

Quantitative Security Analysis using Hypergraph Based Attack Models

*Project report submitted in partial fulfillment of the requirements
for the degree of*
Master of Computer Application

by

Purnendu Manna

Class Roll No.: 002210503043

Examination Roll No.: MCA244042

Registration No.: 163635 of 2022-23

under the supervision of

Dr. Mridul Sankar Barik
Assistant Professor

Department of Computer Science and Engineering
Faculty of Engineering and Technology
Jadavpur University
2024

Faculty of Engineering and Technology Jadavpur University

Certificate of Recommendation

To Whom It May Concern

I hereby recommend that the project report entitled “**Quantitative Security Analysis using Hypergraph Based Attack Models**” has been carried out by **Purnendu Manna** (Registration No.: 163635 of 2022-23, Class Roll No.: 002210503043, Examination Roll No.: MCA244042), under my guidance and supervision and be accepted in partial fulfilment of the requirements for the degree of Master of Computer Application in the Department of Computer Science & Engineering, Jadavpur University, during the academic year 2023-2024.

Dr. Mridul Sankar Barik
Assistant Professor
Dept. of Comp. Sc. & Engg.
Jadavpur University
(**Project Supervisor**)

Prof. Nandini Mukhopadhyay
Head of the Department
Dept. of Comp. Sc. & Engg.
Jadavpur University

Prof. Dipak Laha
Dean,
Faculty of Engineering & Technology
Jadavpur University

Faculty of Engineering and Technology
Jadavpur University

Certificate of Approval

This is to certify that the project report entitled “**Quantitative Security Analysis using Hypergraph Based Attack Models**” is a bonafide record of work carried out by **Purnendu Manna**, in fulfillment of the requirements for the award of the degree of Master of Computer Application in the Department of Computer Science and Engineering, Jadavpur University. It is understood that by this approval the undersigned do not necessarily endorse or approve any statement made, opinion expressed or conclusion drawn therein but approve the project report only for the purpose for which it has been submitted.

Signature of Examiner 1

Date:

Signature of Examiner 2

Date:

Faculty of Engineering and Technology

Jadavpur University

Declaration of Compliance with Academic Ethics

I hereby declare that this project report entitled "**Quantitative Security Analysis using Hypergraph Based Attack Models**" is an outcome of my genuine and independent work, as part of my Master of Computer Application program. All information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, all materials and results that are not original to this work have been fully cited and referenced

Name: Purnendu Manna

Registration No.: 163635 of 2022 – 23

Examination Roll No.: MCA244042

Thesis Title: Quantitative Security Analysis using Hypergraph Based Attack Models

Signature with Date

Acknowledgement

With my most sincere and gratitude, I would like to thank **Dr. Mridul Sankar Barik**, Department of Computer Science & Engineering, my supervisor, for his overwhelming support throughout the duration of the project. His motivation always gave me the required inputs and momentum to continue with my work, without which the project work would not have taken its current shape. His valuable suggestions and numerous discussions have always inspired new ways of thinking. I feel deeply honored that I got this opportunity to work under him.

I would like to express my sincere thanks to all my teachers for providing sound knowledge base and cooperation. I would like to thank all the faculty members of the Department of Computer Science & Engineering of Jadavpur University for their continuous support.

Last, but not the least, I would like to thank my batch mates for staying by my side when I needed them.

Purnendu Manna

Class Roll No.: 002210503043

Exam Roll No.: MCA244042

Department of Computer Science & Engineering,
Jadavpur University.

Contents

1	Introduction	1
1.1	Background	1
1.2	Research Statement	2
1.3	Contribution of the Project	2
1.4	Outline of the Project Report	3
2	Hypergraphs	5
2.1	Introduction	5
2.2	Hypergraph	5
2.3	Undirected Hypergraph	5
2.3.1	Definition of Undirected Hypergraph	6
2.3.2	Cardinality of a Hyperedge	6
2.3.3	Degree of a Vertex	6
2.3.4	Incidence Matrix	7
2.3.5	Adjacency Matrix	7
2.3.6	Entropy of Undirected Hypergraph	7
2.4	Directed Hypergraph	9
2.4.1	Definition of Directed Hypergraph	9
2.4.2	Directed Hypergraph Traversal	9
2.4.3	Entropy of Directed Hypergraph	14
3	Hypergraph Based Attack Models	20
3.1	Attack Graph	20
3.1.1	An Example Attack Graph	20
4	Hypergraph Based Forensic Analysis	23
4.1	Evidence Graphs	25
4.2	Forensics in SDN	27
5	Analytics Based on Attack Hypergraph	29
5.1	Experiment 1	30
5.2	Experiment 2	37

6	Conclusion and Future Work	43
6.1	Conclusion	43
6.2	Future Work	43
6.2.1	Precision of Entropy Calculation	44
6.2.2	Practical Applications in Network Security	44
6.2.3	Optimization of Hyperedge Patching	44
6.2.4	Enhancing Analytical Frameworks	44

List of Figures

2.1	Example of an Undirected Hypergraph	6
2.2	Example of a Directed Hypergraph	10
2.3	Directed Hypergraph with Cost Factor (adapted from [5])	12
2.4	Minimum Cost Path in Figure 2.3	14
2.5	Different types of s-overlaps in Directed Hypergraph	15
3.1	An Example Network Configuration	20
3.2	Attack Graph of Example Network (shown in Figure 3.1)	22
4.1	Example Network	25
4.2	Attack Graph of Network Topology 4.1	26
4.3	Evidence Graph of Figure 4.2	27
5.1	Example of Attack Hypergraph	30
5.2	Attack Hypergraph (adapted from [3])	37

Abstract

Cloud Computing has revolutionized the way IT service are now offered to end users. It has become widespread and is also essential foundation to support current and future technologies of IT industry. This project tackles this challenge by introducing hypergraph-based entropy as a new way to analyze network security. Hypergraphs, unlike traditional models, can capture complex interactions between elements. The research offers a new traversal algorithm for attack scenarios, considers different levels of overlap when measuring uncertainty, and even proposes a method to identify and fix network vulnerabilities. Extensive testing confirms that strategically patching weaknesses based on this analysis significantly improves network security. Through comprehensive experimentation, the research validates the efficacy of these metrics, showcasing that strategic fortification informed by hypergraph entropy analysis substantially fortifies network resilience. Beyond furnishing practical tools for cybersecurity practitioners, this research lays the groundwork for future endeavors that fuse hypergraph theory with advanced mathematical frameworks to fortify cybersecurity defenses further.

Chapter 1

Introduction

1.1 Background

The digital age has seen an unprecedented increase in the complexity and frequency of cyber threats. As information technology continues to evolve, so too do the methods employed by malicious actors. Traditional security measures often fall short in the face of sophisticated, multi-stage cyber attacks. Consequently, there is a pressing need for advanced security models that can provide a more comprehensive understanding and defense against these threats.

One promising avenue of research in this domain is the use of hypergraphs for security analysis. Unlike traditional graphs, which only connect pairs of nodes, hypergraphs can connect any number of nodes with a single hyperedge. This capability allows hypergraphs to model complex relationships and interactions more effectively, making them particularly suitable for representing multi-stage attack scenarios.

In this project, we explore the application of hypergraph-based entropy measures as a quantitative security metric in Attack hypergraph. Entropy, a concept borrowed from information theory, measures the uncertainty or disorder within a system. By applying this concept to hypergraphs, we aim to develop a metric that can quantify the complexity and security of network configurations.

An attack graph is a tool used to assess security risks. It helps identify all the different ways an attacker could break into a system. It does this by considering two things: weaknesses in software (vulnerabilities) and how an attacker might move around once they're inside (lateral movements). Previous research has demonstrated the major impact of attack graph in the field of cybersecurity [2][16] used to formulaise for modelling multistage attacks. Hypergraph is a generalization of ordinary graph and contrary to the ordinary graphs, hyperedges capture multiway relationships that may exist between nodes. Hence, hypergraph based attack models are more expressive compared to its graph based counterpart. This research seeks to bridge this gap by developing and validating hypergraph-based models for security analysis. By leveraging the unique properties of hypergraphs, we aim to provide deeper insights into network vulnerabilities and enhance the effectiveness of cybersecurity measures.

The following sections will delve into the specifics of hypergraphs, outline the research problem, and detail the contributions of this project. Our goal is to establish a robust framework for using hypergraphs in cybersecurity, paving the way for more sophisticated and resilient defense mechanisms against cyber threats.

1.2 Research Statement

The aim of this project is to explore the suitability of hypergraph-based entropy measures as a security metric. In an era where cyber threats are growing both in complexity and frequency, traditional security models often prove inadequate in addressing sophisticated, multi-stage attacks. As information systems become more interconnected and interdependent, the need for advanced, comprehensive security metrics is more critical than ever.

This research seeks to harness the unique properties of hypergraphs to provide a more nuanced and detailed analysis of network security. Unlike traditional graph models, which represent relationships as simple pairwise connections, hypergraphs allow for the representation of higher-order relationships. This capability is particularly advantageous in cybersecurity, where interactions and dependencies between multiple entities are common and often intricate.

By applying entropy measures to hypergraph structures, this project aims to quantify the uncertainty and complexity within a network. Entropy, a concept derived from information theory, serves as a measure of disorder or unpredictability in a system. When integrated with hypergraph models, entropy can provide a powerful tool for assessing the robustness and vulnerability of network configurations. This approach promises to yield deeper insights into the nature of cyber threats, enabling more effective identification and mitigation of potential security risks.

To achieve this goal, the research will involve the development of hypergraph-based models and the application of entropy measures to these models. This involves both theoretical formulation and practical implementation, using real-world data to validate the proposed methods. The effectiveness of hypergraph-based entropy as a security metric will be evaluated through comprehensive experiments.

Ultimately, this project work aspires to contribute to the field of cybersecurity by introducing a novel and robust metric for security analysis. By leveraging hypergraph theory and entropy measures, the research aims to enhance our ability to understand and defend against complex cyber threats. This work will pave the way for future research and development in the application of advanced mathematical models to cybersecurity, offering new avenues for protecting critical information systems in an increasingly digital world.

1.3 Contribution of the Project

This project work makes several significant contributions to the field of cybersecurity by developing advanced methodologies for evaluating and enhancing security using hypergraph-based entropy measures. The main contributions are as follows:

Traversal Algorithm for Directed Attack Hypergraph:

The project introduces novel traversal algorithms tailored for directed attack hypergraphs. These algorithms are designed to efficiently navigate through the hypergraph, identifying critical paths and nodes that are pivotal for understanding and mitigating security threats.

Entropy Measures for Directed Attack Hypergraphs with s-overlaps:

A comprehensive framework for calculating the entropy of directed attack hypergraphs is developed, considering different types of overlaps. The analysis includes head-tail, tail-tail, and other forms

of s-overlaps. These entropy measures provide a quantitative assessment of the security posture, highlighting areas of high complexity and potential vulnerability.

Identifying and Patching Exploits:

The research demonstrates an effective methodology for identifying the most vulnerable nodes within the attack hypergraph. By focusing on nodes with high hyperedge centrality (HDC), the project provides a strategy for prioritizing patches. This approach ensures that patching efforts are concentrated on nodes that significantly reduce the overall entropy when secured, thus enhancing the robustness of the network.

Efficient Patch Deployment:

The project work outlines a systematic approach for deploying patches to minimize the hypergraph's entropy. By analyzing different patching scenarios and their impact on the hypergraph's entropy, it provides a clear strategy for reducing vulnerability. The results show that patching a combination of fan-in and fan-out nodes is more effective than focusing solely on one type, ensuring a balanced reduction in potential exploit paths.

Through these contributions, the project work advances the understanding of how hypergraph-based models can be utilized to enhance cybersecurity measures, offering practical tools and insights for researchers and practitioners in the field.

1.4 Outline of the Project Report

The report is structured to explore and establish the viability of hypergraph-based entropy measures as a security metric, with a focus on enhancing cybersecurity. It begins with an introduction to hypergraphs, highlighting their potential to represent complex relationships that traditional graphs cannot, and sets the stage for their application in modeling multistage attack scenarios.

The report then delves into the mathematical foundation of hypergraphs, distinguishing between undirected and directed hypergraphs. It outlines the definitions, properties, and key components of each type, including hyperedge cardinality, vertex degree, incidence matrix, adjacency matrix, and the concept of entropy as applied to undirected hypergraphs. This theoretical groundwork lays the foundation for understanding the subsequent practical applications in cybersecurity. Building on this, the report introduces a traversal algorithm specifically designed for directed attack hypergraphs, emphasizing its utility in navigating these structures to identify critical paths and nodes. This is followed by a detailed framework for calculating the entropy of directed attack hypergraphs, taking into account various types of overlaps such as head-tail and tail-tail. These entropy measures provide a quantitative assessment of the network's security posture, pinpointing areas of high complexity and vulnerability.

The report further introduces the concept of an attack graph, providing a detailed explanation of its significance and effects on real-world network topology and cybersecurity. This includes an in-depth analysis of how attack graphs can be utilized to understand and mitigate potential security threats. Furthermore, the report illustrates a real-world attack scenario within a network, demonstrating the process of generating an attack graph for that specific network. This example

helps to highlight the practical applications and benefits of using attack graphs in enhancing network security measures.

The report then explores the critical role of forensic analysis in investigating digital crimes, highlighting the tools and methodologies used to uncover, preserve, and analyze digital evidence. With the increasing complexity of cybercrimes, forensic analysis has evolved to manage vast data volumes, identify complex malicious patterns, and maintain the integrity of evidence. The chapter reviews state-of-the-art forensic technologies and techniques, emphasizing operating system forensics, network forensics, and multimedia forensics. It also discusses advanced approaches such as evidence graph construction, vulnerability reasoning, and machine learning in forensics. Various studies and frameworks, including a provenance-based forensic model for IoT environments and an integrated forensic analysis framework for Software-Defined Networks (SDN), are examined to illustrate their effectiveness in modern forensic investigations.

Building on this, the report addresses the identification and prioritization of patches for the most vulnerable nodes within the attack hypergraph by analyzing some experiments on actual attack hypergraph. By focusing on nodes with high hyperedge centrality, the project presents a strategy for efficient patch deployment that aims to minimize the hypergraph's overall entropy, thereby enhancing the network's robustness against potential exploits. Different patching scenarios are analyzed to demonstrate that a combination of fan-in and fan-out node patching is more effective than targeting a single type.

Finally, the report consolidates its findings and contributions, proposing a novel and robust metric for security analysis based on hypergraph theory and entropy measures. It underscores the potential of this approach to offer deeper insights into cyber threats and more effective methods for identifying and mitigating security risks. The work concludes with implications for future research and development in applying advanced mathematical models to cybersecurity, paving the way for improved defense mechanisms in an increasingly digital world.

Chapter 2

Hypergraphs

2.1 Introduction

Hypergraph is a powerful mathematical tool that extends the reach of traditional graphs. Graphs have revolutionized our understanding of relationships, from social networks to transportation systems. However, they can only connect vertices in pairs. Hypergraphs break free from this limitation, allowing for connections between any number of vertices. This opens doors to modeling complex relationships where multiple elements play a crucial role. Despite their potential, hypergraphs haven't received the same level of attention as graphs. This gap in knowledge presents an exciting opportunity. By delving into the world of hypergraphs, we can uncover new ways to represent and analyze intricate connections in various domains.

This paper aims to bridge this gap by introducing hypergraphs to model multistage attack scenarios, analyzing them with different functionalities of hypergraphs, delving into their key components, and unveiling their applications in the field of cybersecurity. Finally, we will present the specific research question (or hypothesis) driving our exploration of hypergraphs in the field of cybersecurity, paving the way for a deeper understanding of this versatile mathematical tool.

2.2 Hypergraph

In contrast to a graph, which can only unite two vertices using an edge, a **Hypergraph** is a broad type of graph that can join any number of vertices within an edge. It facilitates the mapping of intricate relationships between various nodes. There are mainly **two** types of hypergraphs present - **Undirected Hypergraph** and **Directed Hypergraph**.

2.3 Undirected Hypergraph

An *undirected hypergraph* is a generalization of a simple undirected graph in which the connections, known as hyperedges, can link more than two nodes. In traditional undirected graphs, edges connect exactly two nodes, forming pairs. In hypergraphs, hyperedges can connect any number of nodes, allowing for more complex and versatile relationships.

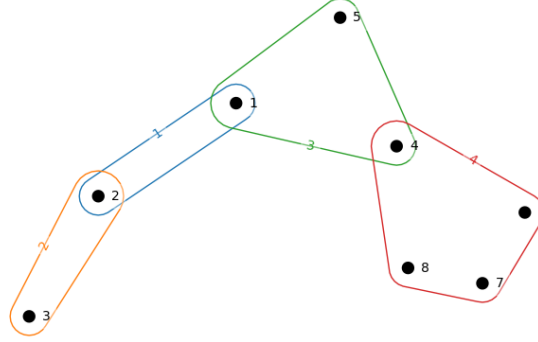


Figure 2.1: Example of an Undirected Hypergraph

2.3.1 Definition of Undirected Hypergraph

An undirected hypergraph H is defined as a pair of (V, E) where V is a finite non-empty set of vertices ($V = \{v_i | 1 \leq i \leq N\}$) and E is the set of finite hyperedges ($E = \{e_i | 1 \leq i \leq M\}$). Each hyperedge $e \in E$ consists of non-empty subsets of V and such that $\bigcup_{i=1}^M e_i = V$.

Example : Consider an example of undirected hypergraph $H = (V, E)$ in Figure 2.1 where $V = \{v_1, v_2, \dots, v_8\}$ and $E = \{e_1, e_2, e_3, e_4\}$. Here we can see, unlike normal graph edges, hyperedges can map more than two vertices in a hyperedge. Here, $e_1 = \{v_1, v_2\}$, $e_2 = \{v_2, v_3\}$, $e_3 = \{v_1, v_5, v_4\}$ and $e_4 = \{v_4, v_6, v_7, v_8\}$.

2.3.2 Cardinality of a Hyperedge

Cardinality of a hyperedge $e \in E$ of a *Hypergraph* $H(V, E)$ indicates the number of vertex participated in a hyperedge, i.e., $|e| = \{n(v_i) | v_i \in e\}$. For example, in Figure 2.1 the cardinality of e_1 , $|e_1| = 2$, as only v_1 and v_2 participated in e_1 . whereas, the cardinality of e_4 , $|e_4| = 4$ as v_4, v_6, v_7 and v_8 participated in e_4 .

2.3.3 Degree of a Vertex

Degree of a vertex v is the number of edges e_j in which the vertex participated in the Hypergraph $H(V, E)$. For example in Figure 2.1, degree of v_4 , $d(v_4) = 2$ as v_4 participated in two hyperedges, i.e., e_3 and e_4 . whereas, degree of v_5 , $d(v_5) = 1$ as v_5 only participated in hyperedge e_3 .

2.3.4 Incidence Matrix

Incidence Matrix of a hypergraph is induced matrix that can be described in matrix form such that it describes the existence of relation between nodes and hyperedges. Let a hypergraph H with vertices $V = \{v_1, v_2, \dots, v_n\}$ and hyperedges $E = \{e_1, e_2, \dots, e_m\}$. Let $I(H) = ((a_{i,j}))_{(i,j) \in \{1..m\} \times \{1..n\}}$ be the induced matrix of H , where $a_{i,j} = 1$ if v_j is present in e_i otherwise $a_{i,j} = 0$. The matrix is a $m \times n$ matrix where m is the number of hyperedges and n is the number of nodes present.

2.3.5 Adjacency Matrix

Adjacency Matrix of a hypergraph H can be induced by dot(.) product of incidence matrix $(I(H))$ with itself. It describes the number of nodes overlapping between two hyperedges.

$$L(H) = I(H)I(H)^t \quad (2.1)$$

where $I(H)$ is the incidence matrix of the hypergraph H and $L(H)$ is a matrix of $m \times m$ where m is the number of hyperedges in the Hypergraph H .

2.3.6 Entropy of Undirected Hypergraph

To quantify the quantity of information contained in or provided by an information source, the notion of entropy has been established. Its goal is to simulate the various combinatorial states that can exist within a given set. It goes without saying that a probability distribution of these various states' occurrences models them. This allows for the creation of a "relatively straightforward" (in terms of summarization) understanding of the informational content of the studied items. As a result, the main focus of this theory of information is probability. Since its debut, this idea has been expanded upon in a variety of scientific fields, including stochastic theory, computer science, and dynamic systems. Shannon is credited with introducing entropy into information theory.

Shanon Entropy of a Undirected Hypergraph

The following derivation has been taken from the work presented in [4]. Let $H(V, E)$ be a simple undirected *Hypergraph* with vertices $V = \{v_1, v_2, \dots, v_n\}$ and hyperedges $E = \{e_1, e_2, \dots, e_m\}$. Let $I(H) = ((a_{i,j}))_{(i,j) \in \{1..m\} \times \{1..n\}}$ be the induced matrix of H , where $a_{i,j} = 1$ if v_j is present in e_i otherwise $a_{i,j} = 0$. Now we consider the matrix $m \times m$ defined as,

$$L(H) = I(H)I(H)^t = ((e_i \cap e_j))_{i,j \in \{1..m\}} \quad (2.2)$$

Here, $L(H)$ is a positive semi-definite matrix and eigenvalues of the matrix $L(H)$ are denoted by $\lambda_i(L(H))$, where $i = 1..m$ and all eigenvalues are positive and can be ordered as,

$$0 \leq \lambda_1(L(H)) \leq \lambda_2(L(H)) \leq \dots \leq \lambda_m(L(H)) \quad (2.3)$$

Trace of Hypergraph(H) : The sum of degree $d(v)$ of each of the vertices v in the matrix H is the trace of the matrix $L(H)$, i.e., sum of all edges that contain v .

$$Tr(L(H)) = \sum_{v \in V} d(v) \quad (2.4)$$

Proof. $Tr(L(H)) = \sum_{i=1}^m \lambda_i = \sum_{e \in E} |e| = \sum_{x \in V} d(x)$. Let us define :

$$\mu_i = \frac{\lambda_i(L(H))}{\sum_{i=1}^m \lambda_i(L(H))} = \frac{\lambda_i(L(H))}{Tr(L(H))} \quad (2.5)$$

The μ_i are the eigenvalues of the normalized matrix

$$\mathcal{L}(H) = \frac{1}{Tr(L(H))} L(H) \quad (2.6)$$

which is also positive and semi-definite.

Classically, the *Shannon entropy* of the hypergraph H is defined as:

$$S(H) = - \sum_{i=1}^m \mu_i \log_2(\mu_i) \quad (2.7)$$

Example : Consider the example of hypergraph $H = (V, E)$ in Figure 2.1 where $V = \{v_1, v_2, \dots, v_8\}$ and $E = \{e_1, e_2, e_3, e_4\}$. Here we can see, unlike normal graph edges, hyperedges can map more than two vertices in a hyperedge. Here, $e_1 = \{v_1, v_2\}$, $e_2 = \{v_2, v_3\}$, $e_3 = \{v_1, v_5, v_4\}$ and $e_4 = \{v_4, v_6, v_7, v_8\}$.

Let us compute the entropy of the Hypergraph 2.1,

Firstly, we have to define incidence matrix of the hypergraph H , i.e.,

$$I(H) = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix} \quad (2.8)$$

where, $I(H)$ be a 4×8 matrix, number of hyperedges(m) is 4 and number of nodes(n) is 8. Now adjacency matrix of hypergraph H be,

$$\begin{aligned} L(H) &= I(H)I(H)^t \\ &= \begin{bmatrix} 2 & 1 & 1 & 0 \\ 1 & 2 & 0 & 0 \\ 1 & 0 & 3 & 1 \\ 0 & 0 & 1 & 4 \end{bmatrix} \end{aligned} \quad (2.9)$$

As the $L(H)$ is 4×4 matrix, there will be 4 eigenvalues for the corresponding adjacency matrix respectively,

$$(\lambda_1, \lambda_2, \lambda_3, \lambda_4) = (0.71600213, 4.75080042, 3.3184589, 2.21473855) \quad (2.10)$$

And Trace of Hypergraph ($Tr(L(H))$) = $\sum_{i=1}^m \lambda_i = \sum_{e \in E} |e| = \sum_{x \in V} d(x) = 11$ Classically, the

Shannon Entropy of the Hypergraph is defined as,

$$\begin{aligned}
S(H) &= - \sum_{i=1}^m \mu_i \log_2(\mu_i) \\
&= - \sum_{i=1}^m \left(\frac{\lambda_i(L(H))}{Tr(L(H))} \right) \log_2 \left(\frac{\lambda_i(L(H))}{Tr(L(H))} \right) \\
&\approx 1.767
\end{aligned} \tag{2.11}$$

2.4 Directed Hypergraph

A *directed hypergraph* is a type of hypergraph where the hyperedges have a specific direction, indicating a more complex relationship among nodes. This structure extends the concept of a traditional directed graph to allow hyperedges to connect multiple nodes, offering greater flexibility and complexity in modeling relationships.

2.4.1 Definition of Directed Hypergraph

A directed hypergraph H is defined as a pair of (V, E) where V is finite non-empty set of vertices ($V = \{v_i | 1 \leq i \leq N\}$) and E is the set of finite hyperedges ($E = \{e_i | 1 \leq i \leq M\}$). Each hyperedge is an ordered pair $e(X, Y)$ where X and Y are disjoint non-empty subsets of V . (X, Y) describes relation from $Tail(X)$ to $Head(Y)$, such that, $e = (X, Y) = (T(e), H(e)) \in E, T(e) \neq \emptyset, H(e) \neq \emptyset$, and $T(e) \cap H(e) = \emptyset$.

Example : Consider an example of directed hypergraph $H = (V, E)$ in Figure 2.2 where $V = \{v_1, v_3, ..v_{26}\}$ and $E = \{e_1, e_2, ..e_7\}$. And every hyperedge can be defined by a set of ordered pair X of $\langle T(X), H(X) \rangle$ where $T(X)$ is the nodes that belong in tail end of the directed hyperedge X and $H(X)$ is the nodes that belong in the head of the directed hyperedge X . Consider the hyperedge e_1 , can be defined by the ordered pair of $\langle T(e_1), H(e_1) \rangle$ where $T(X) = \{v_{22}, v_{23}, v_{26}\}$ and $H(X) = \{v_{18}\}$.

2.4.2 Directed Hypergraph Traversal

Directed hypergraph traversal refers to the process of navigating through a directed hypergraph by following its directed hyperedges from source to target nodes. This traversal can be complex due to the multiple nodes involved in each hyperedge and the potential for complex structures like cycles, branches, or different connectivity patterns.

Directed hypergraphs are a powerful tool for representing complex systems with intricate relationships. Traversal allows you to navigate through these relationships, understanding the structure and connections between different parts. It is useful for various applications, including network analysis, workflow optimization, and data flow analysis.

In essence, traversal is a way to explore and analyze directed hypergraphs. It helps you understand how different parts of the system are interconnected and how data flows through it. This can be useful for a variety of tasks, such as identifying bottlenecks in a process, finding the most efficient route between two points, or understanding how data is transformed as it moves through a system.

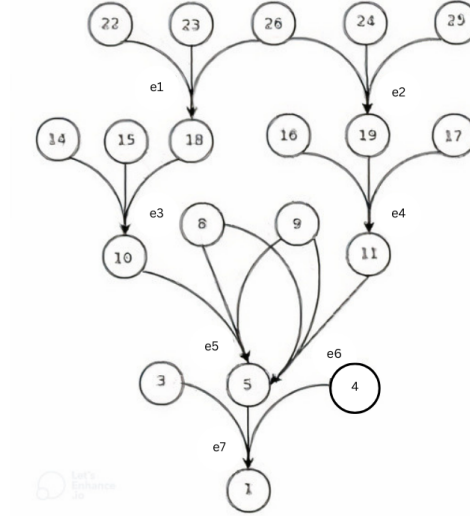


Figure 2.2: Example of a Directed Hypergraph

In this study we are going to use traversal to *Attack Hypergraph* and study the path from source node to destination node so that we can analyze how attacker may go from the source node to destination node so that we can take prevention measures.

Head and Tail of Directed Hyperedge

A hyperedge $e \in E$ has direction from a non-empty set of source nodes to a non-empty set of destination nodes. Head of a hyperedge $H(e)$ is the destination part and Tail of the hyperedge is the source part of the hyperedge e .

FanIn and FanOut Hyperedges of Vertex

Let $v \in V$ be a vertex of directed hypergraph $H = (V, E)$, then fanIn is the number of hyperedges came to the vertex v , and fanOut is number of hyperedges goes from vertex v .

Forward Star of Directed Hyperedge

Forward star [5] as the set of edges in H having v in their tail, that is $FS_H(v) = \{e \in E : v \in T(e)\}$. For a set of nodes $\bar{V} \subset V$, we define their predecessors as the set of edges in H having $\bar{V}$ as their head, that is $prec(\bar{V}) = \{e \in E : \bar{V} = H(e)\}$.

Backward Star of Directed Hyperedge

Backward star [5] as the set of edges in H having v in their head, that is $BS_H(v) = \{e \in E : v \in H(e)\}$. For a set of nodes $\bar{V} \subset V$, we define their successors as the set of edges in H having $\bar{V}$ as

their tail, that is $\text{succ}(\bar{V}) = e \in E : \bar{V} = T(e)$.

For instance, in the hypergraph 2.2, we can see Tail of the hypergraph $T(e_1) = \{n_{22}, n_{23}, n_{26}\}$ and Head of the hypergraph $H(e) = \{n_{18}\}$. And forward star of n_5 is $FS_h(n_5) = \{e_7\}$ and Backward Star of n_5 is $BS_h(n_5) = \{e_5, e_6\}$.

Modified Depth First Search for Hypergraph Traversal with Forward Star FS_h

Depth-First Search (DFS) is a classic algorithm for traversing and exploring directed graphs, where each edge has a specified direction. It's a fundamental algorithm in computer science, used for a variety of tasks such as exploring graphs, finding connected components, detecting cycles, topological sorting, and more.

But in the case of Hypergraph classic DFS algorithm might not work suitably as the complexity of Directed Hypergraph is very much to tackle by classic DFS algorithm. In directed hypergraph multiple there can be multiple nodes in a hyperedge unlike normal directed graph. That why we need to modify the classic DFS algorithm so that we can use it for Directed Hypergraph. In the later section we will also introduce the cost factor and minimum cost path to reach destination node from source node in the hypergraph.

In the previous definitions we came to know about forward star of a node v , that is the set of edges having v in their tail. So we can traverse to the set of forward star edges from the node v . And we will use this concept in the DFS algorithm to traverse in the directed Hypergraph.

Algorithm 1 Modified Depth First Search Traversal Algorithm

Require: Initial node $start$, target node $target$ with value $target.val$

Ensure: A path from $start$ to $target$ (if found), otherwise 'None'

```

1:  $visited \leftarrow \emptyset$  (Initialize an empty set for visited nodes)
2: Function DFS( $node, currEdge = ''$ )
3:  $visited.add(node)$  (Mark current node as visited)
4: if  $node.val == target.val$  then
5:   return [ $currEdge$ ] (Target found, return current path)
6: end if
7:  $fstar \leftarrow node.fstar$  (Get node's frontier)
8: for  $edge \in fstar$  do
9:   for  $vertex \in Head(edge)$  do
10:    if  $vertex \notin visited$  then
11:       $path \leftarrow \text{DFS}(vertex, currEdge + ' ' + edge)$  (Recursive call)
12:      if  $path \neq \text{None}$  then
13:        return [ $currEdge$ ] +  $path$  (Path found, return extended path)
14:      end if
15:    end if
16:  end for
17: end for
18: return  $\text{None}$  (No path found from current node)
19:
20: return DFS( $start$ ) (Call DFS with initial node)
```

Let's consider the directed hypergraph in Figure 2.2, and take source node as node n_{22} and destination node as node n_1 . Initially, the visited set is set to $\emptyset$ as no node is visited initially. So the traversal will go from $e_1 \rightarrow e_3 \rightarrow e_5 \rightarrow e_7$, where node n_1 exists.

Modified Dijkstra Algorithm to find Minimum Cost Path in Directed Hypergraph using Forward Star FS_h

In the previous section, we discussed the directed Hypergraph traversal algorithm without introducing any path cost to the hyperedge. But if we introduce a cost factor to the hypergraph the **Modified Depth First Search algorithm (1)** may not work. And the cost factor is crucial for graphs as it can be used to prioritize different hyperedges with others. So it is very important to traverse the hypergraph with a cost factor, and in most cases, we try to find the minimum cost path in graphs.

In the ordinary graph, we use **Dijkstra's Algorithm** to find the minimum cost path from the source node to the destination node. Dijkstra's algorithm operates on a graph where each edge has a non-negative weight, representing the cost or distance between nodes. It efficiently finds the shortest path from a source node to all other nodes in the graph, constructing a shortest-path tree. The key idea is to iteratively select the node with the shortest known distance and explore its neighbors, updating their distances if a shorter path is found.

But we can't directly use ordinary Dijkstra's algorithm in hypergraphs as multiple nodes in one hyperedge. So we need to modify the Dijkstra's algorithm by introducing the forward star concept of a node in the hyperedge, similar to the modified DFS algorithm, but we will use a greedy approach to find the minimum cost path in the hypergraph.

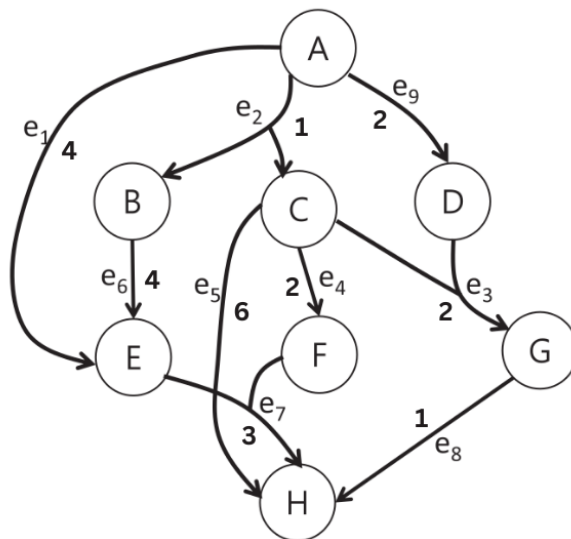


Figure 2.3: Directed Hypergraph with Cost Factor (adapted from [5])

Algorithm 2 Modified Dijkstra Algorithm to find Minimum Cost Path in Directed Hypergraph

```
1: Input: A graph represented by edges dictionary ('self.edges'), cost dictionary ('self.costs'),  
   starting node ('start'), target node ('target'), and all nodes ('nodes') in the graph  
2: Output: A list representing the minimum cost path from 'start' to 'target' (if found), or 'None'  
   (if not found)  
3: Initialize an empty dictionary 'distance' to store distances from 'start' to each node, with all  
   distances set to positive infinity ('+∞')  
4: Initialize an empty dictionary 'parent' to store parent nodes in the path  
5: Set 'distance[start]' to 0 (distance to starting node is 0)  
6: Initialize a priority queue 'pq' using a min-heap (stores tuples of (distance, node))  
7: Insert '(0, start)' into 'pq' (starting node with distance 0)  
8: while pq ≠ ∅ do  
9:   /* Remove node with minimum distance from priority queue */  
10:  (currDist, currNode) = heappop(pq)  
11:  if currNode == target then  
12:    /* Target node reached, reconstruct path */  
13:    Initialize an empty list 'path' to store the path nodes  
14:    while currNode ≠ start do  
15:      Append 'currNode' to 'path'  
16:      Set 'currNode = parent[currNode]' (move to parent in the path)  
17:    end while  
18:    Append 'start' to 'path'  
19:    Reverse the 'path' list (to get the path from start to target)  
20:    return path  
21:  end if  
22:  Let 'fstar' be 'currNode.fstar' (presumably containing edges associated with the node)  
23:  for edge ∈ fstar do  
24:    for vertice ∈ Head(edge) do  
25:      Calculate new distance: 'newDistance = costs[edge] + currDist'  
26:      if newDistance < distance[vertice] then  
27:        Update distance for 'vertice': 'distance[vertice] = newDistance'  
28:        Update parent for 'vertice': 'parent[vertice] = currNode'  
29:        /* Add vertice with new distance to priority queue */  
30:        heappush(pq, (newDistance, vertice))  
31:      end if  
32:    end for  
33:  end for  
34: end while  
35: /* No path found if while loop exits normally */  
36: return None
```

Let's consider the directed hypergraph Figure 2.3 , and use the above algorithm to find the minimum cost path from source node A to destination node H. And the minimum cost path goes through $A \rightarrow C \rightarrow G \rightarrow H$ with path cost of 4. As we can see in the figure 2.4, the red path is the minimum cost path in the hypergraph figure 2.3.

The modified Dijkstra's algorithm might work well on most of the hypergraphs but it also comes

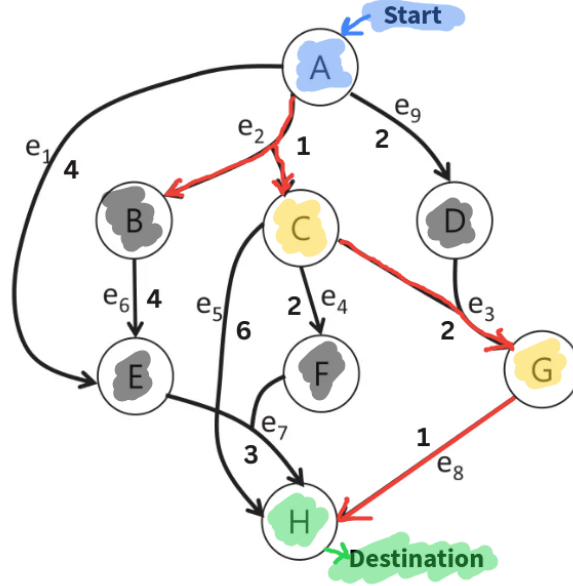


Figure 2.4: Minimum Cost Path in Figure 2.3

with flaws. It might not be able to find the actual minimum cost path for every hypergraph as it works on a greedy approach that might not be the optimal choice for every situation. And we know Dijkstra's algorithm can not detect negative cycles but may work with negative hyperedges. So in the future, we need to develop new traversal algorithms that can incorporate these flaws.

2.4.3 Entropy of Directed Hypergraph

Directed hypergraph entropy is a concept that merges ideas from information theory and graph theory to measure the complexity or uncertainty in directed hypergraphs. A directed hypergraph is a generalization of a directed graph, where an edge (also called a hyperedge) can connect any number of nodes. Hypergraphs are useful in modeling complex relationships in various domains, such as computer networks, biological systems, social networks, and more.

Entropy, in general, quantifies the amount of uncertainty or information content in a system. For a directed hypergraph, entropy can measure the unpredictability of node connections or the distribution of hyperedges. The proposed method captures high-order information through high-order line graphs of hypergraphs and uses the variation of Von Neumann entropy to measure changes in the network's complexity. The more drastic the change, the greater the impact of the node on the network's complexity and the more important the node is. At the same time, to balance the complexity and accuracy of the method, we proposed semi-quadratic approximate Von Neumann entropy centrality (semi-SAVC) by using quadratic approximations of Von Neumann entropy and some high-order information.

Hypergraph and s-overlaps of Directed Hypergraph

Consider a directed hypergraph, which consists of N nodes and M hyperedges and is defined as an ordered pair $H = (V, E)$, where $V = \{v_1, v_2, v_3, \dots, v_N\}$ is a finite set of nodes, v_i ($i = 1, 2, 3, \dots, N$) is a vertex of the hypergraph, $E = \{e_1, e_2, e_3, \dots, e_M\}$ is a set of hyperedges and e_j is a hyperedge of the hypergraph, subject to condition.

$$\begin{cases} e_j \neq \emptyset (j = 1, 2, 3, \dots, M) \\ \bigcup_{j=1}^M e_j = V \end{cases} \quad (2.12)$$

For e_j , its cardinality is denoted as $r_j = |e_j|$. The incidence matrix of the directed hypergraph H is denoted by $I(H) = ((a_{i,j}))_{(i,j) \in \{1..M\} \times \{1..N\}}$, where $a_{i,j} = 1$ if v_j is present in e_i otherwise $a_{i,j} = 0$. The matrix is a $M \times N$ matrix, where M is the number of hyperedges and N is the number of nodes present.

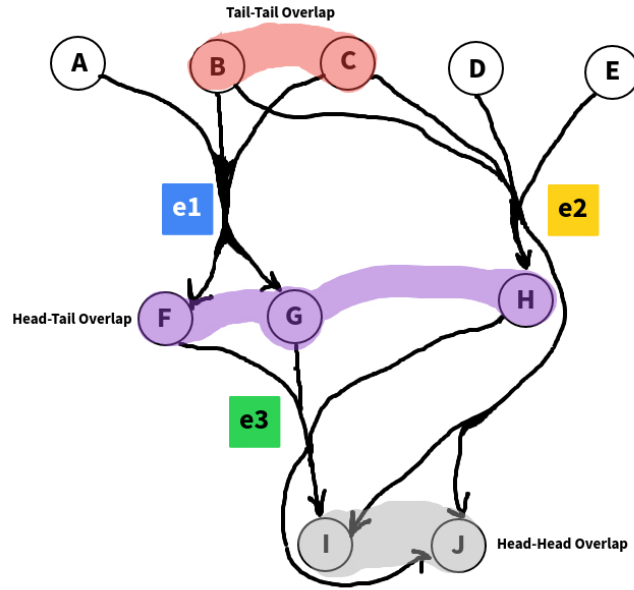


Figure 2.5: Different types of s-overlaps in Directed Hypergraph

In a hypergraph, the **s-overlap** [7] refers to the number of nodes shared between two hyperedges, which is at least s . Figure 2.5 gives an example of a directed hypergraph. Let $\mathcal{H} = (V, E)$ be the hypergraph, then $V = \{A, B, \dots, J\}$ and $E = \{e_1, e_2, e_3\}$. A directed hyperedge has two parts: - **tail** and the **head** -

Adjacency List of Hypergraph 2.5		
Hyperedge	Head Set	Tail Set
e_1	$\{A, B, C\}$	$\{F, G\}$
e_2	$\{B, C, D, E\}$	$\{H, I, J\}$
e_3	$\{F, G, H\}$	$\{I, J\}$

There are three kinds of s-overlaps that can be possible in a directed hypergraph.

Tail-Tail s-Overlap : If $\text{Tail}(T_i)$ of a hyperedge $e_i \in E$ and $\text{Tail}(T_j)$ of another hyperedge $e_j \in E$ of a hypergraph $\mathcal{H}$ have s common nodes in between, then it is called Tail-Tail s-overlap between e_i and e_j . As we can see in figure 2.5, $\text{Tail}(e_1) = \{A, B, C\}$ and $\text{Tail}(e_2) = \{B, C, D, E\}$ and the Tail-Tail overlap between e_1 and e_2 is $(\text{Tail}(e_1) \cap \text{Tail}(e_2)) = \{B, C\}$ and $s = |\text{Tail}(e_1) \cap \text{Tail}(e_2)| = 2$. So there is a tail-tail 2-overlap between e_1 and e_2 in the hypergraph $\mathcal{H}$.

The Tail-Tail s-overlap of a directed hypergraph is written as a symmetric matrix M_{TT} , where $M_{TT} = ((s_{i,j}))_{(i,j) \in \{1..M\} \times \{1..M\}}$, where M is the number of hyperedges and each $s_{i,j}$ represents the tail-tail s-overlap between e_i and e_j . For example, in Figure 2.5, Tail-Tail s-overlap can be represented as,

$$M_{TT} = \begin{array}{c|ccc} & e_1 & e_2 & e_3 \\ \hline e_1 & 0 & 2 & 0 \\ e_2 & 2 & 0 & 0 \\ e_3 & 0 & 0 & 0 \end{array} \quad (2.13)$$

Head-Tail s-Overlap : If $\text{Head}(H_i)$ of a hyperedge $e_i \in E$ and $\text{Tail}(T_j)$ of another hyperedge $e_j \in E$ of a hypergraph $\mathcal{H}$ have s common nodes in between, then it is called Head-Tail s-overlap between e_i and e_j . In figure 2.5, $\text{Head}(e_1) = \{F, G\}$, $\text{Head}(e_2) = \{H, I, J\}$, and $\text{Tail}(e_3) = \{F, G, H\}$. The head-tail overlap between e_1 and e_3 is $(\text{Head}(e_1) \cap \text{Tail}(e_3)) = \{F, G\}$ where value of $s = |\text{Head}(e_1) \cap \text{Tail}(e_3)| = 2$ that is, a Head-Tail 2-overlap, and the head-tail overlap between e_2 and e_3 is $(\text{Head}(e_2) \cap \text{Tail}(e_3)) = \{H\}$ where value of $s = |\text{Head}(e_2) \cap \text{Tail}(e_3)| = 1$ that is, a Head-Tail 1-overlap in the hypergraph $\mathcal{H}$.

The Head-Tail s-overlap of a directed hypergraph is written as a symmetric matrix M_{HT} , where $M_{HT} = ((s_{i,j}))_{(i,j) \in \{1..M\} \times \{1..M\}}$, where M is the number of hyperedges and each $s_{i,j}$ represents the head-tail s-overlap between e_i and e_j . For example, in Figure 2.5, Head-Tail s-overlap can be represented as,

$$M_{HT} = \begin{array}{c|ccc} & e_1 & e_2 & e_3 \\ \hline e_1 & 0 & 0 & 2 \\ e_2 & 0 & 0 & 1 \\ e_3 & 0 & 0 & 0 \end{array} \quad (2.14)$$

Head-Head s-Overlap : If $\text{Head}(H_i)$ of a hyperedge $e_i \in E$ and $\text{Head}(H_j)$ of another hyperedge $e_j \in E$ of a hypergraph $\mathcal{H}$ have s common nodes in between, then it is called Head-Head s-overlap between e_i and e_j . As we can see in figure 2.5, $\text{Head}(e_2) = \{H, I, J\}$ and $\text{Head}(e_3) = \{I, J\}$ and the Head-Head overlap between e_2 and e_3 is $(\text{Head}(e_2) \cap \text{Head}(e_3)) = \{I, J\}$ and $s = |\text{Head}(e_2) \cap \text{Head}(e_3)| = 2$. So there is a Head-Head 2-overlap between e_1 and e_2 in the hypergraph $\mathcal{H}$.

The Head-Head s-overlap of a directed hypergraph is written as a symmetric matrix M_{HH} , where $M_{HH} = ((s_{i,j}))_{(i,j) \in \{1..M\} \times \{1..M\}}$, where M is the number of hyperedges and each $s_{i,j}$ represents the head-head s-overlap between e_i and e_j . For example, in Figure 2.5, Head-Head s-overlap can be represented as,

$$M_{HH} = \begin{array}{c|ccc} & e_1 & e_2 & e_3 \\ \hline e_1 & 0 & 0 & 0 \\ e_2 & 0 & 0 & 2 \\ e_3 & 0 & 2 & 0 \end{array} \quad (2.15)$$

In [Research Paper 1], Bryce introduced the Von Neumann entropy and explained how to quantify unpredictability in digraphs. He did this by defining the diagonal matrix D of the adjacency

matrix Γ and using $D = [d_{ij}]$, where $d_{ii} = d^+(v_i)$, and determining $L = D - \Gamma$ to find the *Laplacian matrix* L . And again, dividing L by its trace gives the normalized Laplacian matrix $\bar{L}$.

But in the case of a directed hypergraph, the paradigm is a bit different; that's why we can't use the adjacency matrix directly, as in the mapping of a hyperedge, multiple edges participate to formulate a more generalized way to bind relations between multiple nodes, unlike in an ordinary digraph where an edge has only two nodes. That's why this paper introduced the concept of the *s-overlap matrix* $\mathcal{M}$ that defines the overlapping relation between two hyperedges instead of the normal adjacency matrix of an ordinary digraph.

Now the diagonal matrix D will be:

$$(D[i][i])_{i=1}^M = \sum_{j=1}^M \mathcal{M}[i][j] \quad (2.16)$$

where $D \in \{D_{TT}, D_{HT}, D_{HH}\}$ and $\mathcal{M} \in \{M_{TT}, M_{HT}, M_{HH}\}$ for different s-overlap diagonal matrix where off-diagonal elements will be 0 ($D[i][j] = 0$ for $i \neq j$).

Now define *Laplacian Matrix* L ,

$$L(\mathcal{H}) = D - \mathcal{M} \quad (2.17)$$

And the *Normalized Laplacian Matrix* $\bar{L}$ will be,

$$\bar{L}(\mathcal{H}) = \frac{1}{Tr(L(\mathcal{H}))} L(\mathcal{H}) \quad (2.18)$$

Directed Hypergraph Entropy

The concept of utilizing entropy for the analysis of graphs is not novel, and its precise application has adapted to various contexts. One such application, upon which our discourse is founded, originates from the domain of quantum information theory. In quantum mechanics, a quantum state is characterized by a density matrix, which possesses specific attributes: it is Hermitian, positively semi-definite, and has a unit trace (where the trace equals 1). This suggests that the density matrix's spectrum, or the set of eigenvalues, can be understood as a discrete probability distribution, according to a theorem in linear algebra and matrix theory. The eigenvalues are guaranteed to be real by their "Hermitian" property, to be positive and semi-definite by their guarantee of nonnegative eigenvalues, and to sum up to one by their unit trace. The von Neumann entropy of the state is derived by applying the Shannon entropy to this spectrum. Similarly, the normalized Laplacian matrix of an undirected graph shares analogous properties, thus suggesting an analogy to a density matrix of a physical system. Hence, it is reasonable to define the von Neumann entropy of a graph in this manner.

In this study, those concepts will be expanded to the setting of directed hypergraphs, where a new method is required because the normalized Laplacian's spectrum is complex-valued and has little chance of being understood as a discrete probability distribution.

If the directed hypergraph $\mathcal{H}$ and the normalized Laplacian matrix is $\bar{L}(\mathcal{H})$, then entropy of the directed hypergraph can be defined as,

$$V(\mathcal{H}) = \sum_{\lambda \in \Lambda} -\lambda \log_2(\lambda) \quad (2.19)$$

where Λ is spectrum of $\bar{L}(\mathcal{H})$; that is, the multiset of its eigenvalues.

The following proposition says that we can talk about $V(\mathcal{H})$ without ambiguity or potential misunderstanding, assuming calculations are done correctly. In other terms, it is saying that if one person calculates $V(\mathcal{H})$, their result will be the same as someone else who calculates $V(\mathcal{H})$, even if the names of vertices or edges of $\mathcal{H}$ are labeled differently.

For example, in Figure 2.5 we previously found the s-overlap matrices $\mathcal{H}$, now diagonal matrices D of these s-overlap matrices will be respectively,

$$D_{TT} = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix}, D_{HT} = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, D_{HH} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix}$$

Now, Laplacian matrices will be for different s-overlaps, respectively.

$$L_{TT}(\mathcal{H}) = \begin{bmatrix} 2 & -2 & 0 \\ -2 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix}, L_{HT}(\mathcal{H}) = \begin{bmatrix} 2 & 0 & -2 \\ 0 & 1 & -1 \\ 0 & 0 & 0 \end{bmatrix}, L_{HH}(\mathcal{H}) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & -2 \\ 0 & -2 & 2 \end{bmatrix}$$

And the normalized Laplacian matrices will be, respectively,

$$\bar{L}_{TT}(\mathcal{H}) = \begin{bmatrix} \frac{2}{4} & \frac{-2}{4} & 0 \\ \frac{-2}{4} & \frac{2}{4} & 0 \\ \frac{0}{4} & \frac{0}{4} & \frac{0}{4} \end{bmatrix}, \bar{L}_{HT}(\mathcal{H}) = \begin{bmatrix} \frac{2}{3} & 0 & \frac{-2}{3} \\ 0 & \frac{1}{3} & \frac{-1}{3} \\ 0 & 0 & 0 \end{bmatrix}, \bar{L}_{HH}(\mathcal{H}) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & \frac{2}{4} & \frac{-2}{4} \\ 0 & \frac{-2}{4} & \frac{2}{4} \end{bmatrix}$$

Now the eigenvalues will be,

	λ_1	λ_2	λ_3
$\bar{L}_{TT}(\mathcal{H})$	1	0	0
$\bar{L}_{HT}(\mathcal{H})$	0.66666667	0.33333333	0
$\bar{L}_{HH}(\mathcal{H})$	1	0	0

(2.20)

then the entropy of the directed hypergraph $\mathcal{H}$ will be,

	$V(\mathcal{H})$
$V_{TT}(\mathcal{H})$	0
$V_{HT}(\mathcal{H})$	0.6365141682948128
$V_{HH}(\mathcal{H})$	0

(2.21)

The above example can describe the complexity and uncertainty of the hypergraph. As the result suggests, there are three different resultant entropy for different s-overlaps in the directed hypergraph, and we can see that for the Head-Tail s-overlap, the resultant entropy is the highest of all. That means, the complexity or uncertainty in directed hypergraphs is highest for head-tail s-overlap.

Hyperedge Centrality (HDC)

In [7], it introduced the concept of hyperedge centrality (HDC), which measures the importance of nodes by the number of incident hyperedges. The more incident hyperedges a node has, the more

important it is. Consider a directed hypergraph, which consists of N nodes and M hyperedges and is defined as an ordered pair $H = (V, E)$, where $V = \{v_1, v_2, v_3, \dots, v_N\}$ is a finite set of nodes, v_i ($i = 1, 2, 3, \dots, N$) is a vertex of the hypergraph, $E = \{e_1, e_2, e_3, \dots, e_M\}$ is a set of hyperedges, and e_j is a hyperedge of the hypergraph, subject to condition. And the incidence matrix of the directed hypergraph H is denoted by $I(H) = ((a_{i,j}))_{(i,j) \in \{1..M\} \times \{1..N\}}$, where $a_{i,j} = 1$ if v_j is present in e_i otherwise $a_{i,j} = 0$. The matrix is a $M \times N$ matrix, where M is the number of hyperedges and N is the number of nodes present. Then, hyperdegree centrality is defined as:

$$HDC(v_i) = \sum_{j=1}^M a_{ji} \quad (2.22)$$

where a_{ji} is the (j, i) -th element of the incidence matrix A of the hypergraph.

Chapter 3

Hypergraph Based Attack Models

3.1 Attack Graph

Attack graphs are popular modeling tool for multistage attacks. It encodes system security states and attacker actions in form of dyadic relations resulting in a graph. Given the information on system components and their vulnerabilities, network topology and configuration, we can generate attack graphs using open source tools such as MulVAL [1].

Formally, an attack graph is defined as a directed graph $G = (V, E)$. The set of vertices comprises of two types, i.e. $V = E_x \cup S_c$, where E_x is the set of exploits and S_c is the set security conditions. Similarly, the set of edges comprises of two types, i.e. $E = R_r \cup R_i$. Require type directed edges connect security conditions to exploits, i.e. $R_r \subseteq S_c \times E_x$ and imply type directed edges connect exploits to security conditions, i.e. $R_i \subseteq E_x \times S_c$.

3.1.1 An Example Attack Graph

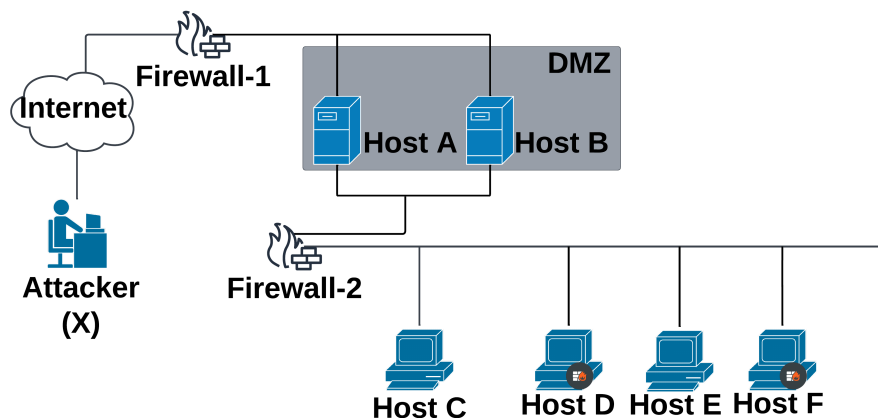


Figure 3.1: An Example Network Configuration

Let us consider an example network configuration as shown in Figure 3.1. The demilitarized zone comprises of Host A and Hosts B. The internal network has Hosts C, D, E, and F. Host F has particular significance as it holds critical resources. The attacker operates from the external network. The attacker’s objective is to attain root-level privileges on Host F. To mitigate potential threats, firewalls (both perimeter and host-based) have been strategically positioned. The internal hosts within the network, including Hosts A, B, C, D, E, and F, and the attacker are limited to connecting solely to specific ports or services based on the comprehensive firewall policies outlined in Table 3.1.

Table 3.1: Firewall Policies of Example Network (shown in Figure 3.1)

Attacker (X)	Attacker (X)	HostA	HostB	HostC	HostD	HostE	HostF
	localhost	All	All	None	None	None	None
HostA	All	localhost	All	Netfilter	None	None	None
HostB	All	All	localhost	None	None	Vmware	None
HostC	All	All	All	localhost	Red Hat	All	None
HostD	All	All	All	All	localhost	All	Kaspersky
HostE	All	All	All	All	Red Hat	localhost	None
HostF	All	All	All	All	All	All	localhost

These policies delineate the constraints on connectivity. The term “All” indicates that a source host is permitted to connect to a destination host on any port to access the services running on those ports. Conversely, “None” signifies that a source host is prohibited from accessing any services on the destination host. In our example network, we assume that the hosts initially possess certain vulnerabilities, which are summarized in Table 3.2. For example, Host A and B have vulnerabilities, CVE-2021-44235 and CVE-2022-22964, which, if exploited successfully, can grant an attacker root-level privileges on Host A and B respectively.

Figure 3.2 shows an attack graph for our example network configuration. It enumerates potential attack paths that an attacker could take to reach the target Host F.

Table 3.2: Vulnerabilities of Example Network (shown in Figure 3.1)

Host	Services	CVE ID
Host A	SAP Netweaver	CVE-2021-44235
Host B	VMware Horizon Agent	CVE-2022-22964
Host C	Netfilter subsystems	CVE-2023-0179
Host D	Red Hat Enterprise	CVE-2018-1111
Host E	Linux kernel	CVE-2020-14386
Host F	Kaspersky Antivirus	CVE-2017-9811

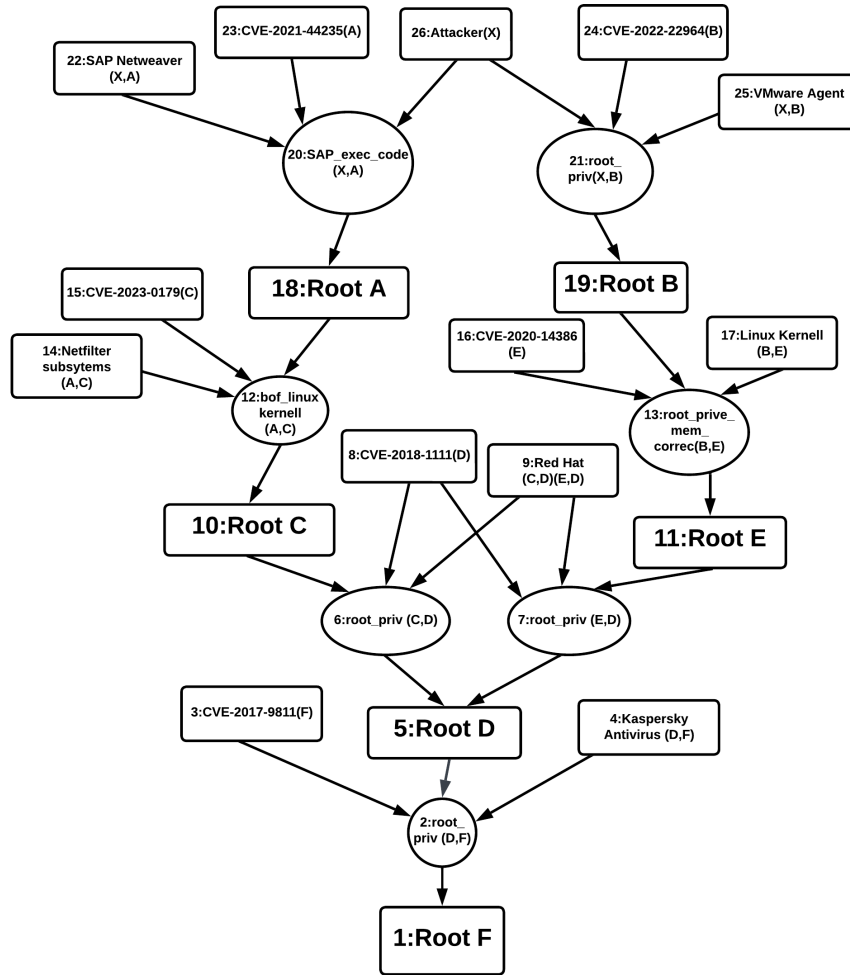


Figure 3.2: Attack Graph of Example Network (shown in Figure 3.1)

Chapter 4

Hypergraph Based Forensic Analysis

Forensic analysis plays a pivotal role in the investigation of digital crimes, providing the necessary tools and methodologies to uncover, preserve, and analyze digital evidence. As cybercrimes become increasingly sophisticated, the field of forensic analysis has evolved to address the complex challenges associated with modern digital environments. This evolution is driven by the need to efficiently handle vast amounts of data, identify intricate patterns of malicious activities, and ensure the integrity of digital evidence throughout the investigative process.

In The paper [8], A. R. Javed et al (2022) provides a comprehensive survey of the state-of-the-art technologies, tools, and techniques used in computer forensics. It examines various forensic domains, including operating system forensics, network forensics, and multimedia forensics, highlighting the essential features and comparative effectiveness of both commercial and open-source forensic toolkits. Among the key technologies covered are forensic tools that assist tasks like memory analysis, network traffic monitoring, and digital evidence extraction from many systems, such as Belkasoft Evidence Center, OSForensics, and Network Miner. The survey introduces a Feature Scoring Model (FSM) for evaluating the capabilities of these tools, aiding forensic investigators in selecting the most appropriate toolkit for different investigative scenarios based on specific features and requirements. The paper also identifies current challenges and future research directions, emphasizing the need for advancements in live forensics, registry forensics, and the integration of adversary machine learning and deep learning techniques in forensic analysis.

A comprehensive approach to forensic analysis in network environments using a combination of evidence graph construction and vulnerability reasoning was presented by J. He et al. (2016) [6]. The core technology involves the use of a Support Vector Machine (SVM) for filtering suspicious data, ensuring the efficient classification of network traffic by distinguishing between benign and suspicious packets. This filtered data is then utilized to construct an evidence graph, which serves as the foundation for the forensic analysis. Additionally, the vulnerability evidence reasoning algorithm (VERA) is employed to reconstruct attack scenarios and trace back the network packets to their original sources. The proposed method was validated through experiments using datasets from the MIT Lincoln Laboratory, specifically LLDOS 1.0 and LLDOS 2.0.2, demonstrating its effectiveness and accuracy in rapid and effective network forensics analysis .

In another paper Leichtnam et al. (2020) [10], introduces a novel approach for forensic analysis

of network attacks by automating the processing of network events. This method utilizes graph-based representations of "Security Objects" derived from network event logs, which capture the essential attributes and relationships between events. The technology employed involves creating graphs where nodes represent security objects (such as IP addresses, port numbers, and network connections) and edges denote the relationships between these objects. By leveraging community detection algorithms within these graphs, the approach can identify clusters of related events, facilitating the discovery of indicators of compromise (IoCs) and providing a comprehensive view of the network activities associated with potential attacks.

Nisioti et al. (2023) [13] discuss forensic analysis using a variety of advanced technologies to enhance the investigation of sophisticated cyber incidents, particularly multi-stage attacks by strategic adversaries. The technologies employed include attack graphs for mapping evidence and identifying missing elements, clustering techniques to analyze large sets of unstructured documents, Visual Security Analytics (VSA) for decision support in live forensics, and Bayesian networks to weigh evidence and aid decision-making during triage stages. Additionally, the paper discusses models such as the Targeted Complex Attack Network (TCAN) for combining dynamic attack graphs with network evolution, and frameworks like MLAPT for machine learning-based reconstruction of multi-stage APT attacks. These methods are evaluated on their efficiency and decision support capabilities, highlighting their strengths and areas for future research to better support digital forensic investigations in the evolving threat landscape .

Sadineni et al. (2022) [15], introduce ProvNet-IoT, a novel provenance-based forensic model aimed at investigating network-level attacks in Internet of Things (IoT) environments. The framework is designed to handle the complexity of IoT network traffic and uses information modeling, functional modeling, and event modeling to create detailed provenance graphs from network captures. ProvNet-IoT's methodology includes preprocessing network traffic data, generating provenance information, and conducting forensic analysis to detect incidents and collect evidence. Technologies used in this approach include Particle Deep Framework (PDF), Particle Swarm Optimization (PSO), Multi-layer Perception (MLP) deep learning algorithms, and various network forensic tools like Netfox Detective and IoT Network Analyzer. These tools enable detailed analysis of network traffic, identification of malicious activities, and reconstruction of attack scenarios, providing reliable and forensically sound evidence .

Raftopoulos et al. (2013) [14], propose a novel approach to forensic analysis of network attacks by restructuring network event logs into graph structures called "Security Objects" (SOs). This methodology automates the identification of relevant information linked to indicators of compromise (IoCs). The process involves generating graphs from network logs, where nodes represent security events and edges denote relationships between these events. By employing community detection techniques on these graphs, the approach isolates strongly connected sub-graphs that highlight both normal activities and potential security incidents. The system's effectiveness was demonstrated using the CICIDS 2017 dataset, showcasing its ability to manage large datasets and identify intricate attack patterns that might be overlooked through traditional methods .

In summary, hypergraph-based forensic analysis offers a sophisticated and versatile approach to understanding complex data relationships. By transcending the limitations of traditional graph models, hypergraphs provide a powerful tool for digital forensics, enabling deeper insights and more effective investigative outcomes. As the field of digital forensics progresses, the integration of hypergraph theory is poised to play a crucial role in advancing analytical capabilities and addressing the challenges of modern data environments.

4.1 Evidence Graphs

An evidence graph is a detailed representation where nodes signify hosts involved in an attack and edges represent forensic evidence linking these hosts. Probabilistic evidence graphs extend this concept by incorporating probabilities that reflect the certainty and relevance of each piece of evidence. These graphs are crucial in forensic analysis as they provide a structured way to visualize and analyze the complex interrelationships and dependencies in attack scenarios, helping investigators understand the sequence of events, identify key actors, and estimate the likelihood of different attack paths.

Definition (Evidence Graph) [12] : An evidence graph is a sextuple $G = (N, E, N\text{-Attr}, E\text{-Attr}, L, T)$ where N is a set of nodes representing host computers, $E \subseteq (N_i \times N_j)$ is a set of directed edges each consisting of a particular data item that indicates activities between the source and target machines, $N\text{-Attr}$ is a set of node attributes that include the host ID, attack states, timestamp and host importance, and $E\text{-Attr}$ is a set of edge attributes each consisting of an event description, evidence impact weight, attack relevance and host importance. Functions $L : N \rightarrow 2^{N\text{-Attr}}$ and $T : E \rightarrow 2^{E\text{-Attr}}$ assign attribute-value pairs to nodes and edges, respectively.

The $N\text{-Attr}$ attack states comprise one or more of attack source, target, stepping-stone and affiliated host computers. Affiliated hosts are computers that have suspicious interactions with an attacker, a victim or stepping-stone host. For example,

Liu, et al. (2013) [11] conducted a multi-state attack on a small-scale experimental network as shown in Figure 4.1. The network setup includes an external firewall controlling Internet access to an enterprise network with an Apache Tomcat webserver, which allows Internet users to use port 8080. An internal firewall regulates access to a MySQL database server on port 3306, accessed by the webserver and workstations. The webserver logs incoming IP addresses, the database server records queries, and the IDS monitors network traffic.

The objective was for an Internet user to gain access to database tables via a SQL injection attack exploiting unsanitized java servlet code: `theStatement.executeQuery("select * from profiles where name='Alice' AND password='"+passWord+"'");`, corresponding to CWE89 in NVD. A Windows XP SP3 workstation with IE6 vulnerability (CVE-2009-1918) was exploited through social engineering, tricking the user to click a malicious link, enabling remote control via TightVNC.

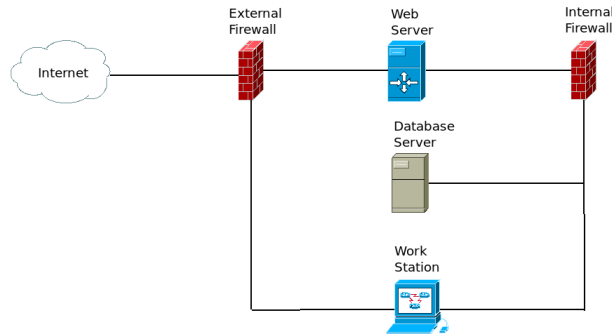


Figure 4.1: Example Network

An attack graph generated by MulVAL is shown in Figure 4.2, with single exploit probabilities from CVSS and human factors in Table 1. A key probability factor, user interaction, was assigned

a value of 0.8. The forensic analysis used attacker roles listed in Table 2, including attacker IP, stepping stone, web server, and victim database server.

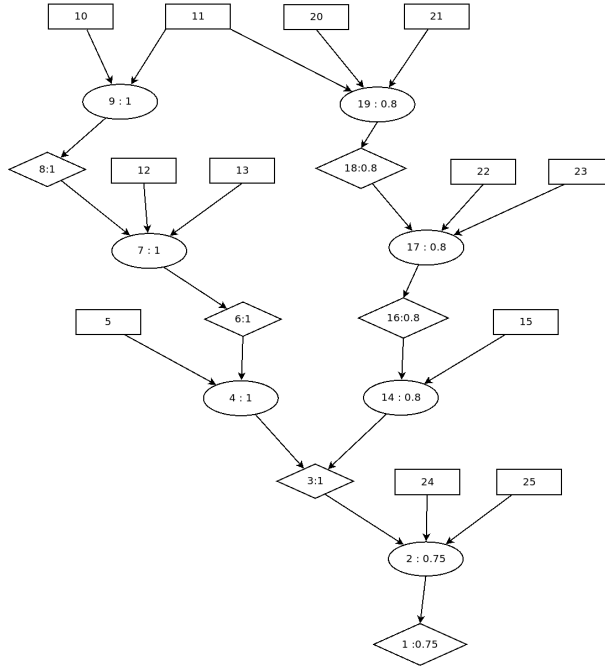


Figure 4.2: Attack Graph of Network Topology 4.1

Number	Configuration/Vulnerability	Probability
1	Direct Network Access	1
2	Social Engineering	0.8
3	CVE-2009-1918	1
4	CWE-89	0.75
5	Access to database from workstation	1

Attacker	129.174.128.148
Stepping Stone	Workstation
Stepping Stone/Affiliated	Web server
Victim	Database server

Our forensic analysis uses the attacker roles stated in Table 2, corresponding to the following scenario.

1. An un-sanitized string 'anything' or '1='1' is fed into the password field of the Webpage causing a SQL injection attack to the database "profiles".

2. Internet Explorer "Aurora" Memory Corruption attack was launched from the attacker machine to the Windows workstation with the IE6 browser by tricking a windows workstation user to click a link sent from the attacker machine.
3. After hijacking the workstation, TightVNC is uploaded to the workstation, allowing to remotely control the workstation that has direct access to the database.

Evidence was categorized, with primary evidence including the SQL injection query timestamped "120416 15:32:51". This was cross-referenced with logs from the webserver and workstations, identifying a suspicious IP "129.174.92.32" as the attacker. Thus, expert knowledge suggested that an attack path from the attacker through the workstation to the database server, which completed another attack path "*attacker* $\rightarrow$ *workstation* $\rightarrow$ *databaseserver*" in the evidence graph.

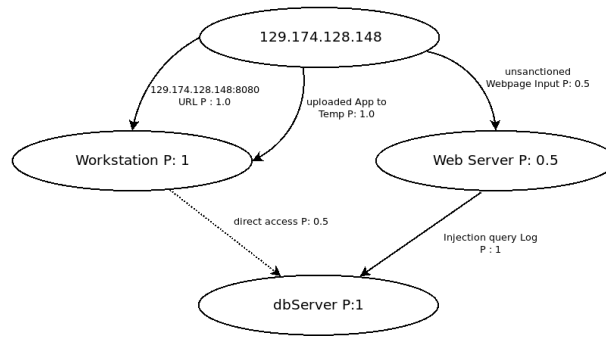


Figure 4.3: Evidence Graph of Figure 4.2

In another paper, Liu, et al. (2013) proposed in [12], an algorithm for merging multiple evidence graphs like **Probabilistic Evidence Graph**, **Logical Evidence Graph** to provide a comprehensive view of network attacks. The core technology used in this paper involves probabilistic evidence graphs and attack graphs. Evidence graphs are utilized to model network intrusion evidence and represent dependencies between various pieces of evidence. Nodes in these graphs denote computers of interest, while edges depict the relationships between evidentiary items. The algorithm integrates multiple evidence graphs, each potentially derived from different systems involved in a network attack, into a single, unified graph. This integrated evidence graph offers a broader context and better understandability of the attack's global scope compared to individual local graphs. The methodology incorporates quantitative metrics from sources like the NIST National Vulnerability Database to estimate the success probabilities of attacks and the reliability of evidence. By merging the graphs, the paper aims to address the limitations of previous approaches that could not effectively combine evidence from distributed attacks, enhancing the accuracy and utility of forensic analysis.

4.2 Forensics in SDN

Software-Defined Networking (SDN) is a major improvement for telecom companies. It separates how data flows (data plane) from how it's controlled (control plane). This allows for more flexible and automated network management. However, this very feature can be a double-edged sword.

Malicious actors can exploit the centralized control of SDN to launch more widespread and damaging attacks.

When such an attack occurs, both legal and network security teams need to figure out what happened (root cause). Both sides rely on information collected after the incident (forensics and incident response). However, these processes are currently separate due to challenges in integrating them with SDN environments. This separation wastes time and resources during investigations.

M. B. Jimenez et al.(2023)[9] presented an in-depth framework for integrating forensic analysis and incident response processes specifically tailored for Software Defined Networks (SDN). SDN architectures decouple the network control plane from the data plane, enabling centralized management and dynamic network configuration, which presents unique challenges and opportunities for cybersecurity forensics. The framework aims to address these challenges by ensuring evidence security, optimizing network performance during forensic processes, and ensuring synchronized logging across multiple SDN controllers.

One of the primary challenges in SDN forensics is maintaining the security and integrity of evidence. The framework employs distributed ledger technology (DLT) to create a secure, immutable record of the evidence chain of custody. This ensures that any collected forensic data remains tamper-proof and verifiable, which is crucial for maintaining the integrity of the investigation and for legal proceedings. The use of DLT also facilitates decentralized validation, enhancing the trustworthiness of the forensic data across different administrative domains.

To optimize network performance during forensic data collection, the framework introduces trigger mechanisms that initiate data collection processes only when specific conditions are met. This approach helps to avoid unnecessary data collection, reducing the load on the network and storage systems. Additionally, the framework employs dimensionality reduction techniques to manage the volume of collected data. By focusing on the most relevant data points and eliminating redundant information, these techniques help in efficiently processing and storing forensic evidence without overwhelming network resources.

The framework also addresses the challenge of log synchronization across multiple SDN controllers. In a distributed SDN environment, ensuring that logs from different controllers are synchronized and consistent is critical for accurate forensic analysis. The framework proposes the use of synchronized timestamps and standardized logging formats to achieve this consistency. By ensuring that all logs are aligned and uniformly formatted, investigators can more easily correlate events and identify the sequence of actions that led to a security incident.

Furthermore, the framework emphasizes the importance of forensic readiness and rapid incident response. Forensic readiness involves preparing the network environment to support efficient forensic investigations before any incidents occur. Rapid incident response mechanisms are also integrated into the framework to detect and contain threats as quickly as possible. By automating certain response actions and enabling real-time monitoring, the framework helps to mitigate the impact of security incidents and preserve critical evidence for subsequent analysis.

Overall, the framework provides a robust solution for addressing the complexities of forensic analysis and incident response in SDN environments. By leveraging advanced technologies such as distributed ledger technology, trigger mechanisms, and dimensionality reduction, and by focusing on both forensic readiness and incident response, the framework enhances the security and reliability of SDN networks. This comprehensive approach ensures that SDN environments are well-equipped to handle sophisticated cyber threats and can effectively support forensic investigations and incident response activities. [9]

Chapter 5

Analytics Based on Attack Hypergraph

The sole objective of generating an attack hypergraph is to enable assessment of security. There are many ways in which information encoded in an attack hypergraph can be used to gain vital insight into the global security posture of a network. This helps security administrators make the correct decisions about mitigation strategies. In the actual network, the security administrator cannot patch all the nodes in the network. That's why we need to find the best way possible to patch the whole system so that we can reduce the majority of the vulnerability in the network.

Measuring security measures through the lens of attack hypergraph entropy provides a comprehensive framework for evaluating system resilience and vulnerability. In this paradigm, each node within the hypergraph represents a distinct aspect of security infrastructure, ranging from individual user accounts and access points to critical system resources and interconnected services. The hyperedges between nodes symbolize relationships or dependencies, reflecting the intricate network of interactions within the security landscape. When analyzing attack hypergraph entropy, higher values signify a greater degree of randomness and complexity in the network topology. This complexity often arises from multiple nodes sharing overlapping functionalities or permissions, leading to a convoluted and less predictable security architecture. Consequently, systems characterized by higher entropy face increased risks of exploitation and compromise, as the presence of numerous overlapping nodes creates potential avenues for attackers to infiltrate or bypass security measures.

Lower entropy levels, on the other hand, are indicative of a more ordered and structured security environment, where the hypergraph shows significant separations between security components. In such scenarios, the attack hypergraph topology is characterized by fewer nodes overlapping in functionalities, resulting in a streamlined and well-defined security architecture. This reduction in complexity simplifies security management processes, facilitating the identification and mitigation of vulnerabilities. Moreover, lower entropy levels enable more efficient patching procedures, as security updates can be applied with greater precision to specific nodes or segments of the hypergraph. As a result, systems with lower entropy levels are inherently more resilient to cyber threats, as they boast a higher degree of control and predictability over security configurations.

By leveraging attack hypergraph entropy as a quantitative metric for assessing security measures, analysts can gain deeper insights into the underlying structure and dynamics of complex security networks. This approach empowers organizations to proactively identify potential weaknesses and

prioritize remediation efforts, thereby bolstering their overall cybersecurity posture. Furthermore, by the constant monitoring and analysis of hypergraph entropy throughout time, businesses may ensure the continued protection of sensitive information and important assets by adapting to developing threats and emerging attack vectors. In essence, attack hypergraph entropy serves as a powerful tool for enhancing situational awareness and informing strategic decision-making in the realm of cybersecurity.

5.1 Experiment 1

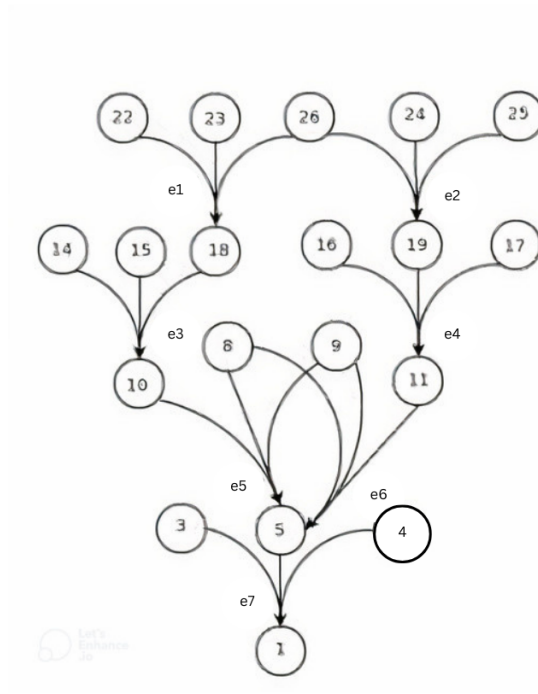


Figure 5.1: Example of Attack Hypergraph

Let's consider the attack hypergraph 5.1 and find the entropy of the hypergraph using the proposed methodology in this paper.

Let $\mathcal{H} = (V, E)$ be the hypergraph, then $V = \{n_1, n_3, \dots, n_{26}\}$ and $E = \{e_1, e_2, \dots, e_7\}$. Adjacency list of this Attack hypergraph is -

Adjacency List of Hypergraph 5.1		
Hyperedge	Head Set	Tail Set
e_1	$\{n_{18}\}$	$\{n_{22}, n_{23}, n_{26}\}$
e_2	$\{n_{19}\}$	$\{n_{24}, n_{25}, n_{26}\}$
e_3	$\{n_{10}\}$	$\{n_{14}, n_{15}, n_{18}\}$
e_4	$\{n_{11}\}$	$\{n_{16}, n_{19}, n_{17}\}$
e_5	$\{n_5\}$	$\{n_{10}, n_8, n_9\}$
e_6	$\{n_5\}$	$\{n_8, n_9, n_{11}\}$
e_7	$\{n_1\}$	$\{n_3, n_5, n_4\}$

Now, there are three kinds of s-overlaps that can be possible in a directed hypergraph -
Tail-Tail s-Overlap :

$$M_{TT} = \begin{array}{c|ccccccc} & e_1 & e_2 & e_3 & e_4 & e_5 & e_6 & e_7 \\ \hline e_1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ e_2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ e_3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ e_4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ e_5 & 0 & 0 & 0 & 0 & 0 & 2 & 0 \\ e_6 & 0 & 0 & 0 & 0 & 2 & 0 & 0 \\ e_7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array} \quad (5.1)$$

Head-Tail s-Overlap :

$$M_{HT} = \begin{array}{c|ccccccc} & e_1 & e_2 & e_3 & e_4 & e_5 & e_6 & e_7 \\ \hline e_1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ e_2 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ e_3 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ e_4 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ e_5 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ e_6 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ e_7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array} \quad (5.2)$$

Head-Head s-Overlap :

$$M_{HH} = \begin{array}{c|ccccccc} & e_1 & e_2 & e_3 & e_4 & e_5 & e_6 & e_7 \\ \hline e_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ e_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ e_3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ e_4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ e_5 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ e_6 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ e_7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array} \quad (5.3)$$

now diagonal matrices D of these s-overlap matrices will be, respectively,

$$D_{TT} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, D_{HT} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix},$$

$$D_{HH} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Now, Laplacian matrices will be for different s-overlaps, respectively.

$$L_{TT}(\mathcal{H}) = \begin{bmatrix} 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & -2 & 0 \\ 0 & 0 & 0 & 0 & -2 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, L_{HT}(\mathcal{H}) = \begin{bmatrix} 1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix},$$

$$L_{HH}(\mathcal{H}) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

And the normalized Laplacian matrices will be, respectively,

$$\bar{L}_{TT}(\mathcal{H}) = \begin{bmatrix} \frac{1}{6} & \frac{-1}{6} & 0 & 0 & 0 & 0 & 0 \\ \frac{-1}{6} & \frac{1}{6} & 0 & 0 & 0 & 0 & 0 \\ \frac{0}{6} & \frac{0}{6} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{2}{6} & \frac{-2}{6} & 0 \\ 0 & 0 & 0 & 0 & \frac{-2}{6} & \frac{2}{6} & 0 \\ 0 & 0 & 0 & 0 & \frac{0}{6} & \frac{0}{6} & 0 \end{bmatrix}, \bar{L}_{HH}(\mathcal{H}) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{-1}{2} & 0 \\ 0 & 0 & 0 & 0 & \frac{-1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 0 & \frac{0}{2} & \frac{0}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\bar{L}_{HT}(\mathcal{H}) = \begin{bmatrix} \frac{1}{6} & 0 & \frac{-1}{6} & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{6} & 0 & \frac{-1}{6} & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{6} & 0 & \frac{-1}{6} & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{6} & 0 & \frac{-1}{6} & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{6} & 0 & \frac{-1}{6} \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{6} & \frac{-1}{6} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Now let us compute the eigenvalues for, $\bar{L}_{TT}(\mathcal{H})$ matrix,

Consider the matrix square matrix A as the $\bar{L}_{TT}(\mathcal{H})$, and the non-zero vector X is called an eigenvector, corresponding to the eigenvalue ' λ '. here we are considering the homogeneous system as follows,

$$\begin{aligned} AX &= \lambda X \\ AX - \lambda X &= 0 \\ (A - \lambda I)X &= 0 \end{aligned}$$

where X is a non-zero eigenvector and $(A - \lambda I)$ is eigenvalue corresponding to eigenvector X. Now,

$$\begin{aligned} &|A - \lambda I| = 0 \\ &\begin{vmatrix} \frac{1}{6} - \lambda & \frac{-1}{6} & 0 & 0 & 0 & 0 & 0 \\ \frac{-1}{6} & \frac{1}{6} - \lambda & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -\lambda & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -\lambda & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{2}{6} - \lambda & \frac{-2}{6} & 0 \\ 0 & 0 & 0 & 0 & \frac{-2}{6} & \frac{2}{6} - \lambda & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -\lambda \end{vmatrix} = 0 \end{aligned}$$

now expanding along column 3, column 4, column 7,

$$\begin{aligned}
(-\lambda)(-\lambda)(-\lambda) & \begin{vmatrix} \frac{1}{6} - \lambda & \frac{-1}{6} & 0 & 0 \\ \frac{-1}{6} & \frac{1}{6} - \lambda & 0 & 0 \\ 0 & 0 & \frac{2}{6} - \lambda & \frac{-2}{6} \\ 0 & 0 & \frac{-2}{6} & \frac{2}{6} - \lambda \end{vmatrix} = 0 \\
-\lambda^3 & \begin{vmatrix} \frac{1}{6} - \lambda & \frac{-1}{6} & 0 & 0 \\ \frac{-1}{6} & \frac{1}{6} - \lambda & 0 & 0 \\ 0 & 0 & \frac{1}{3} - \lambda & \frac{-1}{3} \\ 0 & 0 & \frac{-1}{3} & \frac{1}{3} - \lambda \end{vmatrix} = 0
\end{aligned}$$

Now subtract column 2 multiplied by $6\lambda - 1$ from column 1: $C_1 = C_1 - (6\lambda - 1)C_2$,

$$-\lambda^3 \begin{vmatrix} 0 & \frac{-1}{6} & 0 & 0 \\ 2\lambda(3\lambda - 1) & \frac{1}{6} - \lambda & 0 & 0 \\ 0 & 0 & \frac{1}{3} - \lambda & \frac{-1}{3} \\ 0 & 0 & \frac{-1}{3} & \frac{1}{3} - \lambda \end{vmatrix} = 0$$

now expand along row 1,

$$-\lambda^3 \left(-\frac{1}{6}\right) \begin{vmatrix} 2\lambda(3\lambda - 1) & 0 & 0 \\ 0 & \frac{1}{3} - \lambda & \frac{-1}{3} \\ 0 & \frac{-1}{3} & \frac{1}{3} - \lambda \end{vmatrix} = 0$$

Now expand the matrix by using Sarrus matrix determinant,

$$\begin{aligned}
\left(\frac{1}{6}\lambda^3\right)[2\lambda(3\lambda - 1)\left(\frac{1 - 3\lambda}{3}\right)^2 + 0 + 0 - 0 - \frac{2}{9}\lambda(3\lambda - 1) - 0] &= 0 \\
\left(\frac{1}{6}\lambda^3\right)2\lambda(3\lambda - 1)\left[\frac{1 - 6\lambda + 9\lambda^2}{9} - \frac{1}{9}\right] &= 0 \\
\left(\frac{1}{6}\lambda^3\right)2\lambda(3\lambda - 1)\left[\frac{1 - 6\lambda + 9\lambda^2 - 1}{9}\right] &= 0 \\
\left(\frac{1}{6}\lambda^3\right)2\lambda(3\lambda - 1)\left[\frac{-2\lambda + 3\lambda^2}{3}\right] &= 0 \\
\left(\frac{1}{6}\lambda^3\right)2\lambda(3\lambda - 1)\frac{\lambda}{3}(3\lambda - 2) &= 0 \\
\lambda^5(3\lambda - 1)(3\lambda - 2) &= 0
\end{aligned}$$

Now the eigenvalues will be,

$$\begin{aligned} 3\lambda - 1 &= 0 \\ \lambda &= \frac{1}{3} \end{aligned}$$

So, $\lambda_1 = \frac{1}{3} \approx 0.3333$

$$\begin{aligned} 3\lambda - 2 &= 0 \\ \lambda &= \frac{2}{3} \end{aligned}$$

So, $\lambda_2 = \frac{2}{3} \approx 0.6667$, and for all other eigenvalues will be 0, that is, $\lambda_3 = \lambda_4 = \lambda_5 = \lambda_6 = \lambda_7 = 0$

Similarly, we can calculate eigenvalues of the other normalized Laplacian matrices $\bar{L}_{HH}(\mathcal{H})$ and $\bar{L}_{TT}(\mathcal{H})$. So all eigenvalues that correspond to normalized Laplacian matrices are,

$$\Lambda = \begin{array}{c|ccccccc} & \lambda_1 & \lambda_2 & \lambda_3 & \lambda_4 & \lambda_5 & \lambda_6 & \lambda_7 \\ \hline \bar{L}_{HH}(\mathcal{H}) & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ \bar{L}_{HT}(\mathcal{H}) & 0.1667 & 0.1667 & 0.1667 & 0.1667 & 0.1667 & 0.1667 & 0 \\ \bar{L}_{TT}(\mathcal{H}) & 0.3333 & 0.6667 & 0 & 0 & 0 & 0 & 0 \end{array} \quad (5.4)$$

then the entropy of the directed hypergraph $\mathcal{H}$ will be,

$$V(\mathcal{H}) = \sum_{\lambda \in \Lambda} -\lambda \log_2(\lambda) \quad (5.5)$$

where Λ is the spectrum of $\bar{L}(\mathcal{H})$; that is, the multiset of its eigenvalues.

$$\begin{array}{c|c} & V(\mathcal{H}) \\ \hline V_{HH}(\mathcal{H}) & 0 \\ V_{HT}(\mathcal{H}) & 1.7917594692280547 \\ V_{TT}(\mathcal{H}) & 0.6365141682948128 \end{array} \quad (5.6)$$

By analyzing the results, we can conclude that for **Head-Tail s-overlap**, entropy is the highest. That indicates a higher degree of randomness and complexity in the network topology. This complexity is frequently caused by several nodes sharing rights or functionality that overlap, which results in a complicated and unpredictable security architecture. Higher entropy systems are therefore more vulnerable to exploitation and compromise because several overlapping nodes provide potential pathways for attackers to penetrate or get around security safeguards.

That's why we need to reduce the entropy of the attack hypergraph by **patching** some hyperedges. but it is not feasible to patch all the attack hypergraph. So, we need to find only a few hyperedges that can make the most difference in reducing the entropy of the hypergraph. And network administration needs to only patch those hyperedges or exploits.

Previously, we introduced the concept of **hyperedge centrality (HDC)**, which evaluates the significance of nodes based on the number of hyperedges they are part of. A node's importance increases with the number of hyperedges it is connected to. Now, If we use the concept of HDC, we will get results like this,

$Node(v_i)$	$HDC(v_i)$
n_5	3
n_{10}	2
n_{11}	2
n_8	2
n_9	2
n_{19}	2
n_{18}	2
n_{26}	2
n_{22}	1
n_{23}	1
n_3	1
n_{17}	1
n_{16}	1
n_{15}	1
n_{14}	1
n_1	1
n_{25}	1
n_4	1
n_{24}	1

Here, the result suggests n_5 is the most significant node in the directed hypergraph. By comparing the hypergraph figure 5.1, we can see that in n_5 , two edges (e_5, e_6) are coming to node n_5 and (e_7) is going from n_5 . so n_5 is participating in 3 directed hyperedges. And that's why it is the most significant node in the attack directed hypergraph.

Now the final part is to patch edges belonging to the exploited hyperedge. But in this case, we also need a efficient way to patch only particular nodes that can minimize the overall entropy of the hypergraph. Now let's patch the hyperedges one by one and analyze the entropy of the hypergraph 5.1:

Patch Hyperedge(E)	Max $Entropy(V_{max}(\mathcal{H}))$	Overlap
None	1.7917	Head-Tail
$\{e_6\}$	1.3862	Head-Tail
$\{e_5\}$	1.3862	Head-Tail
$\{e_7\}$	1.3862	Head-Tail
$\{e_5, e_6\}$	0.6931	Head-Tail
$\{e_6, e_7\}$	1.3862	Head-Tail
$\{e_5, e_7\}$	1.3862	Head-Tail
$\{e_5, e_6, e_7\}$	0.6931	Head-Tail

Here, we can see that if we patch $\{e_5, e_6\}$ or $\{e_5, e_6, e_7\}$ directed hyperedges, the overall hypergraph entropy will be mostly reduced to 0.6931 from 1.7917. But it will be most efficient to patch $\{e_5, e_6\}$ hyperedges rather than $\{e_5, e_6, e_7\}$ hyperedges, as the reason is clear that it would be efficient to patch 2 exploits rather than 3 exploits.

Now, if we closely analyze the final results, we can see the *fanIn* hyperedges of node n_5 are 2, which are $\{e_5, e_6\}$, whereas the *fanOut* hyperedge of node n_5 is 1, which is $\{e_7\}$ only. That's why it is efficient to patch *fanIn* nodes in this case, as there are more *fanIn* hyperedges than *fanOut* hyperedges.

5.2 Experiment 2

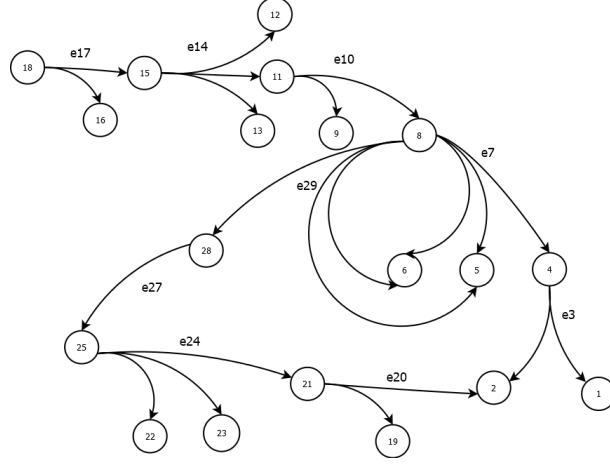


Figure 5.2: Attack Hypergraph (adapted from [3])

Let's take another example of the attack hypergraph 5.2 adapted from [3] and find the entropy of the hypergraph using the proposed methodology in this paper.

Let $\mathcal{H} = (V, E)$ be the hypergraph, then $V = \{n_1, n_3, \dots, n_{28}\}$ and $E = \{e_3, e_7, \dots, e_{29}\}$. The adjacency list of this Attack hypergraph is:

Adjacency List of Hypergraph 5.2		
Hyperedge	Head Set	Tail Set
e_{17}	$\{n_{16}, n_{15}\}$	$\{n_{18}\}$
e_{14}	$\{n_{12}, n_{13}, n_{11}\}$	$\{n_{15}\}$
e_{10}	$\{n_8, n_9\}$	$\{n_{11}\}$
e_7	$\{n_6, n_5, n_4\}$	$\{n_8\}$
e_{29}	$\{n_{28}, n_6, n_5\}$	$\{n_8\}$
e_{27}	$\{n_{25}\}$	$\{n_{28}\}$
e_3	$\{n_2, n_1\}$	$\{n_4\}$
e_{24}	$\{n_{21}, n_{22}, n_{23}\}$	$\{n_{25}\}$
e_{20}	$\{n_{19}, n_2\}$	$\{n_{21}\}$

Now, there are three kinds of s-overlaps that can be possible in a directed hypergraph -

Tail-Tail s-Overlap :

[illegible]

Head-Tail s-Overlap :

[illegible]

Head-Head s-Overlap :

[illegible]

now diagonal matrices D of these s-overlap matrices will be, respectively,

$$D_{TT} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, D_{HT} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix},$$

$$\bar{L}_{HT}(\mathcal{H}) = \begin{bmatrix} \frac{1}{8} & \frac{-1}{8} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{8} & \frac{-1}{8} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{2}{8} & \frac{-1}{8} & \frac{-1}{8} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{8} & 0 & 0 & \frac{-1}{8} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{8} & \frac{-1}{8} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{8} & 0 & \frac{-1}{8} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{8} & \frac{-1}{8} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Now the eigenvalues will be,

	λ_1	λ_2	λ_3	λ_4	λ_5	λ_6	λ_7	λ_8	λ_9
$L_{HH}(\mathcal{H})$	0.6667	0	0.3333	0	0	0	0	0	0
$\bar{L}_{HT}(\mathcal{H})$	0.125	0.125	0.25	0.125	0.125	0.125	0	0.125	0
$\bar{L}_{TT}(\mathcal{H})$	1	0	0	0	0	0	0	0	0

(5.10)

then the entropy of the directed hypergraph $\mathcal{H}$ will be,

$$V(\mathcal{H}) = \sum_{\lambda \in \Lambda} -\lambda \log_2(\lambda) \quad (5.11)$$

where Λ is the spectrum of $\bar{L}(\mathcal{H})$; that is, the multiset of its eigenvalues.

	$V(\mathcal{H})$
$V_{HH}(\mathcal{H})$	0.6365141682948128
$V_{HT}(\mathcal{H})$	1.9061547465398496
$V_{TT}(\mathcal{H})$	0

(5.12)

By analyzing the results, we conclude that the entropy is highest for **Head-Tail s-overlap**. This indicates a higher degree of randomness and complexity in the network topology, often caused by multiple nodes sharing overlapping rights or functionalities, leading to a complex and unpredictable security architecture. Consequently, higher entropy systems are more susceptible to exploitation and compromise because overlapping nodes create multiple potential pathways for attackers to bypass security measures.

To mitigate this, we should reduce the entropy of the attack hypergraph by **patching** certain hyperedges. However, patching the entire attack hypergraph is not feasible. Therefore, we need to identify a select few hyperedges whose patching would most effectively reduce the hypergraph's entropy. Network administrators should focus on patching these critical hyperedges or exploits.

Previously, we introduced the concept of **hyperedge centrality (HDC)**, which assesses the significance of nodes based on the number of hyperedges they belong to. The importance of a node increases with the number of hyperedges it is connected to. Using the HDC concept, we can achieve results like this:

$Node(v_i)$	$HDC(v_i)$
n_8	3
n_4	2
n_{11}	2
n_2	2
n_{25}	2
n_{15}	2
n_{21}	2
n_6	2
n_{28}	2
n_5	2
n_{13}	1
n_1	1
n_{16}	1
n_{18}	1
n_9	1
n_{23}	1
n_{22}	1
n_{19}	1
n_{12}	1

The results indicate that n_8 is the most significant node in the directed hypergraph. Referring to the hypergraph in Figure 5.2, we observe that e_{10} points to n_8 , and two edges, e_7 and e_{29} , originate from n_8 . Thus, n_8 is involved in three directed hyperedges, making it the most critical node in the attack-directed hypergraph.

The next step involves patching the edges associated with the exploited hyperedges. We need an efficient method to patch only specific nodes to minimize the overall entropy of the hypergraph. Let's proceed by patching the hyperedges one at a time and analyzing the resulting entropy of the hypergraph as shown in Figure 5.2:

Patch Hyperedge(E)	Max $Entropy(V_{max}(\mathcal{H}))$	Overlap
None	1.906	Head-Tail
$\{e_{10}\}$	1.609	Head-Tail
$\{e_7\}$	1.791	Head-Tail
$\{e_{29}\}$	1.791	Head-Tail
$\{e_{10}, e_7\}$	1.386	Head-Tail
$\{e_{10}, e_{29}\}$	1.386	Head-Tail
$\{e_7, e_{29}\}$	1.386	Head-Tail
$\{e_7, e_{10}, e_{29}\}$	1.098	Head-Tail

Unlike the previous experiment, here we can see that for different individual hyperedge patches, the entropy reduces differently. So reduction of entropy now not only depends on *fanIn-fanOut* but also depends on the individual hyperedge. As we can see, if we patch $\{e_7, e_{10}, e_{29}\}$, the overall entropy will be mostly reduced to 1.098 from 1.906. So we need to patch 3 exploits in this case to reduce to the minimal entropy.

Now, if we closely analyze the final results, we can see that the *fanIn* hyperedge of node n_8 is 1, which is $\{e_{10}\}$, whereas the *fanOut* hyperedges of node n_8 are 2, which are $\{e_7, e_{29}\}$. So if we patch

the only *fanOut* hyperedges like in the previous experiment, it should give the minimum entropy. but that's not what the results are indicating here. As we can see, if we patch $\{e_{10}, e_7\}$ or $\{e_{10}, e_{29}\}$ or $\{e_{29}, e_7\}$, the entropy reduces to a similar value, but that's not what the optimal results are here. because here the whole entropy is depending on both *fanIn-fanOut* and the contribution of the predecessor hyperedge of node n_8 , that is $\{e_{10}\}$. As we can see, predecessor hyperedges have more precedence here than successor hyperedges of node n_8 in reduction of entropy. That's why we need to patch not only *fanOut* exploits, but also *fanIn* exploits.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

This project has explored the application of hypergraph-based entropy measures as a novel metric for cybersecurity analysis. The research demonstrates that hypergraphs, with their ability to represent higher-order relationships, offer a more nuanced and comprehensive framework for analyzing complex network interactions compared to traditional graph models.

The significant contributions of this work include the development of a traversal algorithm for directed attack hypergraphs, the formulation of entropy measures accounting for different types of overlaps, and the implementation of a methodology to identify and patch the most vulnerable nodes in a network. These advancements provide a systematic approach for enhancing the robustness of networks against sophisticated cyber threats.

Through extensive experiments, the project validates the effectiveness of hypergraph-based entropy measures in quantifying the uncertainty and complexity within a network. The results indicate that strategic patching, informed by hypergraph entropy analysis, can significantly reduce network vulnerability. This research thus offers valuable insights and practical tools for cybersecurity practitioners and researchers, advancing the field by introducing a robust, mathematically grounded metric for security analysis.

The implications of this work extend beyond immediate applications, paving the way for future research in the integration of hypergraph theory with other advanced mathematical models to further enhance cybersecurity measures. This project sets a foundation for developing more sophisticated and resilient defense mechanisms, crucial in an era where cyber threats are increasingly complex and pervasive .

6.2 Future Work

Here are some potential future directions for the research presented in the attached paper, focusing on entropy in directed attack hypergraphs:

6.2.1 Precision of Entropy Calculation

- **Improvement in Entropy Measurement:** Enhance the methods for calculating entropy in directed attack hypergraphs to provide more precise and accurate results. This could involve developing new algorithms or refining existing ones to better capture the complexity and nuances of these structures.
- **High-Resolution Entropy Metrics:** Implement high-resolution entropy metrics that can more effectively differentiate between various levels of network complexity and vulnerability.

6.2.2 Practical Applications in Network Security

- **Integration with Security Systems:** Apply the entropy metrics in real-world network security systems to dynamically monitor and manage network health. This could involve developing software tools that use entropy as a key metric for identifying vulnerabilities.
- **Continuous Monitoring and Adaptation:** Establish protocols for the continuous monitoring of network entropy to adapt to evolving security threats. This would enable proactive rather than reactive security measures.

6.2.3 Optimization of Hyperedge Patching

- **Strategic Hyperedge Patching:** Focus on identifying the most critical hyperedges whose patching would significantly reduce network entropy. Develop optimization models that can prioritize hyperedges based on their impact on overall network security.
- **Automated Patching Systems:** Create automated systems that can apply patches to selected hyperedges efficiently, reducing human intervention and the potential for error.

6.2.4 Enhancing Analytical Frameworks

- **Advanced Analytical Techniques:** Incorporate advanced data analytics and machine learning techniques to enhance the interpretation of entropy metrics. This can help in identifying patterns and anomalies that are not immediately obvious.
- **Visualization Tools:** Develop sophisticated visualization tools that can represent entropy changes and network vulnerabilities graphically, making it easier for network administrators to understand and act upon the data.

By pursuing these directions, the research can significantly contribute to the field of network security, providing more effective tools and methodologies for maintaining robust and resilient networks.

Bibliography

- [1] Mulval: Multi-host, multi-stage vulnerability analysis tool. <https://github.com/risksense/mulval>. Accessed: 2023-09-10.
- [2] J. C. Acosta, E. Padilla, and J. Homer. Augmenting attack graphs to represent data link and network layer vulnerabilities. In *MILCOM 2016 - 2016 IEEE Military Communications Conference*, pages 1010–1015, 2016.
- [3] M. Barik, A. Sengupta, and C. Mazumdar. Attack graph generation and analysis techniques. *Defence Science Journal*, 66(6):559–567, Oct. 2016.
- [4] I. Bloch and A. Bretto. A new entropy for hypergraphs. In *Discrete Geometry for Computer Imagery: 21st IAPR International Conference, DGCI 2019, Marne-La-Vallée, France, March 26–28, 2019, Proceedings*, page 143–154, Berlin, Heidelberg, 2019. Springer-Verlag.
- [5] M. Comuzzi. Optimal directed hypergraph traversal with ant-colony optimisation. *Information Sciences*, 471:132–148, 2019.
- [6] J. He, C. Chang, P. He, and M. S. Pathan. Network forensics method based on evidence graph and vulnerability reasoning. *Future Internet*, 8(4), 2016.
- [7] F. Hu, K. Tian, and Z.-K. Zhang. Identifying vital nodes in hypergraphs based on von neumann entropy. *Entropy*, 25(9), 2023.
- [8] A. R. Javed, W. Ahmed, M. Alazab, Z. Jalil, K. Kifayat, and T. R. Gadekallu. A comprehensive survey on computer forensics: State-of-the-art, tools, techniques, challenges, and future directions. *IEEE Access*, 10:11065–11089, 2022.
- [9] M. B. Jimenez and D. Fernandez. A framework for sdn forensic readiness and cybersecurity incident response. In *2022 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pages 112–116, 2022.
- [10] L. Leichtnam, E. Totel, N. Prigent, and L. Mé. Forensic analysis of network attacks: Restructuring security events as graphs and identifying strongly connected sub-graphs. In *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 565–573, 2020.
- [11] C. Liu, A. Singhal, and D. Wijesekera. Mapping evidence graphs to attack graphs. pages 121–126, 12 2012.

- [12] C. Liu, A. Singhal, and D. Wijesekera. Creating integrated evidence graphs for network forensics. In G. Peterson and S. Shenoi, editors, *Advances in Digital Forensics IX*, pages 227–241, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.
- [13] A. Nisioti, G. Loukas, A. Mylonas, and E. Panaousis. Forensics for multi-stage cyber incidents: Survey and future directions. *Forensic Science International: Digital Investigation*, 44:301480, 2023.
- [14] E. Raftopoulos and X. Dimitropoulos. Understanding network forensics analysis in an operational environment. In *2013 IEEE Security and Privacy Workshops*, pages 111–118, 2013.
- [15] L. Sadineni, E. S. Pilli, and R. B. Battula. Provnet-iot: Provenance based network layer forensics in internet of things. *Forensic Science International: Digital Investigation*, 43:301441, 2022.
- [16] O. Stan, R. Bitton, M. Ezrets, M. Dadon, M. Inokuchi, Y. Ohta, T. Yagyu, Y. Elovici, and A. Shabtai. Extending attack graphs to represent cyber-attacks in communication protocols and modern it networks. *IEEE Transactions on Dependable and Secure Computing*, 19(3):1936–1954, 2022.