GraphQL診断

@salty_byte 2023/04/28

話すこと

- GraphQLとは
- 前提知識
- 攻撃手法と対策
- 診断時の確認項目
- まとめ

GraphQLとは

ざっくり歴史

- Web API向けに作られたクエリ言語およびランタイム
- Facebookによって2012年から始まり、2015年にオープンソース化
- 2018年GraphQL Foundationが設立
  - GraphQLの標準化や普及の推進
- 2018年2月にGraphQL スキーマ定義言語 (SDL) が仕様の一部になった

GraphQL | A query language for your API
What is the GraphQL Foundation? | GraphQL
https://github.com/graphql/graphql-spec/pull/90/

特徴

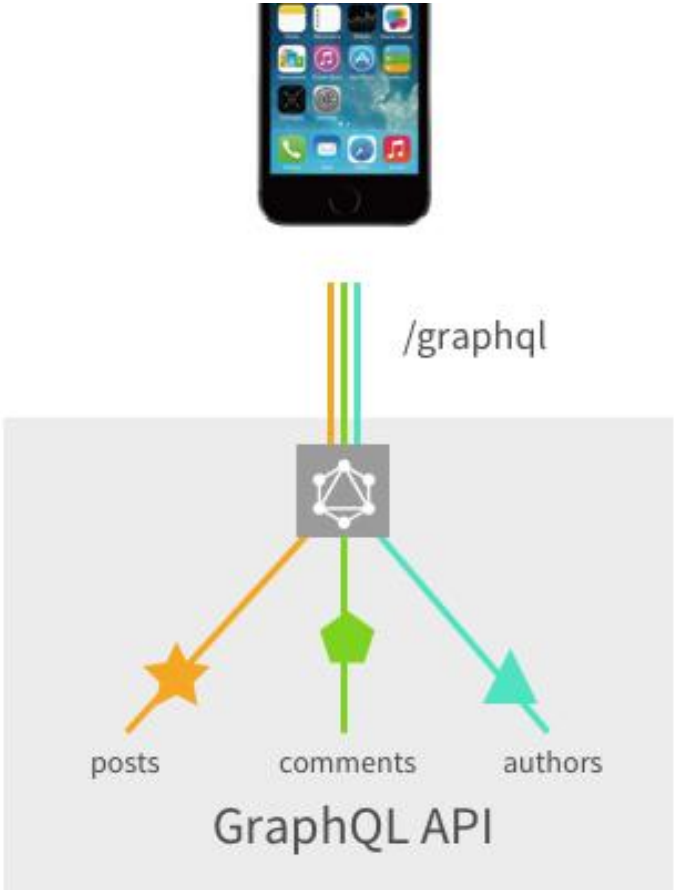
- データの形式/構造は、スキーマによって指定する
  - 型安全なデータ
- クライアント側で、（スキーマに従って）必要なデータの命令を組み立てる
  - データのやり取りが簡単
  - 最小限のデータを取得可能
  - 柔軟な対応が可能
- 一度のリクエストで多くのリソースを取得可能
- 既存のクエリに影響を与えずに、フィールドを追加/削除が可能
  - APIの拡張や管理が容易
  - バージョン管理が不要

GraphQL | A query language for your API

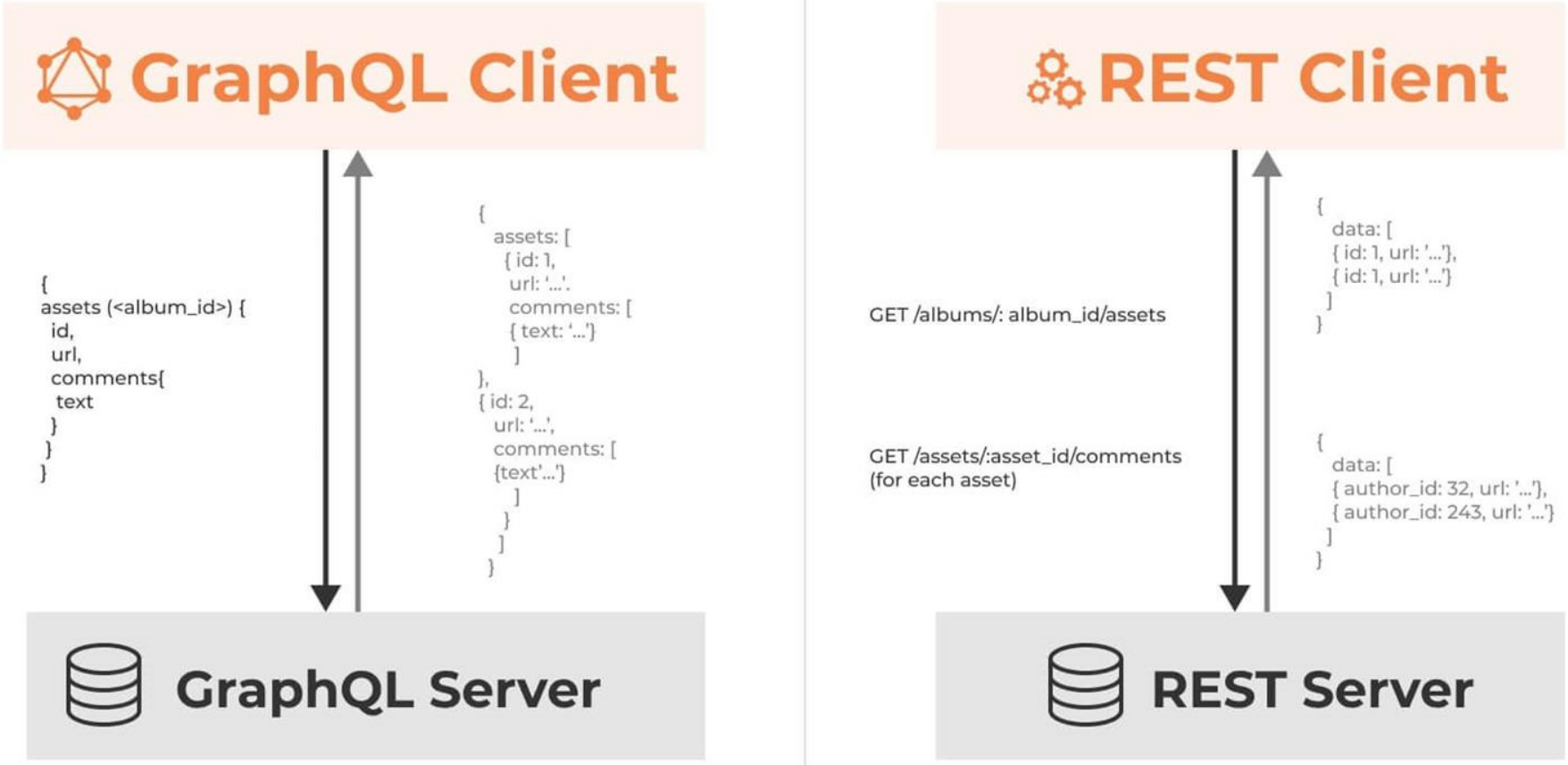
GraphQL vs REST API（ざっくり比較）

比較項目	GraphQL	REST API
エンドポイント数	一つ	リソース毎に複数
取得データの指定方法	クライアント側で指定	サーバ側で指定
表現方法（機能別）	クエリ ミューテーション サブスクリプション	HTTPメソッドで分ける GET/POST/PUT/PATCH/DELETE
ステータスコード	常に200	処理状態によって変える 例）正常時：200 エラー時：400, 401, 403等

GraphQLとREST | GraphQLチュートリアル (hasura.io)



図引用：https://www.apollographql.com/blog/graphql/basics/graphql-vs-rest/



図引用：https://www.mobilelive.ca/blog/graphql-vs-rest-what-you-didnt-know

よく使われているツール（例:JavaScript/TypeScript）

- GraphQLサーバ
  - Apollo Server
  - GraphQL Yoga
- GraphQLクライアント
  - Apollo Client
- 他
  - GraphQL Code Generator

他の言語についてはこちらを参照：GraphQL Code Libraries, Tools and Services

前提知識

用語

- Query（クエリ）
- Mutation（ミューテーション）
- Subscription（サブスクリプション）

GraphQL vs REST API（用途別）

機能	GraphQL	REST API
データ取得	クエリ	GET
データ追加	ミューテーション	POST
データ更新/削除	ミューテーション	PUT/PATCH/DELETE
データ監視/ イベント通知	サブスクリプション	-

サポート状況

- デフォルトで以下が許可されている
  - GET (クエリストリング)
  - POST (x-www-form-urlencoded)
  - POST (json)

スキーマ定義

- スキーマを定義する際に型に注意する
  - ルート型
    - Query / Mutation / Subscription
  - スカラー型
    - String（文字列型）
    - Int（整数型）
    - Float（浮動小数点型）
    - Boolean（論理型）
    - ID（ID型）
  - オブジェクト型
  - 列挙型（Enum）

  …等

https://qiita.com/NagaokaKenichi/items/d341dc092012e05d6606

スキーマ定義

```
type Query {
  users: [User!]!
  user(id: ID!): User
}

type Mutation {
  createUser(name: String!): User!
}

type User {
  id: ID!
  name: String!
}
```

- データの型や構造を定義する
- サーバやクライアントはこのスキーマに従って実装することで、型安全にデータのやり取りが可能となる

https://www.howtographql.com/typescript-apollo/1-getting-started/

スキーマ定義

```
type Query {
  users: [User!]!
  user(id: ID!): User
}

type Mutation {
  createUser(name: String!): User!
}

type User {
  id: ID!
  name: String!
}
```

queryの定義：

- usersクエリ
  - 引数無しでユーザリストを返す
- userクエリ
  - idを引数として、条件にあったユーザを返す

!：nullでないことを示す
[]：リストを示す

スキーマ定義

```
type Query {
  users: [User!]!
  user(id: ID!): User
}

type Mutation {
  createUser(name: String!): User!
}

type User {
  id: ID!
  name: String!
}
```

mutationの定義：

- createUserクエリ
  - 名前（文字列）を引数として、作成したユーザを返す

スキーマ定義

```
type Query {
  users: [User!]!
  user(id: ID!): User
}

type Mutation {
  createUser(name: String!): User!
}

type User {
  id: ID!
  name: String!
}
```

オブジェクトの定義：
- Userオブジェクト
  - idとnameを持つ

クエリ/ミューテーション例

```
# ユーザ情報を取得する
query {
  users {
    id
    name
  }
}
```

```
# idからユーザを取得する
query {
  user(id: "1") {
    id
    name
  }
}
```

```
# 新しいユーザを追加する
mutation {
  createUser(name: "Bob") {
    id
    name
  }
}
```

HTTPリクエスト例

```
POST /graphql HTTP/1.1
Host: 127.0.0.1:5013
Content-Length: 224
sec-ch-ua: "Not:A-Brand";v="99", "Chromium";v="112"
（中略）
Accept-Language: ja,en-US;q=0.9,en;q=0.8
Cookie: env=graphiql:disable
Connection: close

{"query":"query getPastes {¥n        pastes(public:false) {¥n            id¥n            title¥n
content¥n            ipAddr¥n            userAgent¥n            owner {¥n
name¥n            }¥n            }¥n        }"}
```

「¥n」は削除しても動作する

HTTPレスポンス例

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 311
Date: Thu, 27 Apr 2023 16:02:15 GMT

{"data":{"pastes":[{"id":"2","title":"555-555-1337","content":"My Phone Number","ipAddr":"127.0.0.1","userAgent":"User-Agent not set","owner":{"name":"DVGAUser"}},{"id":"1","title":"Testing Testing","content":"My First Paste","ipAddr":"127.0.0.1","userAgent":"User-Agent not set","owner":{"name":"DVGAUser"}}]}}
```

整形後


```
{
  "data": {
    "pastes": [
      {
        "id": "2",
        "title": "555-555-1337",
        "content": "My Phone Number",
        "ipAddr": "127.0.0.1",
        "userAgent": "User-Agent not set",
        "owner": {
          "name": "DVGAUser"
        }
      },
      {
        "id": "1",
        "title": "Testing Testing",
        "content": "My First Paste",
        "ipAddr": "127.0.0.1",
        "userAgent": "User-Agent not set",
        "owner": {
          "name": "DVGAUser"
        }
      }
    ]
  }
```

攻撃手法と対策

攻撃手法（GraphQLの特性により考えられる問題）

- サーバ設定の不備（Introspection Query）
- DoS攻撃
- レースコンディション
- バッチ攻撃

https://github.com/WebAppPentestGuidelines/graphQLGuideLine/blob/master/docs/specific.md

攻撃手法（通常のWeb診断と同様の項目）

- インジェクション系（e.g. SQL、XSS、OSコマンド）
- 認可制御の不備
- 不適切なエラーメッセージ
- クロスサイト・リクエスト・フォージェリ（CSRF）

Introspection Query

- GraphQLのスキーマや型を問い合わせるための特別なクエリ
- 外部からアクセスができると、利用可能なクエリやミューテーション、サブスクリプションなどの詳細な情報を調べられる
- デフォルトで有効になっていることが多いため、設定ミスで意図せず公開されている可能性がある

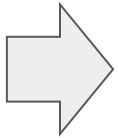
攻撃事例：Introspection Query

- https://hackerone.com/reports/291531

攻撃例：Introspection Query

クエリ

```
query IntrospectionQuery {
  __schema {
    queryType {
      name
    }
    mutationType {
      name
    }
    subscriptionType {
      name
    }
    types {
      name
    }
  }
}
```



結果

```
{
  "data": {
    "__schema": {
      "queryType": {
        "name": "Query"
      },
      "mutationType": {
        "name": "Mutation"
      },
      "subscriptionType": {
        "name": "Subscription"
      },
      "types": [
        {
          "name": "__TypeKind"
        },
...}
```

攻撃例：Introspection Query

```
GET /graphql?query={__schema{types{name,fields{name}}}} HTTP/1.1
Host: 127.0.0.1:3000
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/112.0.5615.50 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Accept-Encoding: gzip, deflate
Accept-Language: ja,en-US;q=0.9,en;q=0.8
Connection: close
Content-Length: 0
```

デモ：Introspection Query

Introspection Query

- 実際にはIntrospection Queryを一から組み立てるのは面倒
- 以下のようなツールを使うと楽
  - GraphQL Voyager
  - GraphQL Playground

Introspection Queryが無効になっている場合

- ブラウザの開発者ツールで検索
  - Inspect/Sources/"Search all files"
  - file:* mutation
  - file:* query

https://book.hacktricks.xyz/network-services-pentesting/pentesting-web/graphql#leaked-graphql-structures

対策：Introspection Query

- Introspection Queryへのアクセスを制限する

例）Apollo Serverの場合

```
const server = new ApolloServer({
  typeDefs,
  resolvers,
  introspection: false
});
```

NODE_ENVがproductionなら指定不要

https://www.apollographql.com/docs/apollo-server/api/apollo-server/#introspection

よくある開発用パス

- 開発用として以下のようなパスにアクセスができる場合がある
  - /graphiql
  - /console
  - /graphql/console
  - /voyager

  ※実際にはパスは任意に指定できるため、外部から判断できないこともある
- APIとして公開していない場合は、ユーザからアクセスできる必要は無い
- もしアクセスできた場合は、Informationで指摘してもよさそう

デバッグモード

- デバッグモードを有効にできる場合がある
  - /graphql.php?debug=1
- 有効になると詳細なエラーが出力されるようになる

不適切なエラーメッセージ

- 不適切な設定によって、発生したエラーの詳細がクライアントに送られる
- スタックトレース
  - サーバ側で発生した例外から内部ロジックの推測ができる
- サジェスト（Field Suggestions）
  - Introspection Queryが無効でも関連のあるキーワードが出力されるため、利用可能なクエリの調査が可能になる
  - Nessusの検査プラグインもある：https://www.tenable.com/plugins/was/112895

不適切なエラーメッセージ：スタックトレース

{"errors":[{"message":"Syntax Error GraphQL (2:20) Unexpected character ¥""¥".¥n¥n1: mutation CreatePaste ($title: String!, $content: String!, $public: Boolean!, $burn: Boolean!) {¥n2: createPaste'(title:$title, content:$content, public:$public, burn: $burn) {¥n ^¥n3: paste {¥n","locations":[{"line":2,"column":20}]}]}
}

不適切なエラーメッセージ：サジェスト

```
http://127.0.0.1:3000/graphql?query={__schema}
```

```
HTTP/1.1 200 OK
X-Powered-By: Express
content-type: application/json; charset=utf-8
Date: Wed, 26 Apr 2023 23:35:45 GMT
Connection: close
Content-Length: 174

{"errors":[{"message":"Field ¥"__schema¥" of type ¥"__Schema!¥" must
have a selection of subfields. Did you mean ¥"__schema
{ ... }¥"?","locations":[{"line":1,"column":2}]}]}
```

対策：不適切なエラーメッセージ

- 入力値の検証をする
- 利用者に不要なエラーメッセージは抑制する

例）Apollo Serverの場合

```
const server = new ApolloServer({
  typeDefs,
  resolvers,
  introspection: false
  includeStacktraceInErrorResponses: true,
});
```

NODE_ENVがproductionなら指定不要

https://www.apollographql.com/docs/apollo-server/data/errors/#omitting-or-including-stacktrace

DoS攻撃

- クエリ展開やオブジェクト解決にリソースを占有させる
- サーバに過剰な負荷をかけることで、レスポンス時間の遅延やサーバダウンを引き起こす

https://github.com/WebAppPentestGuidelines/graphQLGuideLine/blob/master/docs/specific/dos.md

DoS攻撃：巨大ノードの要求

- 複数のフィールドを組み合わせて巨大なノードを要求するクエリを発行する

```
query{
        regions (limit: 1000){
                teams (limit: 1000){
                        users (limit: 1000) {
                                name
                        }
                }
        }
}
```

条件にあうデータが必要

https://github.com/WebAppPentestGuidelines/graphQLGuideLine/blob/master/docs/specific/dos.md

DoS攻撃：再帰的な解決

- 再帰的な解決を引き起こすクエリを利用する


```
query{
    Author (id: 1){
        name,
        books{
            name,
            author{
                books{
                    name,
                    author{
                        name
                    }
                }
            }
        }
    }
}
```

https://github.com/WebAppPentestGuidelines/graphQLGuideLine/blob/master/docs/specific/dos.md

デモ：DoS攻撃

対策：DoS攻撃

- 再帰的解決が可能なクエリを排除する
- クエリの展開深度の最大値を設定する
- クエリの複雑さを制限する
- 同時実行数を制限する

レースコンディション

- 同一リソースに対して、複数スレッドで同時にアクセスすることで競合を発生させる
- 事例：https://hackerone.com/reports/488985

対策：レースコンディション

- 適切に排他制御を実装する

バッチ攻撃

- GraphQLでは、一回のリクエストで複数のクエリを実行できる
- ブルートフォース（総当たり）攻撃に使える
- WAF、RASP、IDS/IPS、SIEM等では、攻撃を検出できない可能性がある
  - 1 つのリクエストに複数の処理を書いて送るため

攻撃：バッチ攻撃

例）ログインの総当たり

```
mutation {
  login(username: "admin", password: "password")
  second: login(username: "admin", password: "admin")
  third: login(username: "administrator", password: "password")
 }
```

対策/軽減策：バッチ攻撃

- リクエストのレート制限を設ける
- 機密オブジェクトのバッチ処理を防止する
- 一度に実行できるクエリの数を制限する

認可制御の不備

- 権限確認が不十分な場合、権限がなくても処理を実行できる場合がある
- 確認方法は、通常の診断と同様
- GET、POST (x-www-form-urlencoded) も試してみると良さそう

対策：認可制御の不備

- 適切な権限管理をする

クロスサイト・リクエスト・フォージェリ（CSRF）

- CSRFの検証が不十分な場合、罠ページにアクセスした利用者に意図しない処理を実行させることが可能

攻撃：CSRF

- CORS設定の不備を確認する
  - 通常はPOST（json）で送られるため、CORSの不備がない限り罠は作れない
- GET、POST (x-www-form-urlencoded) を利用して確認する
  - POSTの場合はCookieのSamesite属性がLax以上だと、異なるオリジンからCookieが送信されない
  - リクエストヘッダにAPIトークンのようなカスタムヘッダがある場合も注意

対策：CSRF

- 通常のCSRF対策と同様
- カスタムヘッダを利用するのが良さそう

インジェクション攻撃

- SQL、XSS、OSコマンド等のインジェクションに対する脆弱性を確認する
- 確認方法は、通常の診断と同様

カスタムスカラー（Custom scalars）

- 用意されていないスカラー型は独自に定義可能
  - 日付、メールアドレス、UUID等

例）日付

```
scalar DateTime
```

- カスタムスカラーは、入力値の検証が足りない可能性がある

対策：インジェクション攻撃

- 各脆弱性の対策を行う
- 入力値の検証
- WAFの導入

診断時の確認項目

確認項目

- ☑ サーバ設定の不備（有効なIntrospection Query）
- ☑ サーバ設定の不備（デバッグ用機能の公開）
- ☑ インジェクション系
- ☑ 認可制御の不備
- ☑ 不適切なエラーメッセージ
- ☑ クロスサイト・リクエスト・フォージェリ（CSRF）

+α：

- ☑ DoS攻撃
- ☑ バッチ攻撃
- ☑ レースコンディション

（具体的な確認項目については、社内診断レギュレーションとなるため割愛）

まとめ

まとめ

- GraphQL自体は前からある技術だが、導入事例が増えてきている
- GraphQLの診断でも基本的には通常の診断項目と同様

参考

- https://cheatsheetseries.owasp.org/cheatsheets/GraphQL_Cheat_Sheet.html
- https://blog.doyensec.com/2018/05/17/graphql-security-overview.html
- https://blog.yeswehack.com/yeswerhackers/how-exploit-graphql-endpoint-bug-bounty/
- https://blog.assetnote.io/2021/08/29/exploiting-graphql/
- https://engineering.mercari.com/blog/entry/20220303-concerns-with-using-graphql/
- https://malware.news/t/the-5-most-common-graphql-security-vulnerabilities/39042

やられサイト

- https://github.com/righettod/poc-graphql
- https://github.com/dolevf/Damn-Vulnerable-GraphQL-Application