



|



# GraphQL Vulnerabilities in the Wild

What scanning real-life GraphQL apps told us  
about exploiting GraphQL in Production



# Table of content

- 01 About Escape
- 02 Why is GraphQL vulnerable?
- 03 Why is it hard to test production GraphQL apps?
- 04 The findings
- 05 Sensitive data made public!



# About Escape.tech

First automated Security Testing platform (DAST) able to test the business logic of APIs and GraphQL.

The screenshot displays the Escape Security Testing platform's user interface. On the left is a dark sidebar with navigation links: 'escape v2 Beta', '+ Invite Team Members', 'Applications', 'Reporting New', 'CI/CD New', and 'Integrations'. Below these are 'Release notes', 'Documentation', 'Support', 'Community', and a 'Steve Jobs' profile section. The main content area is titled 'Escape Security Testing' and shows the 'Overview' for the 'Pied Piper GraphQL App' at <https://piedpiper.escape.tech/graphql>. A large orange circle indicates a 'Your score' of 42%. The interface includes four cards: 'API Coverage' (95%, 16/16 queries, 6/7 mutations), 'API Health' (42% +5pp), 'Issues' (15, -3), and 'Median Response Time' (93ms, -3%). Below these are sections for 'Operation', 'PII', 'Status', 'P50', and 'Security Alerts', listing various GraphQL queries and their results. The bottom of the page features a teal footer bar with the 'escape' logo and the 'B-SIDES MILANO' logo.

1599

production APIs  
scanned

**1599**

production APIs  
scanned

**416**

cumulative scan  
hours

**1599**

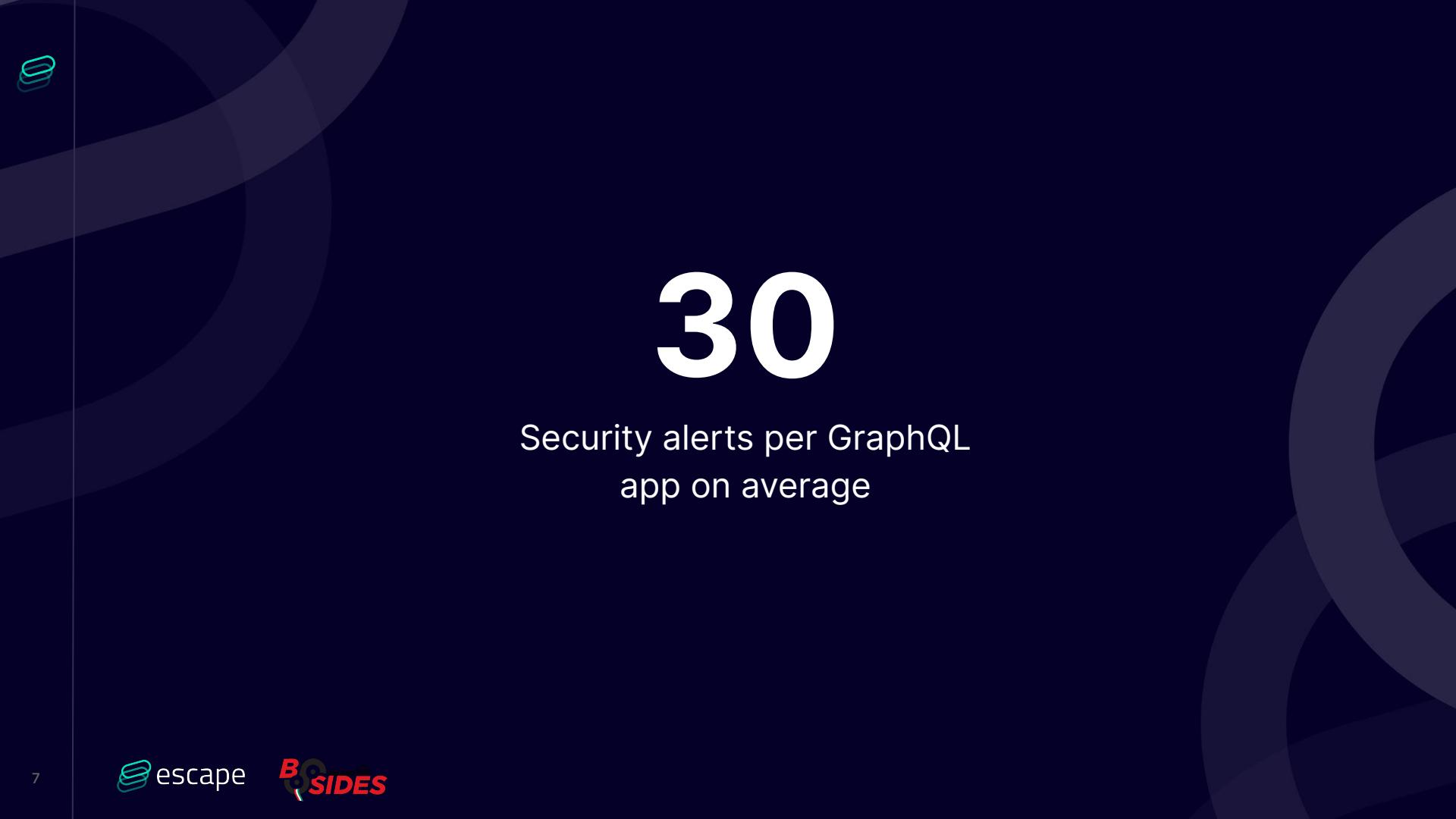
production APIs  
scanned

**416**

cumulative scan  
hours

**46,809**

security alerts  
found



# 30

Security alerts per GraphQL  
app on average



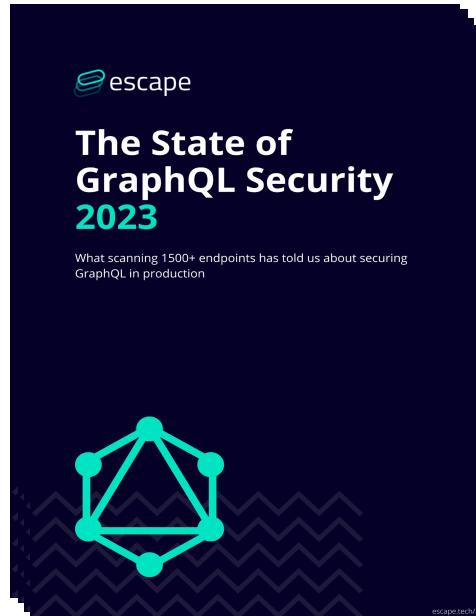
**Tristan Kalos**  
Co-founder & CEO

✉️ [tristan@escape.tech](mailto:tristan@escape.tech)  
👉 [@tristankalos](https://twitter.com/tristankalos)

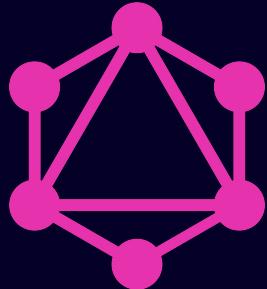


**Antoine Carossio**  
Co-founder & CTO

✉️ [antoine@escape.tech](mailto:antoine@escape.tech)  
👉 [@iCarossio](https://twitter.com/iCarossio)



<https://escape.tech/resources/state-of-graphql-security-2023/>



# Why is GraphQL vulnerable?



# GraphQL is a query language for APIs



GraphQL

Describe your data

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

Ask for what you want

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

Get predictable results

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

 PELOTON

 NETFLIX

 The New York Times

 Walmart

 PayPal



 nerdwallet

 expedia group

 glassdoor

 priceline

 escape

 B-SIDES MILANO



# Why is GraphQL vulnerable?

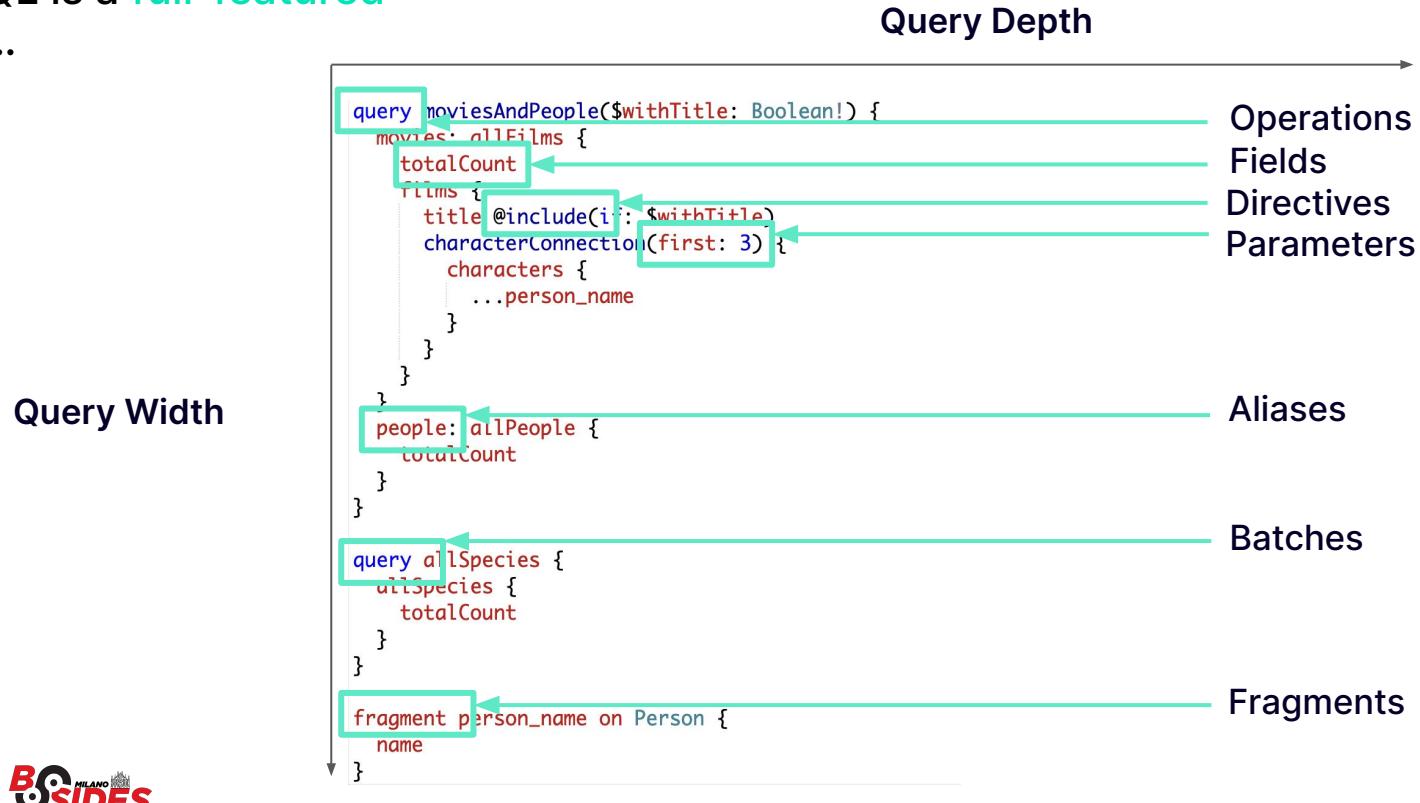
1 – GraphQL is a **full-featured language**...

```
query moviesAndPeople($withTitle: Boolean!) {  
  movies: allFilms {  
    totalCount  
    films {  
      title @include(if: $withTitle)  
      characterConnection(first: 3) {  
        characters {  
          ...person_name  
        }  
      }  
    }  
  }  
  people: allPeople {  
    totalCount  
  }  
}  
  
query allSpecies {  
  allSpecies {  
    totalCount  
  }  
}  
  
fragment person_name on Person {  
  name  
}
```



# Why is GraphQL vulnerable?

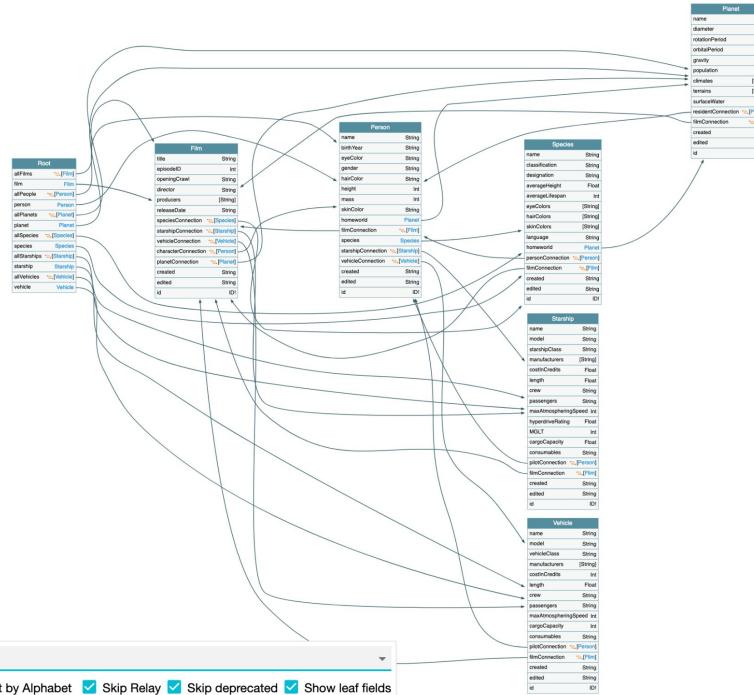
1 – GraphQL is a **full-featured language**...





# Why is GraphQL vulnerable?

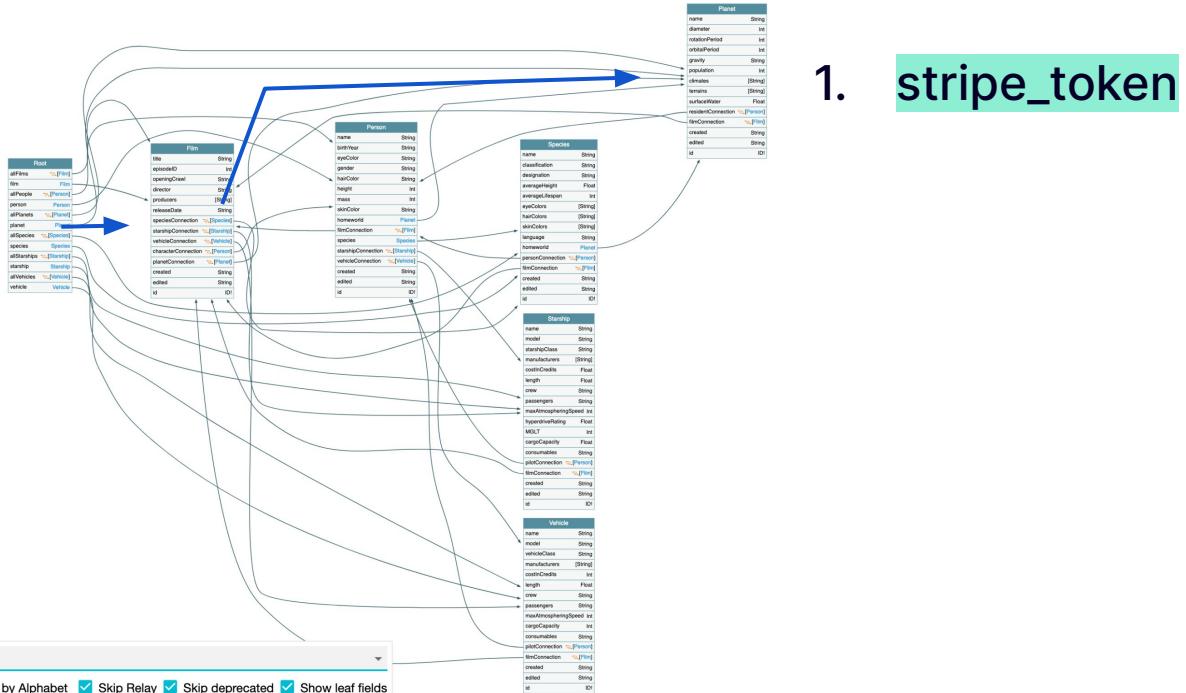
2 - GraphQL is a graph... with a lot of different paths leading to the same resource





# Why is GraphQL vulnerable?

2 - GraphQL is a graph... with a lot of **different paths** leading to **the same resource**

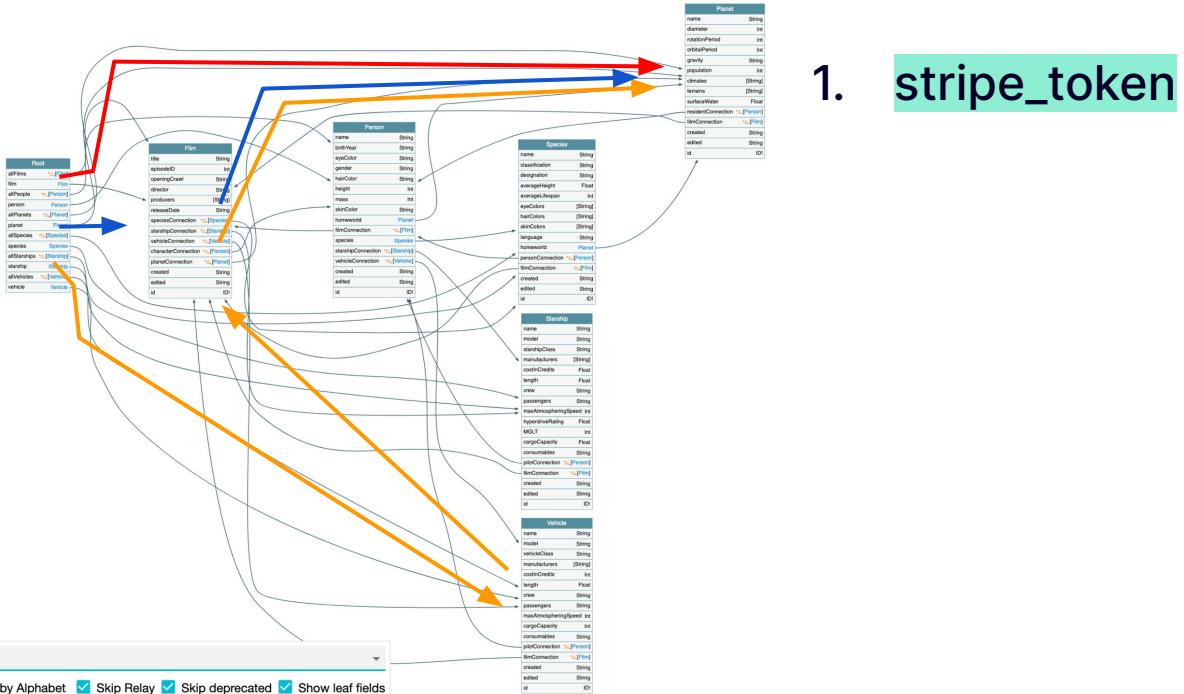


1. stripe\_token



# Why is GraphQL vulnerable?

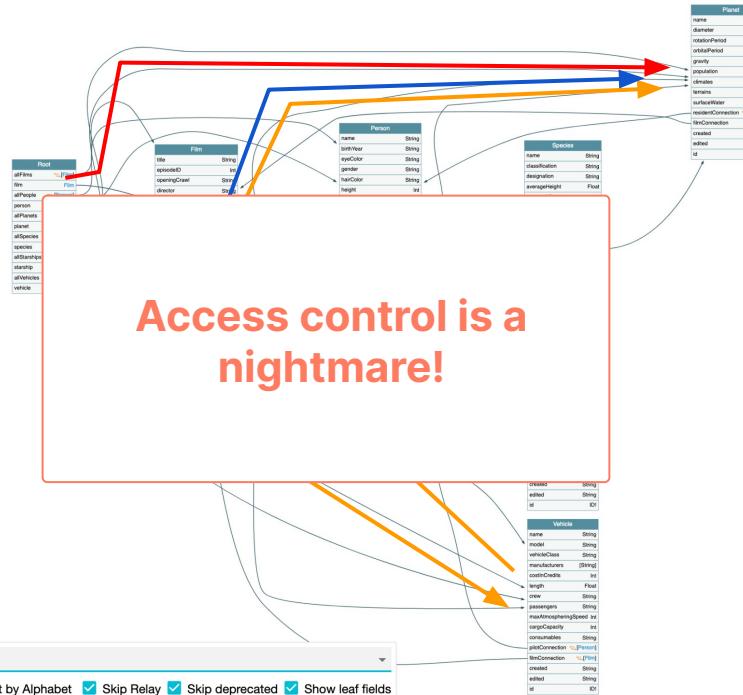
2 - GraphQL is a graph... with a lot of **different paths** leading to the **same resource**





# Why is GraphQL vulnerable?

2 - GraphQL is a graph... with a lot of different paths leading to the same resource



## 1. stripe\_token

# Access control is a nightmare!

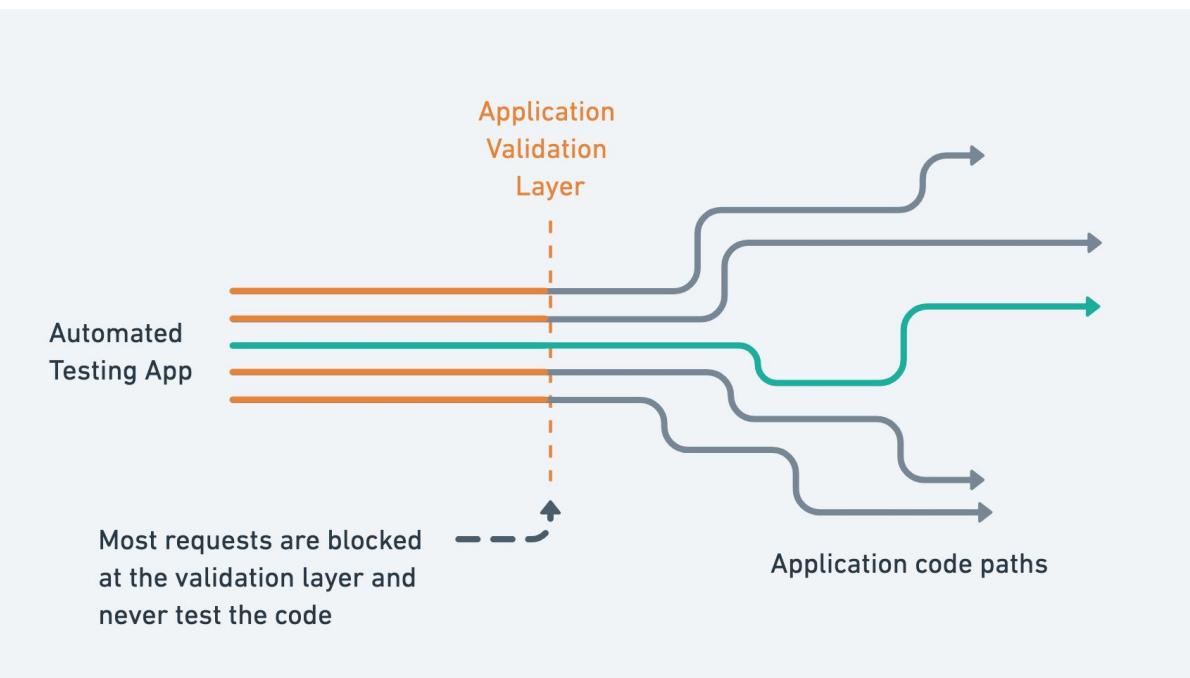


Why is it hard to test  
production GraphQL apps?



# The two problems when testing GraphQL APIs

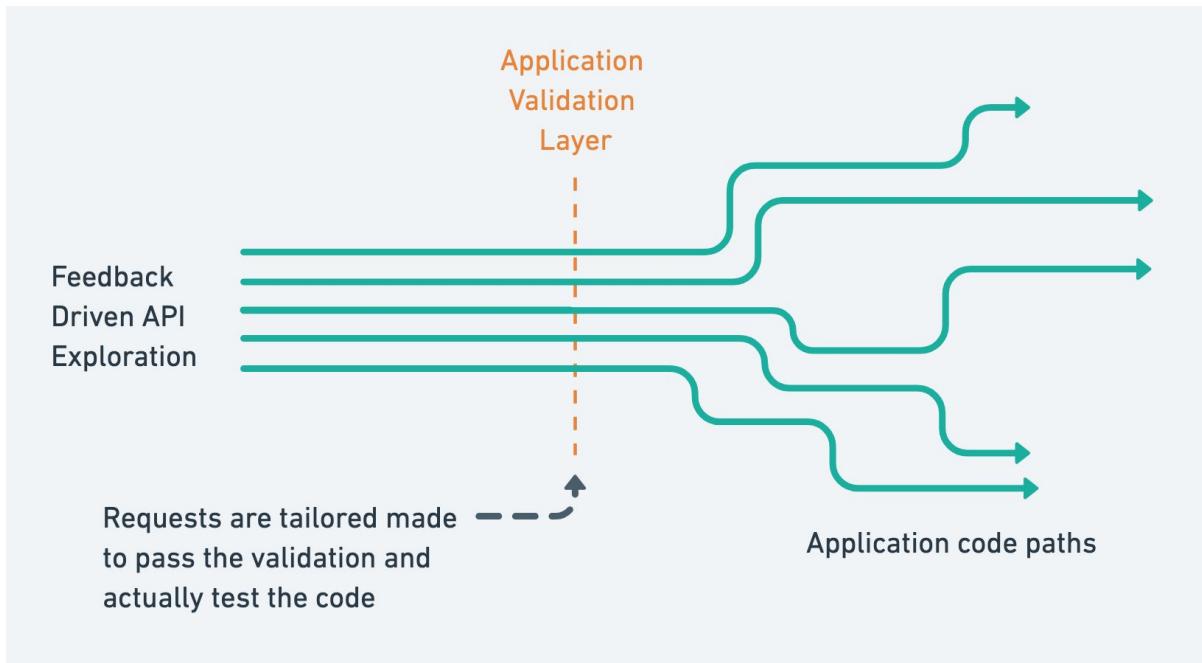
**Problem 1:** Classic testing cannot assess the business logic of APIs





# The two problems when testing GraphQL APIs

## Solution 1: Feedback-Driven API Exploration





# The two problems when testing GraphQL APIs

## Solution 1: Feedback-Driven API Exploration

```
# Write your query or mutation here
mutation {
  editUser(
    userId: "65556.4"
    email: "hello world"
    password: "; or 1=1"
  )
  userName
  email
}
}
```

Incoherent data

Regular API Testing

```
# Write your query or mutation here
mutation {
  editUser(
    userId: "5d139c40-adf0-438b-8063-23a616815ea5"
    email: "john.silverstein@escape.tech"
    password: "Sast<Dast!; or 1=1"
  )
  userName
  email
}
}
```

Coherent data

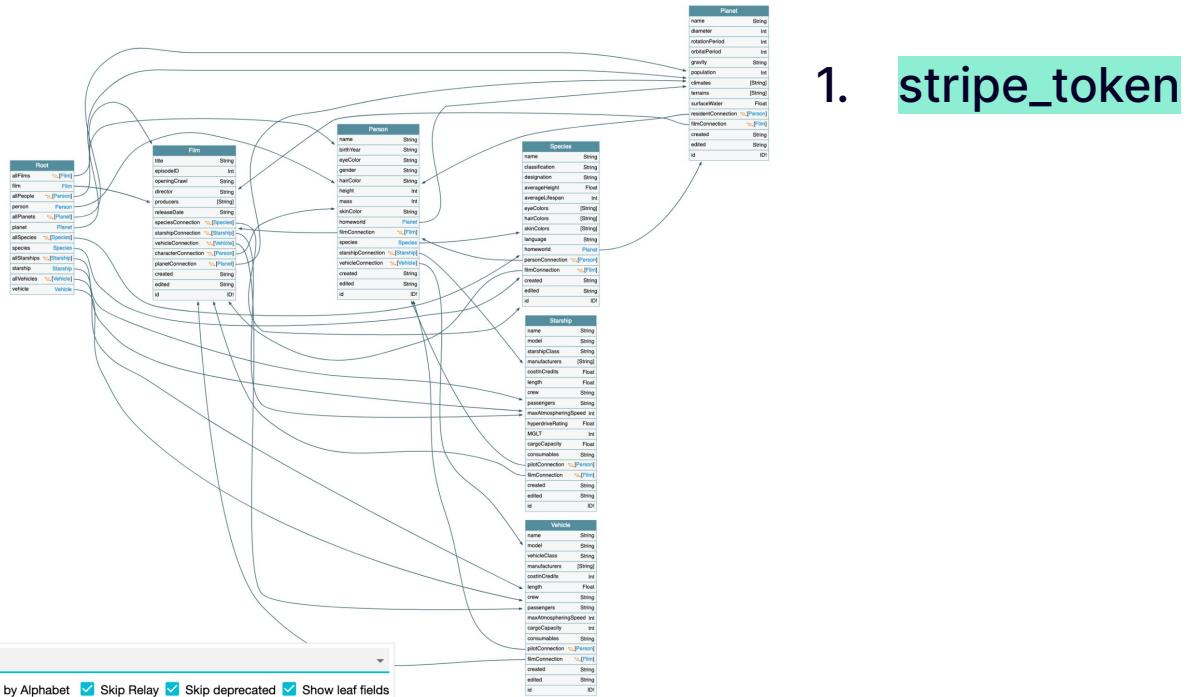
Real data previously  
found in API

Feedback Driven API Exploration



## The **two problems** when testing GraphQL APIs

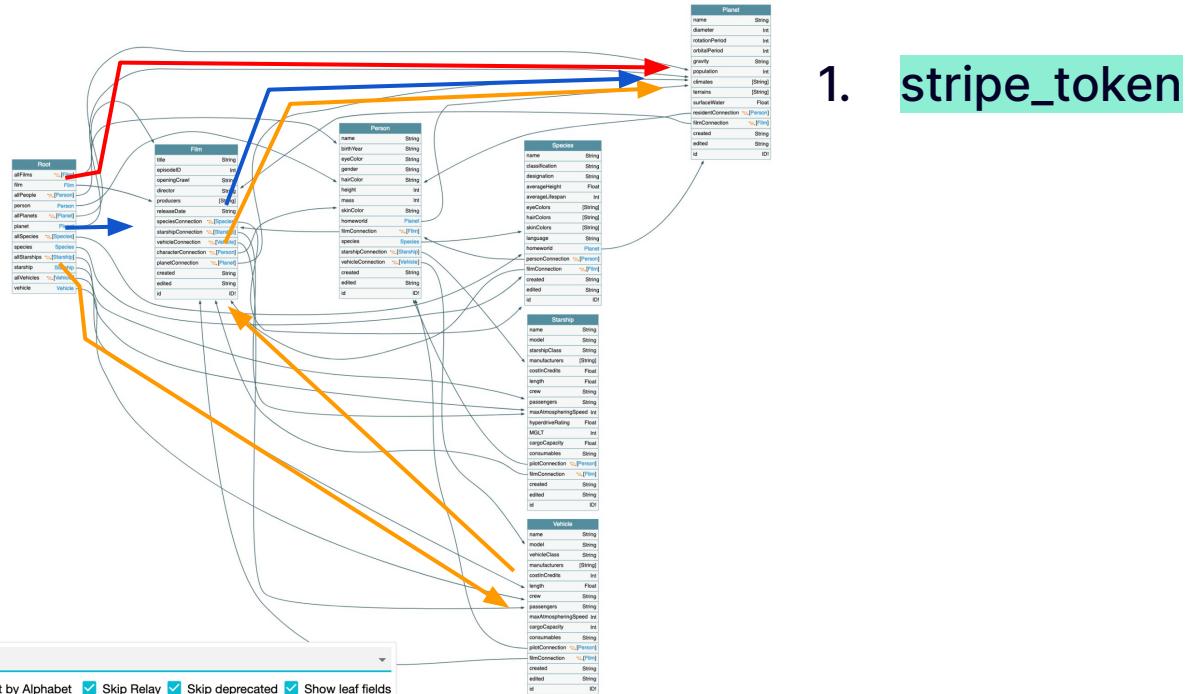
## Problem 2 : GraphQL is a Graph...





# The two problems when testing GraphQL APIs

Solution 2 : Recursively explore all paths in the graph





# The findings

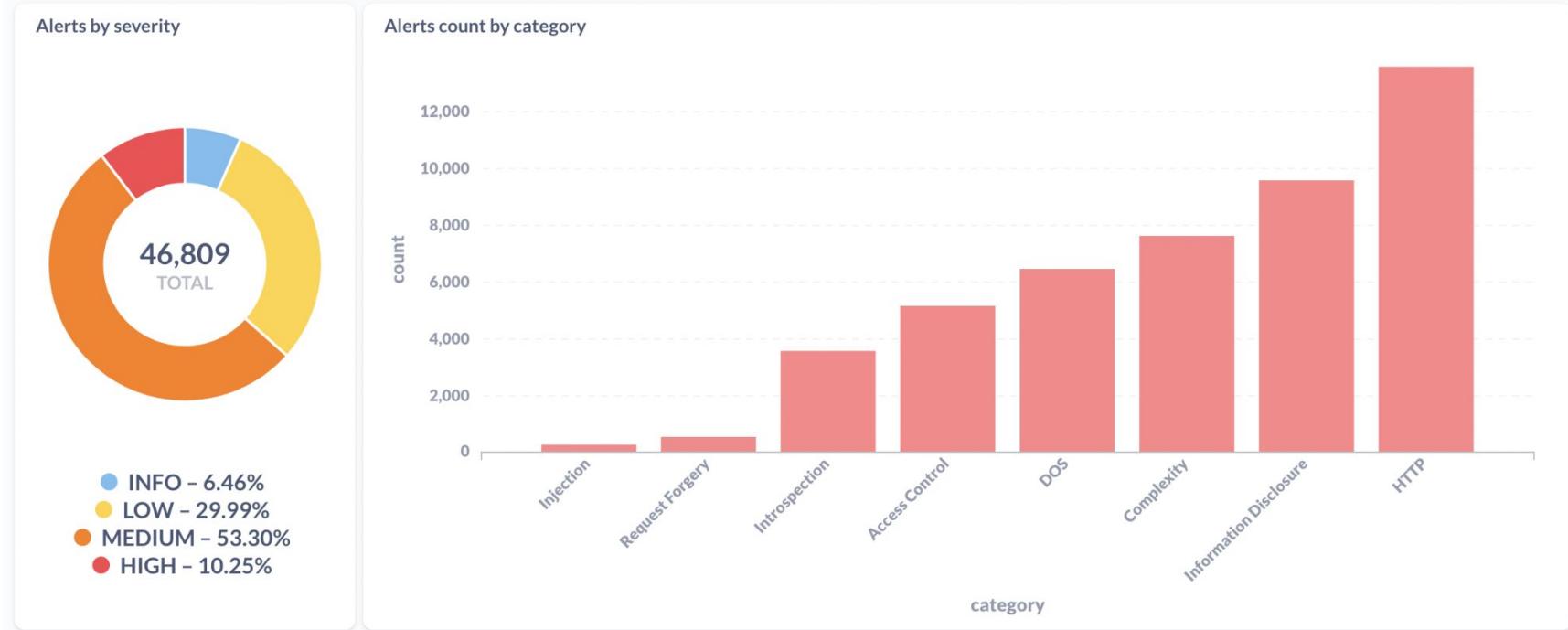


46,809

Security alerts found

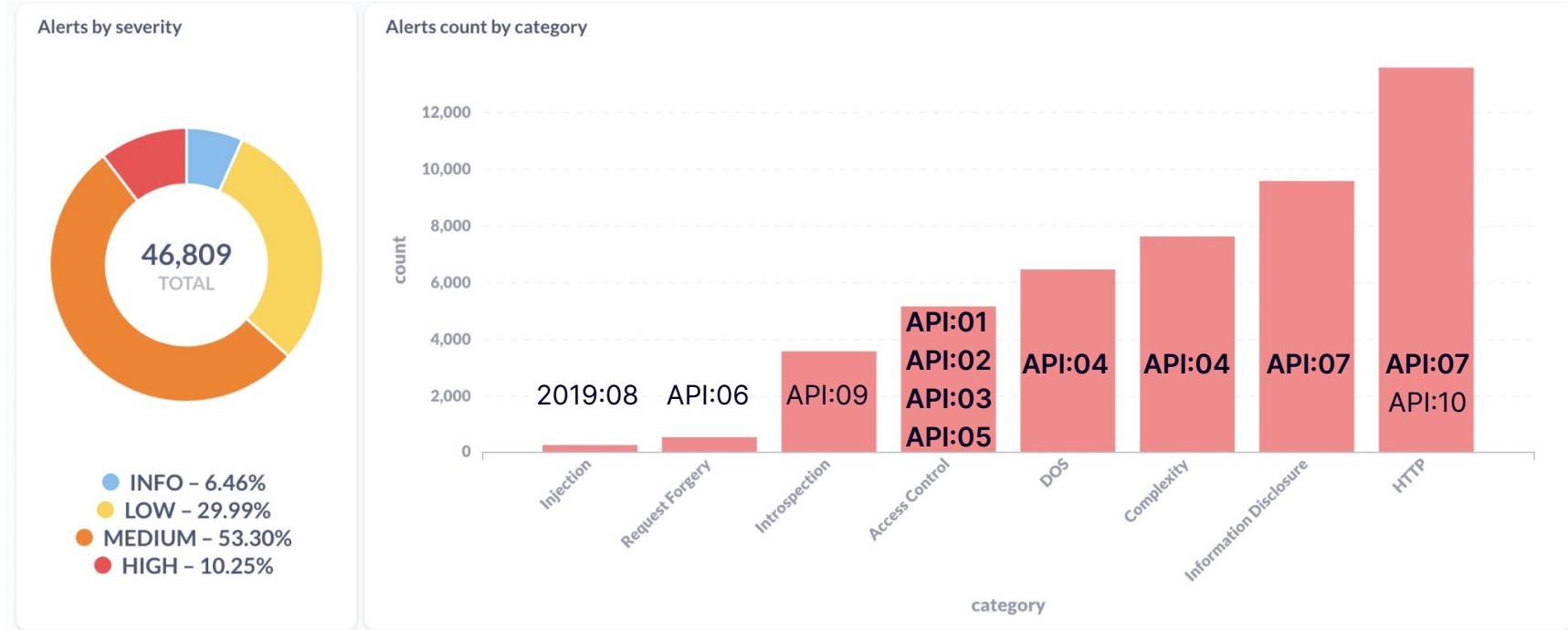


# Which severity and category?



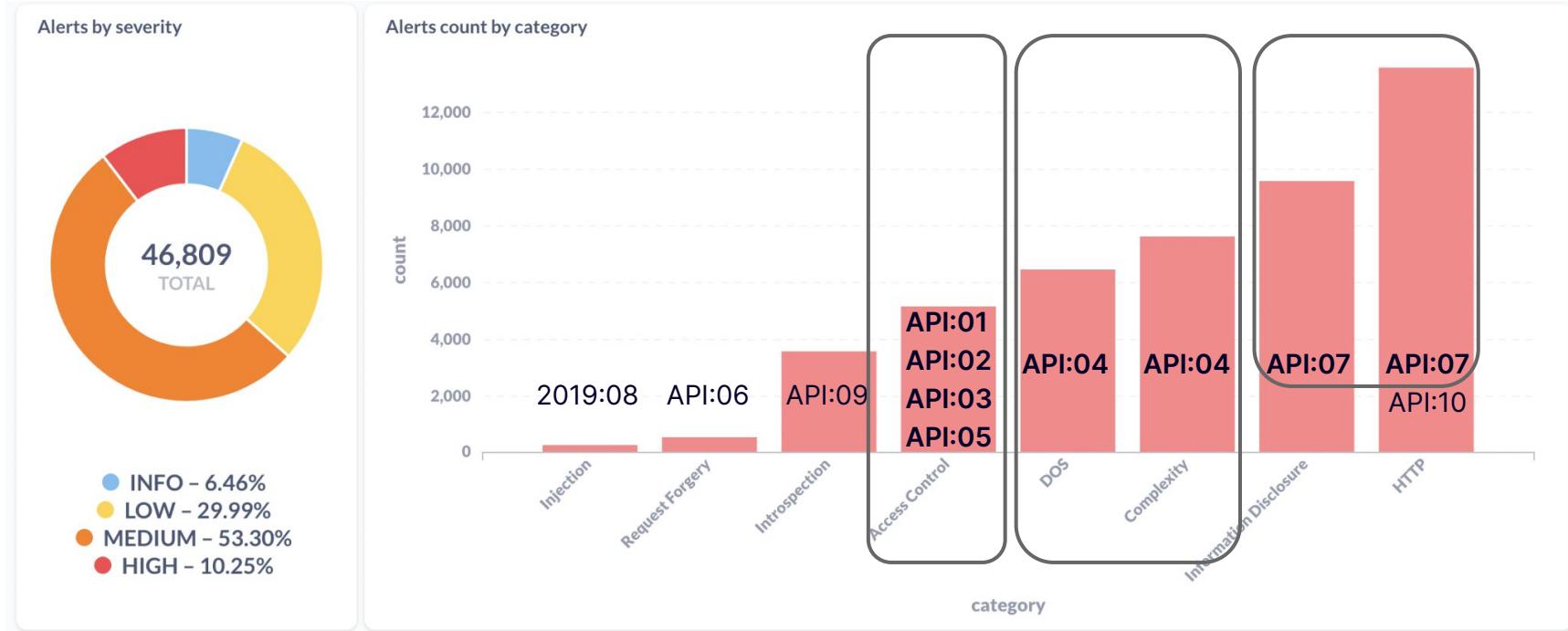


# Which severity and category?





# Which severity and category?

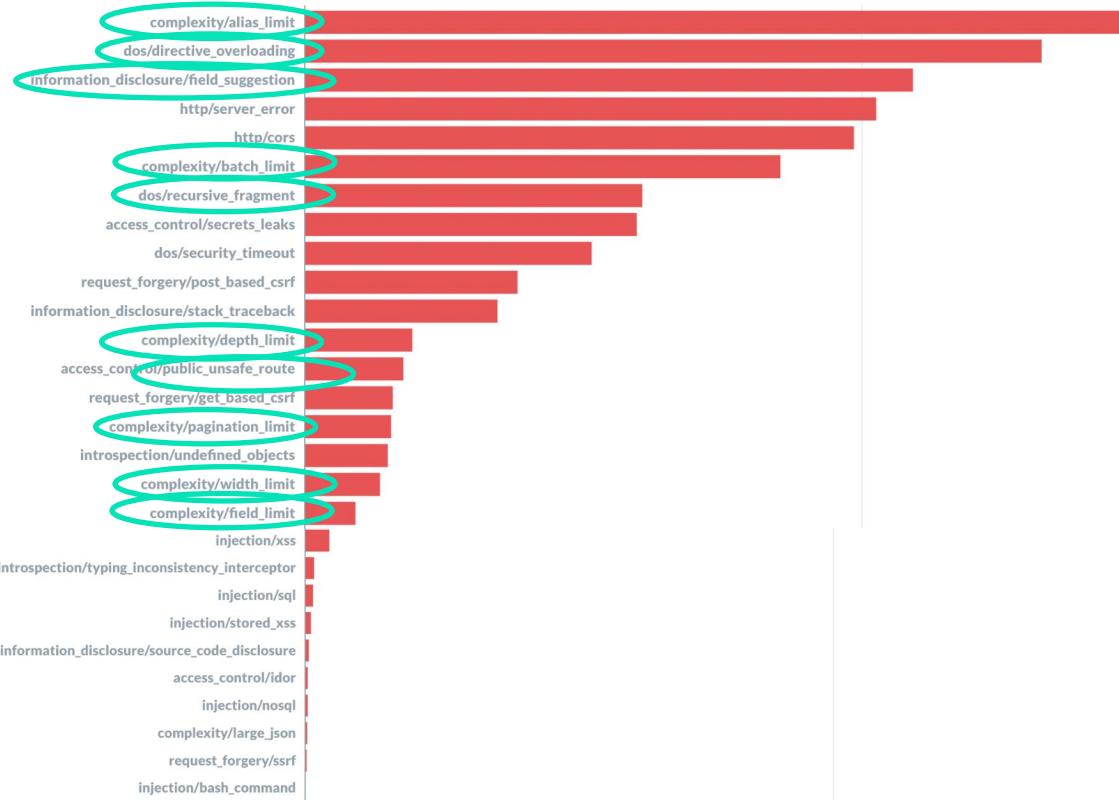




# A lot are GraphQL-specific



# GraphQL-specific Vulnerabilities come first





# Top vulnerability 1: Bruteforcing API requests (API:04)

**Batching:** Multiple GraphQL Queries in one HTTP Request

```
query Hero {  
    hero {  
        name  
    }  
}  
  
query Vilain {  
    villain {  
        name  
    }  
}
```

**Aliasing:** The same fields multiple times with aliases

```
{  
    empireHero: hero(episode: EMPIRE) {  
        name  
    }  
    jediHero: hero(episode: JEDI) {  
        name  
    }  
}
```



# Top vulnerability 1: Bruteforcing API requests (API:04)

The problem: can be used to bypass rate limiting on login mutations = Bruteforce attack

```
{  
    login1: login(email: user0@graphql.org, password: "password1") {  
        username  
        jwt  
    }  
    login2: login(email: user0@graphql.org, password: "password22") {  
        username  
        jwt  
    }  
    login3: login(email: user0@graphql.org, password: "password2211") {  
        username  
        jwt  
    }  
}  
...
```



## Top vulnerability 2: Denial of Service (API:04)

Fragments: a piece of logic that can be shared between multiple queries

```
1 fragment NameParts on Person {  
2   firstName  
3   lastName  
4 }
```



# Top vulnerability 2: Denial of Service (API:04)

The problem: What if a fragment calls itself? Oops, you get an infinite recursion loop.

```
{  
    ...X  
}  
  
fragment X on Query {  
    ...Y  
} →  
  
fragment Y on Query {  
    ...X  
}  
  
{  
    "errors": [  
        {  
            "message": "Maximum call stack size exceeded",  
            "extensions": {  
                "code": "INTERNAL_SERVER_ERROR",  
                "exception": {  
                    "stacktrace": [  
                        "RangeError: Maximum call stack size exceeded",  
                        "    at Map.get (<anonymous>)",  
                        "    at getReferencedFieldsAndFragmentNames (/usr/si  
/app/node_modules/graphql/validation/rules  
/OverlappingFieldsCanBeMergedRule.js:468:45)",  
                    ]  
                }  
            }  
        ]  
    }  
}
```



# Top vulnerability 3: Internal API Schema Leak (API:07)

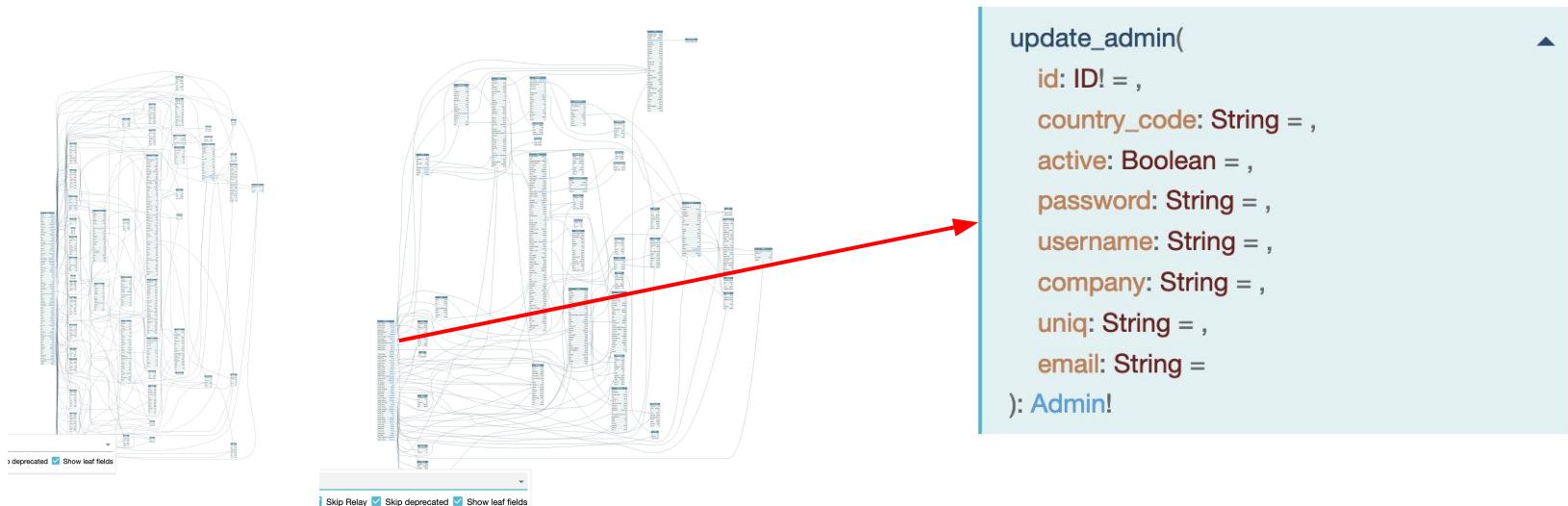
The problem: Endpoints with disabled introspection still leaks underlying schema through field suggestion using

```
{  
  "errors": [  
    {  
      "message": "Cannot query field \"createSesion\" on  
                 type \"RootMutation\". Did you mean  
                 \"createSession\", \"createUser\",  
                 \"createFile\", or \"createImage\"?",  
      "category": "graphql"  
    }  
  ]  
}
```



# Top vulnerability 3: Internal API Schema Leak (API:07)

**The problem:** Endpoints with disabled introspection still leaks underlying API Schema through field suggestion using open source tool [Clairvoyance](#), anybody can build back the full Schema.





# Classic API vulnerabilities are still there



# HTTP Errors, Access Control Issues, Injections





# API Errors and Stacktraces disclosing internal data (API:07)

**The problem:** Stack Traces give too much information on what's vulnerable in your app

```
"extensions": {  
    "code": "GRAPHQL_VALIDATION_FAILED",  
    "exception": {  
        "stacktrace": [  
            "GraphQLError: Boolean cannot represent a non  
boolean value: \"'\"",  
            "    at GraphQLScalarType.parseLiteral  
(/usr/local/app/common/temp/node_modules/.pnpm/@apollo  
/gateway@0.6.1/node_modules/graphql  
/type/scalars.js:271:13)",  
            "    at isValidValueNode (/usr/local/app/common  
/temp/node_modules/.pnpm/graphql@16.5.0/node_modules
```

## Prototype Pollution In

### @Apollo/gateway

HIGH 7.3

Updated at: 11  
months ago

Published: 3  
years ago

Package: @apollo/gateway (npm)

Affected Version: < 0.6.2  
Patch Version: 0.6.2

(12% endpoints misconfigured)



# Access Control issues (API:01,02,03,05)

**The problem:** Most routes writing data (mutations) are not supposed to be accessible without authentication

Access Control    Unsafe route is public M

✖ Ignore ^

**Description**  
A route that mutate application data should not be public.

**Remediation**  
Restrict access to the route, using an authentication middleware for example.

**Details & reproduction (1 related alert)**

Mutation updatePayment

A mutation request has succeeded without authentication.

✖ Ignore More



## Injections are still there (API 2019:08)

- 16 SQL or NoSQL injections
  - 1 bash command injection
  - 3 Server Side Request Forgery

**H** Injection / NoSQL injection detected

**Reproduction**

---

**Request** 21.0ms | **5/16/23, 5:02 AM** **Copy as curl**

▼ Headers (2)

```
X-Escape-Request-ID: 9975315d02c74675aa977b6560a3ba83
Content-Type: application/json
```

▼ Body (2 lines)

```
query graphql {
    printCurrencyUsed(user_id: ">")
}
```

---

**Response** 200

▼ Headers (5)

```
Date: Tue, 16 May 2023 09:02:41 GMT
Content-Type: application/json
Content-Length: 1088
Connection: keep-alive
Strict-Transport-Security: max-age=15724800; includeSubDomains
```

▼ Body (39 lines)

```
{
  "data": {
    "printCurrencyUsed": null
  },
  "errors": [
    {
      "message": "Uncaught
MongoDB\\Driver\\Exception\\CommandException: unknown operator",
      "locations": [
        {
          "line": 1,
          "column": 1
        }
      ],
      "path": [
        "printCurrencyUsed"
      ]
    }
  ]
}
```



# Sensitive data made public!

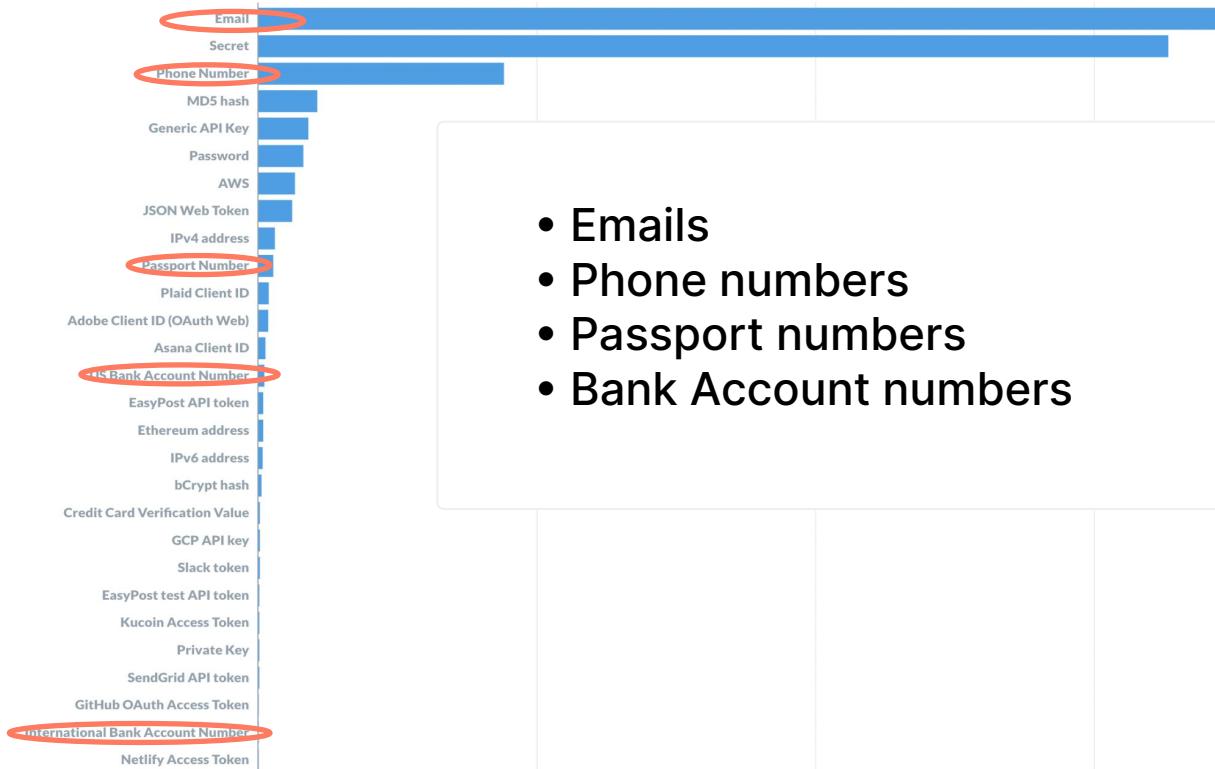


# 4,493 potential PII and token leaks found.

Email	EasyPost API token
Secret	Ethereum address
Phone Number	IPv6 address
MD5 hash	bCrypt hash
Generic API Key	Credit Card Verification Value
Password	GCP API key
AWS	Slack token
JSON Web Token	EasyPost test API token
IPv4 address	Kucoin Access Token
Passport Number	Private Key
Plaid Client ID	SendGrid API token
Adobe Client ID (OAuth Web)	GitHub OAuth Access Token
Asana Client ID	International Bank Account Number
US Bank Account Number	Netlify Access Token

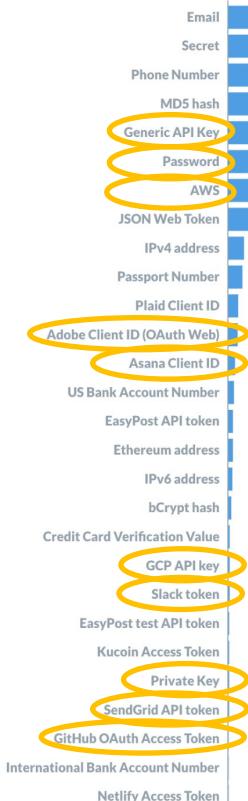


# 4,493 potential PII and token leaks found.





# 4,493 potential PII and token leaks found.



- API Keys: AWS, GCP
- Private RSA Keys
- Auth Tokens: Adobe, Asana, Github...



# How does OWASP TOP 10 reflect in prod-ready GraphQL apps?

**DOS & Complexity** issues

*Due to GraphQL-specific features*



**API:04:** Unrestricted Resource Consumption



# How does OWASP TOP 10 reflect in prod-ready GraphQL apps?

## DOS & Complexity issues

*Due to GraphQL-specific features*



**API:04:** Unrestricted Resource Consumption

## Access Control, Authentication and Authorization:

*Very common because of the graph nature of GraphQL*



**API:01:** Broken Object Level Authorization (BOLA)

**API:02:** Broken Authentication

**API:03:** Broken Object Property Level Authorization (BOPLA)

**API:05:** Broken Function Level Authorization (BFLA)



# How does OWASP TOP 10 reflect in prod-ready GraphQL apps?

## DOS & Complexity issues

*Due to GraphQL-specific features*



**API:04:** Unrestricted Resource Consumption

## Access Control, Authentication and Authorization:

*Very common because of the graph nature of GraphQL*



**API:01:** Broken Object Level Authorization (BOLA)

**API:02:** Broken Authentication

**API:03:** Broken Object Property Level Authorization (BOPLA)

**API:05:** Broken Function Level Authorization (BFLA)

## Security Misconfiguration and Information Disclosure

*By default in GraphQL Server Engines*



**API:07:** Security Misconfiguration

# Conclusion, using only unauthenticated API calls:

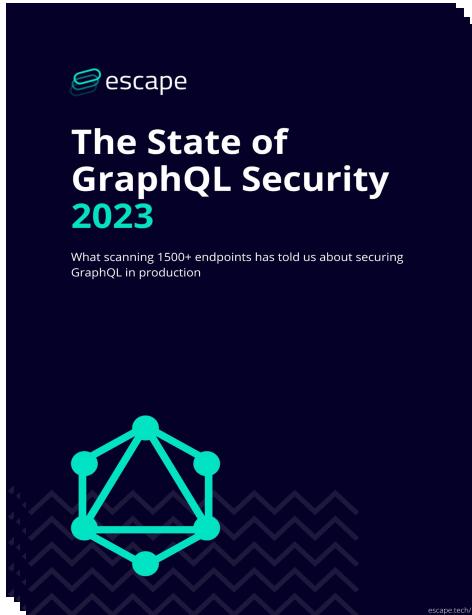
- We found 130k+ public GraphQL APIs into the wild and scanned **1599** of them
- Highlighted numerous vulnerabilities, most frequently GraphQL Specific, but not only
- **Access control flaws** and **secret leaks** are \*very\* frequent in GraphQL



**Tristan Kalos**  
Co-founder & CEO  
✉️ [tristan@escape.tech](mailto:tristan@escape.tech)  
👉 [@tristankalos](https://twitter.com/tristankalos)



**Antoine Carossio**  
Co-founder & CTO  
✉️ [antoine@escape.tech](mailto:antoine@escape.tech)  
👉 [@iCarossio](https://twitter.com/iCarossio)



<https://escape.tech/resources/state-of-graphql-security-2023/>



# Two more things...

# GraphQL Armor

The missing free & open source security middleware for GraphQL applications



Provides  
out-of-the-box  
security best  
practices:

- [Aliases Limit](#)
- [Character Limit](#)
- [Cost Limit](#)
- [Depth Limit](#)
- [Directives Limit](#)
- [Field Suggestion](#)

[escape.tech/graphql-armor](https://escape.tech/graphql-armor)



# Escape learning center

The first free & open-source security learning platform dedicated to **GraphQL** developers.

escape.tech/learn **Beta** 0 points

**Introduction**  
These lessons cover common security concerns with simple solutions.

	<b>AUTHORIZATION</b> <b>Broken Object-Level Authorization</b> Learn how to properly setup object-level authorization in GraphQL. OWASP API1:2023	10 points
	<b>DOS</b> <b>Unrestricted Resource Consumption</b> Learn how to block expensive queries with GraphQL Armor, preventing denial-of-service (DoS) attacks. OWASP API4:2023	10 points
	<b>CONFIGURATION</b> <b>Disable Debug Mode</b> Learn how to disable debug mode in production, to prevent information disclosure. OWASP API7:2023	10 points
	<b>INJECTION</b> <b>SQL Injection</b> While less common nowadays, SQL injections are still a threat. Learn how to exploit and prevent them. OWASP API8:2019	10 points

[escape.tech/learn](https://escape.tech/learn)



We are hiring  
to make DAST cool again.

[jobs.escape.tech/](https://jobs.escape.tech/)



|



Thank you



We are hiring

to bring cybersecurity into all developers' hands.

[jobs.escape.tech/](https://jobs.escape.tech/)