# *Mitigating Broken Access Control Vulnerabilities through Graph-Theoretic Optimization of Web Application Authorization Logic Topologies*

Kelvin Sebastian Yen - 13525068
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: kelvinsyen@gmail.com , 13525068@std.stei.itb.ac.id

***Abstract*—*The advent of web applications brought forth numerous vulnerabilities that could lead to serious repercussions. Among these exploits, Broken Access Control (BAC) stands out as the most prevalent threat to modern web security architecture. This problem primarily stems from the flaws of human design in mapping authorization logic and user permissions. As the application's infrastructure expands, its architectural complexity scales exponentially, thereby increasing the probability of oversight during traditional, manual security audits. As such, the decision was made to model the system topology as a directed weighted graph, which is then evaluated using the Floyd-Warshall algorithm. This mathematical approach determines the path of least resistance based on the weight of connecting edges. The application of this algorithm yields a comprehensive matrix that maps the optimal exploitation routes between all nodes, providing security engineers with a quantifiable framework to prioritize vulnerability patching.*

***Keywords—Broken Access Control; Floyd-Warshall Algorithm; Weighted Graph; Vulnerability Mapping; Path of Least Resistance***

## I. Introduction

In the pursuit of societal advancement, mankind has not ceased its progression in science and technology. This advance brought forth numerous inventions designed to improve mankind's quality of life. One such invention is web applications, a derivative of the internet commonly used on a daily basis. While formerly a luxury only the wealthy are privileged to use, the digital era provides equally distributed access among all social classes. Web applications have since become the crux of modern-day business by being a platform for a wide variety of services, e.g. academics, e-commerce, and public services.

However, as with all breakthroughs, this technological advancement is not flawless. Every web application utilizes a series of components that communicate with each other–namely the frontend, API endpoints, databases–which harbor risks when not given proper security. These risks are further exacerbated by the scaling of a web application's complexity. An example of said risk is Broken Access Control (BAC) vulnerabilities [1], which happen when programmers fail to implement proper access restrictions.

Conventional vulnerability testing methods such as penetration testing are not entirely reliable as they are prone to human error. The vast, complex topology of a web application's communication network poses the risk of an unthorough coverage during testing. As such, a mathematical tool capable of encapsulating every variable that could potentially compromise the application's operational security is necessary. Mapping the complex network with a weighted graph presents the topological relations in a comprehensive and rigorous manner.

This research paper aims to solve authorization logic problems through the application of graph theory and relations. Implementing the Floyd-Warshall Algorithm helps verify whether a user has a direct unauthorized transitive access to a forbidden endpoint, while the Floyd Algorithm analyzes the weakest path based on risk weights of each component.

## II. Literature Study

### *A. Discrete Mathematics*

#### *1)* Relations

A binary relation is mathematically defined as a subset of the Cartesian product between two sets A and B. Set A is defined as the domain, while set B is its co-domain. Relations can be represented in several manners, one of which are matrices. Set A is used as its row elements, whereas set B functions as its columns. Relations are characterized by 4 unique properties, among them is the transitive property [3].

The transitive property mentions that if there exists relations (a, b) and (b, c), there also exists a relation (a, c). This property is further explored in transitive closure, a foundational concept that identifies all indirectly related vertices. The formula for finding the transitive closure of a relation R is defined through the union of its powers [4]:

$$R^* = R \cup R^2 \cup R^3 \cup \ldots \cup R^n$$

Consequently, when translated into a zero-one matrix $M_R$ representing a set R with n elements, the transitive closure of said matrix is obtained through Boolean OR operations [4]:

$$M_{R^*} = M_R \vee M_R^{[2]} \vee M_R^{[3]} \vee \dots \vee M_R^{[n]}$$

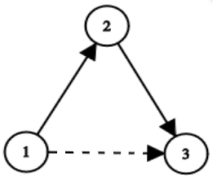


Fig 2.1: Transitive property between nodes 1 and 3

However, computing the transitive closure through successive Boolean matrix multiplications is computationally exhaustive, scaling poorly as the number of elements grows. Therefore, a much more efficient method of finding transitive closures is essential. The Warshall Algorithm expedites the process of finding the transitive closure of a matrix MR by providing a highly optimized approach through the following Boolean logic formula [2]:

$$w_{ij}^{[k]} = w_{ij}^{[k-1]} \vee (w_{ik}^{[k-1]} \wedge w_{kj}^{[k-1]})$$

This algorithm works by iterating through the matrix in a row-column manner. At every iteration, the algorithm performs an AND operation to check whether a path exists between vertices i and k, and vertices k and j in the previous iteration. The Boolean output is then used in an OR operation with the Boolean value of a path between vertices i and j. The OR operator is necessary to conclude if a direct path exists or an indirect path is formed.

*2)* Graphs

Graphs are a set of vertices and edges representing the relations between said vertices [5]. A graph can have no edges, but must have at least one vertex. As graphs contain relations, they can also be represented with matrices, specifically the adjacency matrix. Adjacency matrices relay information on whether or not two vertices are connected via Boolean values [5].

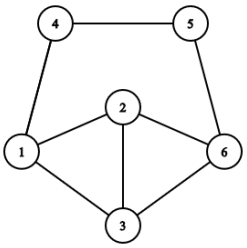


Fig 2.2: Undirected graph representing basic system connectivity

On simple graphs, an edge represents a bidirectional relationship between two vertices. However, directed graphs, a more advanced form of graphs, specify the direction of an edge. These changes affect the specifications of the adjacency matrix, where the rows represent source vertices and the columns function as the destination.

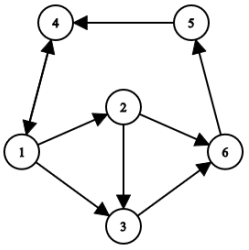


Fig 2.3: Directed graph illustrating flow

Weighted graphs are an even more complex subtype of graphs. In unweighted graphs, the value of an edge regardless of direction--except for loops--is set to one.

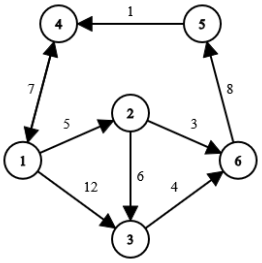


Fig 2.4: Directed graph with weights on each edge

On weighted graphs, each edge is assigned a specific non-negative integer weight. Instead of zeroes and ones, the adjacency matrix is filled with the weight of each edge. Zeroes are only used for the main diagonal, as the weight of a vertex to itself is 0. Additionally, the infinity symbol (∞) replaces zeroes in representing disconnected vertices.

*3)* The Floyd Algorithm

Because weighted graphs utilize integer weights and the infinity symbol rather than simple Boolean values, the standard Warshall algorithm is no longer mathematically applicable. While Warshall efficiently determines the existence of a path, it cannot evaluate the cost of a traversal. To find the transitive reachability and the shortest path between all pairs of vertices within a weighted graph, a modified approach is required. The Floyd Algorithm serves this purpose by iterating through the weight matrix and finding the path with the minimum cumulative weight via the following formula [6]:

$$d_{ij}^{[k]} = min(d_{ij}^{[k-1]}, d_{ik}^{[k-1]} + d_{kj}^{[k-1]})$$

The core mechanism of this algorithm lies in the min() function. The subjects of comparison are the weight of the currently known optimal path and the cumulative weight of each indirect path on previous iterations. The addition operation calculates the total combined weight of this specific route from the source to the destination. The expected output is a matrix with the minimum cumulative weight for each pair of vertices.

### B. Access Control

Fundamentally, access control is an integral part of web application architecture that enforces the restrictions of user actions within the intended boundaries. Its primary function is to strictly prevent users from manipulating sensitive system assets. When this architecture fails, a series of severe security repercussions may occur, including but not limited to information disclosure, modification of data, and the execution of actions outside user limits. Failures in this authorization mechanism are collectively categorized as Broken Access Control (BAC) vulnerabilities [1]. Among the numerous manifestations of BAC, some of the most prominent exploits are privilege escalation, metadata manipulation, Insecure Direct Object References (IDOR), and bypassing access control checks by modifying parameters.

## III. System Design and Methodology

### C. Security Weight Classification

To evaluate the vulnerability level of a web application logically and mathematically, a weighing system is established. This weight represents the expected security level an attacker attempting to breach the web should meet. A lower weight indicates weaker security measures, while better fortified endpoints receive a significantly higher grade.

Table 3.1: Path Weighing Criteria

| Weight | Security Classification | Technical Conditions |
|---|---|---|
| 1 | Vulnerable Security | Publicly accessible paths without authentication (IDOR) |
| 3 | Standard Security | Access requiring basic session tokens/cookies |
| 5 | Elevated Security | Access requiring strict server side Role Based Access Control (RBAC) |
| 10 | Critical Security | High level administrative backend processing or direct connection to core data assets |

### D. Architectural Node Definition

The targeted web application is simulated to encapsulate exactly 10 logical nodes. These nodes are classified into three types: User Roles, Intermediaries/Gateways, and Endpoints.

Table 3.2: Web Application Nodes

| Node | Component Name | Technical Conditions |
|---|---|---|
| $V_1$ | Guest | Unauthenticated anonymous public actor |
| $V_2$ | User | Authenticated actor with standard privileges |
| $V_3$ | Admin | High-privileged actor managing core system functions |
| $V_4$ | Auth Validator | Middleware component verifying JWT tokens and roles |
| $V_5$ | /home | The web application's public home page |
| $V_6$ | /api/v1/profile | REST API endpoint returning regular user data (JSON) |
| $V_7$ | /admin/dashboard | Admin front end control panel |
| $V_8$ | /api/v1/admin/export | Hidden admin endpoint returning sensitive raw JSON payload |
| $V_9$ | General App DB | Database containing non sensitive data and public asset |
| $V_{10}$ | Central Security DB | Isolated database containing security credentials |

### E. Authorization Logic Topology and Edge Mapping

The interaction and data communication between components form a Directed Weight Graph (G = (V, E)). Under ideal business logic, access to administrative endpoints must flow sequentially through the validation intermediaries. However, to simulate a BAC vulnerability, this web application introduces a logical flaw in its authorization flow: the developers have omitted authorization on the backend route for $V_{8,}$ allowing direct unauthenticated attempt queries from standard users.

Table 3.3: Graph edge weighing based on security level

| Edge | Weight | Description |
|---|---|---|
| $V_1$, $V_5$ | 1 | Web guest accessing the home page |
| $V_5$, $V_9$ | 1 | Home page retrieves data from the general database |
| $V_2$, $V_4$ | 3 | Users go through intermediary to validate auth |

| | | |
|---|---|---|
| $V_4$, $V_6$ | 3 | Users are authorized and redirected to the profile page after successful validation |
| $V_6$, $V_9$ | 3 | Profile page retrieves standard user data from the general database |
| $V_3$, $V_4$ | 5 | Admin transmits high privilege credentials to be validated |
| $V_4$, $V_7$ | 5 | After successful validation, admins are granted access to the admin dashboard |
| $V_7$, $V_8$ | 5 | Access to a hidden endpoint is invoked through the dashboard interface |
| $V_8$, $V_{10}$ | 10 | Hidden endpoint executes backend queries to retrieve sensitive data from the isolated, central database |
| $V_2$, $V_8$ | 1 | Standard users are given direct access to the hidden endpoint |

## IV. Results and Analysis

### F. Adjacency Matrix

The relation between nodes can be represented using an adjacency matrix. A value of 1 signifies the existence of an edge from the row vertex to the column vertex, whereas 0 tells of the lack of a direct connection. The adjacency matrix of the previously mapped web application topology is as follows:

Table 4.1: Adjacency Matrix of the Web Application Topology

| | $V_1$ | $V_2$ | $V_3$ | $V_4$ | $V_5$ | $V_6$ | $V_7$ | $V_8$ | $V_9$ | $V_{10}$ |
|---|---|---|---|---|---|---|---|---|---|---|
| $V_1$ | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| $V_2$ | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| $V_3$ | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| $V_4$ | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 |
| $V_5$ | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| $V_6$ | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| $V_7$ | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| $V_8$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| $V_9$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| $V_{10}$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

While the adjacency matrix provides a strict mathematical foundation, visualizing these relations through a directed graph offers a more intuitive perspective of the web application's architecture, as seen in Fig 4.1 below:

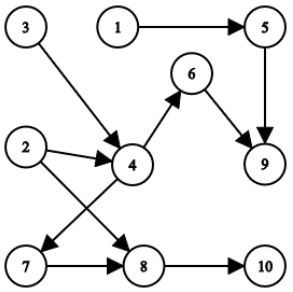


Fig 4.1: Graph representation of the web application topology

### G. Applying the Warshall Algorithm

The Warshall Algorithm is used to determine whether an indirect path exists between two vertices. With this knowledge, we can make use of the algorithm to check if a user has access to forbidden endpoints within the application. The following matrix represents the final result of applying the Warshall algorithm on the adjacency matrix in Table 4.1.

Table 4.2: Transitive Closure Matrix of the Web Topology

| | $V_1$ | $V_2$ | $V_3$ | $V_4$ | $V_5$ | $V_6$ | $V_7$ | $V_8$ | $V_9$ | $V_{10}$ |
|---|---|---|---|---|---|---|---|---|---|---|
| $V_1$ | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| $V_2$ | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 |
| $V_3$ | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 |
| $V_4$ | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 |
| $V_5$ | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| $V_6$ | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| $V_7$ | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 |
| $V_8$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| $V_9$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| $V_{10}$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

Applying the Warshall algorithm introduces numerous changes in the adjacency matrix. For example, there exists a newly established path where the vertex $V_1$ has indirect access to $V_9$. This connection is intended by design, as public data and assets stored within should be accessible to all visitors, guests included.

Most notably, it is visible that numerous endpoints have access to the central, isolated database harboring confidential data. A legitimate connection exists between $V_3$ and $V_{10}$, as

admins should have full governance over sensitive internal data. However, $V_2$, the user endpoint, demonstrates unauthorized reachability to $V_{10}$. There is a clear indication of flawed business logic being present, but a connection does not guarantee access. As such, it would be reasonable to postulate that the information available at present is insufficient to draw any meaningful conclusions, and that further measurements are necessary.

### *H. Further Analysis Through the Floyd Algorithm*

While the Warshall Algorithm gives a practical view on the topological relations of endpoints, it would be inadequate to draw a conclusion so prematurely. Simply knowing whether a path exists is not enough information to properly evaluate the adequacy of a web application's defense mechanisms. By employing the Floyd algorithm, the analysis shifts towards finding the path of least resistance, effectively exposing vulnerable paths that require minimal security clearance to breach. The Floyd algorithm yields the following output:

Table 4.3: Path of Least Resistance Weight Matrix

| | $V_1$ | $V_2$ | $V_3$ | $V_4$ | $V_5$ | $V_6$ | $V_7$ | $V_8$ | $V_9$ | $V_{10}$ |
|---|---|---|---|---|---|---|---|---|---|---|
| $V_1$ | 0 | ∞ | ∞ | ∞ | 1 | ∞ | ∞ | ∞ | 2 | ∞ |
| $V_2$ | ∞ | 0 | ∞ | 3 | ∞ | 6 | 8 | 1 | 9 | 11 |
| $V_3$ | ∞ | ∞ | 0 | 5 | ∞ | 8 | 10 | 15 | 11 | 25 |
| $V_4$ | ∞ | ∞ | ∞ | 0 | ∞ | 3 | 5 | 10 | 6 | 20 |
| $V_5$ | ∞ | ∞ | ∞ | ∞ | 0 | ∞ | ∞ | ∞ | 1 | ∞ |
| $V_6$ | ∞ | ∞ | ∞ | ∞ | ∞ | 0 | ∞ | ∞ | 3 | ∞ |
| $V_7$ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | 0 | 5 | ∞ | 15 |
| $V_8$ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | 0 | ∞ | 10 |
| $V_9$ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | 0 | ∞ |
| $V_{10}$ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | 0 |

As opposed to simply disclosing whether a path exists between two vertices, each cell within the matrix now contains the cumulative security weight required when traversing the validation workflow. The path between $V_2$ and $V_9$ is weighed at 2 security levels, an appropriate value given the open-to-public access. For an administrator, accessing the central database requires passing through a rigorous, multi-layered security route. On the other hand, a standard user is faced with a disproportionately low cumulative weight of 11 in order to infiltrate the exact same core system assets. This stark mathematical contrast ($11 < 25$) proves that the vulnerability acts as a path of least resistance, allowing attackers to completely bypass the intended authorization architecture with minimal effort.

## V. Conclusion

The highly complex architecture of a web application inevitably produces an equally complex topological network of communication nodes. As a result, mapping out its vast security infrastructure may prove to be tough labour, one not free of human error. Consequently, said infrastructure tends to harbor vulnerabilities malicious parties could freely exploit. Therefore, a mathematical method capable of thoroughly mapping a complex network is necessary in order to minimize human error.

The application's security posture first goes through a preliminary evaluation by simply determining whether a path exists between two nodes. The Warshall algorithm is a highly effective tool in mapping paths between nodes. However, the Warshall algorithm is not the solution to all problems, as some paths are a necessary foundation of the systemic functionality. Thus, further evaluation is needed to reach an absolute verdict on a path's security level. Employment of the Floyd algorithm returns the path of least resistance, providing the crucial information to determine the vulnerability severity and its cumulative exploitable weight.

Ultimately, this method only serves to transform the topological network into a quantifiable mathematical model. Whether or not a path poses a risk to the system's integrity comes down to the security engineer's decision. For this reason, it is imperative that context and business logic remain relevant throughout the evaluation process.


## Acknowledgment

The author would like to deliver his utmost thanks, first and foremost, to the Lord for His endless blessings throughout the 2nd semester at Bandung Institute of Technology. He would also like to show similar appreciation towards his parents, whose bottomless and ever-present love for him helped push him to keep on moving forward. The author expresses his sincere gratitude to Mr. Rinaldi Munir for the comprehensive and detailed lecture materials and academic insights that served as the foundation for this study. Lastly, a heartfelt thank you is extended towards my friend Edward Terrance Lie, whose presence and support I am very grateful for.


## References


[1] OWASP Foundation, "A01:2021 – Broken Access Control," *OWASP Top 10:2021*, 2021. [Online]. Available: https://owasp.org/Top10/A01_2021-Broken_Access_Control/. [Accessed: 19 Jun. 2026].

[2] K. H. Rosen, *Discrete Mathematics and Its Applications*, 8th ed. New York, NY, USA: McGraw-Hill, 2019. [Online]. Available: https://cis.temple.edu/~latecki/Courses/CIS2166-Fall25/RosenDiscreteMath8Ed.pdf. [Accessed: 19 Jun. 2026].

[3] R. Munir, "Relasi dan Fungsi Bagian 1," materi kuliah Matematika Diskrit, STEI ITB, Bandung, Indonesia, 2026. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2025-2026/05-Relasi-dan-Fungsi-Bagian1-(2026).pdf. [Accessed: 19 Jun. 2026].

[4] R. Munir, "Relasi dan Fungsi Bagian 3," materi kuliah Matematika Diskrit, STEI ITB, Bandung, Indonesia, 2026. [Online]. Available:

https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2025-2026/07-Relasi-dan-Fungsi-Bagian3-(2026).pdf. [Accessed: 19 Jun. 2026].

[5] R. Munir, "Graf Bagian 1," materi kuliah Matematika Diskrit, STEI ITB, Bandung, Indonesia, 2026. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2025-2026/20-Graf-Bagian1-2026.pdf. [Accessed: 19 Jun. 2026].

[6] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. Cambridge, MA, USA: MIT Press, 2009, p. 695. [Online]. Available: https://www.cs.mcgill.ca/~akroit/math/compsci/Cormen%20Introduction%20to%20Algorithms.pdf. [Accessed: 19 Jun. 2026].

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Jakarta, 19 Juni 2026

Kelvin Sebastian Yen 13525068