escape

The State of GraphQL Security 2023

What scanning 1500+ endpoints has told us about securing GraphQL in production

Executive Summary

In Q1 2023, the Escape Research team found **46,809 security issues** among **1599 public GraphQL endpoints.**

65% of endpoints were vulnerable to **high or critical severity issues** according to the CVSS ranking system.

Of all the issues found, **38.8% were specific to GraphQL**, showing how this technology comes with its own security challenges.

Also, Sensitive data leaks represented **9.38%** of the findings. In particular, we found **296 remote access tokens** of various kinds leaking through unauthenticated GraphQL API calls.

Finally, **91% of the APIs tested** were vulnerable to easily exploitable **Denial of Service issues.**

This research over the last quarter shows that **organizations are yet to tackle the specificities of GraphQL correctly** when it comes to security, with the technology being especially prone to **leaking sensitive data** and **denial of service.**

Key Take Aways

- **Criticality.** The number of high-risk GraphQL vulnerabilities remains dramatically high, with 65% of endpoints vulnerable to high or critical vulnerabilities. This highlights an endemic risk among GraphQL-based production applications.

- **Best Practices.** 47% of the findings could have been fixed by implementing best practices, showing a need for essential knowledge and proactive security around GraphQL.

- **GraphQL Specificity.** 38.8% 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.

- **Attack Vectors.** Information Disclosure, Lack of Resource Limiting, Denial of Service, and Access Control Issues were the attack vectors causing the most risks. HTTP misconfigurations were the most common issues but not the most critical.

- **Exploitability.** All of the vulnerabilities considered in this report were found using unauthenticated requests on public, external endpoints, highlighting the lack of proper authorization mechanisms in numerous GraphQL APIs.

Introduction

Background

GraphQL, a powerful and flexible query language for APIs, has seen rapid adoption since its inception in 2015. Developed by Facebook and later open-sourced, it has become the go-to choice for modern application development, allowing clients to request exactly what they need from backend services.

However, with its growing popularity, it has become increasingly important for organizations to understand and address this technology's unique security challenges.

Purpose of the Report

By analyzing a large sample of public GraphQL endpoints, we have identified the most pressing security issues faced by organizations utilizing GraphQL for their APIs.

This report aims to provide an overview of the results, highlighting key vulnerabilities, trends, and best practices. It has been written with CISOs, CTOs, VP Engineering, and VP Product Security in mind to help them address these security concerns effectively.

```
type Project {
  name: String
  tagline: String
  contributors: [User]
}
```

```
{
  project(name: "GraphQL") {
    tagline
  }
}
```

```
{
  "project": {
    "tagline": "A query language for APIs"
  }
}
```

Methodology

Our research team enumerated 2,449 Public GraphQL APIs online.

We then used our own proprietary GraphQL Scanning technology to audit those endpoints.

Escape can run more than 60 security tests in fully automated mode, with 40 of them being specific to GraphQL. The complete list of security tests can be found in our public documentation.

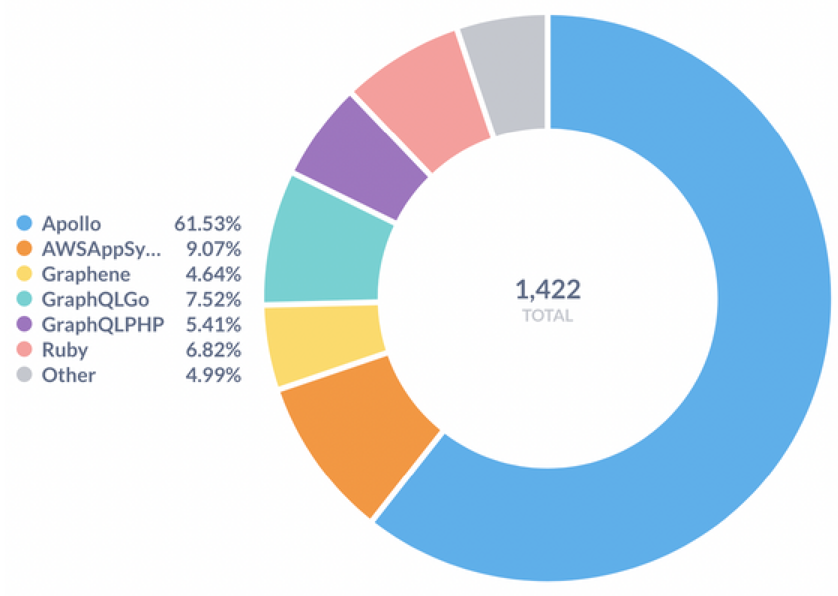
To avoid interfering with the applications we audited, all scans were run in read-only mode, unauthenticated, and with conservative rate limits.

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

Overview of Findings

Our of the 2,499 GraphQL endpoints found, we were able to audit 1588 of them that were fully public. It took us 416 hours of cumulative computing time to complete this report's research.

When analyzing the breakdown of the frameworks powering public GraphQL endpoints, we can observe that Apollo Server stands out as the most used GraphQL engine by powering 61.53% of all public endpoints.

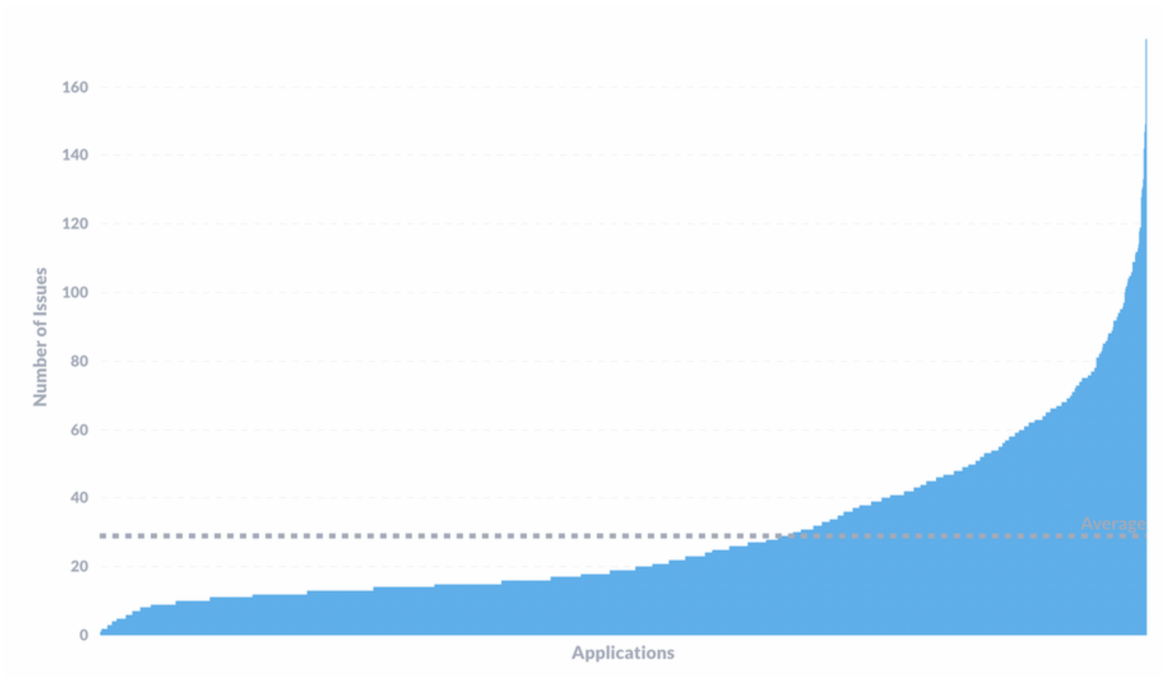


Total Security Issues Found

We highlighted a total of 43,784 security issues among all the scanned applications. On average, each application contained 29 security issues.

Overview of Findings

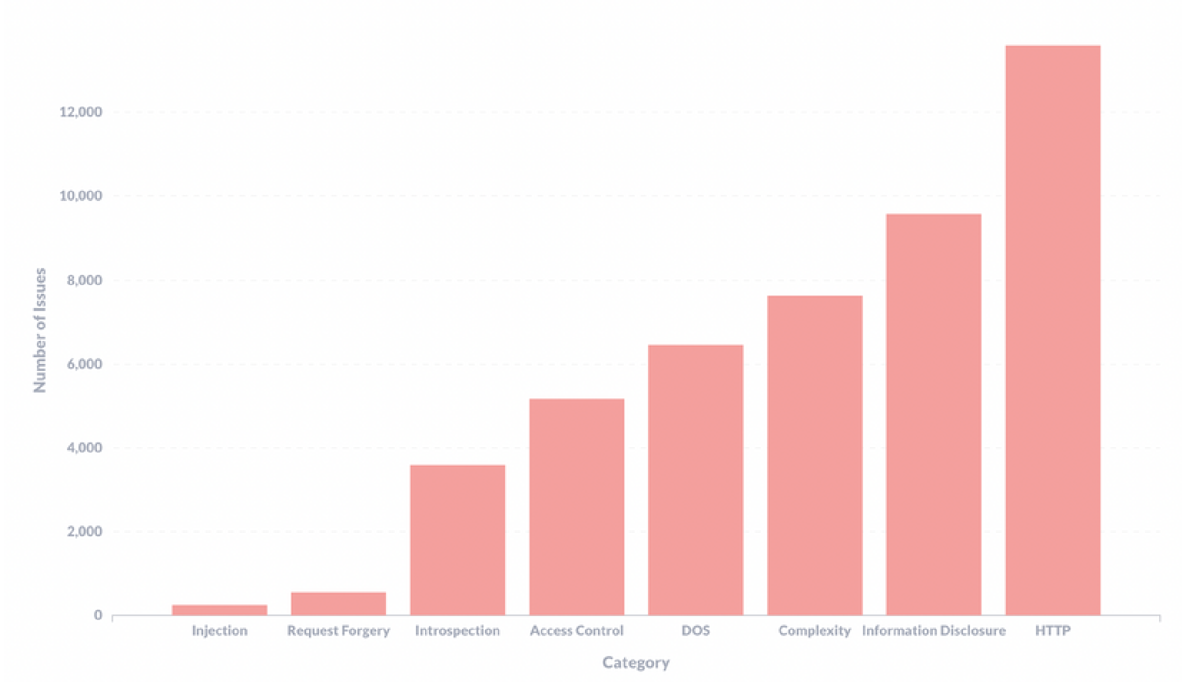
The histogram of the number of alerts per scan shows that most applications are close to the average of 30 alerts. Very few applications contained less than 10 alerts, while a few outsider applications contained way more, up to 174 issues for a single application.



Overview of Findings

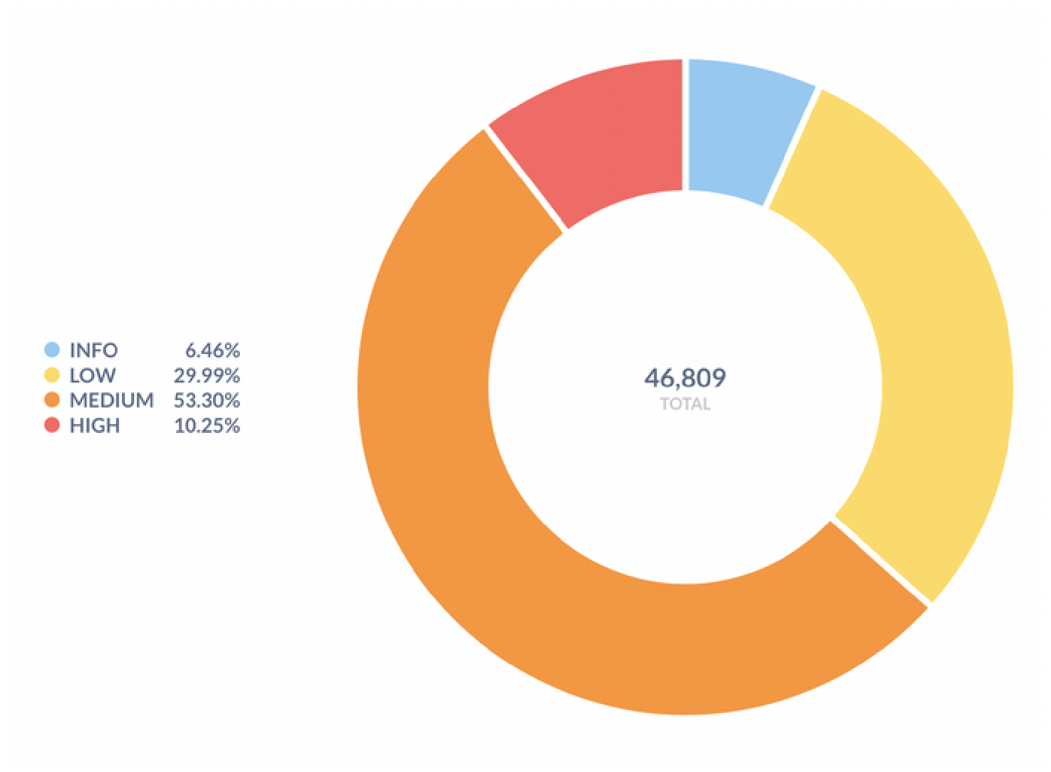
Breaking down the issues per category, we can quickly observe what the common risks for GraphQL APIs are:

- HTTP-level security misconfigurations are the most common issues
- Information Disclosure issues are the second most common, showing that GraphQL engines are prone to revealing too much information to unauthenticated users.
- Then come two categories linked to OWASP API4:2023 (Unrestricted Ressource Consumption): complexity attacks, a GraphQL-specific threat, and Denial of Services.
- Access Control issues are significantly present among our results, accounting for 10,8% of the total issues alone.
- Introspection issues are GraphQL specific and related to API9:2023 (Improper Inventory Management)
- Finally, we found 794 potential injections and requests forgeries, about 2% of the total issues, which is still alarming considering the possible consequences of those issues in modern APIs.



Overview of Findings

Taking a closer look at what matters, we observe that most of those issues are of significant severity. 10.25% of all the issues have a severity of High or above, and 53.30% have a Medium severity.



In-Depth Analysis of Key Vulnerabilities

In this section, we analyse and discuss the key vulnerabilities discovered in public GraphQL APIs by highlighting 3 categories of issues:

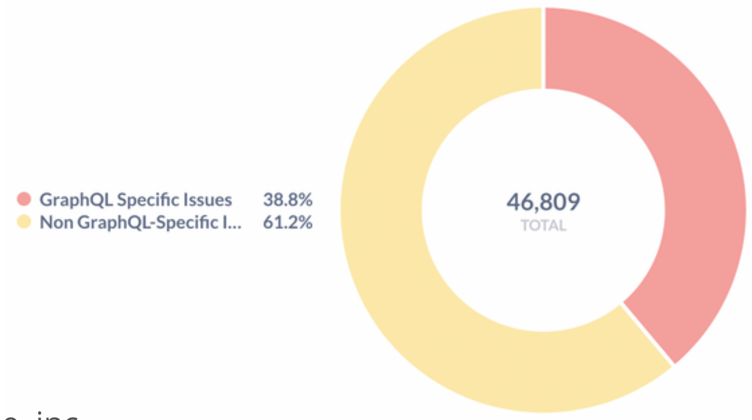
- The issues that are specific to GraphQL
- Denial of Services issues, and why they are overrepresented in GraphQL APIs compared to their REST counterparts
- Sensitive Data Leaks and Access Control issues

GraphQL-specific Issues

Contrarily to REST, GraphQL is a full-featured query language. From a security standpoint, this leaves malicious actors a wide attack surface and various ways to craft queries that abuse the application.

Moreover, parsing GraphQL is more complex than parsing formal API requests.

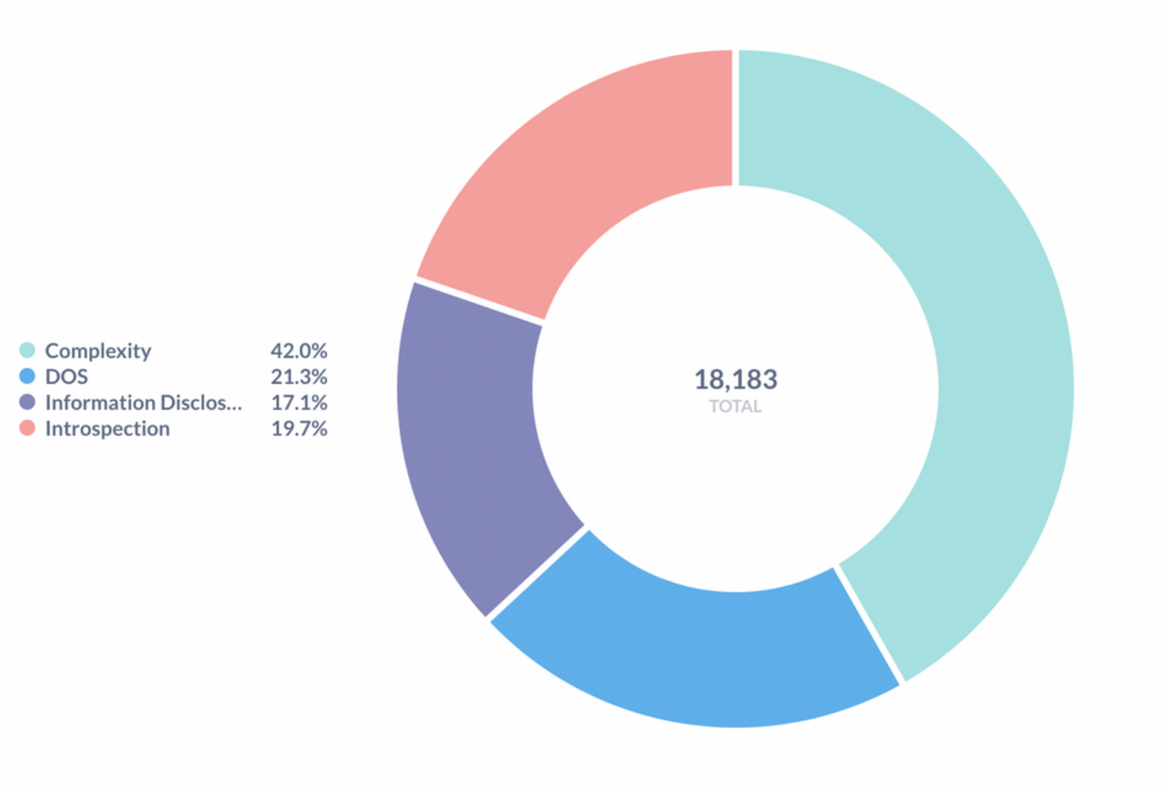
In our research, 38.8% of the issues found were GraphQL-specific. This prominent proportion highlights that, in practice, the specific risk associated with GraphQL still needs to be better handled by engineering and security teams, leaving GraphQL APIs more at stake than their REST counterparts.



In-Depth Analysis of Key Vulnerabilities

When taking a closer look at the GraphQL-specific issues in our results, we can see four majors types of GraphQL-specific attack vectors:

- Complexity attacks, based on abusing the features of the GraphQL language itself
- Denial of Services by exploiting bugs in the GraphQL framework
- Information Disclosure, when the GraphQL framework discloses underlying APIs and sensitive information
- Introspection, when the GraphQL Schema itself is vulnerable



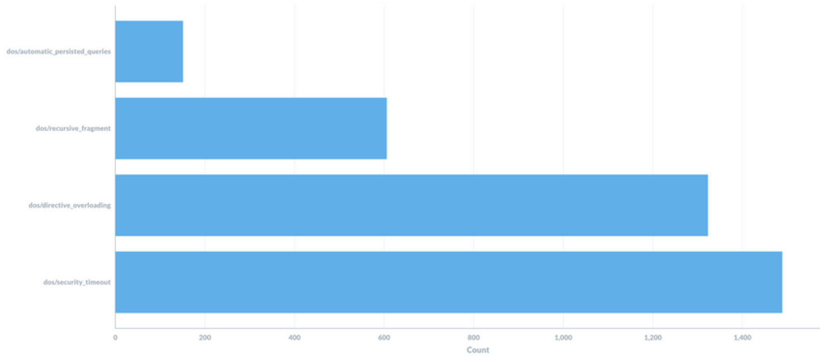
In-Depth Analysis of Key Vulnerabilities

Denial of Services

Most GraphQL implementations are specifically vulnerable to Denial of Service.

This weakness is due to the complexity of putting proper resource usage limits in a full-featured query language. Unlike REST or SOAP, predicting the computation power needed to satisfy a specific GraphQL request in advance is hard. Thus, implementations tend to let clients consume more resources than necessary.

Out of 1,588 endpoints in this study, 1,581 contained some form of denial of service vulnerability. Only 7 GraphQL applications had proper protection against DoS attacks.



More than 600 endpoints are vulnerable to recursive fragments DoS, and more than 1300 to Directive Overloading. Those are two common attacks abusing the GraphQL parser to create easy Denial of Services.

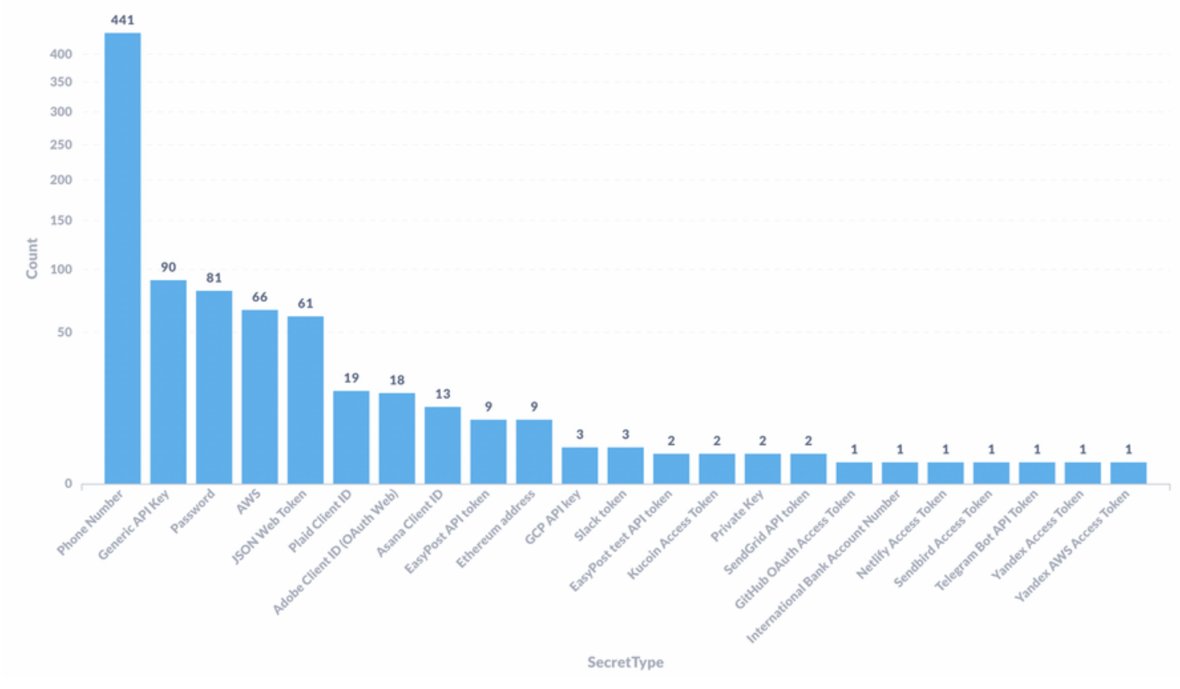
Also, our scanner generated almost 1500 requests that took more than 5 seconds for the endpoint to process, which we defined as a "Security Timeout." While it is not a Denial of Service per se, it shows how GraphQL is also vulnerable to DDoS without proper resource limitations.

In-Depth Analysis of Key Vulnerabilities

Sensitive Data Leaks

Sensitive data leaks represented 9.38% of the findings. In particular, we found 296 secret access tokens of various kinds leaking through unauthenticated GraphQL API calls.

We found 90 Generic API Keys, 81 passwords, 66 AWS Tokens, and 61 JSON Web Tokens.



Those are secrets and should not be publicly accessible. Most of them grant access to the internal system of companies, paving the way for privilege escalation.

The amount of secrets leaked by our tested endpoint dataset is concerning and highlights the lack of proper authorization mechanisms in GraphQL.

Prevention and Mitigation

In order to address the security vulnerabilities and risks outlined in this report, organizations should implement a combination of proactive security measures, best practices, and continuous security assessments. The following recommendations aim to provide a foundation for securing GraphQL APIs against the highlighted threats:

Avoid sensitive data leaks by implementing proper Authentication and Authorization mechanisms

- Implement robust authentication mechanisms to ensure only authorized users can access the API. The present research has been conducted using unauthenticated scanning only, yet we discovered many security issues that could have been mitigated by requiring proper authentication.
- Utilize role-based access control (RBAC) to restrict access to sensitive data and operations based on user roles. In particular, isolation between user accounts must be tested in GraphQL. The technology is prone to creating breaks in tenant isolation.

Mitigate the risk of Denial of Services by implementing GraphQL Rate Limiting and Throttling

- Implement GraphQL-aware rate limiting and throttling to protect against Denial of Service attacks and other endpoint abuse. While technically more complex to implement than in REST, specialized packages like GraphQL Armor have appeared to solve that problem.
- Consider using adaptive rate limiting based on user behavior and legitimate request complexity.

Prevention and Mitigation

3. Train your team and raise awareness on GraphQL specificities

- Educate development and security teams on GraphQL-specific security risks and best practices.
- Encourage a culture of security-aware development and continuous improvement.

4. Ensure the security of the underlying API Schema

- Limit the exposure of sensitive data by carefully designing the GraphQL schema and restricting access to certain fields.
- Validate and sanitize all user inputs to prevent injection attacks and other malicious actions.

5. Monitor your GraphQL APIs

- Regularly monitor and analyze GraphQL API logs to identify potential security issues and respond to incidents quickly.
- Implement real-time alerting to notify security teams of potential threats or suspicious activities.

6. Perform Regular Security Audits and Assessments

- Conduct periodic security audits and vulnerability assessments to proactively identify and address potential risks.
- Use automated security testing tools, like Escape, to continuously evaluate the security posture of your GraphQL APIs.

Organizations can significantly reduce the likelihood of security incidents involving their GraphQL APIs by adopting these prevention and mitigation strategies. Treating security as an ongoing process and continuously adapting to new threats and vulnerabilities as they emerge is essential.

Conclusion

Conclusion

GraphQL's adoption is snowballing among organizations of all sizes, especially in the enterprise. Yet, it comes with new types of cyber risks.

Security professionals and GraphQL developers are unaware of the specific threats to GraphQL. Meanwhile, traditional security tools do not have proper support for the technology yet.

Consequentially, the risk landscape for GraphQL is alarming.

We expect organizations that adopted GraphQL to evolve their security practices and tooling to adapt to the specificities and unique challenges of this technology in the near future.

About Escape

Escape is the GraphQL Security company. We are on a journey to help organizations migrate to GraphQL securely by providing them with the proper knowledge and tooling.

To learn more about Escape and GraphQL Security, visit our website:

https://escape.tech

Escape is a proud member of the GraphQL foundation board.

GraphQL Foundation

escape

escape

escape.tech
ping@escape.tech
+1 (707) 615 6448