2023 IEEE International Conference on Cryptography, Informatics, and Cybersecurity (ICoCICs) | 979-8-3503-3943-7/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICoCICs58778.2023.10276627

Knowledge Graphs for Cybersecurity: A Framework for Honeypot Data Analysis

1st Yevonnael Andrew
Information Technology Department
Swiss German University
Tangerang, Banten, Indonesia
yevonnael.andrew@sgu.ac.id

2nd Charles Lim
Information Technology Department
Swiss German University
Tangerang, Banten, Indonesia
charles.lim@sgu.ac.id

3rd Eka Budiarto
Information Technology Department
Swiss German University
Tangerang, Banten, Indonesia
eka.budiarto@sgu.ac.id

***Abstract*—In the field of cybersecurity, the complexity and diversity of data present significant challenges for effective analysis. This paper explores the use of knowledge graphs as a tool to enhance the analysis of honeypot data. We detail the entire process, from data collection and pre-processing to the creation of the knowledge graph and its subsequent analysis. Our approach enables complex query analysis and provides insights into the sequence and patterns of attacker commands in a specific session, as well as a summary of activities originating from a specific IP address. However, the adaptability of the transformation process may vary depending on the characteristics of the source documents. Our paper underscores the importance of knowledge graphs in enabling more effective threat detection and response mechanisms through a more comprehensive and deeper analysis of honeypot data. Future research could explore real-time graph updating, pattern recognition with machine learning, threat prediction, and attack attribution.**

***Keywords*—honeypot, cybersecurity, knowledge graph, data analysis**

I. Introduction

In the cybersecurity field, understanding and countering cyber threats is essential. This paper presents an approach that uses honeypots and knowledge graphs to analyze attacker behaviour and link activities to specific IP addresses, offering practical applications of theoretical concepts in real-world scenarios.

A honeypot is a decoy computer resource, the value of which arises from being probed, attacked, or compromised [1]. Several popular honeypots are available, such as Cowrie [2], a medium-to-high interaction SSH/Telnet honeypot, and Dionaea [3], which supports several services such as HTTP, SMB, SIP, and MQTT, among others.

Deploying these honeypots generates substantial amounts of data. The complexity and volume of this data present significant challenges for manual analysis, increasing the risk of overlooking crucial information such as specific attack patterns or significant relationships between different data points.

To address these challenges, the graph database, such as Neo4j [4], offers an appealing solution. Graph databases enable data to be represented in an interconnected structure, highlighting relationships between various entities, and thus enabling more intuitive data analysis.

Building upon our prior work, in which we proposed a framework for analyzing and categorizing threats from honeypots [5], developed a more detailed schema for characterizing attacker behaviours [6], and utilized Natural Language Processing (NLP) to link Linux commands in honeypot data with the MITRE ATT&CK framework [7], we now aim to explore the use of knowledge graphs as a new avenue for advanced analysis and threat detection.

The objective of this paper is to explore the use of knowledge graphs to enhance our understanding and analysis of honeypot data. By leveraging knowledge graphs, we aim to enable better identification and understanding of relationships between various entities. By transforming raw honeypot data into structured, interconnected knowledge, we aim to provide a more comprehensive and intuitive platform for cybersecurity analysis.

The use of knowledge graphs in the field of cybersecurity has been a topic of interest in recent years. Knowledge graphs provide a structured and semantic approach to representing data, which can be particularly useful in the complex and dynamic field of cybersecurity. A recent study by Liu et al. [8] provides an overview of the core concepts, schemas, and construction methodologies of cybersecurity knowledge graphs. They highlight the need for more research on the practical application of these graphs in real-world cyberattacks and defence scenarios.

Building on the observation by Liu et al. [8] that the practical application of knowledge graphs in real-world cyberattack and defence situations remains largely unexplored, our paper seeks to fill this gap. This paper contributes to cybersecurity data analysis by introducing a practical methodology that applies knowledge graphs to honeypot data. Therefore, our approach enables detailed insights into attacker behaviour and activities linked to specific IP addresses, thus providing a practical application of theoretical concepts. This also paves the way for future investigations into real-time graph updating, machine learning-based pattern recognition, and threat prediction.

The remainder of the paper is organized as follows: Section II discusses related research on graph implementation in cybersecurity. Section III describes the process of building a knowledge graph from honeypot data, including data collection, pre-processing, knowledge graph creation, linking with external data, and analysis. Section IV provides examples of analysis techniques to derive insights from the constructed knowledge graphs.

II. Related Works

In this section, we discuss related works in the usage of knowledge graphs in cybersecurity and our choice of graph database.

For instance, Narayanan et al. [9] leveraged semantically rich knowledge representation and reasoning integrated with machine learning for a cognitive cybersecurity system. Their system ingests information from various textual sources, which is stored in a knowledge graph. This enables actionable intelligence for security administrators.

Additionally, Wu et al. [10] proposed an ontology and graph-based approach for network security assessment. They designed an ontology to represent security knowledge in a common form, resulting in an efficient system framework for attack graphs and security assessment.

These works underscore the effectiveness of knowledge graphs in cybersecurity, reinforcing our decision to explore knowledge graphs in our study.

A. Knowledge Graph Construction

Knowledge graphs are typically domain-specific. Thus, there is no single schema that can solve every problem. According to [11], there are four main types of graphs:

- Type I: Network Infrastructure and Cyberattack Graphs
- Type II: Cyberthreat Intelligence
- Type III: Controlled Vocabulary or Ontology
- Type IV: Multi-Graph Data Amalgamation

The most suitable classification for a graph-based knowledge base representing honeypot data is Type II: Cyberthreat Intelligence. As a decoy system, honeypots provide information on the behaviour of attackers, the attributes of malware, and other threat-associated details. This approach can provide insights into emerging threats, attack patterns, and malicious activities observed within the honeypot environment by security researchers and analysts.

B. Knowledge Graphs and Reasoning

A knowledge graph is a structured representation of information consisting of nodes and edges representing entities and their relationships. In the context of cybersecurity, the structured representation of a knowledge graph can be demonstrated using a simple scenario. Consider four key entities: an IP address (1.2.3.4), a malware variant (Malware A), a domain name (malware.com), and a threat actor (Threat Actor X).

The relationships in this scenario could be represented as follows:

- IP address 1.2.3.4 is associated with Malware A.
- Threat Actor X uses Malware A.
- The domain name malware.com is linked to IP address 1.1.1.1.

This structured representation of information allows complex querying and pattern recognition, essential in understanding cybersecurity threats and enhancing the capability of threat detection and response. The transformation of raw honeypot data into such a graph is outlined in the following subsections and sections.

Knowledge graphs offer several advantages over traditional databases in the cybersecurity field. One of the key benefits is the ability to integrate and link critical information on real-world vulnerabilities, weaknesses, and attack patterns from various publicly available sources, providing a comprehensive view of the cybersecurity landscape [12], [13], [14]. Table I shows the various types of data used to build the knowledge graph.

TABLE I. TYPES OF DATA

Data Type	Characteristics	References
Textual Data	Unstructured data sources like text documents, reports, and articles	[12], [15], [16]
Structured Data or Standards	Structured data sources or data derived from well-established standards in the cybersecurity domain	[12], [17]
Machine-Generated Data	Data generated by machines, systems, or processes without human intervention, such as log files or sensor data	[18]

Knowledge graphs also have the capacity to handle the semantics of data, enabling more complex queries and inferences. This is particularly useful in cybersecurity, where understanding the context and relationships between different entities can be crucial for threat detection and prevention [19], [17], [20].

Cybersecurity is a highly dynamic field with new threats emerging constantly. Knowledge graphs can more easily evolve and adapt to these changes compared to traditional databases, making them more suitable for the cybersecurity domain [16], [8], [9].

Furthermore, knowledge graphs allow for the efficient association and deep mining analysis of cybersecurity logs, improving the efficiency of log analysis [21]. Table II shows several methods that can be used to construct the knowledge graph.

Finally, the ability to visualize knowledge graphs can aid in understanding complex relationships and patterns in the data. This can be particularly useful in cybersecurity for identifying potential threats and vulnerabilities [22].

Knowledge reasoning involves deducing new knowledge or identifying incorrect information based on the existing knowledge within a knowledge graph. In cybersecurity, knowledge reasoning can support the identification of novel attack patterns, attributing attacks to specific threat actors or groups, and validating hypotheses about attacker behaviour. Various reasoning techniques can be applied to knowledge graphs to enhance their analytical capabilities [24], such as identifying frequent patterns, community detection, centrality analysis (most significant nodes), anomaly detection (unusual patterns or outliers), temporal analysis (change over time) and also

TABLE II. METHODS FOR KNOWLEDGE GRAPH CONSTRUCTION

Methods	Implementation Details	References
Text Analysis Techniques	Utilizes NLP methods, such as Named Entity Recognition (NER), Part-of-Speech (POS) tagging, and Dependency Parsing	[12], [15], [16], [19]
Machine Learning and Deep Learning	Applies ML/DL algorithms, and neural networks for data processing/analysis. Includes various learning techniques and models like Word2Vec, GloVe, BERT, GPT, etc.	[12], [15], [16], [23]
Knowledge Representation	Utilizes techniques like rule-based, expert systems, ontologies, automated frameworks, and methodologies for constructing knowledge graphs from raw data sources.	[17], [18]

more complex techniques, such as rule-based reasoning (predefined rule), ontological reasoning (leveraging ontologies and semantic relationships between concepts in the graph), probabilistic reasoning (Bayesian, Markov) and machine learning-based reasoning.

C. Graph Database Selection

We have selected Neo4j for our research needs due to its performance and user-friendly features. Neo4j's proprietary language, Cypher, facilitates powerful data analysis, and its speed in loading large volumes of data outperforms other graph databases [25], [26]. Its ability to handle large data volumes with high scalability and low latency, along with a flexible data schema for easy data access, makes it particularly beneficial for analysts [26]. Compared to other graph databases like TigerGraph and OrientDB, Neo4j demonstrates better performance in terms of load time, query execution, and certain operations [26].

III. Methodology

A. General Overview

The methodology used in this study involves several steps, which are illustrated in Figure 1, including data collection and pre-processing, entity extraction, knowledge graph creation, and linking with external data.

B. Data Collection and Pre-processing

The data used for building the knowledge graph originates from the pre-processed logs of our deployed Cowrie and Dionaea honeypots. The pre-processing step, aimed at standardizing the log format, was conducted using a modified version of ewsposter [27]. Our modifications included adding an export functionality to MongoDB and making improvements to the command recording feature in Cowrie.

Once standardized, these logs were then stored in MongoDB [28], a document-oriented NoSQL database that provides a flexible schema for storing diverse datasets. MongoDB was specifically chosen due to its ability to handle high volumes of read and write traffic, as well as its scalability to accommodate large data loads. This makes it ideal for managing the vast and varied data typically amassed by honeypots.

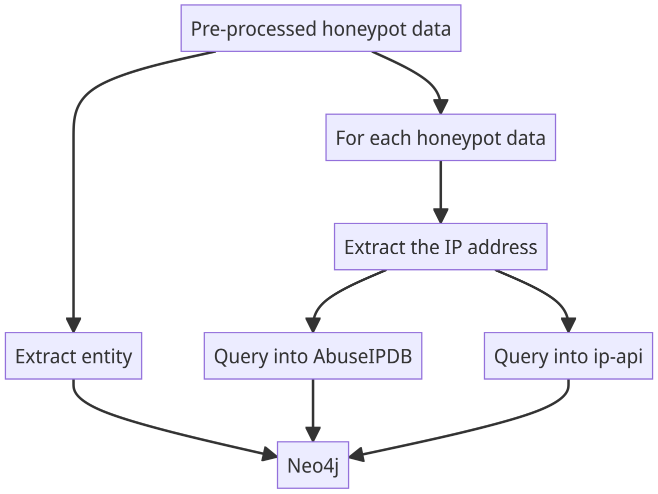

Fig. 1: Methodology flowchart

The datasets used include:

- Pre-processed Cowrie data
- Pre-processed Dionaea data
- Results from AbuseIPDB [29]
- Results from ip-api [30]

C. Entity Extraction and Knowledge Graph Creation

From the collected data, we identify and extract the following information:

- IP addresses of attacker
- Command executed by attackers
- Session information
- Username and password used

These entities are integral to our research as they enable the understanding of attacker profiles, behaviours, and techniques, ultimately aiding in the enhancement of cyber defence strategies and prediction of potential threats.

We utilized Python scripts to process the data from MongoDB into the Neo4j graph database, which involves extracting, transforming, and loading (ETL) processes to generate a knowledge graph that encapsulates myriad relationships within the dataset. These processes are crucial for ensuring the right format and structure for the knowledge graph in our study.

D. Linking with External Data

The extracted IP addresses are used to query AbuseIPDB and ip-api for additional information. The results from these queries are then combined into the Neo4j graph, enriching the existing data with external information.

IV. Results and Evaluation

The task of converting MongoDB documents into a Neo4j graph database has been successfully executed. This transformation generated a knowledge graph encapsulating a myriad of relationships existing within the dataset.

The MongoDB data, composed of diverse types of honeypot records (i.e., 'Dionaea' and 'Cowrie'), presented inherent structural heterogeneity. However, the transformation process managed these variations proficiently, ensuring an accurate graph representation of the underlying data.

In addition to the MongoDB data, external data was fetched from AbuseIPDB and ip-api. These sources provided additional data like geolocation information, network information, and abuse reports, enriching the original dataset and aiding in constructing a more comprehensive graph representation. This additional data was used to populate certain nodes in the graph, such as ‘Address‘, ‘Country‘, ‘AS‘, and ‘UsageType‘.

A sample of data fetched from ip-api is shown in Listing 1.

Listing 1: IP-API Result

```
{
   'status': 'success',
   'country': 'Netherlands',
   'countryCode': 'NL',
   'region': 'DR',
   'regionName': 'Drenthe',
   'city': 'Meppel',
   'zip': '7942',
   'lat': 52.706,
   'lon': 6.1876,
   'timezone': 'Europe/Amsterdam',
   'isp': 'XHOSTIS',
   'org': 'Xhost Internet Solutions',
   'as': 'AS208091 XHOST INTERNET SOLUTIONS LP',
   'query': <IP_ADDRESS>
}
```

Similarly, a sample from AbuseIPDB is shown in Listing 2.

Listing 2: AbuseIPDB Results

```
{
  'ipAddress': <IP_ADDRESS>,
  'isPublic': True,
  'ipVersion': 4,
  'isWhitelisted': False,
  'abuseConfidenceScore': 89,
  'countryCode': 'NL',
  'usageType': 'Data Center/Web Hosting/Transit
      ',
  'isp': 'Alexander Valerevich Mokhonko',
  'domain': 'serverlux.ru',
  'hostnames': [],
  'isTor': False,
  'totalReports': 156,
  'numDistinctUsers': 70,
  'lastReportedAt': '2023-06-25T23:43:04+00:00'
}
```

The information fetched from these sources was matched with the corresponding records in the MongoDB data by IP address and used to enrich the dataset and construct the graph.

Figure 2 shows the constructed Neo4j graph, distinct node types such as ‘Session‘, ‘Address‘, ‘Honeypot‘, ‘Command‘, ‘Country‘, ‘UsageType‘, and ‘AS‘ were created to represent various entities existing within the data. Concurrently, unique relationships namely, ‘ORIGINATED_FROM‘, ‘CONNECTED_TO‘, ‘BELONGS_TO‘, ‘HAS_USAGE_TYPE‘, ‘LOCATED_IN‘, ‘EXECUTED‘, and ‘NEXT‘ were established to interconnect these nodes. This structuring mirrored the interrelationships present within the original data.

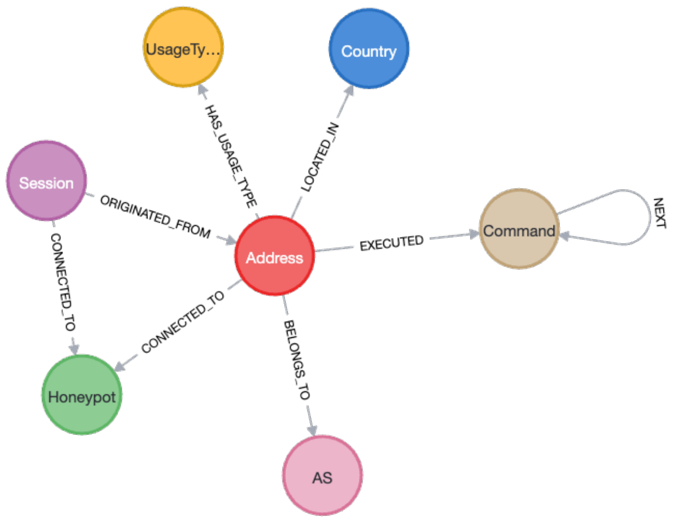

Fig. 2: Graph data schema

Listing 3 illustrates the transformation of a ‘Cowrie‘ document containing an ‘input‘ field. For such ‘Cowrie‘ documents, a linked chain of ‘Command‘ nodes is constructed, each node corresponding to a command in the ‘input‘ list, interconnected through the ‘NEXT‘ relationship. This structuring effectively captures the order of commands, which is essential for understanding the sequence of events during the session.

Listing 3: Cypher query for Cowrie with ‘input‘ field

```
MERGE (s:Session {
   sessionid: sessionid,
   logintime: logintime,
   endtimetime: endtimetime,
   version: version,
   login: login,
   username: username,
   password: password
})
MERGE (a:Address {
   address: source_address,
   port: source_port,
   protocol: source_protocol,
   abuseConfidenceScore: abuse_confidence_score,
   lastReportedAt: last_reported_at
})
MERGE (c:Country {
   name: country
})
MERGE (as:AS {
   name: as_field
})
MERGE (ut:UsageType {
   name: usage_type
})
MERGE (h:Honeypot {
   type: honeypot,
   analyzer_id: analyzer_id
})
MERGE (s)-[r1:ORIGINATED_FROM]->(a)
MERGE (s)-[r2:CONNECTED_TO]->(h)
MERGE (a)-[:LOCATED_IN]->(c)
MERGE (a)-[:BELONGS_TO]->(as)
MERGE (a)-[:HAS_USAGE_TYPE]->(ut)
```

Listing 4 shows the Python code used to construct the linked chain.

Listing 4: Python code to construct the linked chain

```
if idx == 0:
      query += f"""
      WITH a as a
      MATCH (cm:Command {{command: '{
          command_escaped}'}})
      MERGE (a)-[r3:EXECUTED]->(cm)
      """
elif prev_command_node is not None:
      query += f"""
      WITH true as pass
      MATCH (cm:Command {{command: '{
          command_escaped}'}})
      MATCH (p:Command {{command: '{
          prev_command_node}'}})
      MERGE (p)-[r3:NEXT]->(cm)
```

On the other hand, the transformation of documents without an 'input' field (from both 'Dionaea' and 'Cowrie' types) is shown in Listing 5.

Listing 5: Cypher code for Dionaea and Cowrie

```
MERGE (a:Address {
   address: source_address,
   port: source_port,
   protocol: source_protocol,
   abuseConfidenceScore: abuse_confidence_score,
   lastReportedAt: last_reported_at
})
MERGE (c:Country {
   name: country
})
MERGE (as:AS {
   name: as_field
})
MERGE (ut:UsageType {
   name: usage_type
})
MERGE (h:Honeypot {
   type: honeypot,
   analyzer_id: analyzer_id
})
MERGE (a)-[:LOCATED_IN]->(c)
MERGE (a)-[:BELONGS_TO]->(as)
MERGE (a)-[:HAS_USAGE_TYPE]->(ut)
MERGE (a)-[r1:CONNECTED_TO]->(h)
```

Despite the absence of command data, the relationship between the 'Session', 'Address', 'Honeypot', 'Country', 'UsageType', and 'AS' nodes was upheld, providing valuable insights about the session.

It is crucial to note that while our transformation process exhibited effective results with the current dataset, the adaptability to other data structures depends on the specificities of the MongoDB documents. Complex or diverse document structures may require modifications in the transformation process.

V. Application Scenarios

This section highlights how our approach can be used to understand the sequence and pattern of commands employed by an attacker in a specific session, summarize activities originating from a particular source address, assess the effectiveness and performance of one particular honeypot, uncover common patterns in malicious activities, and visualize where attacks are originating from geographically.

1) Session-oriented Command Analysis: For scenarios where the sequence and pattern of commands employed by an attacker in a specific session need to be understood, the Cypher code in Listing 6 can be executed.

Listing 6: Cypher for session-oriented command analysis

```
MATCH (s:Session {sessionid: 'particular-session-
    id'})-[h:ORIGINATED_FROM]-(a:Address)-[e:
    EXECUTED]-(c:Command)
RETURN s,h,a,e,c
```

2) Comprehensive Overview of Source Address Activity: When summarizing activities originating from a specific source address is necessary, the query in Listing 7 can provide insights into the behaviour of certain actors in the network.

Listing 7: Cypher for source address activity

```
MATCH (n)-[e:EXECUTED]-(a:Address {address: '
    specific-ip-address'})-[r:ORIGINATED_FROM]-(
    s:Session)
RETURN n,e,a,r,s
```

3) Count of Honeypot Interactions: When assessing the effectiveness and performance of a specific honeypot by analyzing interactions with it, the query in Listing 8 can be used as a useful tool.

Listing 8: Cypher for honeypot interactions count

```
MATCH (h:Honeypot {analyzer_id: 'particular-
    analyzer-id'})<-[r:CONNECTED_TO]-(s:Session)
RETURN count(s) as InteractionCount
```

4) Sequential Pattern Analysis: In order to uncover common patterns or routines in malicious activities by analyzing command sequences, the query in Listing 9 can be utilized.

Listing 9: Cypher for sequential pattern analysis

```
MATCH path = (c1:Command)-[:NEXT*..5]->(c2:
    Command)
WHERE c1.command = 'particular-command'
RETURN path
```

This query returns paths where a specific command is followed by any sequence of up to five commands, revealing potential attack routines.

5) Geographical Analysis of Attack Origins: For visualizing where attacks are originating from geographically, the query shown in Listing 10 can be utilized.

Listing 10: Cypher for attack origins geographical analysis

```
MATCH (c:Country)<-[:LOCATED_IN]-(a:Address)-[:
    ORIGINATED_FROM]-(s:Session)
RETURN c.name as Country, count(distinct a) as
    NumberOfAttacks
ORDER BY NumberOfAttacks DESC
```

This query returns a list of countries and the number of unique attacks originating from each, ordered by the number of attacks in descending order.

VI. Conclusion

Utilizing knowledge graphs, we have demonstrated a robust framework for the transformation and analysis of honeypot data. Our methodology converted raw data into a structured knowledge graph and enriched it with external information, significantly enhancing our understanding and the utility of such data in cybersecurity.

The implementation of the graph database facilitated complex querying and analysis, highlighting the practical applications of knowledge graphs in this context. Our results emphasize the crucial role of these graphs in enabling more effective threat detection and response mechanisms, achieved through faster and deeper analysis of honeypot data.

However, the transformation process's adaptability is heavily contingent on the characteristics of the source documents, posing certain challenges. Therefore, further research is needed to explore methods that address these variations, making the process more versatile. Furthermore, integrating real-time graph updating, pattern recognition with machine learning, and threat prediction techniques hold immense potential for further advancements in this field.

Acknowledgment

This work was supported by The Information Society Innovation Fund (ISIF Asia), under the Knowledge Research Grant for 2021.

References

[1] L. Spitzner, *Honeypots: tracking hackers*, vol. 1. Addison-Wesley Reading, 2003.

[2] M. Oosterhof, "Cowrie ssh/telnet honeypot https://github.com/cowrie/cowrie [online]," 2023. [Accessed: Jul. 26, 2023].

[3] Dionaea, "https://github.com/DinoTools/dionaea [online]," 2023. [Accessed: Jul. 26, 2023].

[4] Neo4j, "https://neo4j.com [online]," 2023. [Accessed: Jul. 26, 2023].

[5] Ryandy, C. Lim, and K. E. Silaen, "XT-Pot: EXposing Threat Category of Honeypot-Based Attacks," in *Proceedings of the 2021 International Conference on Engineering and Information Technology for Sustainable Industry*, ICONETSI '21, (New York, NY, USA), Association for Computing Machinery, 2020.

[6] R. Djap, C. Lim, K. E. Silaen, and A. Yusuf, "XB-Pot: Revealing Honeypot-based Attacker's Behaviors," in *2021 9th International Conference on Information and Communication Technology (ICoICT)*, pp. 550–555, 2021.

[7] Y. Andrew, C. Lim, and E. Budiarto, "Mapping linux shell commands to mitre att&ck using nlp-based approach," in *2022 International Conference on Electrical Engineering and Informatics (ICELTICs)*, pp. 37–42, 2022.

[8] K. Liu, F. Wang, Z. Ding, S. Liang, Z. Yu, and Y. Zhou, "Recent progress of using knowledge graph for cybersecurity," *Electronics*, vol. 11, no. 15, p. 2287, 2022.

[9] S. N. Narayanan, A. Ganesan, K. Joshi, T. Oates, A. Joshi, and T. Finin, "Early detection of cybersecurity threats using collaborative cognition," in *2018 IEEE 4th international conference on collaboration and internet computing (CIC)*, pp. 354–363, IEEE, 2018.

[10] S. Wu, Y. Zhang, and W. Cao, "Network security assessment using a semantic reasoning and graph based approach," *Computers & Electrical Engineering*, vol. 64, pp. 96–109, 2017.

[11] L. F. Sikos, "Cybersecurity knowledge graphs," *Knowledge and Information Systems*, pp. 1–21, 2023.

[12] G. Shen, W. Wang, Q. Mu, Y. Pu, Y. Qin, and M. Yu, "Data-driven cybersecurity knowledge graph construction for industrial control system security," *Wireless Communications and Mobile Computing*, vol. 2020, pp. 1–13, 2020.

[13] Z. Wang, H. Zhu, P. Liu, and L. Sun, "Social engineering in cybersecurity: a domain ontology and knowledge graph application examples," *Cybersecurity*, vol. 4, pp. 1–21, 2021.

[14] A. Pingle, A. Piplai, S. Mittal, A. Joshi, J. Holt, and R. Zak, "Relext: Relation extraction using deep learning approaches for cybersecurity knowledge graph improvement," in *Proceedings of the 2019 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, pp. 879–886, 2019.

[15] L. Du and C. Xu, "Knowledge graph construction research from multi-source vulnerability intelligence," in *Cyber Security: 19th China Annual Conference, CNCERT 2022, Beijing, China, August 16–17, 2022, Revised Selected Papers*, pp. 177–184, Springer Nature Singapore Singapore, 2022.

[16] A. Piplai, S. Mittal, A. Joshi, T. Finin, J. Holt, and R. Zak, "Creating cybersecurity knowledge graphs from malware after action reports," *IEEE Access*, vol. 8, pp. 211691–211703, 2020.

[17] E. Kiesling, A. Ekelhart, K. Kurniawan, and F. Ekaputra, "The sepses knowledge graph: an integrated resource for cybersecurity," in *The Semantic Web–ISWC 2019: 18th International Semantic Web Conference, Auckland, New Zealand, October 26–30, 2019, Proceedings, Part II 18*, pp. 198–214, Springer, 2019.

[18] A. Ekelhart, F. J. Ekaputra, and E. Kiesling, "The slogert framework for automated log knowledge graph construction," in *The Semantic Web: 18th International Conference, ESWC 2021, Virtual Event, June 6–10, 2021, Proceedings*, pp. 631–646, Springer, 2021.

[19] Y. Jia, Y. Qi, H. Shang, R. Jiang, and A. Li, "A practical approach to constructing a knowledge graph for cybersecurity," *Engineering*, vol. 4, no. 1, pp. 53–60, 2018.

[20] G. Agrawal, Y. Deng, J. Park, H. Liu, and Y.-C. Chen, "Building knowledge graphs from unstructured texts: Applications and impact analyses in cybersecurity education," *Information*, vol. 13, no. 11, p. 526, 2022.

[21] Y. Tao, M. Li, and W. Hu, "Research on knowledge graph model for cybersecurity logs based on ontology and classified protection," in *Journal of Physics: Conference Series*, vol. 1575, p. 012018, IOP Publishing, 2020.

[22] Y. Deng, Z. Zeng, K. Jha, and D. Huang, "Problem-based cybersecurity lab with knowledge graph as guidance," *Journal of Artificial Intelligence and Technology*, 2021.

[23] Y. Ren, Y. Xiao, Y. Zhou, Z. Zhang, and Z. Tian, "Cskg4apt: A cybersecurity knowledge graph for advanced persistent threat organization attribution," *IEEE Transactions on Knowledge and Data Engineering*, 2022.

[24] Y. Chen, H. Li, H. Li, W. Liu, Y. Wu, Q. Huang, and S. Wan, "An overview of knowledge graph reasoning: key technologies and applications," *Journal of Sensor and Actuator Networks*, vol. 11, no. 4, p. 78, 2022.

[25] R. Alm and L. Imeri, "A performance comparison between graph databases: Degree project about the comparisonbetween neo4j, graphdb and orientdb on different operations," 2021.

[26] J. Monteiro, F. Sá, and J. Bernardino, "Experimental evaluation of graph databases: Janusgraph, nebula graph, neo4j, and tigergraph," *Applied Sciences*, vol. 13, no. 9, p. 5770, 2023.

[27] T. Security, "https://github.com/telekom-security/ewsposter [online]," 2023. [Accessed: Jul. 26, 2023].

[28] MongoDB, "Why use mongodb and when to use it? https://www.mongodb.com/why-use-mongodb [online]," 2023. [Accessed: Jul. 26, 2023].

[29] AbuseIPDB, "https://abuseipdb.com [online]," 2023. [Accessed: Jul. 26, 2023].

[30] IP-API, "https://ip-api.com [online]," 2023. [Accessed: Jul. 26, 2023].