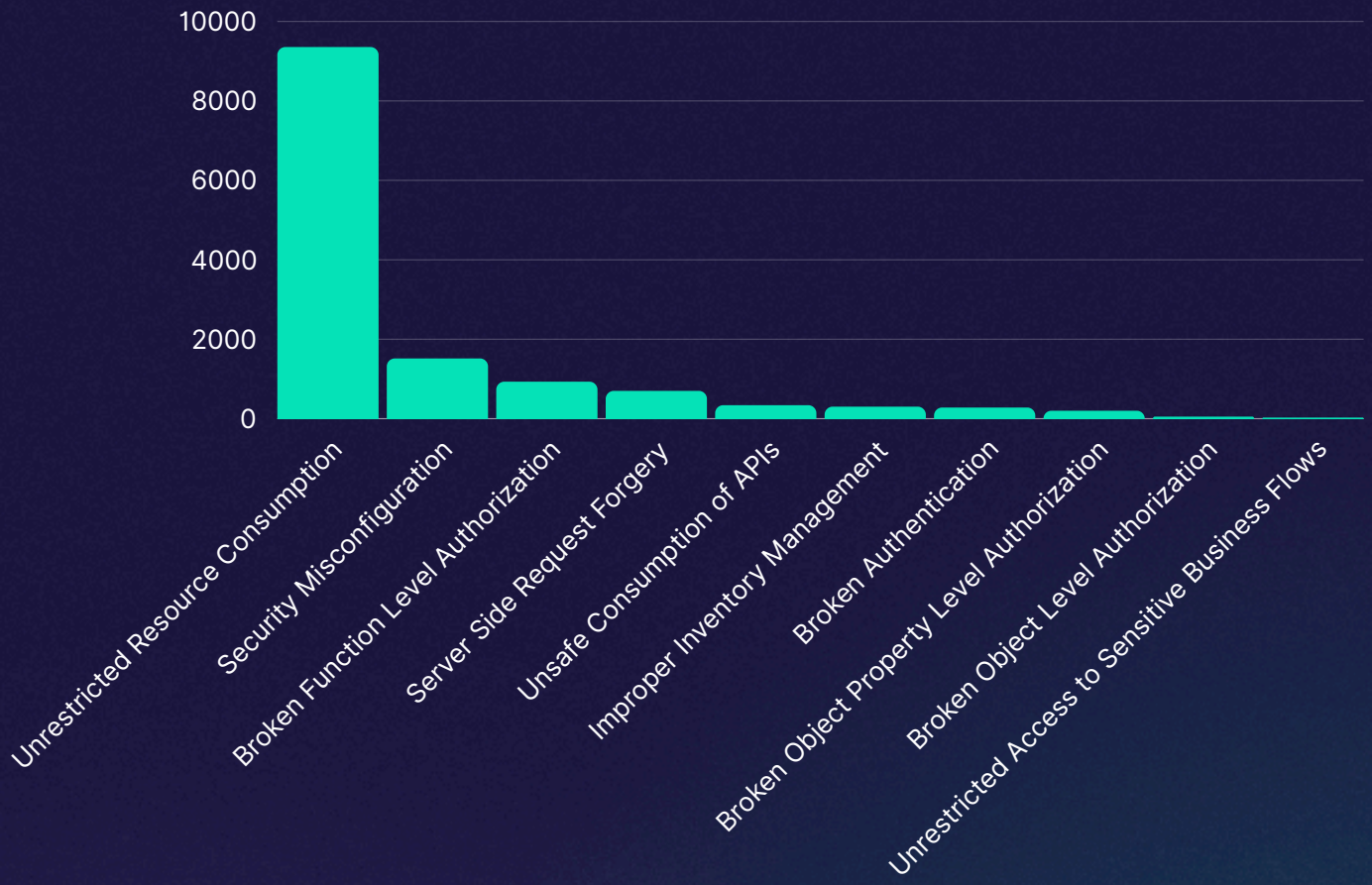
We categorized the issues found using CVSS v3.1 ratings and discovered that almost **99%** of scanned API services had one or more issues of varying severity.

- **High Severity**: Over **33%** had at least one high-severity issue, posing significant security risks if not promptly addressed.
- **Medium Severity**: Over **72%** had medium-severity issues, which still require timely remediation.
- **Low Severity**: Around **78%** had low-severity issues, which were less critical but still needed attention.

It's important to tackle all severity levels when securing GraphQL. High and medium issues are crucial and should be prioritized by development teams. However, each vulnerability, no matter how small, plays a role in maintaining overall security.

Analysis of Key Vulnerabilities

Breaking down the issues per category, we can quickly observe what the common risks for GraphQL APIs are:



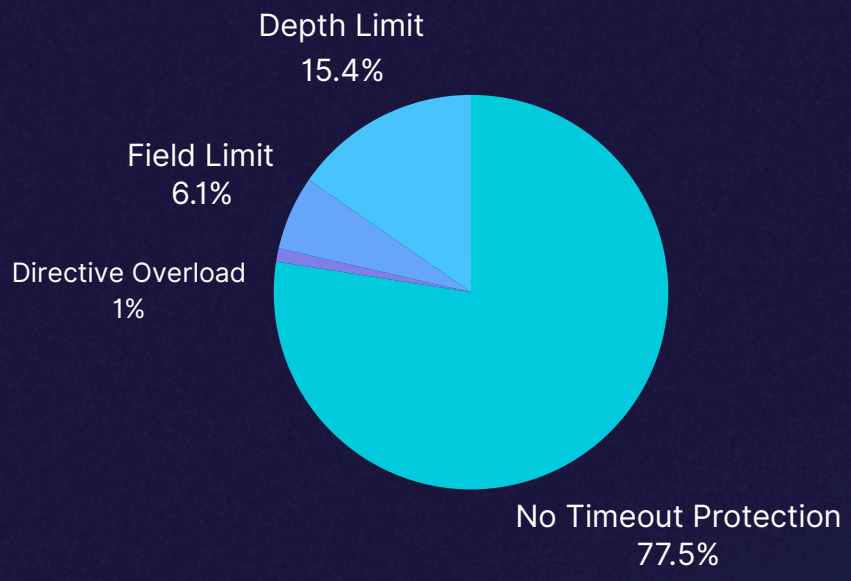
The chart illustrates the distribution of compliance issues related to the OWASP API Top 10 for 2023.

- **Unrestricted Resource Consumption API4:2023 (68.3%):** The majority of issues are linked to this category, suggesting that many APIs lack proper rate limiting and resource allocation mechanisms, making them prone to Denial of Service (DoS) attacks. GraphQL's ability to craft complex queries exacerbates this risk, as attackers can exploit this flexibility to overwhelm the server.

- **Security Misconfiguration API8:2023 (11.1%):** These issues stem from complex, often improperly set configurations, leading to various vulnerabilities. The extensive customization and configuration options of GraphQL can lead to security gaps if not properly managed, increasing the likelihood of misconfigurations.

- **Broken Function Level Authorization API5:2023 (6.8%):** Flaws in complex access control policies allow unauthorized access to resources or administrative functions. GraphQL's dynamic query capabilities make it challenging to implement consistent authorization checks across all endpoints, increasing the risk of such vulnerabilities.

Unrestricted Resource Consumption



Unrestricted Resource Consumption Breakdown

Unrestricted resource consumption accounts for **over 68% of the total issues found**. The graph illustrates the various subcategories of issues causing unrestricted resource consumption in GraphQL APIs.

Timeout issues, representing 77.5% of resource limitation problems, occur when requests take too long to process, leading to potential DoS attacks. Implementing a server timeout, such as a 5-second limit, can mitigate this issue. For detailed guidance on configuring timeouts and managing query complexity in GraphQL, refer to this article.

Depth limit and field limit contribute to a combined 21% of the causes. These issues are GraphQL-specific due to its design and querying capabilities.

Directive overload accounts for around 1% of the causes. This occurs when a user sends a query with many consecutive directives, overloading the engine handling those directives.

Schema Availability

6.3%
Leaking Schema

45%
Closed Schema

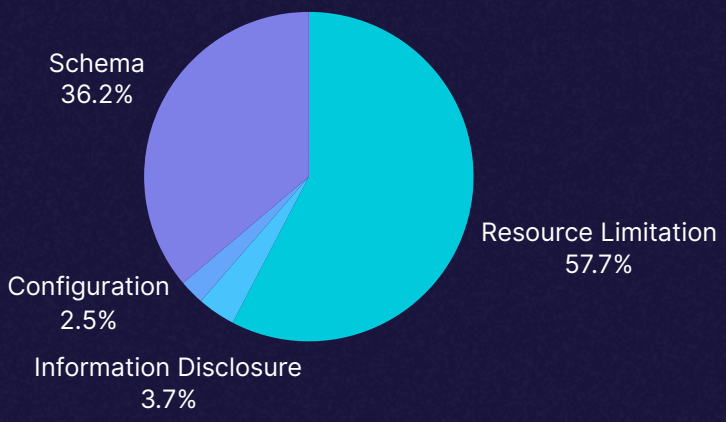
We found that 6.3% of schemas were leaking, i.e. exposing or making accessible the detailed structure and organization of the GraphQL API's data model, including field names, types, and possibly even underlying database structures. Attackers can use the exposed schema to better understand potential vulnerabilities or weak points in the API's design, facilitating more effective exploitation attempts.

45% of schemas were closed with no suggestions, effectively hiding schema details. While closing schemas isn't inherently bad, relying on security by obscurity is ineffective. APIs should be developed assuming schemas are always visible.

GraphQL-specific Issues

13.4%

GraphQL specific issues



Breakdown of GraphQL specific issues

Unlike REST, GraphQL is a full-featured query language, offering a wide attack surface for malicious actors to craft abusive queries. The complexity of parsing GraphQL adds to the security challenges. Our research found that 13.4% of the issues were specific to GraphQL, highlighting that the unique risks associated with GraphQL are not yet well-handled by engineering and security teams, making GraphQL APIs more vulnerable than their REST counterparts.

GraphQL-Specific Risks:

- Resource Limitations: The most common risk, caused by the absence of limits on breadth, depth, fields, aliases, or batch queries, and circular introspection, leading to DoS attacks.
- Schema Issues: Challenges such as positive integer validation errors, duplicated objects, typing misconfigurations, and zombie objects. For instance, zombie objects can reveal legacy or unused parts of the codebase, posing a severe security risk as they are often unmaintained and unpatched.
- Information Disclosure: These issues primarily arise when introspection is disabled. Tools like Clairvoyance can infer the entire schema through “Field Suggestions” when a query contains typos.
- Configuration Issues: Potential vulnerabilities from GraphQL IDEs, absence of Automatic Persisted Queries causing backend performance issues, and large query strings leading to increased latency and degraded client performance.

Addressing these risks is crucial for improving the security of GraphQL APIs and ensuring they are as robust as their REST counterparts.

Industry-wise Breakdown

123
Financial Services

64
Technology & IT

Top 2 industries with the highest number of issues per API service

By mapping chosen GraphQL services to their respective industries, we've tried to identify patterns in how GraphQL is used across different industries and understand the particular vulnerabilities that may arise.

Our findings highlight that two industries are more prone to GraphQL security risks than others - Financial services and Technology & IT.

Banks and financial institutions are increasingly looking to deploy APIs for all areas. According to the 2023 McKinsey survey on APIs in banking, *"Large banks are launching API programs and allocating about 14 % of their IT budget to APIs on average."* This trend highlights the critical role APIs play in enhancing the agility, efficiency, and innovation capabilities of financial institutions.

However, despite this rapid growth, financial institutions continue to face multiple GraphQL API security risks. Discussions around the need for stronger API governance and security controls are frequent, yet concrete actions are often delayed until an emergency arises. This reactive approach is problematic, as it leaves institutions vulnerable to potential breaches and attacks.

When we discussed this issue with an API security professional in the banking sector, she noted, ***"many organizations still don't have a clear view of their inventory".***

This gap in visibility and proactive security measures can lead to significant vulnerabilities, exposing sensitive financial data. Therefore, it is crucial for financial institutions to prioritize the implementation of strong security practices for modern development frameworks like GraphQL and to adopt proactive governance measures.

"Some of the main challenges are that security takes time and money, so it doesn't get the attention it needs until an emergency occurs. Some firms are more proactive, but overall more need to be proactive to really change the game. Stronger and more consistent security controls will make a difference when applied."

- Jack Hart,
API Security | VP, Information Security Architect at City National Bank

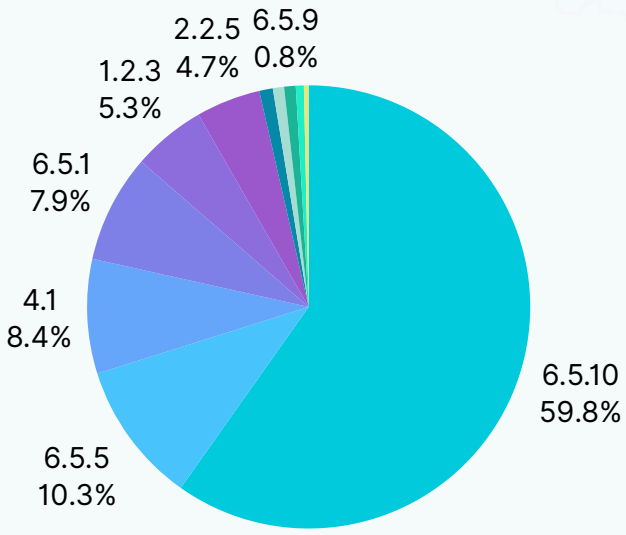
Alignment with Compliance Frameworks

Compliance with security standards and regulations is crucial for organizations leveraging GraphQL APIs. Adhering to these standards helps protect sensitive data and ensures organizations meet legal and regulatory requirements, avoiding severe penalties and reputational damage.

Almost all the APIs we tested **were non-compliant with at least one type of compliance standards:**

- NIS2 (Network and Information Systems Directive 2)
- GDPR (General Data Protection Regulation)
- PCI DSS (Payment Card Industry Data Security Standard)
- PSD2 (Payment Services Directive 2)
- ISO 27001 (International Organization for Standardization 27001)
- NIST (National Institute of Standards and Technology)
- SOC 2 (System and Organization Controls 2)
- FedRAMP (Federal Risk and Authorization Management Program)
- HIPAA (Health Insurance Portability and Accountability Act)

Focus on PCI DSS



The chart shows the distribution of PCI DSS compliance issues, with 59.8% linked to Requirement 6.5.10 - Broken authentication and session management.
This statistic underscores vulnerabilities from insecure coding practices such as weak password policies, inadequate session management, and insufficient use of multi-factor authentication.

Sensitive Data Exposed

4,428
Secrets Exposed

49
Passwords

2
Credit Cards

1,396
Access Tokens

Our analysis uncovered **more than 4k exposed secrets** through GraphQL API responses, including access tokens, passwords, and credit card numbers. These should never be publicly accessible, as they grant entry to internal company systems, paving the way for privilege escalation and unauthorized access.

We also discovered significant amounts of Personally Identifiable Information (PII) such as emails, phone numbers, and bank account numbers. The exposure of this data can lead to severe consequences including phishing attacks, identity theft, and financial scams.

The volume of secrets leaked by our tested endpoints highlights a concerning lack of proper authorization mechanisms in GraphQL.

We've already highlighted the urgent need to address secret spawl in our *"How we discovered over 18,000 API secret tokens"* report.

To address these security vulnerabilities and risks, organizations should implement proactive security measures, adhere to best practices, and conduct continuous security assessments.

You can follow our in-depth recommendations to mitigate these issues.

RECOMMENDATIONS FOR
GRAPHQL SECURITY

To help you understand, prevent, and address your GraphQL vulnerabilities, we've compiled the infographic below.

GRAPHQL-SPECIFIC VULNERABILITIES

API Bruteforcing

While batching and aliasing offer convenience by allowing multiple queries in a single HTTP request, they can be exploited to bypass rate limits and even cause servers to crash through the creation of GraphQL bombs.

Denial of Service

Fragments, which function similarly to functions in GraphQL, can create infinite recursion loops if not properly handled, leading to server crashes and can be exploited.

API Schema Leak

Endpoints with disabled introspection can still leak the underlying API schema through field suggestions. Using the OS tool Clairvoyance, anybody can rebuild the full schema.

BEST PRACTICES FOR GRAPHQL SECURITY

1. **Limit Access control with Authorization and Authentication**
   Without the appropriate authorization-check layer, private data and high-access features may be exposed to unauthorized users. Ensure enforcement of authorization and authentication rules through a cleaner approach using resolver middleware.
2. **Input Validation**
   The best way to protect your API from injections is to use input validation for all incoming requests, write custom validators for domain-specific and more complex validations.
3. **Rate Limiting to Block Brute Force Attacks**
   Use the graphql-limit-plugin to set limits on your queries and mutations. Set a longer time window between queries/mutations for highly vulnerable actions (like sign-ins) and a shorter one for less vulnerable actions. This approach limits attackers without affecting legitimate users.
4. **Depth Limiting**
   You can use the **graphql-armor** package to easily limit the depth of queries. First, check how deep you expect queries to be, and then set a maximum depth accordingly.
5. **Schema Whitelisting**
   Limit the exposed schema to only include necessary types and fields, reducing the attack surface. You can use **persistgraphql** by Apollo to auto-generate a list of approved queries at build time.
6. **Limit the Cost of GraphQL Queries**
   You can use the **graphql-armor** package to easily limit the cost of queries. First, make an estimate of your resolvers' cost for your database and third-party services, then implement a hard limit on each query.

USEFUL OPEN SOURCE TOOLS

Defense

GraphQL Armor
The missing GraphQL security security layer for Apollo GraphQL and Yoga / Envelop servers

GraphQL Protect
A dead-simple yet highly customizable security sidecar compatible with any HTTP GraphQL Server or Gateway

GraphQL Shield
A GraphQL tool to ease the creation of permission layer.

Offense

Goctopus
Blazing fast GraphQL discovery & fingerprinting toolbox.

CrackQL
GraphQL password brute-force and fuzzing utility.

BatchQL
GraphQL security auditing script with a focus on performing batch GraphQL queries and mutations.

GraphQL Wordlist
Operations, field names, type names...

LEARNING RESOURCES

GraphQL Security Academy
Hands-on, interactive and open-source lessons that teach various vulnerabilities and best practices in GraphQL security

Damn Vulnerable GraphQL App
An intentionally vulnerable implementation of Facebook's GraphQL technology to learn and practice GraphQL Security.

Escape Security blog
Learn about GraphQL security, performance, testing and building production-ready APIs with lots of hands-on walkthroughs

Acceis GraphQL for pentesters
Introduction to Basic Concepts, Security Considerations & Reconnaissance, Vulnerabilities and Attacks, Offensive Tools

GraphQL Security Vulnerabilities in the Wild
An excellent talk about GraphQL vulnerabilities

escape

Best practices per role

For security engineers

- Take back control of the API attack surface. Build an API inventory.
    - You should be able to answer the following questions:
    - How many APIs does my company expose externally?
    - Which of them are critical to the business?
    - Which of them manipulates sensitive data?
    - Do we have any development endpoints exposed on the internet? Do we have any unnecessarily exposed endpoints?

- Do some GraphQL API threat modeling. “If I was an attacker, how could I use GraphQL APIs to threaten the business?”

- Talk with the developers. Understand how they build their APIs. Work closely with them to ensure they understand the security implications of the features they implement. Promote a security-focused culture that prioritizes API security from the design phase through to deployment. Give examples.

- Implement best practices throughout the API lifecycle:
    - Implement an API Gateway
    - Implement secret scanning, especially on front-end repositories
    - Test every API release for OWASP API Top 10 and business logic flaws. If possible, do it as early as in the CI/CD

This webinar provides an in-depth exploration of risk management for your GraphQL APIs.

Best practices per role

For developers

1. Limit Access control with Authorization and Authentication

Without the appropriate authorization-check layer, private data and high-access features may be exposed to unauthorized users. Ensure enforcement of authorization and authentication rules through a cleaner approach using resolver middleware.

2. Input validation

The best way to protect your API from injections is to use input validation for all incoming requests, write custom validators for domain-specific and more complex validations. graphql-scalars can help

3. Rate limiting to block brute force attacks

Use the graphql-limit-plugin to specify this limit on your queries and mutations. The best way to set it up is to set a large time window between queries/mutations when they are highly vulnerable (like a sign in) and a shorter one for less vulnerable queries/mutations. That way you only limit attackers and not your users.

4. Depth Limiting

You can use the graphql-armor package to easily limit the depth of queries. First, check how deep you expect queries to be, and then set a maximum depth accordingly.

5. Schema Whitelisting

Limit the exposed schema to only include necessary types and fields, reducing the attack surface. You can use persistgraphql by Apollo or graphql-codegen from The Guild to auto-generate a list of approved queries at build time.

6. Limit the Cost of GraphQL Queries

You can use the graphql-armor package to easily limit the cost of queries. First, make an estimate of your resolvers' cost for your database and third-party services, then implement a hard limit on each query.

This webinar provides an in-depth exploration of risk management for your GraphQL.

Conclusion

The analysis of GraphQL APIs reveals significant adoption alongside considerable security challenges, with unrestricted resource consumption and security misconfigurations being prominent issues.

It shows that securing GraphQL APIs requires a detailed understanding of their unique features and potential vulnerabilities. Meanwhile, traditional security tools do not have proper support for the technology yet.

As GraphQL APIs continue to evolve, so too should the strategies used to secure them, ensuring safe and reliable operations.

Organizations can protect their APIs and stay compliant by maintaining a comprehensive API inventory, conducting regular security audits, implementing resolver middleware for authorization and authentication, and using tools to limit query complexity.

But remember, things can get really complex really fast—that's why companies love automated solutions like Escape that evolve with them.