[GraphQL Security](https://confluence.marketintelligence.spglobal.com/display/TPM/GraphQL+Security)

* Created by Unknown User (kiran.sahoo2@spglobal.com), last modified on [Feb 27, 2023](https://confluence.marketintelligence.spglobal.com/pages/diffpagesbyversion.action?pageId=138641726&selectedPageVersions=31&selectedPageVersions=32)

Purpose of this doc is to help defend graph apis by designing the api right and building automated security testing in the pipeline.

Github Link- <https://dev.azure.com/spglobal/SNLServiceModules/_git/graphql-poc?path=/graphqlsecurity&version=GBusers/kiran.sahoo2/graphqlauth>

**GraphQL Attack Surface**

 Different Parts of GraphQL

|  |  |  |
| --- | --- | --- |
| 1 | Operation type | query, mutation, or subscription |
| 2 | Operation name | Arbitrary client-created label used to provide a unique name to an operation |
| 3 | Top-level field | Function that returns a single unit of information or object requested within an operation (may contain nested fields) |
| 4 | Argument (of a top-level field) | Parameter name used to send information to a field to tailor the behavior and results of that field |
| 5 | Value | Data related to an argument sent to a field |
| 6 | Field | Nested function that returns a single unit of information or object requested within an operation |
| 7 | Directive | Feature used to decorate fields to change their validation or execution behavior, altering a value returned by a GraphQL server |
| 8 | Argument (of a directive) | Parameter name used to send information to a field or object to tailor its behavior and results |
| **9** | **Argument (of a field)** | **Parameter name used to send information to a field to tailor the behavior and results of the field** |

**Operation Names-**

**Operation names are client-driven inputs, they could also potentially be used as attack vectors for injection. Some implementations of GraphQL allow special characters in operation names. The applications might store these names in their audit logs, third-party applications, or other systems. These could cause mayhem if not properly sanitized.**

**Field-**

**This field suggestion feature makes GraphQL a convenient, friendly, and simple tool not only for API consumers but also for hackers. We can exploit this feature to find fields we may not have known about otherwise.**

**Arguments-**

**Arguments give clients a lot of power to manipulate the behavior of their requests and are another great attack vector. Because the value of an argument is client driven, it can potentially be stuffed with malicious content in injection-based attacks**

**Fragments-**

**Fragments can be constructed such that they reference one another, allowing for a circular fragment condition that could lead to DoS conditions**

**Directives-**

**The use of custom directives to expand GraphQL opens implementations to customized attack vectors that we hackers can exploit. A vulnerability in a custom directive  could impact a lot.**

**Interfaces-**

**Interfaces could pose a problem in applications that implement authorization using  directives. Because an interface defines fields to be used by other objects, any sensitive field that isn’t properly decorated could be exposed unintentionally.**

**Introspection-**

**Introspection, the built-in tool helps clients to discover actions they can take using a GraphQL API. Introspection lets clients query a GraphQL server for information about its underlying schema, which includes data like queries, mutations, subscriptions, directives, types, fields, and more. As hackers, this feature can be a gold mine in supporting profiling, data collection, and attack-vector analysis efforts**

**GraphQL Security threat matrix**

* ***Field Suggestions* informs a client whenever they send a query with a spelling mistake and suggests alternative options. This can be leveraged for information disclosure.**
* ***Query Depth Limit* is a security control to prevent DoS attacks that may abuse conditions such as cyclical node relationships in schemas.**
* ***Query Cost Analysis* is a security control to prevent DoS attacks that stem from computationally complex queries.**
* ***Automatic Persisted Queries* is a caching mechanism. It allows the client to pass a hash representing a query as a way to save bandwidth and can be used as a security control with an allow list of safe queries.**
* ***Introspection*provides access to information about queries, mutations, subscriptions, fields, objects, and so on through the \_\_schema meta-field. This can be abused to disclose information about the application’s schema.**
* ***Debug Mode*is a mode in GraphQL that provides additional information in the response for debugging purposes. This can potentially introduce information disclosure issues.**
* ***Batch Requests* is a feature that provides clients with the ability to send a sequence of queries in a single HTTP request. Batch queries are a great vector for DoS attacks.**

**Threat Matrix of Apollo**

[**https://github.com/nicholasaleks/graphql-threat-matrix/blob/master/implementations/apollo.md**](https://github.com/nicholasaleks/graphql-threat-matrix/blob/master/implementations/apollo.md)

**Common GraphQL Vulnerabilities**

**Denial Of Service**

**Field and object relationships, aliases, directives, could potentially be used as attack vectors against a GraphQL service, because these capabilities clients with lot of control over the query structure and execution behavior. Very complex queries can degrade a GraphQL server’s performance if the right security countermeasures are not put in place.**

* **Abusing Directives**
  + **The directive overloading vulnerability requires us to send many directives via several continuous requests. It is very effective at  degrading the server’s performance.**

**Example-**

**query { articles { title @aa@aa@aa@aa # add as many directives as possible content @aa@aa@aa@aa } }**

* **Field Duplication**
  + **Field duplications are queries that exhaust the server because of the amount of time they take to process and resolve. Here is an example**
    - **Chose a  field that you is expensive to resolve, and stuff the query with additional copies of that field’s name**
* **Circular Relationships/Queries**
  + **Circular relationships are common in GraphQL APIs. While not an anti-pattern when it comes to schema design, they should be avoided unless the application is able to gracefully handle complex queries.**
  + **Example-**

**type Article { title: String content: String owner: Owner } type Owner articles: [Article] name: String }  
The following circular query can cause DOS**

**query { articles { owner { articles { owner { articles { owner { name } } } } } } }  
Tools that can help identify circular queries  
1) GraphQL Voyager  
2)InQL**

**Protection Against DOS-**

**1) Leverage  query cost analysis provided by GraphQL Server and set the Maximum cost allowed**

**2) Set Query Depth Limits in the server to stop abuse from Circular queries- Depth is typically calculated per query, so If an attacker sends multiple recursive queries simultaneously, this can still impact the server**

**3) WAFs are must to protect from SQL injections, OS injections, XSS.**

**Information Disclosure-**

**When introspection is enabled, tools like InQL can  automatically extract the schema from GraphQL targets. When introspection is disabled, you can exploit a built-in GraphQL feature known as field suggestions and “stuff” fields by using a tool called Clairvoyance.**

**Authentication  and Authorization**

**Authentication-**

**OAuth 2.0 can also be used to protect graphical GraphQL clients. GraphQL APIs that don’t validate JWT signature by using their secret key will be prone to forgery-based attacks.**

**Authorization-**

***GraphQL Shield* for generating an authorization layer in GraphQL APIs. It allows developers to define rules that either permit or deny client access. Example listed below defines the permissions and roles required to access each query.**

**Schema Directives can also be used for the auth layer**[**https://dev.azure.com/spglobal/SNLServiceModules/\_git/graphql-poc?path=/graphqlauth&version=GBusers/kiran.sahoo2/graphqlauth**](https://dev.azure.com/spglobal/SNLServiceModules/_git/graphql-poc?path=/graphqlauth&version=GBusers/kiran.sahoo2/graphqlauth)**.**

**Injection Vulnerabilities**

**GraphQL’s language supports multiple avenues for a malicious client to send injection data to a server, such as query arguments, field arguments, directive arguments, and mutations.**

* **The application should implement security checks on the input it receives.**
* **The application shouldn't pass the received user input to a third system without proper security checks and validation**

**GraphQL Security API Testing**

**Information Disclosure/Introspection**

1. **Perform a port scan using Nmap to identify open web application ports.**
2. **Scan the web server for GraphQL endpoints by using Graphw00f’s detection mode.**
3. **Perform server fingerprinting with Graphw00f’s fingerprint mode.**
4. **Search for server-level vulnerabilities on MITRE’s CVE database.**
5. **Search for server-level security features on GraphQL Threat Matrix.**
6. **Search for GraphQL IDEs such as GraphiQL Explorer or GraphQL Playground by using EyeWitness.**
7. **Send an introspection query and document all available queries, mutations, and subscriptions.**
8. **Visualize the introspection query response with GraphQL Voyager.**
9. **Extract the GraphQL schema by using field stuffing when introspection is disabled.**
10. **Identify debug errors in query responses by sending malformed queries.**
11. **Identify query tracing in GraphQL responses.**
12. **Test for any PII submitted using queries over the GET method.**

**Denial of Service**

1. **Review the API’s schemas for bidirectional relationships.**
2. **Test for the following:**
   1. **Circular queries or mutations**
   2. **Circular fragments**
   3. **Field duplication**
   4. **Alias overloading**
   5. **Directive overloading**
   6. **Array-based or alias-based query batching**
   7. **Object limit overriding in API pagination arguments such as filter, max, limit, and total**

**Authentication and Authorization**

1. **Test access to the following:**
   1. **The API without authentication headers**
   2. **Restricted fields by using alternate paths**
   3. **The API by using both the GET and POST methods**
2. **Test signature validation in JSON Web Token (JWT).**
3. **Attempt to brute-force mutations or queries that accept secrets, such as tokens or passwords, using the following:**
   1. **Alias-based query batching**
   2. **Array-based query batching**
   3. **CrackQL**
   4. **Burp Suite**

**Injection**

1. Test for injection in the following:
   1. Query arguments
   2. Field arguments
   3. Query directive arguments
   4. Operation names
2. Test for SQLi automatically by using SQLmap.
3. Test for OS command injection automatically by using Commix.

**Forging Requests**

1. Test for the following:
   1. The existence of anti-CSRF tokens in HTTP headers or bodies
   2. Possible anti-CSRF token bypasses
   3. The availability of GET-based queries
   4. Support for GET-based mutations
2. Perform state-changing mutations over GET.
3. Perform state-changing mutations over POST.

**Hijacking Requests**

1. Identify whether the GraphQL server does the following:
   1. Supports subscriptions
   2. Validates the Origin header during a WebSocket handshake

**Graphql Security Testing tools**

**Burp Suite**

Burp Suite can be used  manually interact with target by observing and modifying GraphQL queries before they are sent to the target server.

**Clairvoyance**

This Python-based reconnaissance tool for GraphQL APIs, allows you to discover schema information when introspection is disabled. It works by abusing a GraphQL feature called *field suggestions*. Essentially, it reconstructs the underlying schema by sending queries crafted from a dictionary of common English words and observing the server’s responses

Installation-

# **cd ~**

# **git clone https://github.com/nikitastupin/clairvoyance.git**

# **cd clairvoyance**

**InQL**

This security testing tool, based on Python, relies on the introspection query. InQL can export any information it finds about the GraphQL schema to a variety of formats, making the application’s schema easier to read and understand. InQL also can perform other tasks, such as detecting potential DoS conditions.

Installation

# **cd ~**

# **git clone https://github.com/doyensec/inql.git**

# **cd inql**

# **sudo python3 setup.py install**

**BatchQL**

Allows clients to send multiple queries in a single HTTP request.

Installation-

**# cd ~ # git clone https://github.com/assetnote/batchql.git  
  
GraphQL-Enum  
  
graphql-path-enum is a security testing tool that finds various ways to construct queries that reach a specific piece of data.  
This is good in identifying authorization flaws.**

Installation-

# **cd ~**

# **wget "https://gitlab.com/dee-see/graphql-path-enum/-/jobs/artifacts/v1.1/raw**

**/target/release/graphql-path-enum?job=build-linux"**

**-O graphql-path-enum**

# **chmod u+x graphql-path-enum**

**CrackQL**

**A specialized brute-forcing tool for GraphQL that uses GraphQL language features to better optimize brute-force attacks against API actions that may require authentication.**

Installation-

# **git clone https://github.com/nicholasaleks/CrackQL.git**

# **cd CrackQL**

# **pip3 install -r requirements.txt**

# **python3 CrackQL.py -h**