CyberSecurity dan Forensik Digital e-ISSN: 2615-8442
Vol. 7, No. 2, November 2024, hlm. 62-68

Peningkatan Keamanan Server GraphQL Terhadap Serangan DDOS Dengan Tipe *Batch Attack* Menggunakan Metode *Rate Limiting*

Diash Firdaus[1], Idi Sumardi[2], Ginanjar Nugraha[3]

[1]Informatics, Institut Teknologi Nasional, Bandung, Indonesia
[2,3] Informatics Engineering, STMIK JABAR, Bandung, Indonesia

[*]Email: [1]Diash@itenas.ac.id, [2]Idis@stmikjabar.ac.id, [3]ginz@stmikjabar.ac.id,

Abstrak

GraphQL telah memperkenalkan pergeseran paradigma tentang bagaimana aplikasi berkomunikasi dengan data, menawarkan opsi yang lebih efisien dan ampuh dibandingkan dengan RESTful API tradisional. Namun, atribut yang membuat GraphQL fleksibel dan efisien juga dapat membuatnya rentan terhadap ancaman siber yang ditargetkan, termasuk serangan batch. Eksploitasi ini memanfaatkan kemampuan untuk menggabungkan beberapa kueri atau mutasi ke dalam satu permintaan HTTP, yang dapat menyebabkan server kelebihan beban. Di berbagai industri, termasuk di Facebook, tempat kelahiran GraphQL, teknologi ini digunakan untuk menangani pertukaran data yang rumit antara aplikasi dan basis pengguna yang luas di seluruh dunia. Pembatasan kecepatan muncul sebagai penanggulangan yang tangguh terhadap ancaman serangan batch. Dengan membatasi frekuensi permintaan yang dapat dilakukan pengguna dalam interval waktu tertentu, pembatasan laju melindungi kinerja dan waktu aktif server sekaligus menggagalkan penyalahgunaan. Pendekatan ini tidak hanya membantu dalam manajemen sumber daya server yang bijaksana tetapi juga bertindak sebagai pencegah terhadap aktor jahat yang ingin memanfaatkan sistem. Data empiris mengungkapkan bahwa pembatasan laju efektif dalam mengurangi beban CPU dan Memori secara substansial, mengurangi penggunaan CPU rata-rata dari 4,8% menjadi 0,86% dan penggunaan Memori dari 87MB menjadi 49,6MB selama serangan. Sebaliknya, server tanpa pembatasan kecepatan mengalami lonjakan konsumsi CPU dan Memori setiap beberapa detik, sedangkan dengan pembatasan kecepatan, lonjakan seperti itu terbatas pada 5 detik awal. Bukti ini menggarisbawahi bahwa pembatasan kecepatan memungkinkan server untuk mempertahankan kinerja dan ketersediaan dalam menghadapi potensi serangan.

Kata kunci: DdoS, GraphQL, Batch Attack

Enhancing GraphQL Server Security Against Batch Attack Using The Rate-Limiting Method

Abstract

GraphQL has introduced a paradigm shift in how applications communicate with data, offering a more streamlined and potent option compared to traditional RESTful APIs. However, the very attributes that make GraphQL flexible and efficient can also render it vulnerable to targeted cyber threats, including batch attacks. These exploits leverage the capability to bundle multiple queries or mutations into a single HTTP request, which can lead to server overload. Across various industries, including at Facebook, the birthplace of GraphQL, this technology is employed to handle intricate data exchanges between applications and a vast user base worldwide. Rate limiting emerges as a formidable countermeasure to the threat of batch attacks. By capping the frequency of requests a user can initiate within a specified time interval, rate limiting safeguards server performance and uptime while thwarting misuse. This approach not only aids in the judicious management of server resources but also acts as a deterrent against malicious actors seeking to take advantage of the system. The empirical data reveals that rate limiting is effective in substantially reducing the strain on CPU and Memory, decreasing average CPU usage from 4.8% to 0.86% and Memory usage from 87MB to 49.6MB during an attack. In contrast, servers without rate limiting experience a surge in CPU and Memory consumption every few seconds, whereas with rate limiting, such a spike is confined to the initial 5 seconds. This evidence underscores that rate limiting enables servers to sustain performance and availability in the face of potential attacks.

***Keywords**: DdoS, GraphQL, Batch Attack*

1. INTRODUCTION

GraphQL has revolutionized the way applications interact with data by providing a more efficient and powerful alternative to RESTful APIs (Brito and Valente 2020). However, its very flexibility and efficiency can also make it susceptible to specific types of cyber attacks, such as batch attacks. These attacks exploit the ability to send multiple queries or mutations in a single HTTP request, potentially overwhelming the server. In various industrial sectors, for instance, Facebook, which pioneered GraphQL, utilizes it to manage complex data interactions between its applications and millions of global users (Hanif et al. 2022). By employing GraphQL, Facebook ensures that applications only fetch the precise data required, avoiding over-fetching, which is crucial given the scale and complexity of the data they manage (Quiña-Mera et al. 2023).

On another front, GitHub has integrated GraphQL into their public API to offer developers greater control over the data they request(Github n.d.). This enables developers to make highly specific requests and reduce the volume of unnecessary data, which in turn facilitates more efficient integration and improved performance.

In all these cases, GraphQL offers significant advantages over traditional approaches such as REST. By allowing clients to precisely specify the data they need, applications can reduce network overhead and accelerate responses to user requests, while also simplifying the development process by minimizing the number of API requests that need to be made and reducing complexity on the server side(Muzaki and Salam 2024).

Batch attacks on GraphQL servers can degrade performance, lead to denial of service (DoS), and even compromise sensitive data(McFadden et al. 2024). As the adoption of GraphQL continues to grow across various industries, the imperative to safeguard these systems against such vulnerabilities becomes increasingly critical(Ogboada, VIE, and Matthias 2021).

Rate limiting is a robust defense mechanism that can mitigate the risk posed by batch attacks. Limiting the number of requests a user can make within a certain time frame, it helps maintain server performance and availability while protecting against abuse. This method not only helps in managing server resources efficiently but also serves as a deterrent against attackers looking to exploit the system(Ren et al. 2020)(Mahjabin et al. 2017).

In this paper, we will explore how rate limiting can be strategically implemented to enhance the security of GraphQL servers(Yunus 2019). We will discuss the technical challenges posed by batch attacks, the principles behind rate limiting, and how they can be effectively implemented in a GraphQL context. We will also examine case studies where rate limiting has successfully mitigated batch attacks, drawing lessons on best practices and potential pitfalls. The insights gained will be valuable for developers, security professionals, and IT managers responsible for maintaining GraphQL servers.

This study focuses on strengthening GraphQL server security against batch attacks by implementing the rate-limiting method. Batch attacks exploit the ability of GraphQL to handle multiple queries in a single request, potentially overwhelming server resources and affecting overall system performance. By limiting the frequency of requests allowed within a certain timeframe, rate limiting acts as a targeted defense mechanism that can significantly reduce CPU and memory usage during an attack. This approach not only preserves server functionality under high demand but also helps mitigate resource exhaustion, ensuring a stable and secure environment for GraphQL operations.

2. Theoretical Foundations

2.1. GraphQL Technology

GraphQL, developed by Facebook in 2012 and released publicly in 2015, revolutionizes API querying and data manipulation by treating data as a graph, allowing clients to retrieve exactly what they need through a single API call. This approach not only minimizes over-fetching and under-fetching but also enhances performance by reducing unnecessary data transfer and processing (Meta 2015).

In GraphQL, a strongly typed schema acts as a contract between the client and the server, ensuring that both sides understand the structure of the data being exchanged. This schema defines various types, including scalars, enums, and objects, which detail the data's shape and the operations available. Queries in GraphQL are used for fetching data and are highly customizable, letting clients specify exactly which data fields to retrieve, which is particularly beneficial for complex systems with interconnected data. On the other hand, mutations provide a structured way to modify data, similar to POST, PUT, and DELETE methods in REST APIs, but with clearer expectations of the input and output (Graphql 2024).

Resolvers play a crucial role in GraphQL's architecture, acting as functions that connect API queries and mutations to the actual data in databases or other data sources. This setup allows GraphQL to be extremely flexible in how data is stored and retrieved, supporting various backend structures without requiring changes on the client side.

Despite its advantages, GraphQL introduces challenges in performance management, security, and caching. Complex queries can potentially generate significant loads on servers by requesting large amounts of interconnected data or deeply nested relationships. Unlike REST, which can leverage simple HTTP caching mechanisms, GraphQL requires more sophisticated, often custom, caching solutions to maintain efficiency and performance.

Moreover, the powerful querying capabilities also necessitate robust security measures to prevent abusive requests and data breaches.

2.2. Batch Attack / DoS Attack

Batch attacks represent a significant cybersecurity threat that capitalizes on the specific characteristics of batch processing systems, where multiple transactions or operations are collected and processed together at scheduled intervals. Unlike systems that process transactions in real-time, batch systems are inherently vulnerable because they often lack immediate feedback mechanisms, which delays the detection of anomalous activities. This characteristic makes them attractive targets for cyber attackers looking to exploit any delay in response or gaps in security monitoring (Firdaus and Rianti 2023).

The primary method of exploitation in batch attacks involves leveraging the predictable nature of batch processing schedules. Attackers who understand when batch processes run can time their malicious activities to coincide with these processes, potentially maximizing the damage or theft before detection. For instance, an attacker might insert malicious commands or corrupt data into a batch of transactions knowing that the system will automatically process all included items at a specific time.

2.3. Security in GraphQL APIs

The study by Brito and Valente examines the performance comparison between GraphQL and RESTful APIs, as well as how GraphQL enhances data efficiency while introducing additional security risks. The flexibility and querying capabilities of GraphQL make it vulnerable to exploitation through batch attacks that can overload servers. This highlights the importance of implementing security measures such as rate limiting to address these vulnerabilities in GraphQL APIs (Brito and Valente 2020).

2.4. Rate Limiting as a Defense Mechanism against DDoS Attacks

The study conducted by Mahjabin et al. discusses various techniques for preventing and mitigating DDoS attacks, including the use of rate limiting as an effective approach. Rate limiting helps control the number of requests within a specific period, maintaining server stability and preventing damage from excessive requests, particularly in GraphQL environments that are susceptible to DDoS-like attack patterns (Mahjabin et al. 2017).

2.5. Techniques and Strategies for Mitigating Attacks on GraphQL

McFadden et al. explore the use of reinforcement learning to detect malicious queries in GraphQL to prevent Denial-of-Service (DoS) attacks. This machine learning-based approach allows for dynamic identification of attack patterns, which can complement rate limiting in managing security within GraphQL environments. This study emphasizes the importance of combining mitigation approaches to strengthen the resilience of GraphQL servers against cyber threats (McFadden et al. 2024).

3. RESEARCH METHOD

The flowchart illustrates a structured process for research or project development, delineating a sequence of steps from initiation to conclusion. Figure 1 is a flow for the research method.

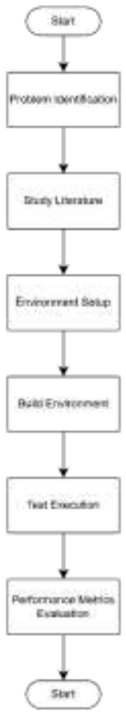


Figure 1 Research Method Flow

3.1. Problem Identification

in this step, the specific problem or research question that needs to be addressed is identified. This involves recognizing and defining the key issues or challenges that the project aims to solve or investigate.

3.2. Study Literature

This phase involves conducting a comprehensive review of existing literature related to the identified problem. The goal is to gather relevant information, theories, and previous research findings that can provide a foundation for understanding the problem and guiding the project.

3.3. Environment Setup

During this step, the necessary environment for conducting the project is established. This includes setting up tools, software, hardware, and any other resources required to carry out the research or development activities.

Table 1. Environment Setup

No	Name	Version
1	Operating System	Windows 11
2	Client	Node JS v20
3	Server	
4	Supporting Tools	Library Express
		Library GraphQL
		Library Pidusage Library Express-Rate-Limit
5	Hardware	CPU AMD Ryzen 7 5800 RAM 16 GB

Figure 2 represents the topology for the research conducted.

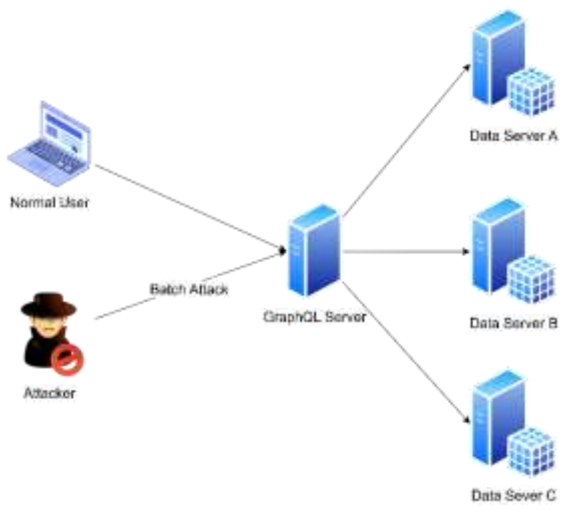


Figure 2 Topology GraphQL Experiment

3.4. Build Environment

This stage involves the actual construction or configuration of the environment as per the requirements identified in the previous step. It ensures that all components are correctly installed and operational for subsequent phases. When constructing the environment from section 2.3, we utilized one normal client, one client acting as a batch attack attacker, one server without rate limiting, and one server employing rate limiting. The attacker will send 100 GET messages within 5 seconds.

3.5. Test Execution

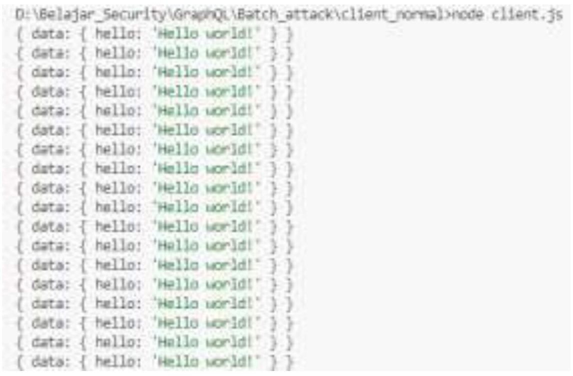
After setting up the environment, we'll run tests to make sure everything is working right. This means doing experiments, simulations, or other tests to collect data and check that our setup is good. During this time, regular users will send messages like they normally would, but we'll also have an attacker sending a bunch of GET data to the server—100 messages every 5 seconds for a whole minute. This could cause problems for the server.

3.6. Performance Evaluation Metrics

In this phase, the performance metrics obtained from the test execution are analyzed and evaluated. This involves assessing the data against predefined criteria or benchmarks to determine the success and effectiveness of the project in addressing the identified problem. When measuring performance, we will look at CPU usage and Memory Usage.

4. RESULT AND DISCUSSION

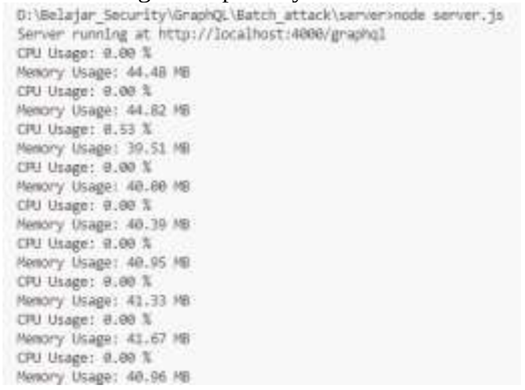
In Figure 3, we can see messages being sent through the normal client at a rate of 1 request per second.



```
D:\Belajar_Security\GraphQL\Batch_attack\client_normal>node client.js
{ data: { hello: 'Hello world!' } }
{ data: { hello: 'Hello world!' } }
{ data: { hello: 'Hello world!' } }
{ data: { hello: 'Hello world!' } }
{ data: { hello: 'Hello world!' } }
{ data: { hello: 'Hello world!' } }
{ data: { hello: 'Hello world!' } }
{ data: { hello: 'Hello world!' } }
{ data: { hello: 'Hello world!' } }
{ data: { hello: 'Hello world!' } }
{ data: { hello: 'Hello world!' } }
{ data: { hello: 'Hello world!' } }
{ data: { hello: 'Hello world!' } }
{ data: { hello: 'Hello world!' } }
{ data: { hello: 'Hello world!' } }
{ data: { hello: 'Hello world!' } }
{ data: { hello: 'Hello world!' } }
{ data: { hello: 'Hello world!' } }
```

Figure 3 Normal Client connect to GraphQL

Meanwhile, Figure 4 shows the CPU usage and Memory usage when data is sent using the normal client. The data set shows how the system's resources are being used while running a server, possibly during a batch attack. The CPU usage stays at 0.00%, meaning there's little processing activity happening. Memory usage starts at 44.48 MB and varies slightly between 39.51 MB and 44.82 MB. This suggests the system is managing tasks without much stress, as seen by the steady CPU and small changes in memory use. The lack of CPU spikes means the server isn't facing heavy computation, possibly because it's early in the attack and the system is being tested. The stable memory usage shows good memory management. However, it's important to keep monitoring and analyzing the system to catch any issues if the attack continues, using strategies like rate-limiting and real-time monitoring to keep the system safe and stable.



```
D:\Belajar_Security\GraphQL\Batch_attack\server>node server.js
Server running at http://localhost:4000/graphql
CPU Usage: 0.00 %
Memory Usage: 44.48 MB
CPU Usage: 0.00 %
Memory Usage: 44.82 MB
CPU Usage: 0.53 %
Memory Usage: 39.51 MB
CPU Usage: 0.00 %
Memory Usage: 40.00 MB
CPU Usage: 0.00 %
Memory Usage: 40.39 MB
CPU Usage: 0.00 %
Memory Usage: 40.95 MB
CPU Usage: 0.00 %
Memory Usage: 41.33 MB
CPU Usage: 0.00 %
Memory Usage: 41.67 MB
CPU Usage: 0.00 %
Memory Usage: 40.96 MB
```

Figure 4 CPU and Memory usage when data send from normal client

Figure 5 is a chart of CPU and Memory usage when the GraphQL server is accessed by the Normal client. The graph shows how a server's CPU and memory are used over time. The blue line for memory usage stays pretty steady around 40 MB, meaning the system is managing its tasks well without much stress. The red line, showing CPU usage, mostly stays near 0% with a few small spikes, indicating occasional processing activity that doesn't strain the system. Overall, both CPU and memory use are stable, suggesting the server is running smoothly. It's important to keep watching these patterns to catch any changes early and keep the system running well and securely.

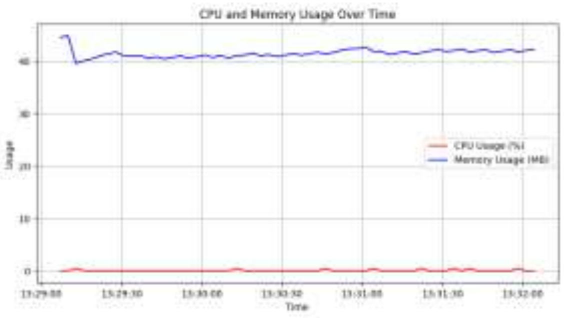
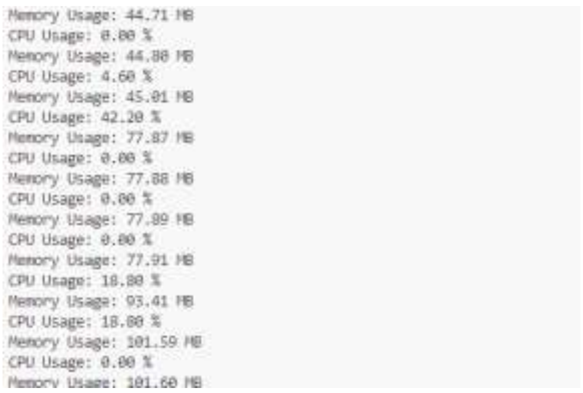


Figure 5 Comparison Chart of CPU and Memory Usage by the Normal client

Figure 6 shows a quick increase in memory usage from 44.71 MB to 101.60 MB in just 10 seconds, indicating a heavy load on the system, likely due to a batch attack. At first, CPU usage is at 0.00%, but it occasionally jumps up to 42.20%, showing bursts of activity. This pattern suggests the system might slow down, as high memory use can drain resources and fluctuating CPU activity can lead to unreliable performance. To tackle these issues, it's recommended to use strategies like rate-limiting, better memory management, and real-time monitoring to keep the system stable and prevent disruptions.



```
Memory Usage: 44.71 MB
CPU Usage: 0.00 %
Memory Usage: 44.88 MB
CPU Usage: 4.60 %
Memory Usage: 45.01 MB
CPU Usage: 42.20 %
Memory Usage: 77.87 MB
CPU Usage: 0.00 %
Memory Usage: 77.88 MB
CPU Usage: 0.00 %
Memory Usage: 77.89 MB
CPU Usage: 0.00 %
Memory Usage: 77.91 MB
CPU Usage: 18.80 %
Memory Usage: 93.41 MB
CPU Usage: 18.80 %
Memory Usage: 101.59 MB
CPU Usage: 0.00 %
Memory Usage: 101.60 MB
```

Figure 6 CPU and Memory usage when under a Batch Attack

It can be seen in Figure 7 that there is an increase in CPU and Memory usage, with the highest Memory usage reaching 110MB, compared to 48MB for a normal user. The highest CPU usage during a Batch Attack is 43%, whereas for a normal user, it remains at 1,6%. Therefore, Batch Attacks significantly impact CPU and Memory availability.

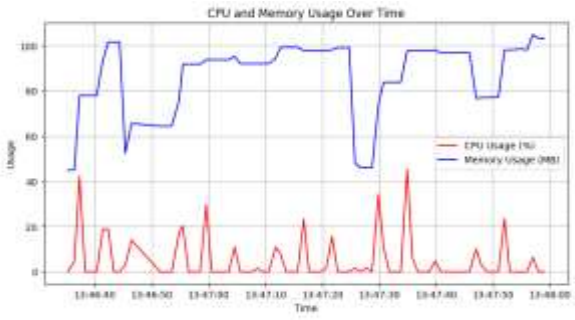
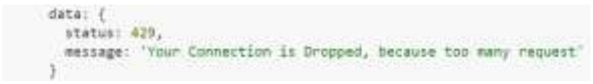


Figure 7 Server's graph when experiencing a Batch Attack

Figure 8 shows the status received by the attacker when rate limiting is added to the server, limiting to a maximum of 100 packets per 5 seconds. Consequently, if a client sends more than 100 packets within 5 seconds, packet dropping will occur, and the attacker will not receive any data from the GraphQL server.



```
data: {
  status: 429,
  message: 'Your Connection is Dropped, because too many request'
}
```

Figure 8 status received by the attacker when rate limiting is added to the server

Figure 9 shows the CPU and Memory usage graph when the server is protected by rate limiting. The spike in CPU and Memory usage only occurs at the beginning when the GraphQL server is first attacked. This happens because the rate-limiting process kicks in when the GraphQL server receives more than 100 packets within the first 5 seconds.

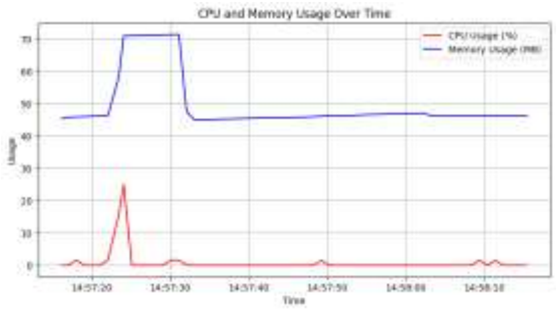


Figure 9 CPU and Memory usage graph when the server is protected by rate limiting

This research demonstrates that the rate-limiting method is highly effective in enhancing the security of GraphQL servers against Batch Attacks. By imposing limits on the number of requests that can be processed within a certain timeframe, the server can protect itself from the excessive load caused by such attacks. The results of the measurements indicate that rate-limiting successfully reduces the workload on the CPU and Memory significantly, from an average of 4.8% CPU usage and an average of 87MB Memory usage during an attack, to just an average of 0.86% CPU and an average of 49.6MB Memory usage. On servers without rate-limiting, there is an explosion in CPU and Memory usage every few seconds, whereas with rate-limiting, such an explosion only occurs in the first 5 seconds.

Please be aware that Rate-limiting can restrict legitimate user access, especially if the limits set are too strict. This can cause frustration for users who are not performing malicious activities (Clark 2019). Furthermore, on systems that have many users or high demand, implementing effective rate-limiting can be challenging. Ensuring that the system can handle a large number of requests and limit enforcement requires a good infrastructure. Rate-limiting can also be complicated to implement properly, especially if there is a need to customize the limits based on the type of user or service being accessed.

Another impact for users is that they may have to wait before being able to take further action if they reach the set limit (Serbout et al. 2023). If the limits are not adapted to reasonable usage patterns, users may experience difficulties in accessing the services they need, which may lead to dissatisfaction.

It is worth comparing with other security such as Firewalls that serve as a barrier between internal and external networks and focus more on blocking unauthorized access based on set rules (Anwar, Abdullah, and Pastore 2021). MFA provides an additional layer of security by requiring more than one form of verification before allowing access, which is a different approach from rate-limiting.

5. CONCLUSION

This research shows that rate-limiting is very effective in securing GraphQL servers from Batch Attacks. By limiting the number of requests a server can handle in a certain time, the server can protect itself from being overloaded. The study found that rate-limiting significantly reduces CPU and memory usage during an attack, from an average of 4.8% CPU and 87MB memory to just 0.86% CPU and 49.6MB memory. Without rate-limiting, CPU and memory usage spikes every few seconds, but with it, spikes occur only in the first 5 seconds. This means rate-limiting helps keep the server running smoothly during attacks.

Based on these findings, several steps are recommended to improve security. First, optimize rate-limiting settings to find the best balance between protection and performance. Second, develop a real-time monitoring system to quickly respond to threats. Third, increase awareness among developers and administrators about Batch Attacks and the importance of security measures like rate-limiting. Additionally, combine rate-limiting with other security methods like authentication and encryption for stronger defense. Finally, further research on future attacks and new security techniques is encouraged to better protect GraphQL servers against complex threats. By following these steps, GraphQL servers can be better equipped to handle evolving security challenges.

BIBLIOGRAPHY

Anwar, Raja Waseem, Tariq Abdullah, and Flavio Pastore. 2021. "Firewall Best Practices for Securing Smart Healthcare Environment: A Review." *Applied Sciences (Switzerland)* 11(19). doi: 10.3390/app11199183.

Brito, Gleison, and Marco Tulio Valente. 2020. "REST vs GraphQL: A Controlled Experiment." *Proceedings - IEEE 17th International Conference on Software Architecture, ICSA 2020* (Dcc):81–91. doi: 10.1109/ICSA47634.2020.00016.

Clark, Scott. 2019. "Guide to the General Data Protection Regulation (GDPR)." *Guide to the General Data Protection Regulation* (May):n/a.

Firdaus, Diash, and Resa Rianti. 2023. "DETEKSI ANOMALI DAN SERANGAN LOW RATE DDOS DALAM LALU LINTAS JARINGAN MENGGUNAKAN NAIVE BAYES." 05(02):140–48.

Github. n.d. "About the GraphQL API - GitHub Docs." Retrieved August 5, 2024 (https://docs.github.com/en/graphql/overview/about-the-graphql-api).

Graphql. 2024. "Schemas and Types | GraphQL." Retrieved August 26, 2024 (https://graphql.org/learn/schema/).

Hanif, Fahri, Imam Ahmad, Dedi Darwis, Ichtiar Lazuardi Putra, and Muhammad Fauzan Ramadhani. 2022. "Analisa Perbandingan Metode Graphql Api Dan Rest Api Dengan Menggunakan Asp.Net Core Web Api Framework." *Jl. ZA. Pagar Alam* 3(2):2774–5384.

Mahjabin, Tasnuva, Yang Xiao, Guang Sun, and Wangdong Jiang. 2017. "A Survey of Distributed Denial-of-Service Attack, Prevention, and Mitigation Techniques." *International Journal of Distributed Sensor Networks* 13(12). doi: 10.1177/1550147717741463.

McFadden, Shae, Marcello Maugeri, Chris Hicks, Vasilios Mavroudis, and Fabio Pierazzi. 2024. "WENDIGO: Deep Reinforcement Learning for Denial-of-Service Query Discovery in GraphQL." *Proceedings - 45th IEEE Symposium on Security and Privacy Workshops, SPW 2024* 68–75. doi: 10.1109/SPW63631.2024.00012.

Meta. 2015. "GraphQL: A Data Query Language - Engineering at Meta." Retrieved August 26, 2024 (https://engineering.fb.com/2015/09/14/core-infra/graphql-a-data-query-language/).

Muzaki, Rizki Nuzul, and Abu Salam. 2024. “Reducing Under-Fetching and Over-Fetching in Rest Api With Graphql for Web-Based Software Development.” 5(2):447–53.

Ogboada, J. G., ANIREH VIE, and D. Matthias. 2021. “A Model for Optimizing the Runtime of GraphQL Queries.” *Vol* 9(3):11–39.

Quiña-Mera, Antonio, Pablo Fernandez, José María García, and Antonio Ruiz-Cortés. 2023. “GraphQL: A Systematic Mapping Study.” *ACM Computing Surveys* 55(10). doi: 10.1145/3561818.

Ren, Kui, Tianhang Zheng, Zhan Qin, and Xue Liu. 2020. “Adversarial Attacks and Defenses in Deep Learning.” *Engineering* 6(3):346–60. doi: 10.1016/j.eng.2019.12.012.

Serbout, Souhaila, Amine El Malki, Cesare Pautasso, and Uwe Zdun. 2023. *API Rate Limit Adoption – A Pattern Collection*. Vol. 1. Association for Computing Machinery.

Yunus, Moh. 2019. “Analisis Kerentanan Aplikasi Berbasis Web Menggunakan Kombinasi Security Tools Project Berdasarkan Framework Owasp Versi 4.” *Jurnal Ilmiah Informatika Komputer* 24(1):37–48. doi: 10.35760/ik.2019.v24i1.1988.