Damn GraphQL

Attacking & Defending APIs

nsec

Who am I

- Principal Security Engineer at Wealthsimple
- Background is primarily in the security engineering space
- Founder of Toronto’s DC416
- First time speaking and participating at NorthSec

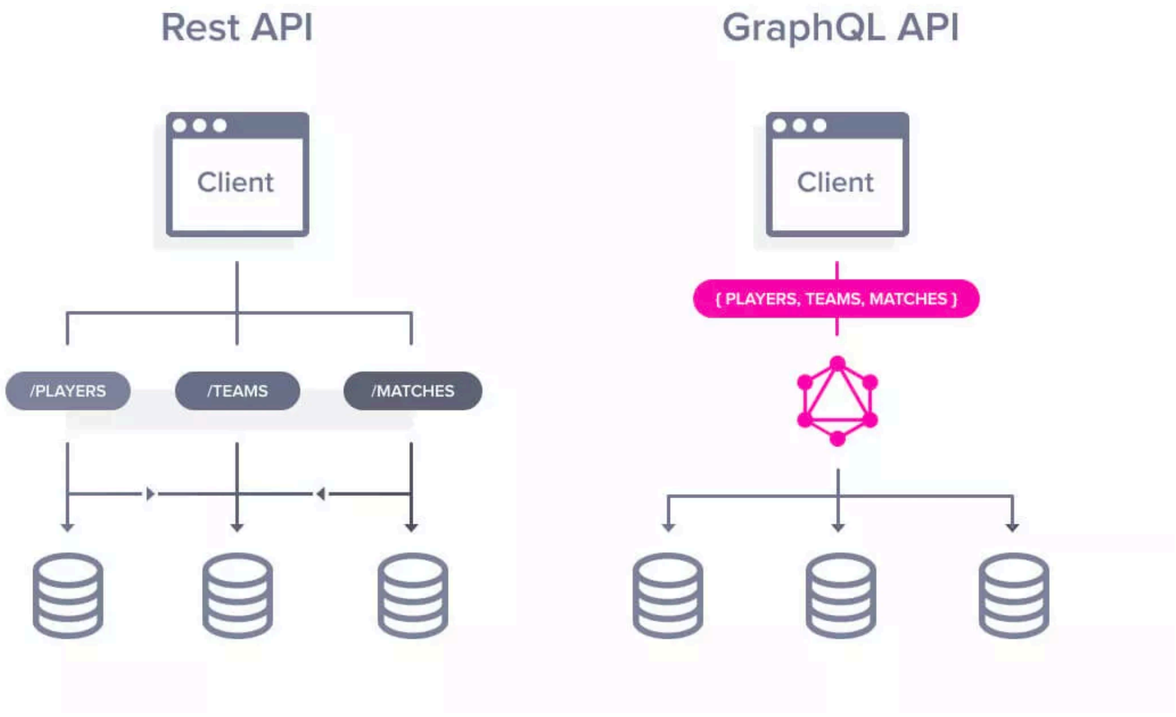
Agenda

- What is GraphQL
- GraphQL Attack Surface
  - Vulnerabilities
  - Mitigations
  - Tips & Tricks
- GraphQL Attack Demo
- GraphQL Hacking Playground

What is GraphQL?

What is GraphQL?

- Query Language for APIs
- Uses **Queries**, **Mutations** and **Subscriptions**
- Solves over-fetching and under-fetching
- Reduces round trip



What is GraphQL?

Schema

- Data definition
- Constructed from **objects** and **fields**

GraphQL schema

```
type Paste {
  id: ID!
  owner: String!
  title: String!
  content: String!
  is_public: Boolean!
  tags: [String!]
}
```

What is GraphQL?

Queries

- Used for reading data
- Example: **Fetching** the **title** and **content** of **all pastes**

Request

```
query {
  pastes {
    title
    content
  }
}
```

Response

```
"data": {
  "pastes": [
    {
      "title": "My New Paste",
      "content": "Hello!"
    }
  ]
}
```

What is GraphQL?

Mutations

- Used for writing / modifying data
  - A fetch follows immediately after
- Example: **Creating** a paste with a specific **title** and **content**

Request

```
mutation {
  createPaste(title:"Hello!", content:"Good bye!"){
    paste {
      title
    }
  }
}
```

Response

```
"data": {
  "createPaste": {
    "paste": {
    "title": "Hello!"
    }
  }
}
```

What is GraphQL?

Just GraphQL things

- Requests carried over POST
- Single endpoint - /graphql
  - Discover routes by fuzzing: /v1/graphql, /v2/graphql, /playground, /console, etc.
  - Nmap NSE for GraphQL
- HTTP response codes are usually 200 OK

https://github.com/dolevf/nmap-graphql-introspection-nse

GraphQL Attack Surface

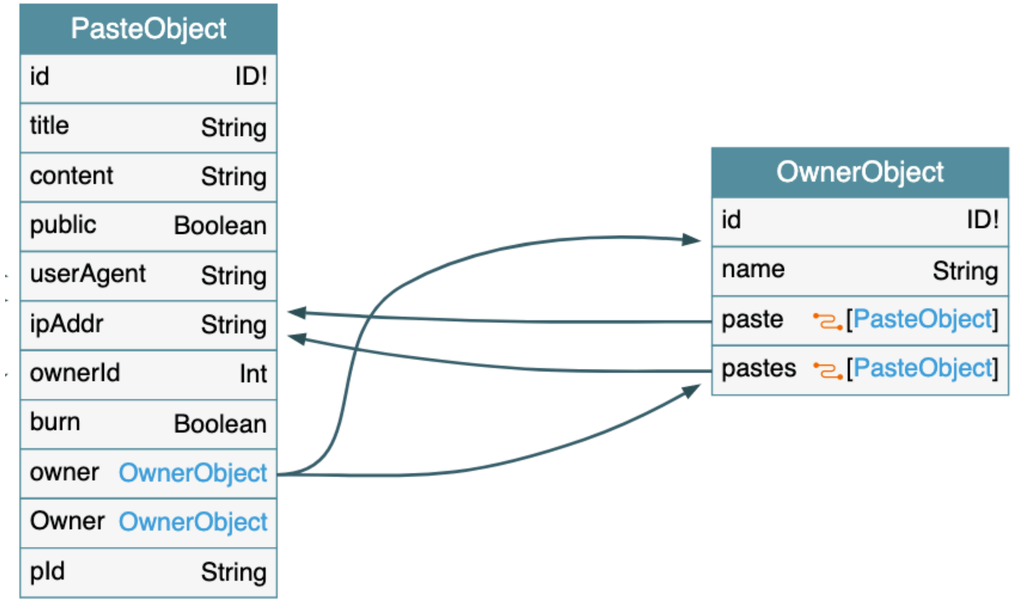
GraphQL Attack Surface

Information Gathering

Introspection

- Introspection is telling GraphQL to describe itself
- Some implementations enable it by default

- Test using an introspection query[1]
- Use Voyager[2] to visualize data model



[1] https://gist.github.com/craigbeck/b90915d49fda19d5b2b17ead14dcd6da

[2] https://apis.guru/graphql-voyager/

GraphQL Attack Surface

Information Gathering

Introspection

- Introspect your own GraphQL
- Place behind authentication and authorization
- Disable introspection in production

GraphQL Attack Surface

Information Gathering

Field Suggestions

- GraphQL will hint on what it supports upon typos
- Will almost always be available!
- Fuzzing opportunity[1]

Request

Typo →

```
query {
  past {
    title
    content
  }
}
```

Response

```
"errors": [
  {
    "message": "Cannot query field \"past\" on type \"Query\". Did you mean \"paste\" or \"pastes\"?"
  }
]
```

GraphQL Suggestions

[1] https://github.com/nikitastupin/clairvoyance

GraphQL Attack Surface

Information Gathering

Field Suggestions

- Most implementations don't offer a disable option
  - Self patch
  - UX Impact

```
# If there are no suggested types, then perhaps this was a typo?
if not suggestion:
  suggestion = did_you_mean(get_suggested_field_names(type_, field_name))
```

[1] https://github.com/graphql-python/graphql-core/blob/main/src/graphql/validation/rules/fields_on_correct_type.py#L23

GraphQL Attack Surface

Denial of Service Amplification

Query Batching

- GraphQL can batch multiple queries for processing
- Processed **in sequence**
- Abuse it for Denial of Service or Enumeration!
- Evades network-level rate limiting

```
query {
  backupSystem
}
```

→

```
[
  {
    query: backupSystem,
    variables: {},
  },
  {
    query: backupSystem,
    variables: {},
  }
]
```

Denial of Service Amplification

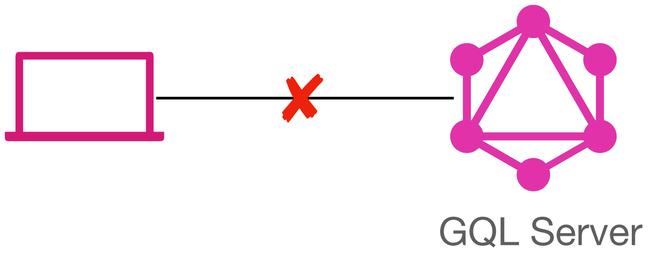
GraphQL Attack Surface

Query Batching

- Array length checks
- Use cost based analysis checks after array unpacking
- Only use where it makes sense!
- Disable if possible

```
[
    {                10
      query: backupSystem,
      variables: {},
    },
    {                10
      query: backupSystem,
      variables: {},
    }
]
```

Query Cost Analysis



max_cost = 10

GraphQL Attack Surface

Denial of Service Amplification

Query Aliasing

- GraphQL allows sending multiple queries using aliases
- The <u>same query can be repeated</u> N times

🕷 Can be an alternative to batch queries based attacks

🕷 Evades network-level rate limiting

```
query {
  backupSystem
}
```

→

```
query {
  alias1: backupSystem
  alias2: backupSystem
  alias3: backupSystem
  alias4: backupSystem
}
```

GraphQL Attack Surface

Denial of Service Amplification

Query Aliasing

- Cost based analysis
- Maximum alias allowance

GraphQL Attack Surface

Denial of Service

Circular Queries

- Circular References may cause a Denial of Service condition
- e.g. an object of type **Paste** may have an **Owner** field and **vice versa**.

👾 Abuse it for Denial of Service

👾 Chain it with a batched query or aliases for maximum impact

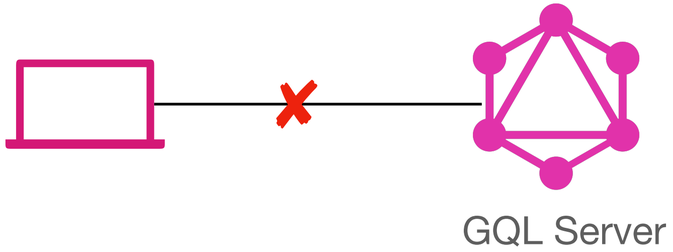
```
query {
  pastes {
    owner {
      pastes {
        owner {
          pastes {
            ...
```

GraphQL Attack Surface

Circular Queries

- Set maximum depth limit[1]
- Can be set at the schema or query level (override)

```
query {
    pastes {          # 1
      owner {         # 2
        pastes {      # 3
          owner {     # 4
            pastes { # 5
                …    # 99
```



max_depth = 10

[1] https://graphql-ruby.org/queries/complexity_and_depth.html

GraphQL Attack Surface

Spoofing
Injection

Operation Name Tampering

- GraphQL allows settings an **Operation Name**.
- Apps may use Operation Name for analytics or naively use them for decision making.

- User controlled input
- Some implementations allow special characters
- Spoofing / Injecting opportunities

Operation Name

```
query GetPastes {
    getUsers {
        username
        email
    }
}
```

GraphQL Attack Surface

Spoofing
Injection

Operation Name Tampering

- Treat it as any other untrusted input
- Have an allow-list of supported operation names
- Reject anything that isn’t on the list.
- Log violations (safely)

GraphQL Attack Surface

Denial of Service

Field Duplication

- GraphQL does not de-dup repeating fields

🕹 Opportunity to amplify the queries and cause resource exhaustion havoc

```
query {
  pastes {
    owner {
      name
    }
  }
}
```

100ms

```
query {
  pastes {
    owner {
      name
      name
      name
      name
      name
      name
    }
  }
}
```

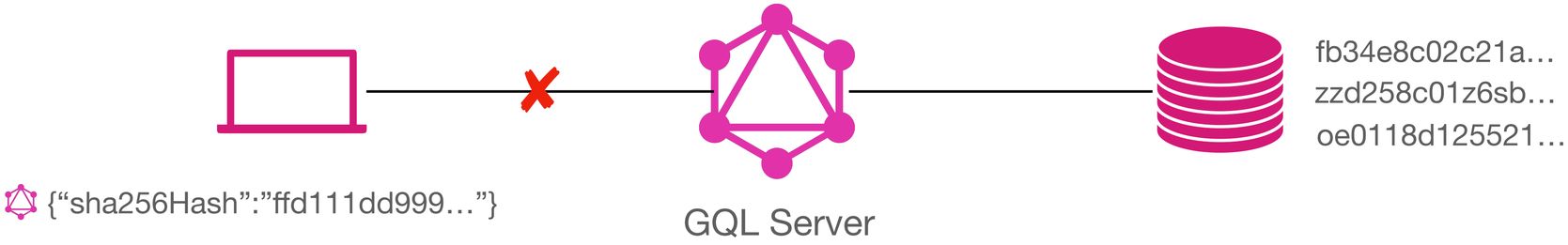
160ms

GraphQL Attack Surface

Field Duplication

- De-dup the fields
- Cost-based analysis
- Persistent Queries
- Log violations (safely)

Persistent Queries (APQ)



GraphQL Attack Surface

Summary

- OWASP Top 10 still applies to GraphQL
- Security tools exist but in limited capacity
  - OWASP ZAP Add-on
  - Burp InQL Extension
- GraphQL is still fairly young
- Gaps in security features across the various languages

GraphQL Attack Demo

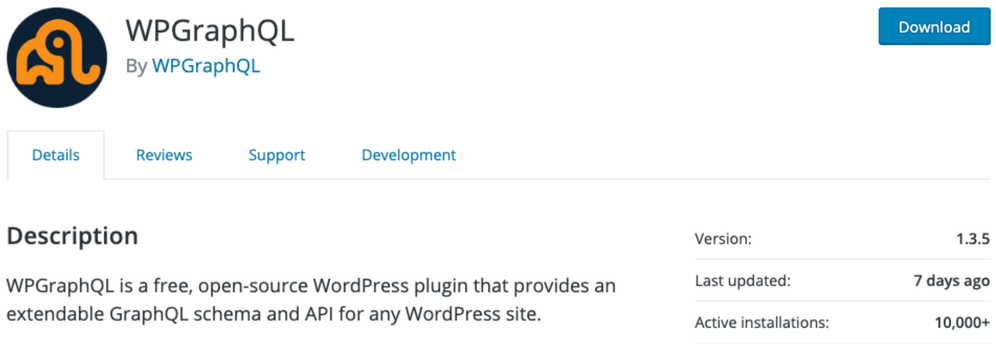
CVE-2021-31157

WordPress GraphQL 1.3.5 - Denial of Service

WordPress GraphQL

About WordPress GraphQL

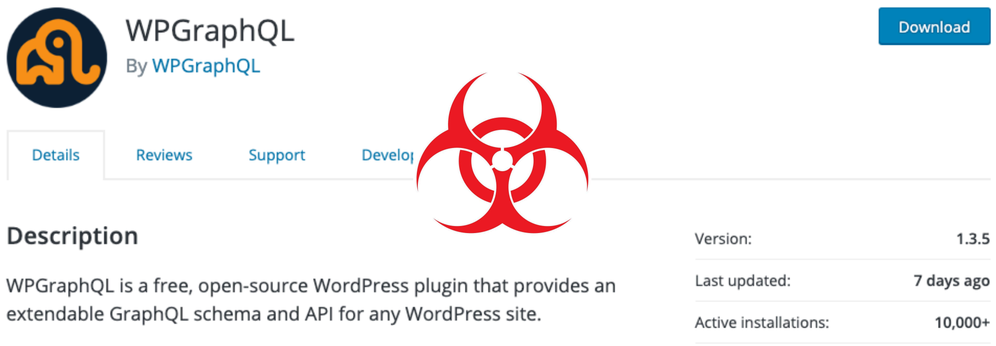
- GraphQL API layer for WordPress
- 100,000+ downloads



CVE-2021-31157

About the Vulnerability

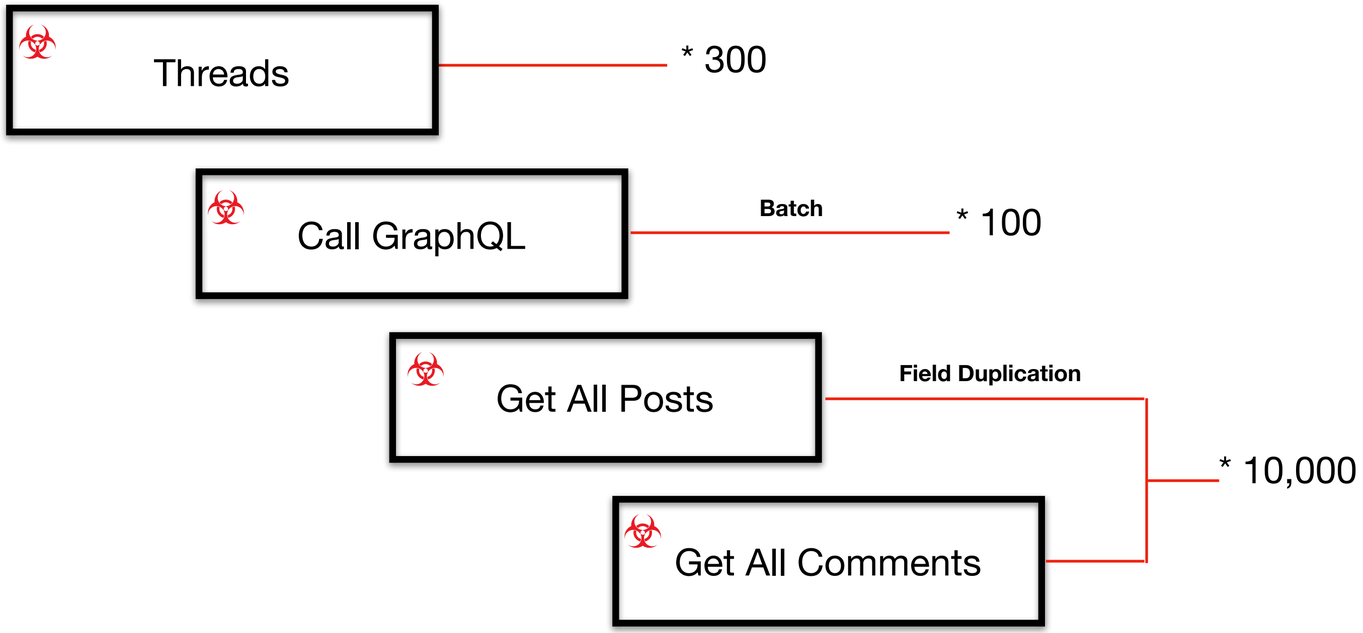
- WPGraphQL has batching enabled by default
- No option to turn off
- Duplicated fields are not de-duped
- Unauthenticated API



https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-31157

CVE-2021-31157

About the Exploit



GraphQL Hacking Playground

Damn Vulnerable GraphQL Application

Like DVWA, but for GraphQL

- Vulnerable Application based on GraphQL
- Covers most of what you've learnt today!
- Emphasis on education
- Choose your own adventure
  - Beginner Mode
  - Expert Mode (Hardened)
- Docker / Server Installation

DVGA

DAMN VULNERABLE GRAPHQL APPLICATION

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

Damn Vulnerable GraphQL Application

DVGA
DAMN VULNERABLE GRAPHQL APPLICATION

- Home
- Private Pastes
- Public Pastes
- Create Paste
- Import Paste
- Upload Paste

Star us on GitHub

Damn Vulnerable GraphQL Application

Welcome!

Damn Vulnerable GraphQL Application, or DVGA, is a vulnerable GraphQL implementation. DVGA allows learning how GraphQL can be exploited as well as defended in a safe environment.

Getting Started

If you aren't yet familiar with GraphQL, see the GraphQL Resources section below. Otherwise, start poking around and find loopholes! There are GraphQL Implementation flaws as well as general application vulnerabilities.

If you are interacting with DVGA programmatically, you can set a specific game mode (such as Beginner, or Expert) by passing the HTTP Request Header `X-DVGA-MODE` with either `Beginner` or `Expert` as values.

If the Header is not set, DVGA will default to Easy mode.

GraphQL Resources

To learn about GraphQL, and common GraphQL weaknesses and attacks, the following resources may be beneficial:

Videos

- GraphQL in 40 Minutes
- Hacking GraphQL For Beginners
- LevelUp 0x05 - REST in Peace

Articles

- OWASP GraphQL Defense
- BishopFox Labs - Design Considerations
- Leap Graph - How to secure GraphQL APIs

Got Stuck?

Head over to the Solutions page to reveal the challenge answers.

Bug Reporting

Found a bug? submit an issue on GitHub.

DVGA
DAMN VULNERABLE GRAPHQL APPLICATION

Damn Vulnerable GraphQL Application

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

DVGA
DAMN VULNERABLE GRAPHQL APPLICATION