Bypassing GraphQL brute force protections

Executive Summary

This report highlights a vulnerability in the authentication mechanism of a web application using **GraphQL**. Despite the implementation of rate limiting to prevent brute force attacks, the protection can be bypassed by exploiting GraphQL aliasing. Using this method, multiple login attempts can be made in a single request, allowing an attacker to enumerate credentials and successfully log in as another user. The severity of this issue is **high** due to the ease of account compromise.

Introduction

This test was conducted to evaluate the effectiveness of brute force protection on a GraphQL-based login API. The goal was to identify if the login endpoint could be manipulated to bypass rate-limiting controls and compromise user credentials through a multi-request attack strategy.

Methodology

1. Logged into the target application via Burp's browser and accessed the login form.

2. Attempted standard login attempts to observe the behaviour of failed authentication.

3. Captured the GraphQL login mutation using Burp Suite's HTTP history.

4. Sent the login request to Repeater for further testing.

5. Noticed rate-limiting errors after multiple individual login attempts.

6. Constructed a single GraphQL request using aliases to submit multiple login mutations simultaneously.

7. Used a list of common passwords in the request.

8. Identified the successful login mutation by searching for `"success": true` in the response.

9. Used the discovered password to log in as ‘user’ and complete the challenge.

Vulnerability Findings

- **Type:** Brute Force Protection Bypass

- **Location**: GraphQL login mutation endpoint

- **Severity**: High

Description:

The login mechanism for the web application was implemented via a GraphQL mutation. While the server enforces rate limiting on a per-request basis, it failed to account for GraphQL aliasing, which allows multiple operations in a single request. An attacker could exploit this by submitting dozens of credential guesses in one GraphQL mutation, effectively bypassing the rate limiter and brute forcing user passwords undetected.

Proof of Concept (PoC)

Example GraphQL request using aliases:

graphql

CopyEdit

Steps

1. Sent the request via Repeater.

2. Scanned the response for `"success": true`.

3. Identified the correct password.

4. Logged in successfully as the user.

Impact Assessment

- Full user account takeover.

- Bypass of intended security controls (rate limiting).

- Potential for large-scale credential stuffing attacks if not mitigated.

- High risk of unauthorized access to sensitive data and user privileges.

Recommendations

1. Implement server-side detection of excessive login attempts, even within a single GraphQL request.

2. Restrict the number of GraphQL aliases or operations allowed per request.

3. Apply per-user and per-IP rate limits based on authentication events rather than request count.

4. Validate GraphQL payload size and complexity to prevent abuse.

5. Monitor login patterns and generate alerts for unusual authentication behaviour.

Conclusion

This assessment revealed a critical weakness in the application's authentication flow due to improper handling of complex GraphQL requests. Strengthening GraphQL protections and applying layered rate-limiting mechanisms are essential steps to prevent such exploitation and safeguard user accounts.