INTEK Jurnal Penelitian
Vol. 12, No. 1, pp. 11-18, April 2025

Denial-of-Service Modeling: Time and CPU Utilization During GraphQL Attacks

Debora Natalia Ginting [1,a], A. Widjajarto [1,b], and Umar Yunan Kurnia Septo Hediyanto [1,c]

[1]Faculty of Industrial Engineering, Telkom University, Jalan Telekomunikasi No. 1, Bandung, 40257, Indonesia
[a]deborantlya@student.telkomuniversity.ac.id (Corresponding Author), [b]adtwjrt@telkomuniversity.ac.id,
[c]umaryunan@telkomuniversity.ac.id

***Abstract*—GraphQL offers flexible, client-driven data retrieval and is increasingly adopted across modern platforms. However, its expressiveness (e.g., deep nesting, introspection, and aliases) can expose APIs to denial-of-service (DoS) vectors. This study empirically evaluates four DoS patterns against a GraphQL API, circular queries, field duplication, alias overloading, and object limit overriding, on a Kali Linux testbed using DVGA and Altair, and analyzes attack paths via an attack tree. Execution time and CPU utilization were evaluated before and after hardening. Field duplication yielded the shortest execution time (2.50 to 1.86 s) with a CPU reduction (88.5% to 75.5%), whereas alias overloading remained the most CPU-intensive despite improvements (time 1,412.05 to 691.29 s; CPU 99% to 93%). Object limit overriding reduced CPU (90% to 74%) with a modest time trade-off (16.61 to 18.74 s). Overall, hardening significantly mitigated resource burden across attacks, though alias-based payloads continued to pressure the server. The results provide actionable baselines for GraphQL hardening strategies and highlight the need for combined countermeasures (depth/complexity limits, alias caps, and rate limiting) to sustain service availability under adversarial query patterns.**

***Keywords*—API GraphQL; Attack Tree; Denial-of-Service; Exploitation; CPU; Time**

I. Introduction

The continuing digital transformation has led to a significant increase in the complexity of data management, prompting the need for novel approaches that can deliver precise data retrieval while maintaining efficiency [1-3]. GraphQL, introduced in 2015, has emerged as a promising alternative to traditional REST APIs by enabling clients to fetch exactly the data they require, thus minimizing unnecessary data transmission and reducing network load [3, 4]. This level of precision has been particularly pertinent in the modern era, where flexibility in querying is paramount for sustaining performance in data-intensive applications.

Recent reports indicate that nearly 47% of developers are incorporating GraphQL into their workflows, with major organizations such as Facebook, GitHub, Shopify, Twitter, and Airbnb leveraging its capabilities. Alongside its growing adoption, a range of performance optimization techniques, including schema stitching, real-time subscriptions, advanced caching strategies, and query complexity analysis, has been explored to harness GraphQL's full potential [5]. These strategies not only enhance the efficiency of data retrieval but also contribute to the overall scalability of applications that depend on a dynamic and flexible data model.

Despite these advantages, GraphQL's inherent flexibility can also introduce significant security challenges. Its capability to support nested and introspective queries may be exploited by attackers to conduct DoS attacks. For example, an attacker could craft a query that excessively burdens the underlying server, leading to resource exhaustion and degraded performance [6]. Although various countermeasures such as query depth limiting, complexity scoring, adaptive rate limiting, and guarded introspection have been proposed, these methods often remain unvalidated against realistic workloads, leaving a gap in empirical evidence that is critical for robust security assurance [7-9].

To address these concerns, the current study proposes an innovative empirical framework designed to simulate a range of DoS attack patterns on GraphQL

DOI: http://dx.doi.org/10.31963/intek.v12i1.5174

APIs. This framework intends to measure key performance indicators such as CPU usage and exploitation time, thereby establishing benchmark reference points for GraphQL vulnerabilities [5]. By systematically profiling the impact of various DoS attack sequences, the framework aims to provide practical guidance for developers and security teams in selecting and implementing APIs that minimize the risk of compromise. Furthermore, recent literature discusses the challenges of testing GraphQL's schema-driven structure, highlighting the need for such empirical investigations [10].

While GraphQL offers significant advantages in terms of data query flexibility and efficiency, its adoption has also introduced new vectors for DoS attacks, primarily due to its flexible and introspective querying capabilities [11, 12]. The proposed empirical framework not only fills the gap in current security validation research but also contributes to the development of actionable benchmarks [13], thus aiding in the implementation of more secure GraphQL APIs. This approach represents a critical step toward ensuring that the benefits of GraphQL are not undermined by emerging security vulnerabilities in an increasingly complex digital ecosystem.

II. Research Methodology

This research employs the threat modeling method with an attack tree approach to explore DoS attacks on GraphQL APIs. Threat modeling is a systematic approach aimed at understanding potential threats to a system, while the attack tree is a graphical representation of various ways to attack a system, with each specific attack broken down into branches representing different scenarios [14,15].

Additionally, this research utilizes time metrics to measure the efficiency of each attack on the GraphQL API. These metrics are used to compare which attacks are most efficient in exploiting system vulnerabilities. The results of this study will provide insights into the effectiveness and efficiency of various attack techniques against GraphQL APIs, ultimately contributing to the improvement of defense strategies and threat mitigation.

This study also leverages CPU usage to analyze the extent of the server's workload when subjected to different types of exploitation attacks. By observing CPU usage during the attacks, this research evaluates the degree to which the server is burdened and assesses the effectiveness of the implemented mitigation measures.

1. Theoretical Review

Denial-of-Service (DoS) attacks are designed to disrupt the availability of online services by overwhelming targeted servers or network resources with excessive traffic, preventing them from handling legitimate client requests. Experimental and simulation-based evaluations are widely used to examine system resilience and identify potential weaknesses related to DoS exploitation. GraphQL is a declarative API query language that enables clients to precisely define the structure of requested data. Although this capability enhances efficiency and developer productivity, it may also introduce attack surfaces that can be abused to degrade system performance. Circular or recursive queries occur when nodes reference each other repeatedly, potentially causing continuous processing loops that exhaust server resources if not limited properly. Field duplication is another DoS technique, in which attackers submit queries containing large numbers of repeated fields, forcing the server to perform redundant computations and consume significant processing power. Alias overloading exploits GraphQL's aliasing feature, allowing duplicated fields to be treated as unique operations. Without strict enforcement, this approach significantly increases resolver execution and may overload the backend. Furthermore, object limit overriding is executed by modifying query limits to extract excessively large datasets, resulting in memory overuse and serious performance degradation. These attack vectors demonstrate that GraphQL's flexible query model requires robust validation policies.

2. Systematics of Problem Solving

The problem-solving methodology outlines the steps taken to address the issue, following the established framework and leveraging the expertise of each specialized group.

DOI: http://dx.doi.org/10.31963/intek.v12i1.5174

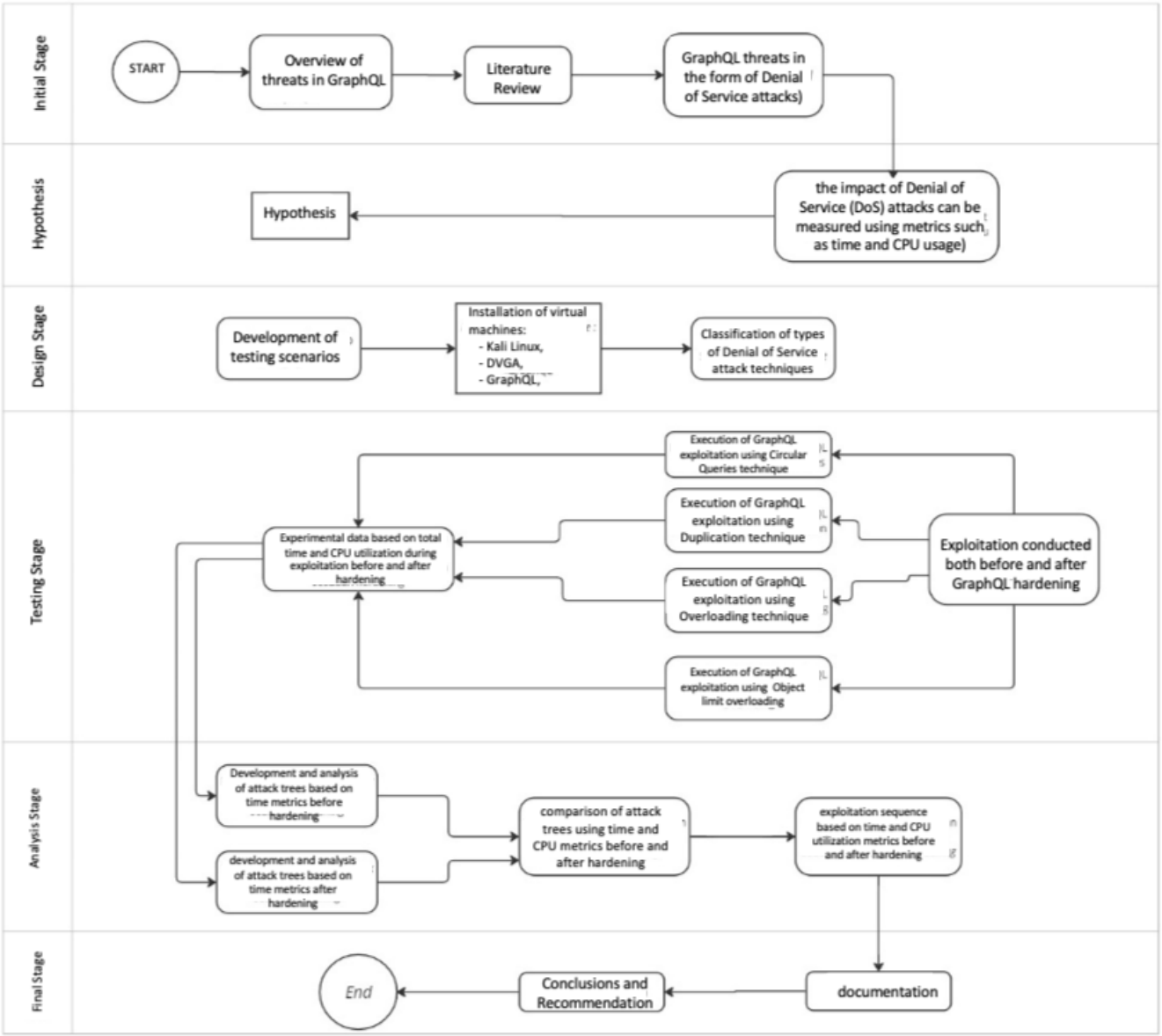


Figure 1. Systematics of problem solving

3. Data Collection

Data collection was conducted through experiments that include the following test scenarios:

a) Attacks using the circular queries technique
b) Attacks using the field duplication technique
c) Attacks using the abusing aliases for DoS technique
d) Attacks using the object limit overriding technique

An analysis of these DoS attack types was performed to assess the extent to which the GraphQL server is vulnerable to DoS attacks.

4. Data Processing

Data processing was carried out by comparing the time and CPU usage on the server within GraphQL. After conducting the experiments, the impact of each DoS attack technique on the GraphQL API was evaluated.

DOI: http://dx.doi.org/10.31963/intek.v12i1.5174

5. Evaluation Method

The evaluation method involved testing after the attacks by comparing the server's capacity to handle resource loads, GraphQL's built-in prevention measures, server response time, and CPU utilization. This allows for the identification of which DoS attack technique has the most significant impact on the GraphQL API.

III. Results and Discussion

1. Experiment Implementation

The implementation of the experiment involves the results from the scenarios before and after the experiment, which will include input data and output data from each DoS attack conducted.

Implementation of the Circular Queries Experiment

In the circular queries experiment, before hardening, the server crashed after 110.78 seconds, utilizing 84.5% of the CPU. After hardening, execution time decreased to 51.78 seconds with a reduction in CPU usage to 76%. This demonstrates that while hardening reduced the server's vulnerability, the technique still imposed a significant load, indicating areas where further optimization may be needed.

Implementation of the Field Duplication experiment

Field duplication proved to be the most efficient attack before hardening, with an execution time of 2.5 seconds and CPU utilization of 88.5%. After hardening, both metrics improved significantly, with the execution time reduced to 1.86 seconds and CPU usage dropping to 75.5%. These results suggest that the hardening measures were particularly effective against this type of attack.

Implementation of the Alias Overloading Experiment

The alias overloading experiment revealed that, although the CPU usage remained high (99% before and 93% after hardening), the execution time was drastically reduced from 1,412.05 seconds to 691.29 seconds post-hardening. While hardening helped mitigate the impact, this attack continued to exert substantial pressure on the server, suggesting that additional mitigation strategies may be necessary.

Implementation of the Object Limit Overriding Experiment

For object limit overriding, the pre-hardening execution time was 16.61 seconds with a CPU utilization of 90%. Post-hardening, the execution time slightly increased to 18.74 seconds, while CPU usage dropped to 74%. This slight increase in execution time indicates that the hardening measures prioritized security, even if it introduced a minimal performance trade-off.

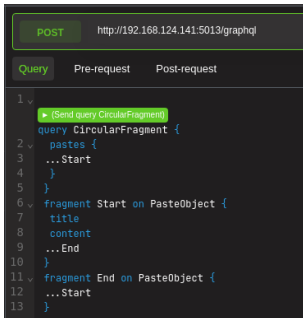


Figure 2. Implementation of the Circular Queries Experiment before Hardening

Figure 3. Implementation of field duplication experiment after exploitation and hardening

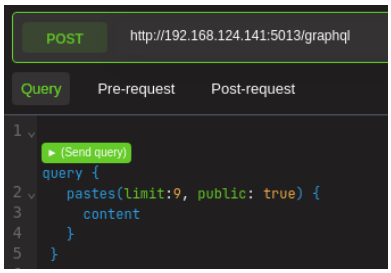


Figure 4. Implementation of object limit overriding experiment after exploitation

2. Measurement of Time and CPU Utilization

Time measurement is based on the use of functions to measure, record, and observe each time scale with different scales for each element.

DOI: http://dx.doi.org/10.31963/intek.v12i1.5174

Results of Time Measurement Comparison Analysis

The time measurement is conducted based on how long the exploitation process takes and follows the predetermined steps. Metric graph time before and after hardening show in Figure 5 and 6. The time measurement is performed in seconds (s).

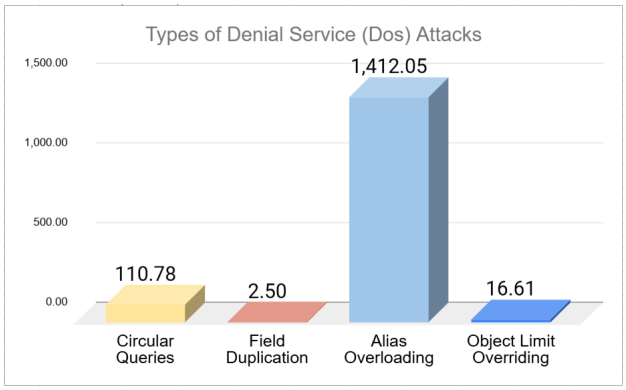


Figure 5. Metric graph time before hardening

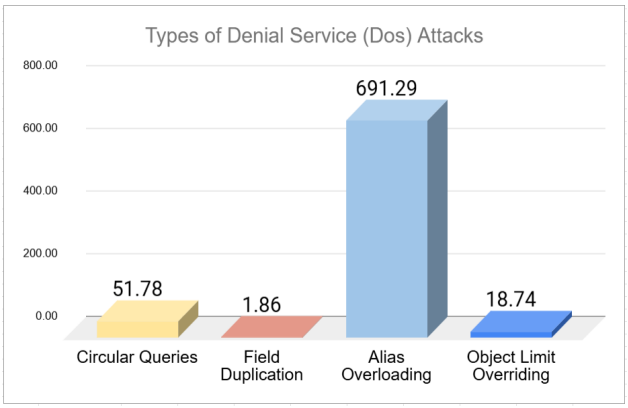


Figure 6. Metric graph time after hardening

Table 1 shows the time metrics for the exploitation conducted before hardening, and Table 2 shows after hardening.

Table 1. Results of time analysis for comparison of exploitation before hardening

No.	Attack	Description	Time Exploitation 1 (s)		
			Real	*User*	*System*
1.	Circular Queries	Successful Exploitation	110.78	25.23	42.27
2.	Field Duplication	Successful Exploitation	2.5	0.58	1.21
3	Alias Overloading	Successful Exploitation	1,412.05	3.21	3.81
4	Object Limit Overriding	Successful Exploitation	16.61	1.39	0.98

Table 2. Results of time analysis for comparison of exploitation after hardening

No.	Attack	Description	Time Exploitation *1* (s)		
			Real	*User*	*System*
1.	*Circular Queries*	Successful Exploitation	51.78	25.23	42.27
2.	*Field Duplication*	Successful Exploitation	1.86	0.49	0.57
3	*Alias Overloading*	Successful Exploitation	691.29	1.8	4.04
4	*Object Limit Overriding*	Successful Exploitation	18.74	1.58	3.05

Results of CPU Utilization Comparison Analysis

The CPU utilization measurement is conducted to observe the load imposed by the attacks on CPU resources, which is a key indicator of whether the server can continue operating normally or begins to experience overload [16, 17]. CPU measurements were taken by averaging multiple trials to obtain the results presented in Table 3.

Table 3. CPU utilization table before and after hardening

Attacks	Low (%)	Hard (%)
Circular Queries	84.50	76
Field Duplication	88.50	75.50
Alias Overloading	99	93
Object Limit Overriding	90	74

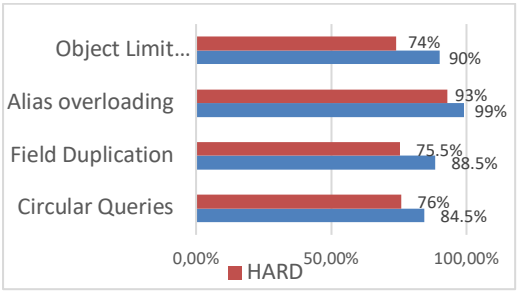


Figure 7. CPU utilization graph before and after hardening

In Figure 7, the vertical y-axis represents the names of the attack types, while the horizontal x-axis represents the percentage of CPU utilization. The horizontally arranged red bars in the chart indicate the results after hardening, while the blue bars indicate the results before hardening.

DOI: http://dx.doi.org/10.31963/intek.v12i1.5174

The results show that the CPU utilization reached its highest percentage during the Alias Overloading attack, with a pre-hardening percentage of 99% and a post-hardening percentage of 93%. The lowest pre-hardening percentage was observed in the Circular Queries attack at 84.5%, while the lowest post-hardening percentage was seen in the Field Duplication attack.

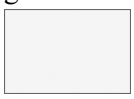
3. Construction of the Attack Tree

An attack tree is a method of visualizing and analyzing information security risks used to identify potential threats to a system or network [18, 19]. It is a graphical representation that illustrates the ways in which an attacker can achieve their objective.

The attack tree created in this study consists of several nodes with different functions, as follows:

1. 

Or = This node represents the testing of an exploitation and can be depicted as variations of steps in the testing process.
2. 

And = This node represents the sequence of processes in an attack that requires one or more processes to run simultaneously to achieve the desired goal or outcome for the attacker
3. 
Sub-goal = This node represents the goal or endpoint of the attack steps.

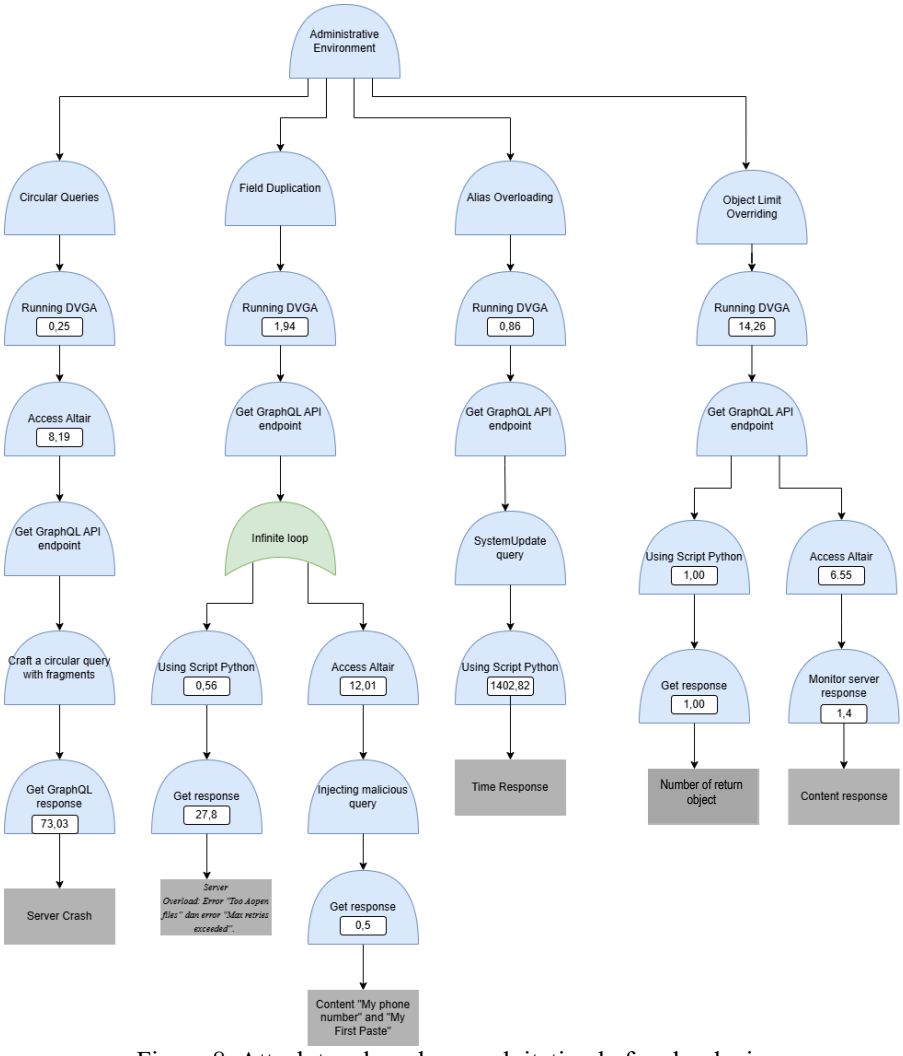


Figure 8. Attack tree based on exploitation before hardening

DOI: http://dx.doi.org/10.31963/intek.v12i1.5174

Results of Attack Tree Analysis Based on Exploitation Before Hardening

The previously constructed Attack Tree for each attack type is combined into a single diagram. The purpose of merging these diagrams is to present the overall results of the research. Figure 8 show the unified diagram from the combined Attack Trees before hardening.

In Figure 8, the attack tree illustrates the process of increasing the load on the target server. First, access is made to the DVGA server for all types of attacks. Circular Queries caused the server to crash in 73.03 seconds after initiation; Field Duplication resulted in a server overload in 27.8 seconds; alias overloading took 1,402.82 seconds to execute the system update query; and Object Limit Overriding triggered the server's content request limitation mechanism in 1.4 seconds. Each attack demonstrated different execution times and impacts, with circular queries and alias overloading having the most significant impact compared to other types of attacks.

Results of Attack Tree Analysis Based on Exploitation After Hardening

In Figure 9, the attack tree illustrates the process of increasing the load on the target server with the presence of hardening measures. Initially, the DVGA server was

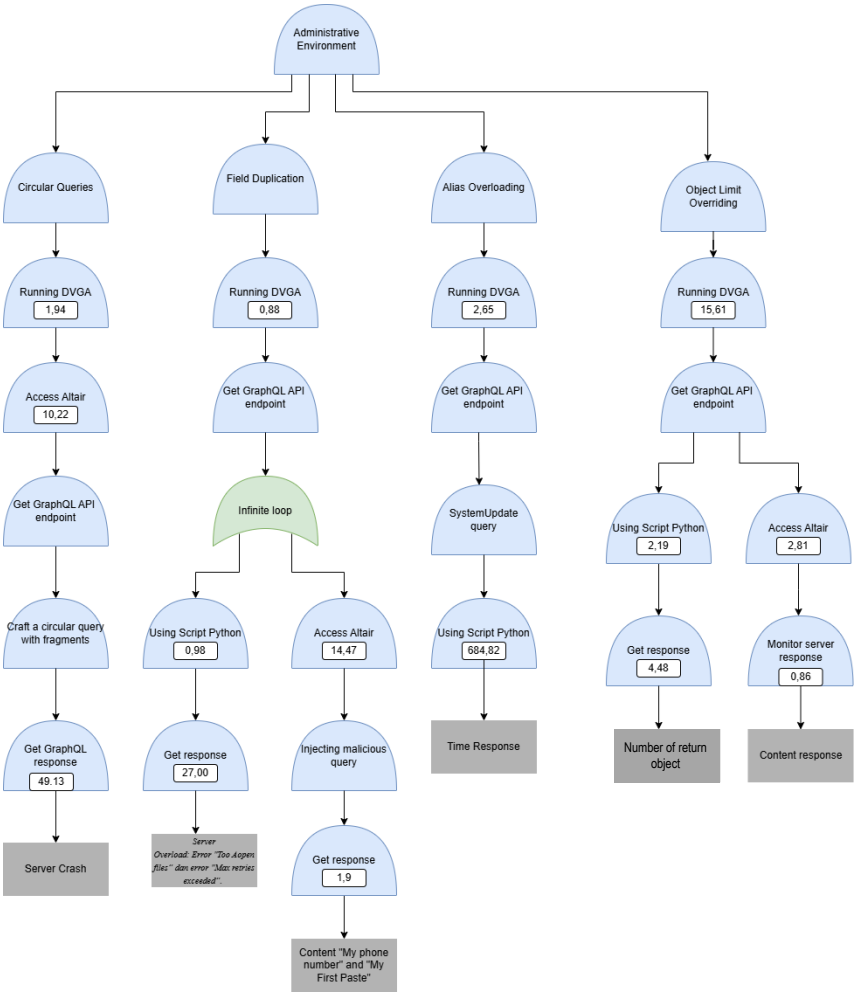


Figure 9. Attack tree based on exploitation after hardening

accessed for all types of attacks. Circular queries caused the server to crash in 49.13 seconds after initiation; field duplication resulted in a server overload in 27 seconds; alias overloading took 684.82 seconds to execute the system update query and object limit overriding imposed a content request limit in 4.48 seconds. Each attack

DOI: http://dx.doi.org/10.31963/intek.v12i1.5174

demonstrated different execution times and impacts, with circular queries and alias overloading having the most significant impact on server performance, followed by field duplication, and finally object limit overriding.

IV. Conclusion

Based on the analysis conducted, the results of the experiments indicate that each attack technique has varying impacts; however, all techniques successfully exploited vulnerabilities in GraphQL, causing server overload or crashes and rendering the server unresponsive.

After the hardening process, there were notable improvements in execution time and CPU usage for circular queries (from 110.78 s / 84.5% to 51.78 s / 76%), field duplication (from 2.5 s / 88.5% to 1.86 s / 75.5%), and alias overloading (from 1412.05 s / 99% to 691.29 s / 93%). Object limit overriding, while reducing CPU usage from 90% to 74%, saw an increase in execution time from 16.61 s to 18.74 s. This longer time reflects tighter security controls rather than hardening failure. Overall, field duplication was the most effective attack, followed by object limit overriding and circular queries, with alias overloading being the least efficient due to its prolonged execution time.

References

[1] U. Lichtenthaler, "Data management efficiency: major opportunities for shared value innovation," *Management Research Review*, vol. 45, no. 2, pp. 156–172, Jan. 2022, doi: 10.1108/MRR-10-2020-0639.

[2] E. Tungadi, I. K. Yusri, S. Serpian, I. Irmawati, M. Olivya, and M. N. Y. Utomo, "Access right to module and data attachment for company profile administrator module," in *AIP Conference Proceedings*, AIP Publishing, 2024.

[3] S. Kanthed, "Rest vs. GraphQL: Comparative analysis of API design approaches," *International Journal of Multidisciplinary Research and Growth Evaluation.*, vol. 4, no. 1, pp. 984–991, 2023, doi: 10.54660/.IJMRGE.2023.4.1.984-991.

[4] Nagaraju Thallapally, "Enhancing data query flexibility with GraphQL: I mplementation and best practices," *Journal of Computer Science and Technology Studies*, vol. 6, no. 2, pp. 176–182, Jun. 2024, doi: 10.32996/jcsts.2024.6.2.20.

[5] S. Cheng and O. Hartig, "LinGBM: A performance benchmark for approaches to build GraphQL servers," 2022, pp. 209–224. doi: 10.1007/978-3-031-20891-1_16.

[6] C. Zhao, "API common security threats and security protection strategies," *Frontiers in Computing and Intelligent Systems*, vol. 10, no. 2, pp. 29–33, Nov. 2024, doi: 10.54097/k5djs164.

[7] J. Liu, J. Yang, L. Xiong, and J. Pei, "Secure skyline queries on cloud platform," in *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, IEEE, Apr. 2017, pp. 633–644. doi: 10.1109/ICDE.2017.117.

[8] A. El Malki, U. Zdun, and C. Pautasso, "Impact of API rate limit on reliability of microservices-based architectures," in *2022 IEEE International Conference on Service-Oriented System Engineering (SOSE)*, IEEE, Aug. 2022, pp. 19–28. doi: 10.1109/SOSE55356.2022.00009.

[9] T. Pavleska, G. P. Sellitto, and H. Aranha, "Crafting organizational security policies for critical infrastructures: an architectural approach," *Journal of Surveillance, Security and Safety*, vol. 5, no. 2, pp. 116–39, May 2024, doi: 10.20517/jsss.2023.40.

[10] Vinaysimha Varma Yadavali, "Testing GraphQL APIs: challenges, tools and best practices," *International Journal of Advanced Research in Science, Communication and Technology*, pp. 1–10, Oct. 2021, doi: 10.48175/IJETIR-8011.

[11] G. Mavroudeas *et al.*, "Learning GraphQL query cost," in *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, IEEE, Nov. 2021, pp. 1146–1150. doi: 10.1109/ASE51524.2021.9678513.

[12] N. Karalis, A. Bigerl, and A.-C. Ngonga Ngomo, "Native execution of GraphQL queries over RDF graphs using multi-way joins," 2023. doi: 10.3233/SSW230007.

[13] S. Serpian, S. Syam, A. N. Istiyana, and M. Marwan, "Internet-based surveillance to evaluate employee loyalty measurement in micro, small, and medium enterprises (MSMEs) through the application of expert judgment," 2024, p. 040012. doi: 10.1063/5.0221040.

[14] I. Agrafiotis, J. R. Nurse, O. Buckley, P. Legg, S. Creese, and M. Goldsmith, "Identifying attack patterns for insider threat detection," *Computer Fraud & Security*, vol. 2015, no. 7, pp. 9–17, Jul. 2015, doi: 10.1016/S1361-3723(15)30066-X.

[15] R. C. Nurse *et al.*, "Understanding insider threat: A framework for characterising attacks," in *2014 IEEE Security and Privacy Workshops*, IEEE, May 2014, pp. 214–228. doi: 10.1109/SPW.2014.38.

[16] L. Duflot, "CPU bugs, CPU backdoors and consequences on security," *Journal in Computer Virology*, vol. 5, no. 2, pp. 91–104, May 2009, doi: 10.1007/s11416-008-0109-x.

[17] E. Kalyvianaki, T. Charalambous, and S. Hand, "Self-adaptive and self-configured CPU resource provisioning for virtualized servers using Kalman filters," in *Proceedings of the 6th international conference on Autonomic computing*, New York, NY, USA: ACM, Jun. 2009, pp. 117–126. doi: 10.1145/1555228.1555261.

[18] T. C. Yang, M. Fang, and Q. F. Shao, "A distributed network risk assessment method based on attack graph," *Applied Mechanics and Materials*, vol. 241–244, pp. 2335–2342, Dec. 2012, doi: 10.4028/www.scientific.net/AMM.241-244.2335.

[19] D. Vitkus, J. Salter, N. Goranin, and D. Čeponis, "Method for attack tree data transformation and import into IT risk analysis expert systems," *Applied Sciences*, vol. 10, no. 23, p. 8423, Nov. 2020, doi: 10.3390/app10238423.