**GraphQL Vuln**

**2 Công nghệ đang làm việc với API là : GRAPHQL và REST API   
mục đích chính tìm hiểu về GRAPHQL VÀ GRAPHQL VULN  
REST Là gì :**

**REST -> Representational State Transfer**Là một khái niệm, là một phong cách kiến trúc (architectural style) dùng cho việc thiết kế dịch vụ web., định nghĩa một tập ràng buộc (tức phải tuân theo khi mà phát triển một web service).

REST API (hay RESTful API) là một API tuân theo nguyên tắc REST, cho phép client và server giao tiếp qua HTTP một cách thống nhất, dễ hiểu.

Quy tắc chính của REST

1. Client–Server:
   * Client và server tách biệt. Client lo UI/UX, server lo xử lý logic và dữ liệu.
2. Stateless (Không trạng thái):
   * Mỗi request từ client phải chứa đầy đủ thông tin để server hiểu và xử lý.
   * Server không lưu trạng thái phiên làm việc của client (không session-based như truyền thống).
3. Cacheable:
   * Response có thể cache để giảm tải server và tăng hiệu suất.
4. Uniform Interface (Giao diện thống nhất):
   * Cách giao tiếp giữa client và server phải chuẩn hoá, nhất quán.
   * Ví dụ: sử dụng URL để định danh resource, JSON/XML để định dạng dữ liệu.
5. Layered System (Hệ thống phân tầng):
   * Client không cần biết server xử lý trực tiếp hay qua nhiều tầng trung gian (load balancer, proxy, firewall).
6. Code on Demand (tuỳ chọn):
   * Server có thể gửi code (ví dụ: script) để client thực thi. (Ít dùng trong thực tế).

Cấu trúc REST API

* Endpoint (URL): đại diện cho resource (tài nguyên).
  + Ví dụ:
    - /users → danh sách người dùng
    - /users/1 → thông tin người dùng có ID = 1
* **HTTP Methods (CRUD):**

| **Method** | **Chức năng** | **Ví dụ** |
| --- | --- | --- |
| GET | Lấy dữ liệu (Read) | GET /users/1 |
| POST | Tạo mới (Create) | POST /users |
| PUT | Cập nhật toàn bộ (Update) | PUT /users/1 |
| PATCH | Cập nhật một phần | PATCH /users/1 |
| DELETE | Xoá dữ liệu | DELETE /users/1 |
|  |  |  |

--------------------------------------------------------------------------------------------------------------------------------------------

**GraphQL là gì?**

* **GraphQL** là một ngôn ngữ truy vấn (**Query Language**) cho API và một runtime để thực thi các truy vấn đó.
* Nó được Facebook phát triển năm 2012, sau đó mã nguồn mở vào năm 2015.

**Nguồn gốc và lịch sử**

* 2012: Facebook bắt đầu dùng GraphQL nội bộ cho ứng dụng di động.
* 2015: Facebook công bố mã nguồn mở GraphQL.
* 2018: GraphQL Foundation được thành lập (thuộc Linux Foundation) để cộng đồng quản lý, không còn phụ thuộc riêng Facebook.
* Hiện nay được dùng bởi nhiều công ty lớn: GitHub, Shopify, Twitter, Pinterest, v.v.

**Schema trong GraphQL là gì?**

* **Schema** = “bản thiết kế” (blueprint) của API.
* Nó mô tả:
  + Dữ liệu nào có trong API.
  + Các kiểu dữ liệu (types) và mối quan hệ giữa chúng.
  + Các thao tác mà client có thể gọi: **Query**, **Mutation**, **Subscription**.

-> Nói ngắn gọn: **Schema định nghĩa tất cả những gì client có thể hỏi từ server**.

**Các loại operation trong GraphQL**có 3 loại: **1. Query**

Dùng để lấy dữ liệu (tương tự GET trong REST).

Ví dụ:

|  |
| --- |
| query {  user(id: 1) {  id  name  email  }  } |

Khi trả về :

|  |
| --- |
| {  "data": {  "user": {  "id": "1",  "name": "huy",  "email": "huy@example.com"  }  }  } |

**2. Mutation**

Dùng để tạo, sửa, hoặc xoá dữ liệu (tương tự POST, PUT, DELETE trong REST).

Ví dụ:

|  |
| --- |
| mutation {  createUser(name: "huy", email: "huy@example.com") {  id  name  }  } |

Kết quả trả về:

|  |
| --- |
| {  "data": {  "createUser": {  "id": "2",  "name": "huy"  }  }  } |

3. **Subscription**

Dùng để **nhận dữ liệu realtime** khi có thay đổi (thường dùng WebSocket).

Ví dụ:

|  |
| --- |
| subscription {  newUser {  id  name  }  } |

Khi có user mới, server sẽ **đẩy dữ liệu về client ngay lập tức**.

**Công Dụng:**

Tối ưu dữ liệu truyền tải

* Client chỉ lấy đúng field cần, không bị *over-fetching* (dư dữ liệu) hay *under-fetching* (thiếu dữ liệu).

Endpoint duy nhất, linh hoạt hơn REST

* REST: /users, /users/1/posts, /posts/1/comments …
* GraphQL: chỉ cần /graphql, nhưng query được tùy ý tổ hợp.

Tăng hiệu suất client (đặc biệt mobile)

* Ít request hơn, vì một query có thể lấy nhiều loại dữ liệu cùng lúc.

Hỗ trợ realtime qua subscription (WebSocket).

Schema rõ ràng

* Mọi API đều được mô tả bằng schema, giúp client tự biết có gì để gọi (tự-document).

**Khác nhau giữ REST API và GRAPHQL**

| **Tiêu chí** | **REST API** | **GraphQL** |
| --- | --- | --- |
| **Số endpoint** | Nhiều endpoint (ví dụ /users, /users/1/posts) | Chỉ 1 endpoint (thường /graphql) |
| **Lấy dữ liệu** | Mỗi endpoint trả dữ liệu cố định, có thể dư hoặc thiếu (*over-fetching / under-fetching*) | Client chỉ định rõ cần field nào, không dư, không thiếu |
| **HTTP methods** | Dùng nhiều phương thức: GET, POST, PUT, DELETE | Chủ yếu POST (chứa query/mutation trong body) |
| **Schema** | Không có schema chính thức, thường dựa vào docs | Có Schema mạnh kiểu (typed), client biết chính xác API có gì |
| **Realtime** | Cần thêm cơ chế khác (WebSocket, SSE) | Có sẵn Subscription cho realtime |
| **Hiệu suất client** | Có thể phải gọi nhiều request để lấy dữ liệu liên quan | Một query có thể gom nhiều dữ liệu trong 1 request |

**Ví dụ minh hoạ**

**Trường hợp: muốn lấy User và Posts của User đó.**

**🔹 REST API**

* Gọi 2 request:
  1. GET /users/1 → trả thông tin user.
  2. GET /users/1/posts → trả danh sách bài viết của user.

**🔹 GraphQL**

* **Chỉ 1 request đến /graphql với query:**

|  |
| --- |
| {  user(id: 1) {  name  email  posts {  title  content  }  }  } |

**Kết quả trả về:**

|  |
| --- |
| {  "data": {  "user": {  "name": "huy",  "email": "huy@example.com",  "posts": [  { "title": "Bài 1", "content": "Nội dung 1" },  { "title": "Bài 2", "content": "Nội dung 2" }  ]  }  }  } |

**GraphQL Attacks**

**Tìm endpoint GraphQL**

Các endpoint thường thấy :

* /graphql
* /api
* /api/graphql
* /graphql/api
* /graphql/graphql

Nếu các điểm cuối chung này không trả về phản hồi GraphQL, bạn cũng có thể thử thêm /v1 vào đường dẫn.

Lưu ý

Các dịch vụ GraphQL thường sẽ phản hồi bất kỳ yêu cầu nào không phải GraphQL bằng lỗi "truy vấn không tồn tại" hoặc lỗi tương tự. Bạn nên lưu ý điều này khi kiểm tra các điểm cuối GraphQL.  
**Request methods**điểm cuối GraphQL production là chỉ chấp nhận các yêu cầu POST có kiểu nội dung là application/json, vì điều này giúp bảo vệ khỏi các lỗ hổng CSRF. Tuy nhiên, một số điểm cuối có thể chấp nhận các phương thức thay thế, chẳng hạn như yêu cầu GET hoặc yêu cầu POST sử dụng kiểu nội dung là x-www-form-urlencoded.

Nếu bạn không thể tìm thấy điểm cuối GraphQL bằng cách gửi yêu cầu POST đến các điểm cuối chung, hãy thử gửi lại truy vấn chung bằng các phương thức HTTP thay thế.  
**GraphQL vuln thường xoay quanh:**

1. **Lộ schema** (introspection).
2. **Truy vấn vượt quyền** (broken access control).
3. **Lộ dữ liệu nhạy cảm** (over-fetching).
4. **Tấn công DoS** (nested query, complex query).
5. **Injection** (SQL/NoSQL).
6. **Batching / alias abuse**.
7. **CSRF**.
8. **Input validation errors**.

**1) Introspection / Information Disclosure**

**Ý tưởng khai thác:** nếu introspection bật, attacker có thể lấy toàn bộ schema (types, fields, args, mutations) — biết chính xác API có gì để tìm field nhạy cảm hoặc mutation quản lý user/roles.

**Ví dụ query (introspection):**

|  |
| --- |
| # standard introspection  {  \_\_schema {  types {  name  fields {  name  args { name type { name } }  }  }  }  } |

**Kết quả:** trả về toàn bộ các type và field, có thể thấy User.passwordHash, adminCreateUser...

**Phát hiện (pentest):** gửi introspection query; nếu nhận schema → lộ.  
**Khắc phục:** tắt introspection ở production hoặc require auth + role check trước khi trả schema; hạn chế thông tin lỗi.

**2) Broken Access Control / Field-level authorization (IDOR)**

**Ý tưởng khai thác:** server kiểm tra quyền ở cấp operation nhưng không kiểm tra từng field/resolver. Attacker dùng query hợp lệ để lấy dữ liệu của user khác hoặc các field nhạy cảm.

|  |
| --- |
| # attacker có access token user A  {  user(id: "42") {  id  name  email # email có thể là thông tin nhạy cảm  salary # field chỉ admin mới được xem, nhưng không kiểm tra  }  } |

Nếu resolver chỉ kiểm tra “user authenticated” mà không check quyền xem salary, attacker lấy được dữ liệu.

**Phát hiện:** thử truy vấn các field nhạy cảm với token user thường; fuzz các id khác; thử mutation đọc resource không thuộc quyền.  
**Khắc phục:** thực hiện authorization **ở từng resolver/field**, kiểm tra ownership + role; áp dụng “deny by default”.

**3) Excessive Data Exposure / Over-fetching of sensitive fields**

**Ý tưởng khai thác:** schema chứa field nhạy cảm (ví dụ passwordHash, ssn), client có thể request trực tiếp nếu không có kiểm soát. Attacker đoán tên field từ schema.

Ví dụ query:

|  |
| --- |
| {  user(id: "1") {  id  name  passwordHash # nếu trả về -> lộ  }  } |

**Phát hiện:** scan schema để tìm field với tên gợi ý nhạy cảm; thử query từng field.  
**Khắc phục:** loại bỏ field nhạy cảm khỏi schema hoặc luôn kiểm tra quyền trước khi trả field; mask/truncate dữ liệu.

**4) Denial of Service — Complex / Deeply Nested Queries**

**Ý tưởng khai thác:** GraphQL cho phép nested queries và large selection sets. Attacker gửi query rất sâu hoặc nhiều alias để làm server tốn CPU/DB.

|  |
| --- |
| {  user(id: "1"){  friends { # friends có resolver quay lại DB  friends {  friends {  ... (lồng nhau sâu)  }  }  }  }  } |

Hoặc dùng nhiều alias để gây nhiều truy vấn DB trong 1 request.

**Phát hiện:** gửi crafted queries với độ lồng/độ phức tạp cao và đo phản hồi/timeouts.  
**Khắc phục:** áp đặt **query depth limit**, **complexity scoring**, giới hạn kích thước request, rate-limiting, caching, throttle resolver expensive fields.

**5) Injection (SQL / NoSQL / Command)**

**Ý tưởng khai thác:** resolver chèn trực tiếp input vào truy vấn DB mà không sanitize/parametrize. Tùy impl, có thể gây SQL injection hoặc NoSQL injection.

Ví dụ (mô tả GraphQL mutation dẫn tới injection):

|  |
| --- |
| mutation {  updateUser(id: "1", bio: "...user input...") {  id  }  } |

Nếu server concat bio vào câu SQL/NoSQL query mà không parameterize, input độc hại có thể thao túng DB.

**Phát hiện:** thử các input có ký tự đặc biệt, kiểm tra lỗi/behavior khác thường; theo dõi logs DB khi gửi payloads (pentest có quyền).  
**Khắc phục:** luôn dùng parameterized queries / ORM prepared statements, validate/escape input, dùng least-privilege DB account.

**6) Batching / Alias Abuse để Bypass Rate-Limit hoặc Brute-force**

**Ý tưởng khai thác:** GraphQL cho phép aliasing — nhiều trường giống nhau trong một request. Kẻ tấn công có thể thực hiện nhiều truy vấn khác nhau trong 1 POST, né rate-limit trên endpoint per-request hạn chế.

|  |
| --- |
| {  a1: user(id: "1"){ email }  a2: user(id: "2"){ email }  a3: user(id: "3"){ email }  ...  } |

**Phát hiện:** gửi nhiều alias truy vấn trong 1 request và xem server trả; kiểm tra giới hạn request vs tổng số ops.  
**Khắc phục:** áp dụng rate-limiting dựa trên *operation complexity* hoặc *field count* chứ không chỉ số request; giới hạn số items/aliases.

**7) CSRF (với cookie-based auth)**

**Ý tưởng khai thác:** nếu endpoint dùng cookie/session auth và không có CSRF token, trang độc hại có thể ép tải query/mutation từ trình duyệt nạn nhân.

**Minh họa:** trang bên thứ 3 gửi POST tới /graphql với mutation thay đổi dữ liệu — trình duyệt kèm cookie session → mutation thực thi.

**Phát hiện:** kiểm tra auth dùng cookie không có CSRF protection; thử gửi POST từ trang khác (pentest phải cẩn trọng).  
**Khắc phục:** dùng CSRF tokens, chuyển sang token-based auth (Authorization header), kiểm tra Origin/Referer.

**8) Input Validation & Stored XSS**

**Ý tưởng khai thác:** mutation cho phép lưu HTML/JS vào trường (comment, profile), và server không sanitize → khi nội dung đó được render lên web client → stored XSS.

Ví dụ mutation:

|  |
| --- |
| mutation {  createComment(postId: "10", body: "<img onerror=alert(1)>") { id }  } |

Nếu front-end hiển thị body mà không escape, XSS xảy ra.

**Phát hiện:** thử gửi payloads có tag/script và kiểm tra nơi hiển thị.  
**Khắc phục:** sanitize input khi lưu / escape khi render ở client, CSP, validate field types/length.

**Hướng tiếp cận khi pentest GraphQL (ngắn gọn)**

1. **Tìm endpoint** (/graphql, /api/graphql, …).
2. **Thử introspection** (nếu bật → thu thập schema).
3. **Map schema**: list queries/mutations/fields/types.
4. **Kiểm tra field-level auth**: truy vấn field nhạy cảm bằng token user thấp quyền.
5. **Test injection**: input validation, observe errors.
6. **Complexity/DoS**: gửi nested/large selection queries (thận trọng, legal scope!).
7. **Batching/alias**: probe rate-limit bypass.
8. **CSRF/XSS**: test mutation side-effects and rendering.

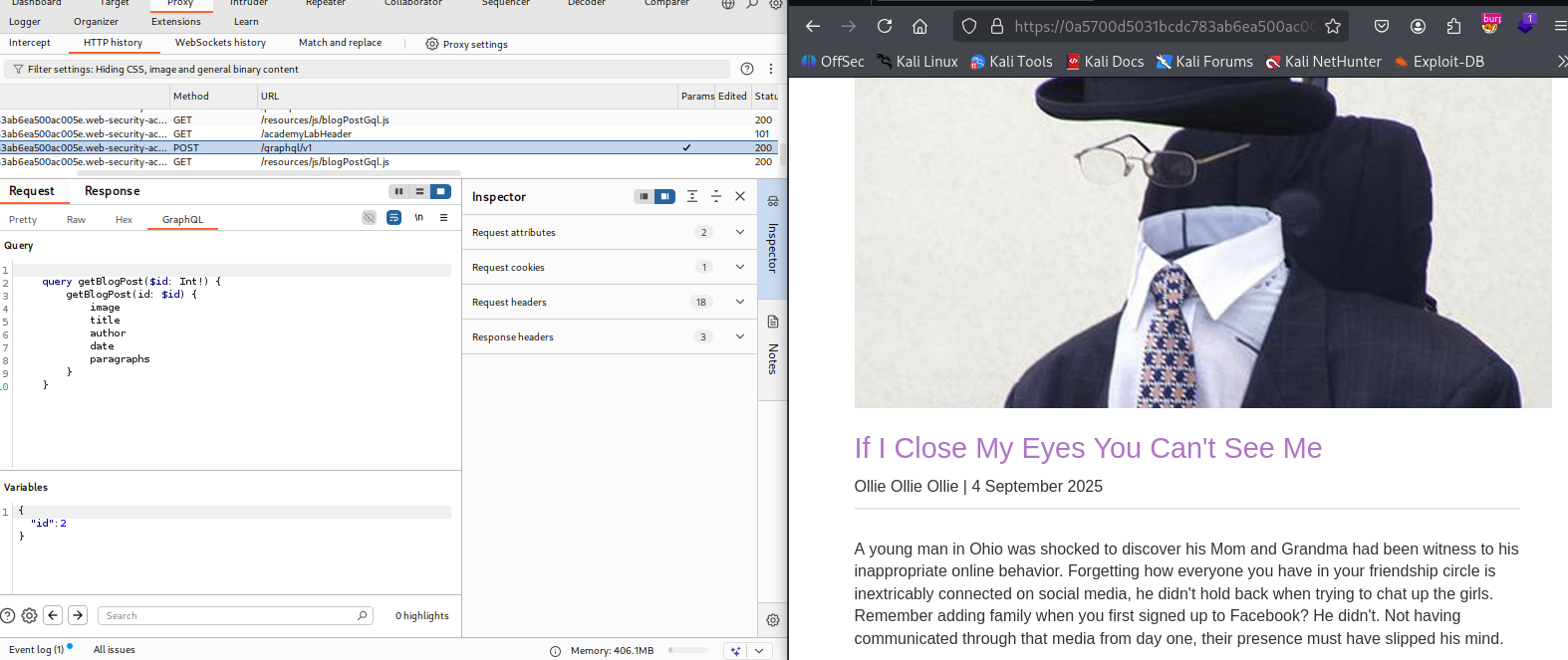
**Lab GraphQL trên PortSwigger**

**Lab: Accessing private GraphQL posts:**

Mô tả:  
The blog page for this lab contains a hidden blog post that has a secret password. To solve the lab, find the hidden blog post and enter the password.

PoC:

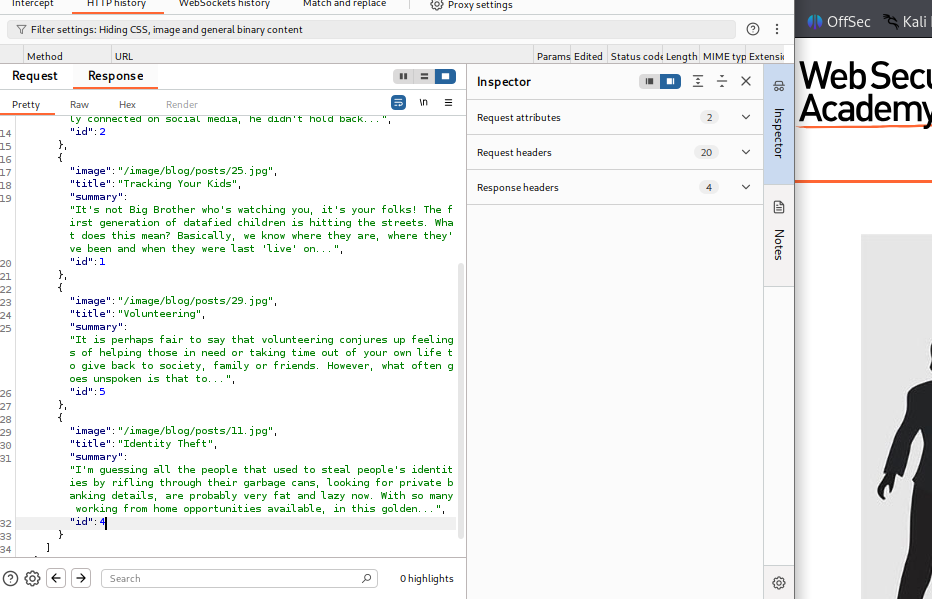
1. Thử vào xem post đâu tiên và đồng thời quan sát burp



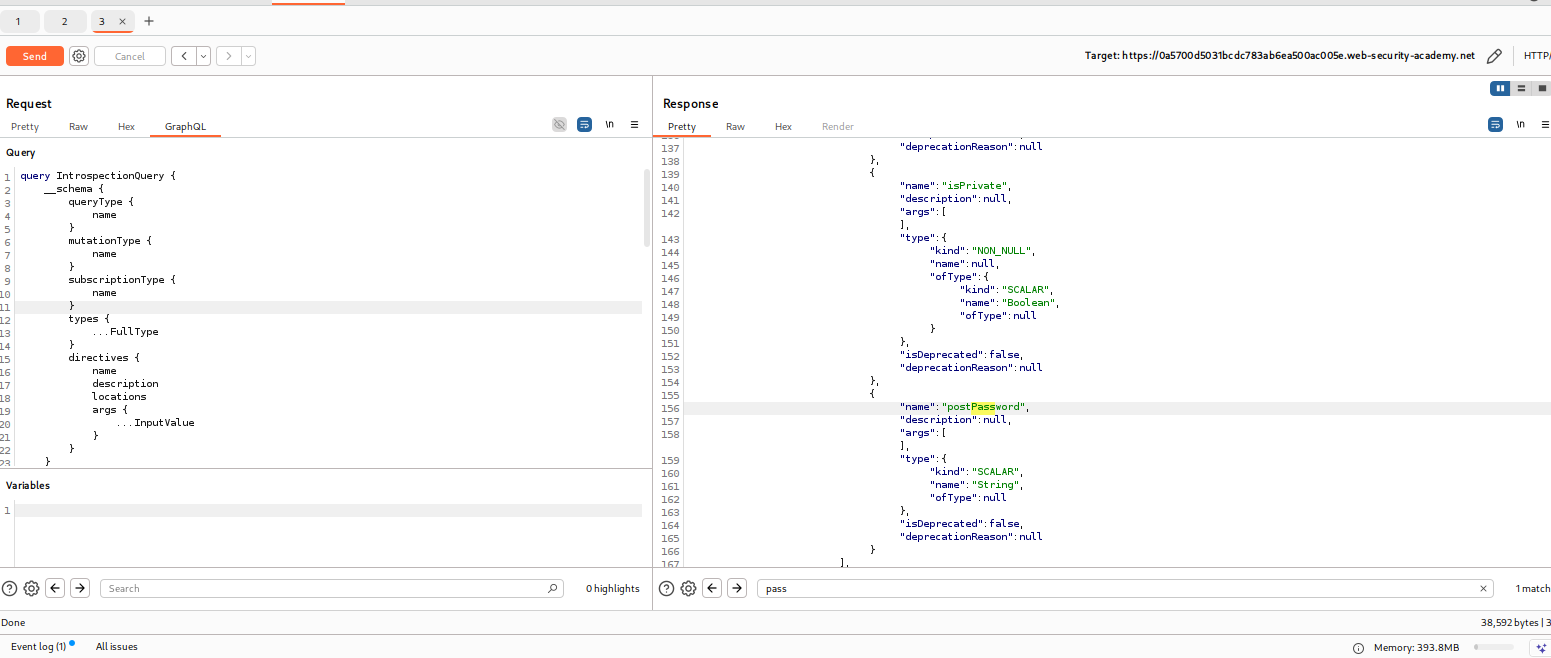
2. Trong request POST /graphql/v1  
với query:

|  |
| --- |
| "query":"\nquery getBlogSummaries {\n getAllBlogPosts {\n image\n title\n summary\n id\n }\n}","operationName":"getBlogSummaries"} |

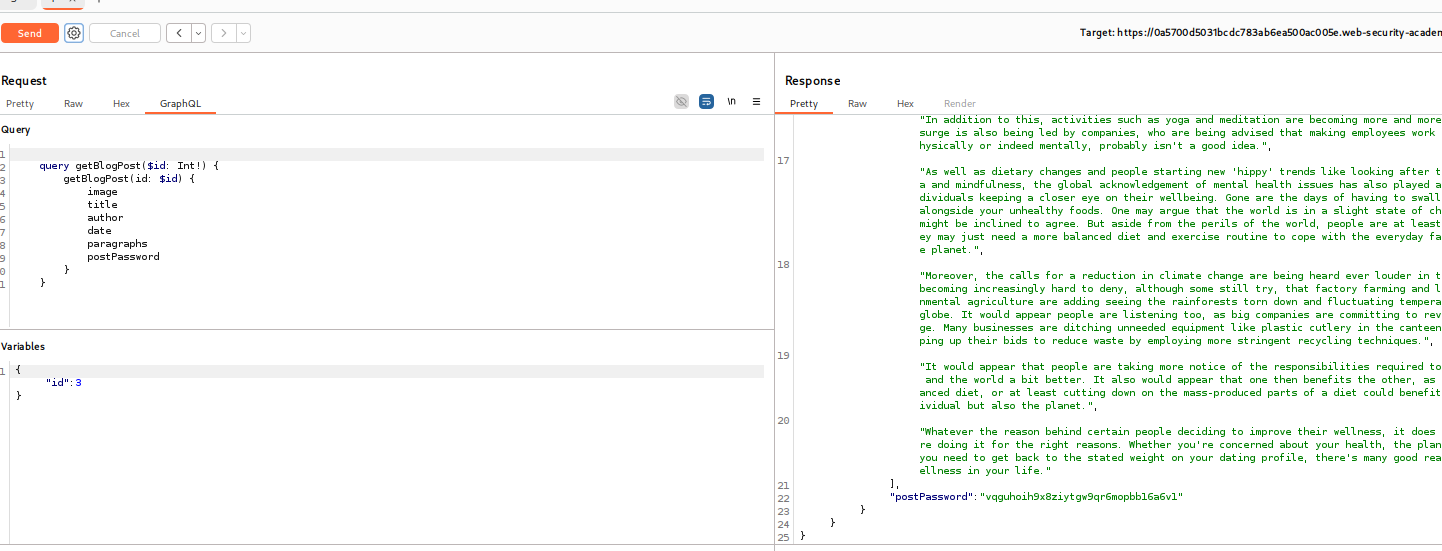
Respone: (trong phản hồi có id các post từ 1 tới 5 nhưng không có id 3 )



3. Lấy request gửi sang repeater chuyển sang graphQL chuột phải vào chọn GraphQL > Set introspection query và gửi đi .



4. Ta có id hidden là 3 và name Postpassword thử request chứa 2 giá trị này trong graphQL



**Lab: Accidental exposure of private GraphQL fields**

Mô tả :

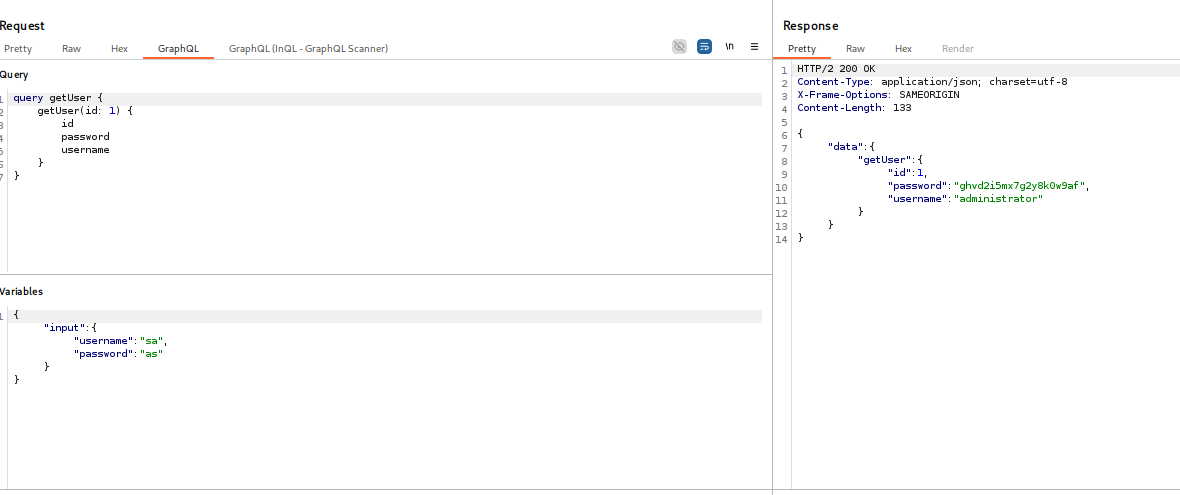
The user management functions for this lab are powered by a GraphQL endpoint. The lab contains an access control vulnerability whereby you can induce the API to reveal user credential fields.

To solve the lab, sign in as the administrator and delete the username carlos.

**Poc:**

1. Tìm endpoint graphql sau đó gửi sang repeater đồng thời copy url gửi sang InQL  


2. Copy query GetUser và paste vào GraphQl với id :1



**Lab: Finding a hidden GraphQL endpoint**

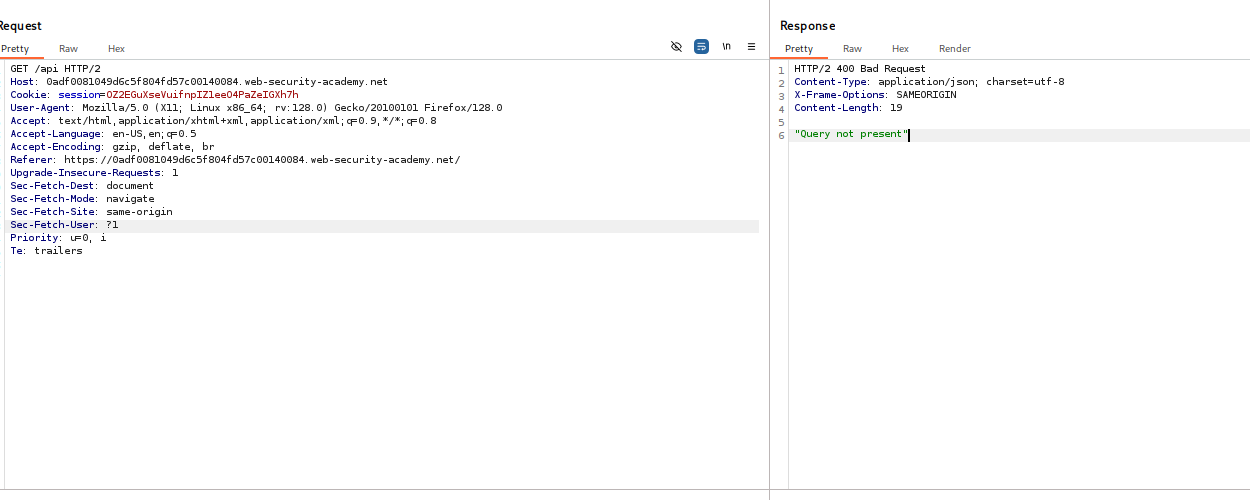
Mô tả:

The user management functions for this lab are powered by a hidden GraphQL endpoint. You won't be able to find this endpoint by simply clicking pages in the site. The endpoint also has some defenses against introspection.

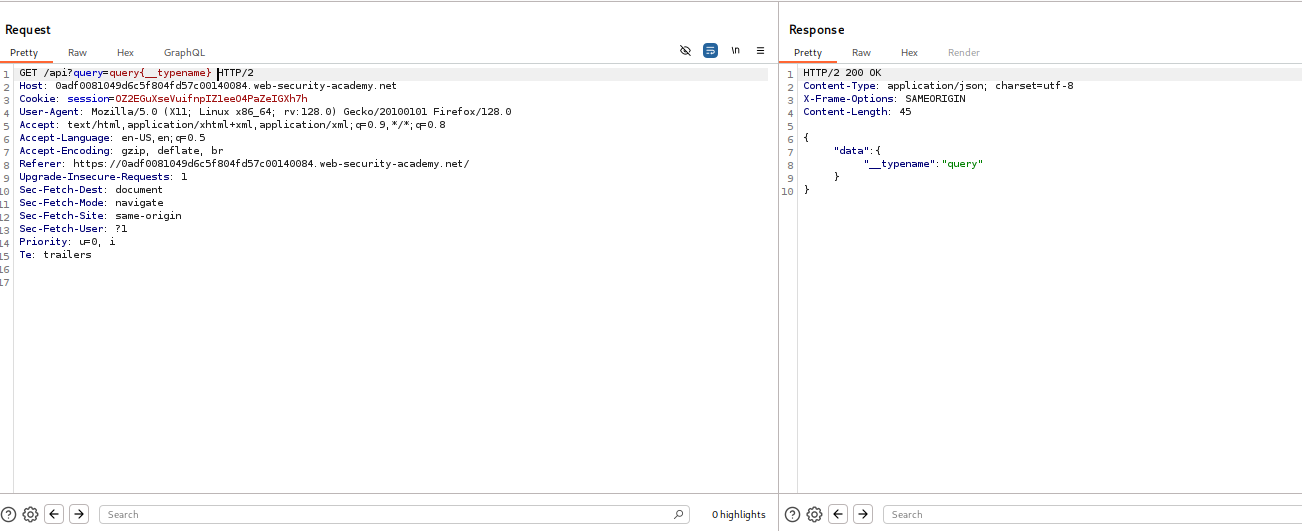
To solve the lab, find the hidden endpoint and delete carlos.

Poc:

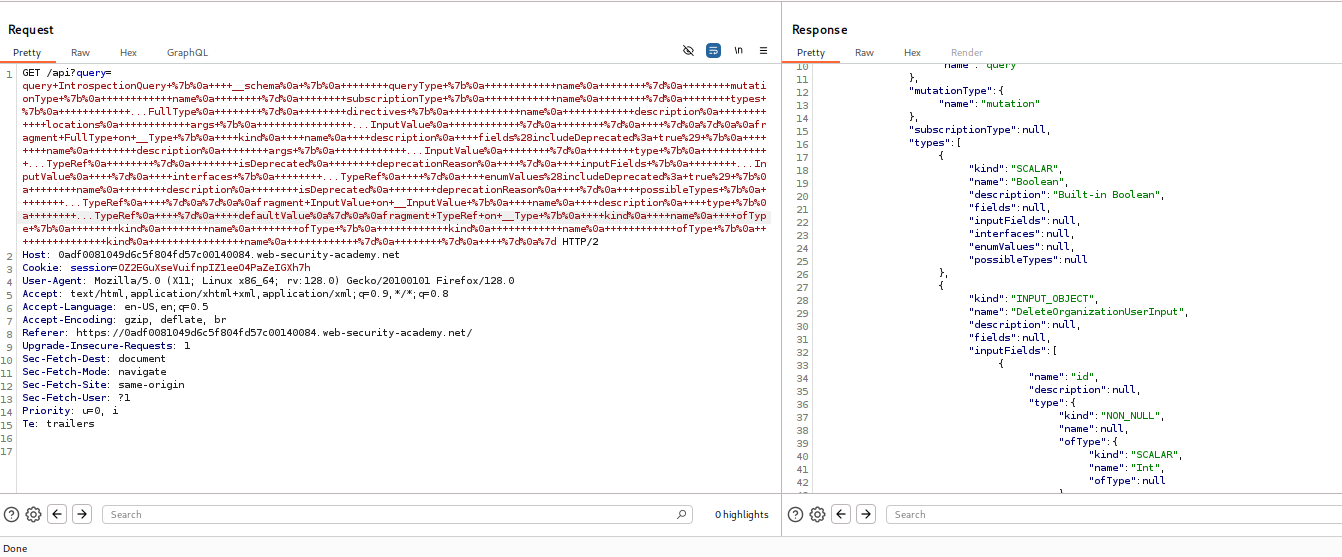
1.Tìm hidden endpoint thử gửi một endpoint api (Lưu ý rằng khi bạn gửi yêu cầu GET đến /api, phản hồi sẽ hiển thị lỗi "Không có truy vấn". Điều này cho thấy có thể có một điểm cuối GraphQL đang phản hồi các yêu cầu GET tại vị trí này.)



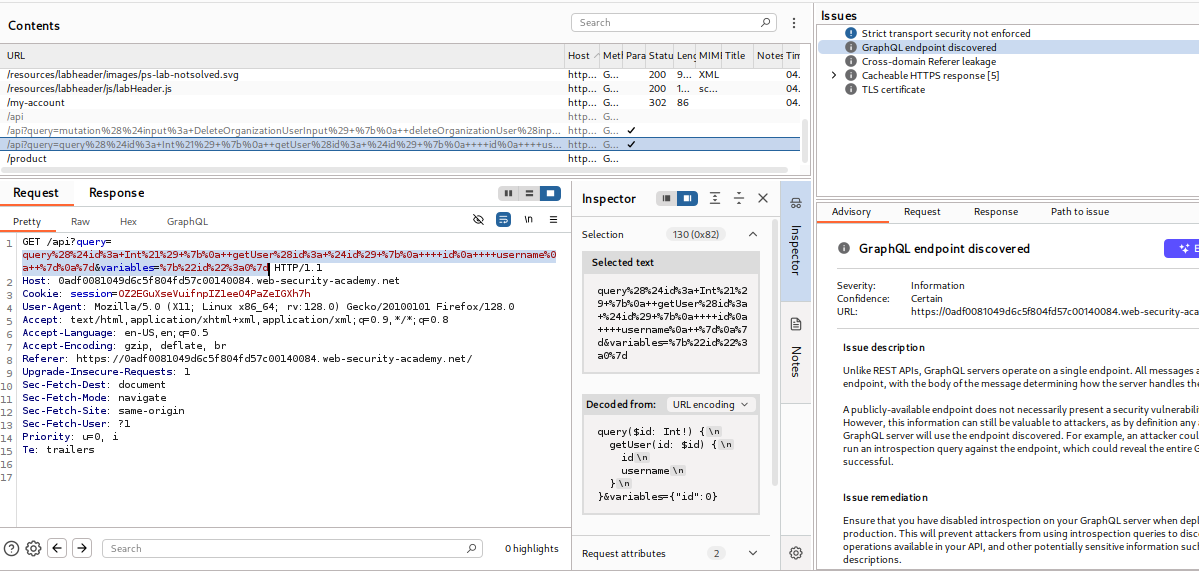
2. Tiếp theo thử một query (/api?query=query{\_\_typename}) và xem phản hồi

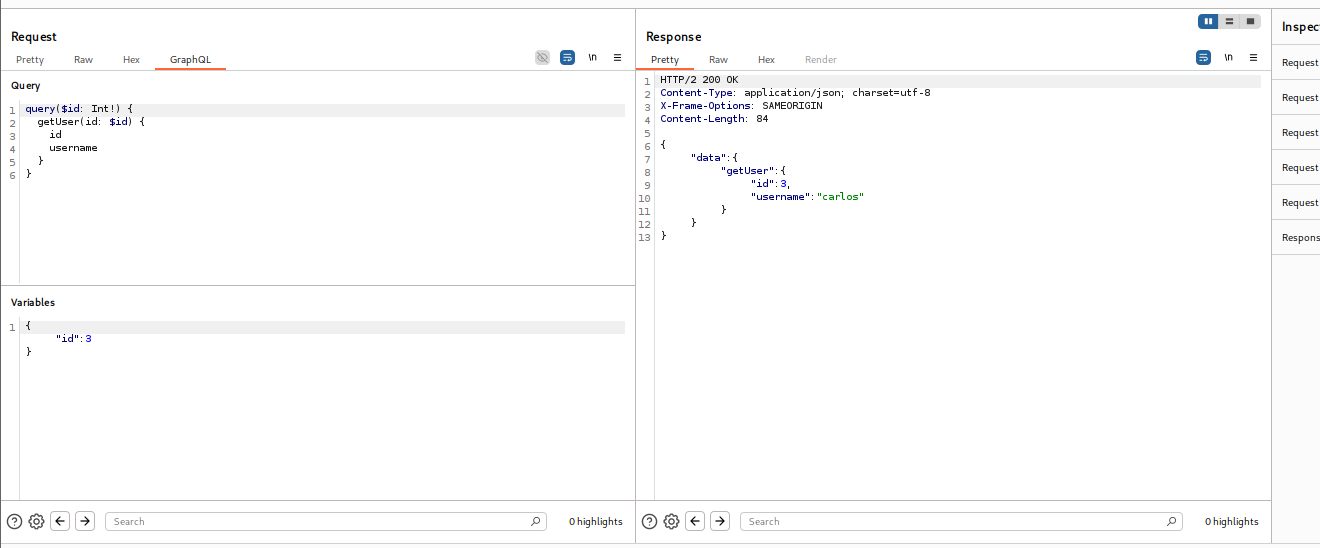


3. Chuột phải và nhấn chọn **GraphQL > Set introspection query**(Sửa đổi truy vấn để bao gồm ký tự xuống dòng sau \_\_schema%0a)

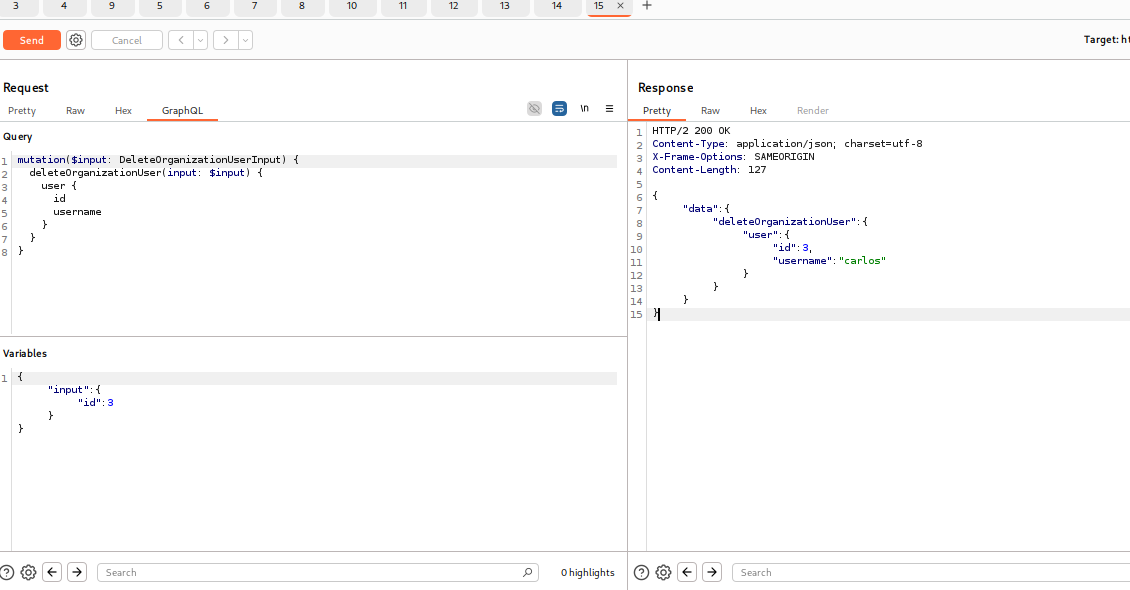


4. Sau đó **GraphQL > Save GraphQL queries to site map**. Thấy 2 request 1 là getuser 2 là delete trước hết gửi request Getuser sang repeater



5. Gửi sang repeater vào graphql thay đổi value của id và xem respone đâu là id của carlos  


6. Tiếp theo gửi request delete sang repeater và thực hiện xóa id: 3



**Lab: Bypassing GraphQL brute force protections**

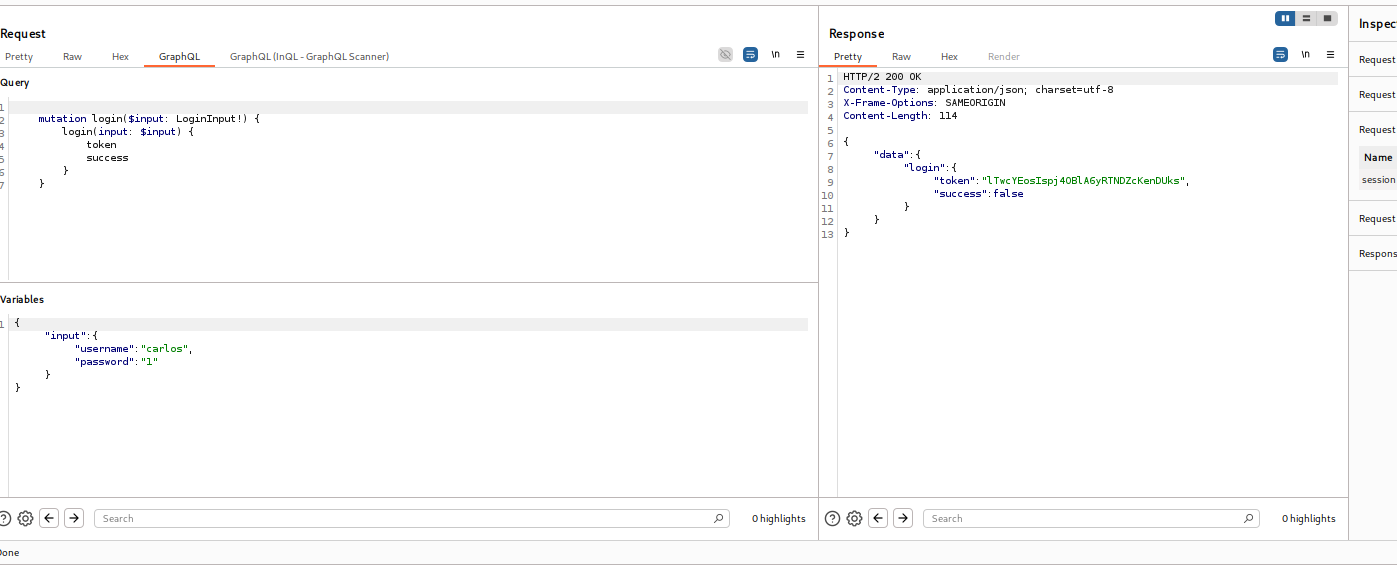
Mô tả:

The user login mechanism for this lab is powered by a GraphQL API. The API endpoint has a rate limiter that returns an error if it receives too many requests from the same origin in a short space of time.

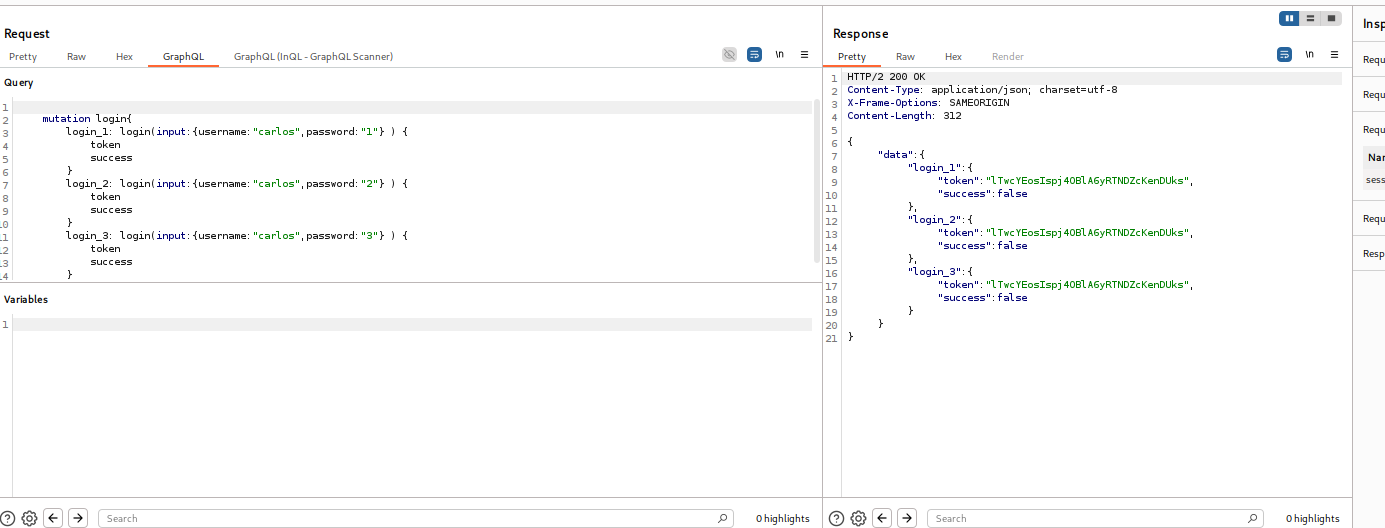
To solve the lab, brute force the login mechanism to sign in as carlos. Use the list of [authentication lab passwords](https://portswigger.net/web-security/authentication/auth-lab-passwords) as your password source.

Poc:

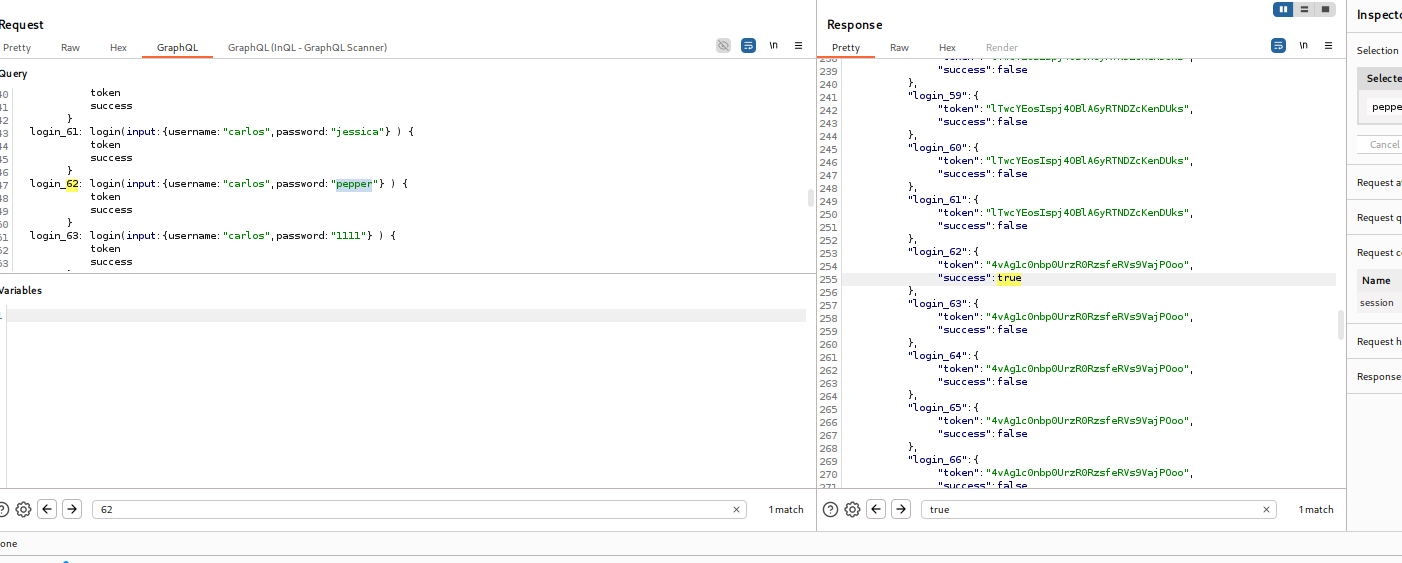
1. Thử đăng nhập và ta có được graphQL



2. Ở đây có rate limit nên không thể gửi liên tục (brude force theo cách thông thường), tuy nhiên có thể gửi 1 request với nhiều giá trị



3. Dùng python để tạo ra payload sau đó gửi request đi



**Lab: Performing CSRF exploits over GraphQL**

Mô tả:

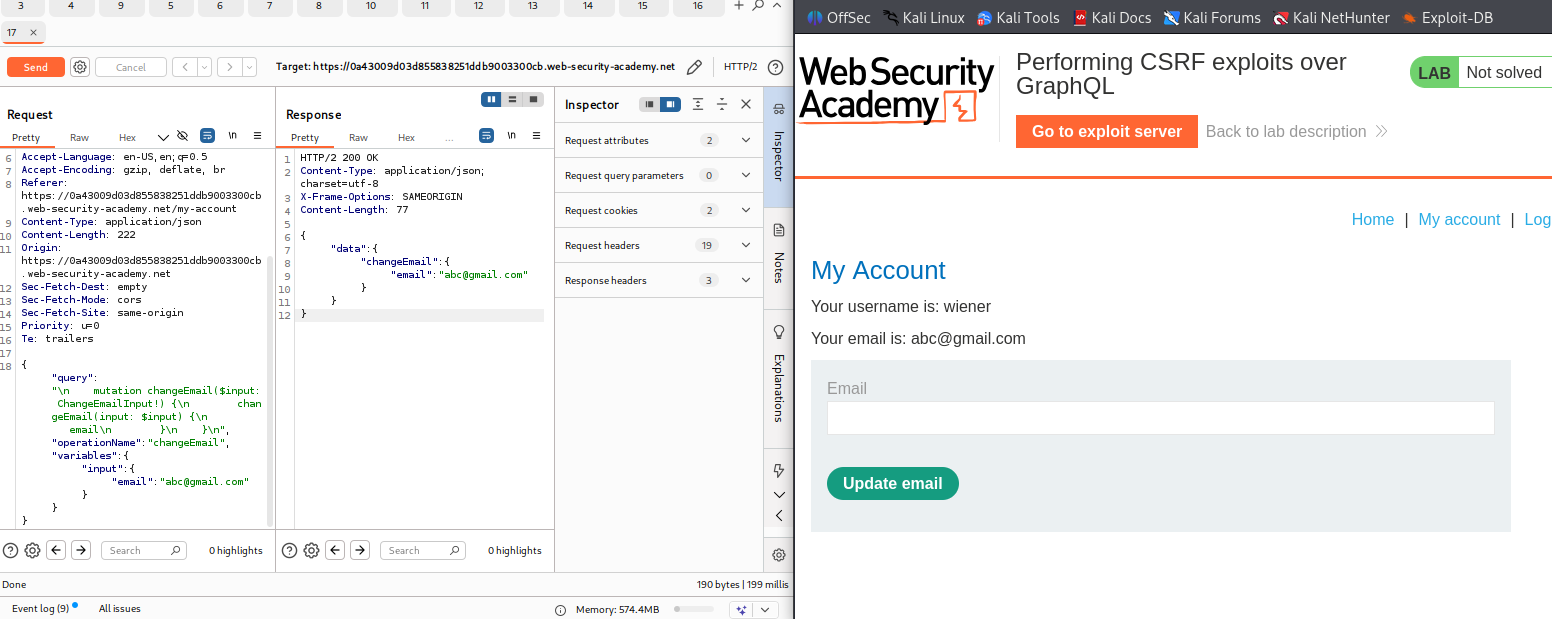
The user management functions for this lab are powered by a GraphQL endpoint. The endpoint accepts requests with a content-type of x-www-form-urlencoded and is therefore vulnerable to cross-site request forgery (CSRF) attacks.

To solve the lab, craft some HTML that uses a CSRF attack to change the viewer's email address, then upload it to your exploit server.

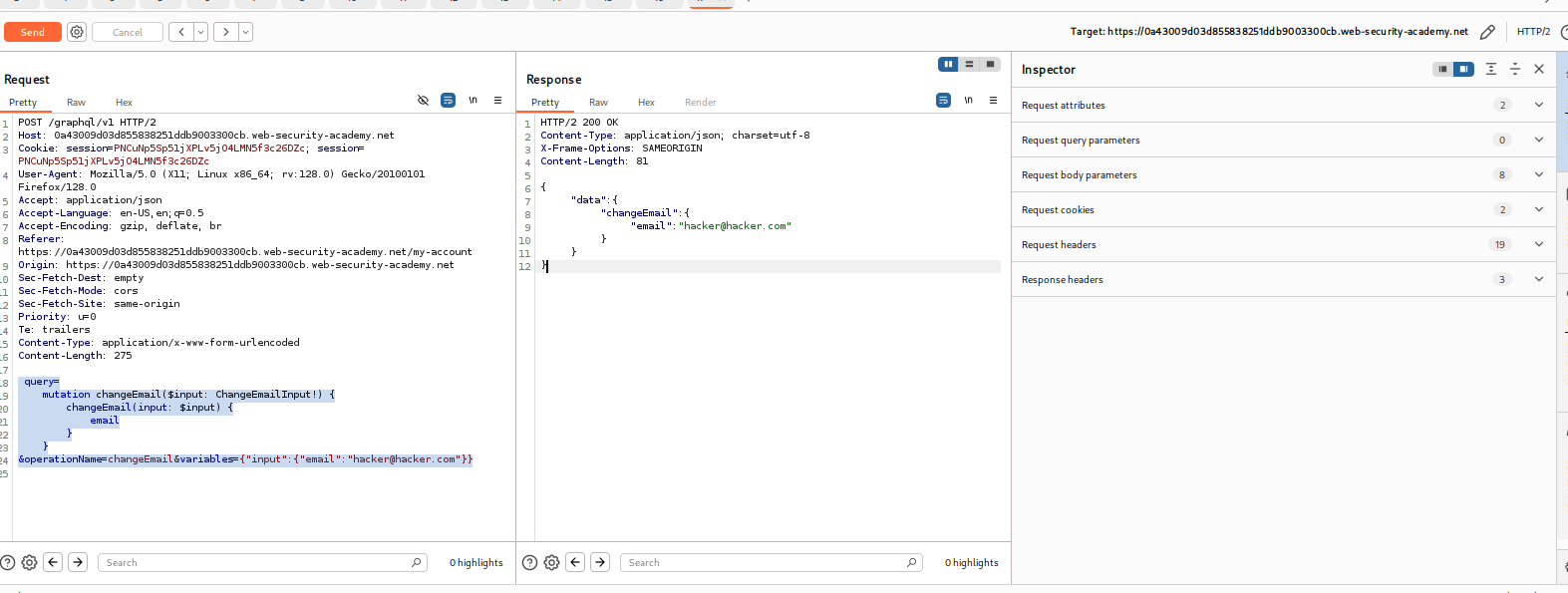
You can log in to your own account using the following credentials: wiener:peter.

Poc:

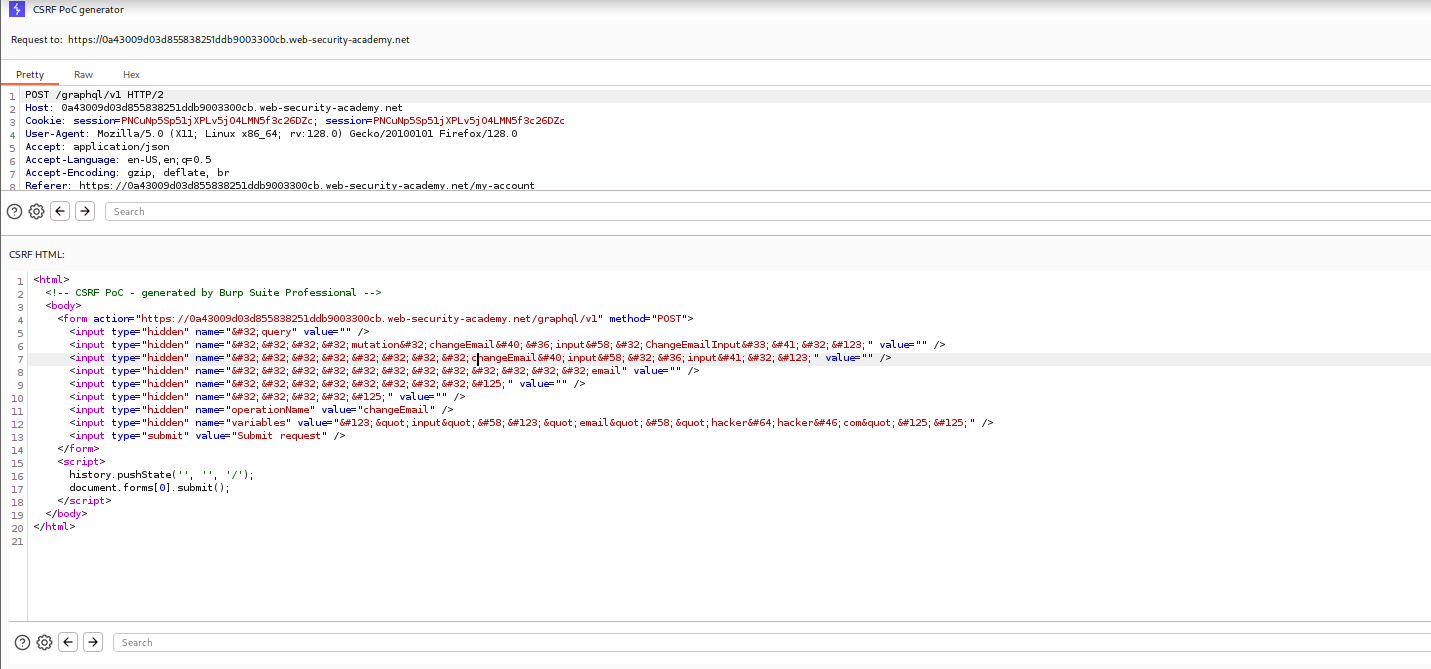
1. truy cập vào account và thay đổi email đồng thời burp sẽ nhận được 1 request graphql

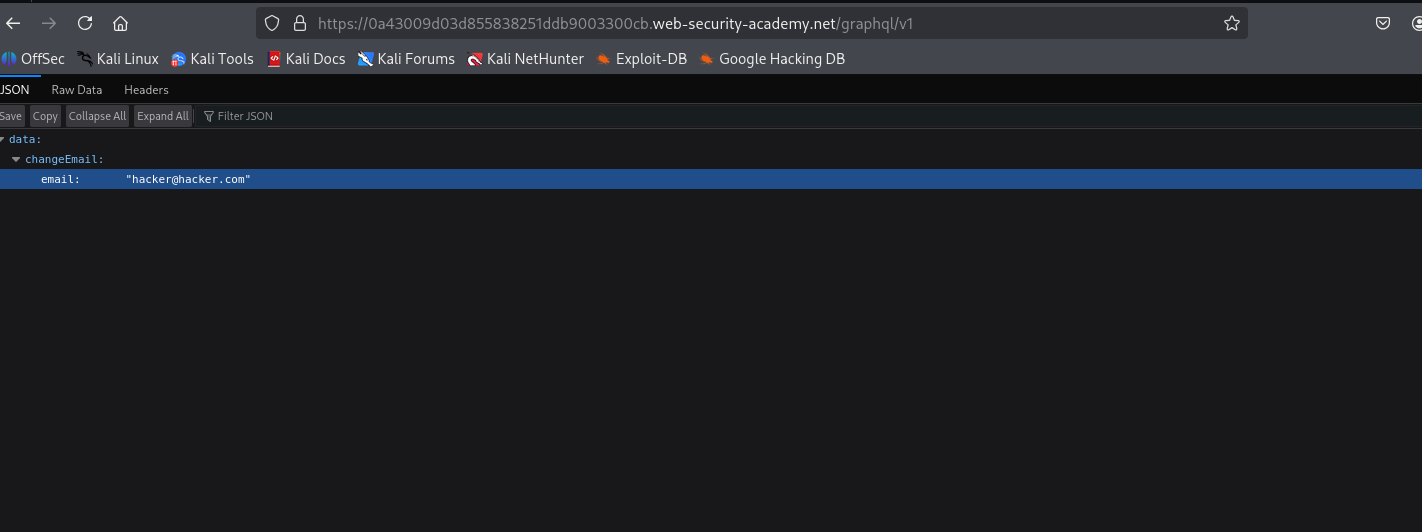


2. change request method 2 lần và thấy được x-www-form-urlencoded (tự chỉnh sửa lại body và thêm vào)



3. chuột phải chọn **Engagement tools > Generate CSRF PoC**



4. kiểm trả exploit: 

5. gửi exploit web sang victim