LOG FILE ANOMALY DETECTION USING KNOWLEDGE GRAPHS AND GRAPH NEURAL NETWORKS

By

Lucas A. Payne

Mengjun Xie
Associate Professor of Computer Science and Engineering
(Chair)

Hong Qin
Professor of Computer Science and Engineering
(Committee Member)

Li Yang
Guerry Professor of Computer Science and Engineering
(Committee Member)

LOG FILE ANOMALY DETECTION USING KNOWLEDGE GRAPHS AND GRAPH NEURAL NETWORKS

By

Lucas A. Payne

A Thesis Submitted to the Faculty of the University of
Tennessee at Chattanooga in Partial
Fulfillment of the Requirements of the Degree
of Master of Computer Science

The University of Tennessee at Chattanooga
Chattanooga, Tennessee

August 2023

ABSTRACT

Log files contain valuable information for detecting abnormal behavior. To detect anomalies, researchers have proposed representing log files as knowledge graphs (KGs) and using KG completion (KGC) techniques to predict new facts. However, current research in this area is limited, and there is no end-to-end system that includes both KG generation and KGC for log-based anomaly detection. This study presents an end-to-end system that utilizes graph neural networks (GNNs) and KGC to detect anomalies in log files. The proposed system has two main components. The first component employs templates to generate a KG from logs that capture normal behavior. The second component applies KG embedding models enhanced with GNN layers to the generated KG and employs KGC to determine suspiciousness of new information through binary classification. The proposed method is evaluated using two public datasets with standard KGC metrics. The experimental results demonstrate its promising potential.

DEDICATION

In loving memory of my mamaw, Gladys Dyer, who passed away while I was writing this thesis. She always put others before herself and always made sure no one went hungry.

ACKNOWLEDGEMENTS

This endeavor would not have been possible without Dr. Mengjun Xie, my committee chair and advisor, who helped decide my thesis topic and offered valuable aid throughout the research, development, and writing process. I am also extremely grateful to the other members of my committee, Drs. Hong Qin and Li Yang, for their feedback and support. Additionally, I could not have undertaken this project without the generous support from the National Security Agency, who funded my research.

Special thanks should go to my wife for her unwavering love, support, and encouragement. I am also grateful to my friends and family, especially my parents, for holding me accountable, working around my odd hours, and for their love and support. Of course, I would be remiss in not mentioning my dog Bailey, who encouraged me to take frequent breaks from my work.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

CSV, Comma-Separated Value
CWA, Closed World Assumption
FN, False Negative
FNR, False Negative Rate
FOAF, Friend of a Friend
FP, False Positive
FPR, False Positive Rate
GCN, Graph Convolutional Network
GDN, Graph Deviation Network
GGNN, Gated Graph Neural Network
GNN, Graph Neural Network
IDS, Intrusion Detection System
KG, Knowledge Graph
KGC, Knowledge Graph Completion
KGEM, Knowledge Graph Embedding Model
KGG, Knowledge Graph Generation
LCWA, Local Closed World Assumption
MR, Mean Rank
MRR, Mean Reciprocal Rank
OWA, Open World Assumption
OWL, W3C Web Ontology Language
RDF, Resource Description Framework
sLCWA, Stochastic Local Closed World Assumption
SLOGERT, Semantic Log Extraction Templating
SPARQL, SPARQL Protocol and RDF Query Language
STIX, Structured Threat Information Extension
TABEL, Tables Extracted as Linked Data
TN, True Negative
TNR, True Negative Rate
TP, True Positive
TPR, True Positive Rate
TSV, Tab-separated Value
UCO, Unified Cybersecurity Ontology
URI, Uniform Resource Identifier
W3C, World Wide Web Consortium
XML, Extensible Markup Language

LIST OF SYMBOLS

$\mathcal{K}$, knowledge graph
$\mathcal{E}$, set of all entities
$\mathcal{R}$, set of all relations

CHAPTER 1

INTRODUCTION

Computer systems and applications write important and rich information to log files, including configurations, status messages, errors, and security events. Many log messages are the result of normal system or application behavior; however, they can also contain abnormal messages that suggest an attack has occurred. Thus, detecting anomalies in log files can lead to increased computer system security. Several research efforts have been conducted on this topic using traditional machine learning methodologies (e.g., [1, 2]). Recent advancements in deep learning have attracted additional attention to the topic (e.g., [3]).

Knowledge graph completion (KGC) has recently been proposed for detecting anomalies in log files [4, 5]. In this approach, information from a set of log files representing the normal operation of a computing environment is organized into a graph structure known as a knowledge graph (KG), and a KGC machine learning model is trained on this knowledge graph. When new log data arrives, the model determines whether the inferred behavior is anomalous.

This thesis proposes a full end-to-end KGC based anomaly detection system and has two main components. The first component presents a framework and implementation of a method to transform unstructured log file data into a structured KG format. The second component applies several KGC methods to perform anomaly detection, using both traditional neural network and graph neural network (GNN) augmented knowledge graph embedding models (KGEMs). The proposed system was evaluated with the common KGC metrics of mean rank (MR), mean reciprocal rank (MRR), and Hits@k, as well as standard classification metrics such as accuracy and F1-score. The experimental results demonstrate that the proposed approach to anomaly detection is promising.

Figure 1.1 shows an overview of the proposed method. In the knowledge graph generation (KGG) step, raw log lines are matched with templates. For each match that is found, several triples are generated using the extracted parameters from the raw log line. The log lines are also labeled to indicate whether they contain normal or suspicious behavior. The goal of the KGEM is to learn a representation of normal behavior within the computing environment and evaluate new information according to this representation to determine whether the new behavior is anomalous. Therefore, the training dataset is made up of only normal behavior, while the validation and test datasets are made up of both normal and suspicious behavior. These datasets are then used to train and evaluate a knowledge graph embedding model (KGEM), which may contain only the scoring function or a number of GNN layers and the scoring function. The scoring function outputs a score representing the confidence that the evaluated triple belongs in the training KG (i.e., the likelihood that the triple represents normal behavior). Each model's KGC performance is evaluated based on Hits@k, MR, and MRR, and classification performance is evaluated based on accuracy and F1-score.

I. Background

Knowledge graphs are a representation of knowledge as a directed graph, where the nodes of the graph are entities, and the edges are the relations between them. Formally, a knowledge graph $\mathcal{K}$ can be defined as $\mathcal{K} = (\mathcal{E}, \mathcal{R})$, where $\mathcal{E}$ is the set of entities and $\mathcal{R}$ is the set of relations. A KG is made up of triples $(subject, relation, object)$ or (s, r, e), where $s, o \in \mathcal{E}$ and $r \in \mathcal{R}$. An example of a knowledge graph triple is (Tom, IsFatherOF, Mike), which states that Tom is Mike's father.

KGs are often incomplete. That is, additional relations can be inferred from the knowledge graph. Knowledge graph completion or link prediction is the task of predicting these additional relations. For example, given (Tom, IsFatherOf, Mike) and (Tom, IsBrotherOf, Charlie), at least four additional facts can be inferred: (Mike, IsSonOf, Tom), (Charlie, IsBrotherOf, Tom), (Charlie, IsUncleOf, Mike), and (Mike, IsNephewOf, Charlie). This example is illustrated in Figure 1.2.

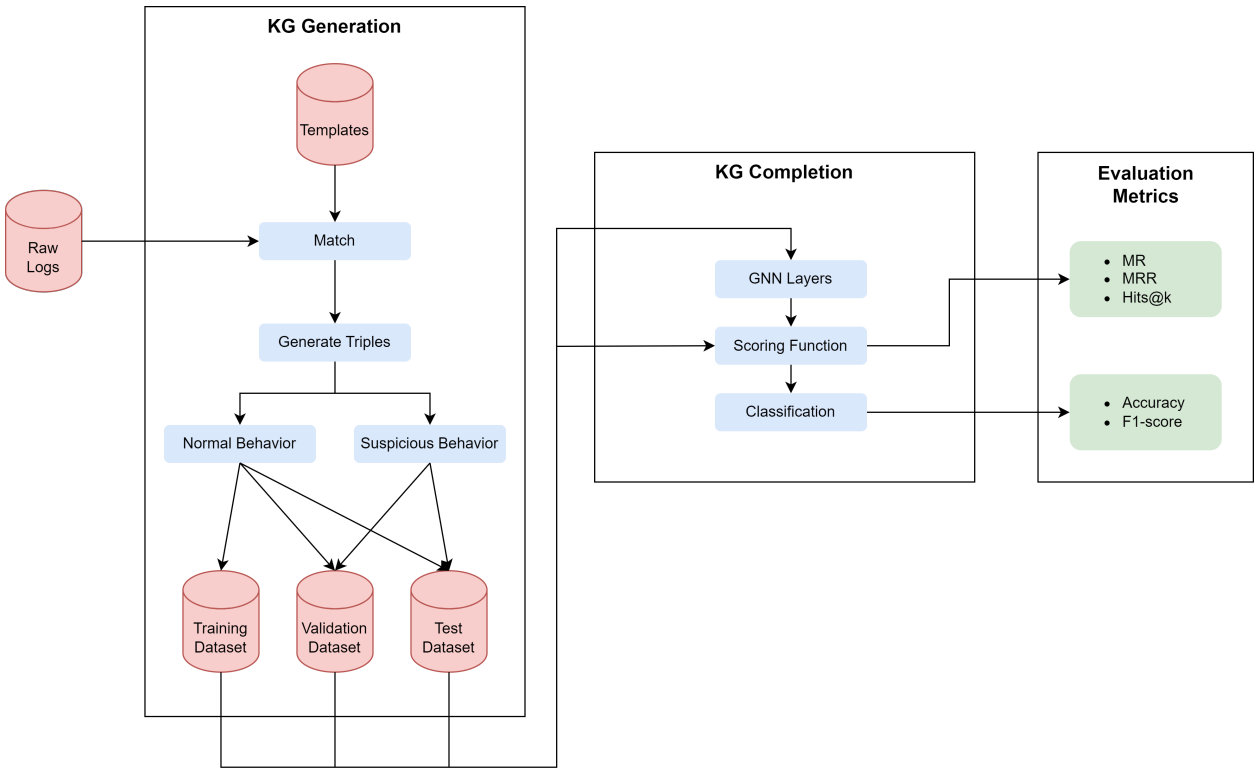

Figure 1.1 Anomaly Detection Overview

The use of knowledge graphs in log file anomaly detection is quite new [4, 5], and the focus of existing papers is on training novel KGEMs to detect anomalies on an already-constructed knowledge graph, while the process of transforming log file data into a KG format is not explained. Additionally, results in these initial papers were not given in the standard KGC metrics and are difficult to interpret.

There are several existing papers (e.g., [6–12]) that present methods for generating KGs from log files, normally into the resource description framework (RDF) format. However, many of these methods are constructed considering the downstream task to be SPARQL (SPARQL Protocol and RDF Query Language) queries. Certain properties of the KGs resulting from these methods make them unsuitable for the KGC task.

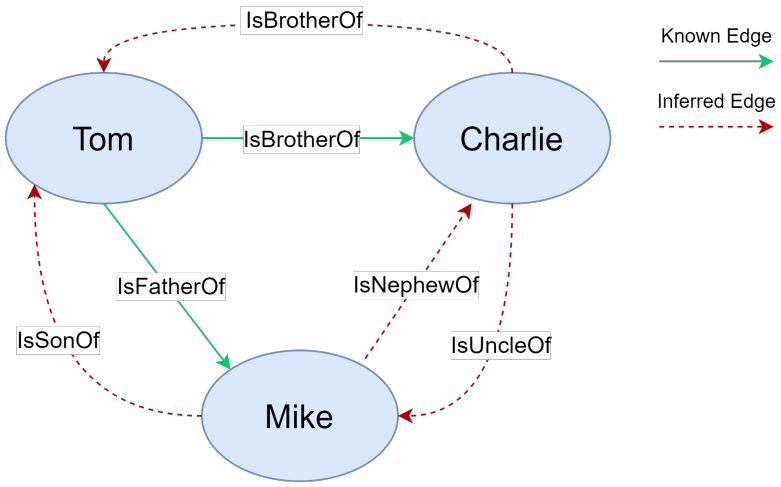

Figure 1.2 Knowledge Graph Completion Example

II. Statement of the Problem

While log files contain information vital to cybersecurity operations, their semi-structured nature means that they cannot be directly processed by computers. Thus, it is necessary to transform log data into a more structured format. Recently, researchers have proposed representing log data as a knowledge graph and using knowledge graph completion to detect anomalies in log files. This is a promising research direction; however, current research is quite limited. In particular, there are currently no suitable methods for generating KGs from log files that consider KGC as the downstream task. The performance of current KGC methods for log file anomaly detection is also difficult to gauge due to the lack of standard metrics. Finally, current models are quite shallow and may benefit from additional layers, especially GNNs.

III. Contributions of the Study

Following are the contributions of this thesis.

- First, a complete end-to-end KGC-based anomaly detection system is proposed. This includes, first, generating knowledge graphs from raw log file data, considering knowledge graph completion as the downstream task. Although methods of generating knowledge graphs from

log files have been proposed in the past, to the knowledge of the author, no method has been proposed considering knowledge graph completion as the downstream task.

- Next, both traditional and GNN-augmented KGEMs are applied to the generated knowledge graph data to identify anomalies within the log data. To the knowledge of the author, no study so far has applied KGEMs that include GNN layers to this problem.

- Finally, results are reported with the standard KGC metrics of MR, MRR, and Hits@k, and a comparison of performance between traditional and GNN-augmented machine learning models is given. Previous studies on anomaly detection in log files using knowledge graph completion report findings using non-standard metrics, and using the standard knowledge graph completion and classification metrics allows the results to fit in more easily with previous work in both knowledge graph completion and classification problems.

IV. Thesis Structure

Chapter 2 conducts a literature review on the topics of KGG, KGC, and log file anomaly detection. Chapter 3 discusses the methods used to implement the proposed anomaly detection system. Chapter 4 provides the experimental results from the anomaly detection system. Finally, Chapter 5 provides a discussion of the results, explore future areas of research, and conclude the thesis.

CHAPTER 2

LITERATURE REVIEW

In this chapter, a literature review is conducted on several topics related to log file anomaly detection and knowledge graph completion to give a richer understanding of the subject before proceeding to the particular methods of this work. First, Section I discusses knowledge graphs in much greater detail. Then, Section II, discusses the incompleteness of knowledge graphs and methods to complete them. Section III discusses how knowledge graphs can be generated from text and, in particular, how they can be generated from log files. Next, Section IV discusses recent literature on using knowledge graph completion to detect anomalies in log files. Finally, Section V summarizes the literature review and conclude this chapter.

I. Knowledge Graphs

Modern knowledge graph research began with the Google Knowledge Graph [13], which was followed by similar announcements from other companies (e.g., Amazon [14], Facebook [15], and Microsoft [16]). Since their adoption in industry, much research has been conducted on the topic. At the heart of knowledge graph research is the graph representation of data.

Knowledge graphs organize data into a labeled, directed graph structure which contains a set of entities $\mathcal{E}$ and a set of relations $\mathcal{R}$. The entities are the nodes of the graph, while the relations are the edges. Thus, a knowledge graph can be represented as $\mathcal{K} = (\mathcal{E}, \mathcal{R})$. Any fact on a knowledge graph can be written as a triple with two entities and a relation that connects them. The first entity is referred to as the subject or head entity, while the second entity is referred to as the object or tail entity. A triple can be written as $(subject, relation, object) \implies (s, r, e)$, where $s, o \in \mathcal{E}$

and $r \in \mathcal{R}$, or $(head, relation, tail) \implies (h, r, t)$, where $h, t \in \mathcal{E}$ and $r \in \mathcal{R}$. This thesis uses the former notation.

A. Storage Formats

There are several ways in which knowledge graph data can be stored on disk, and some of the most popular are discussed here. Certainly the easiest way to store knowledge graph data is as simple text in, for example, a tab-separated value (TSV) or comma-separated value (CSV) file. Here, one knowledge graph triple is stored per line with the subject entity, relation, and object entity being separated by some delimiter, such as a tab or comma. Another step that can be taken is to map entities and relations to identifiers. For example, the following entity mappings could be created: $\{\text{Tom} \rightarrow 0, \text{Mike} \rightarrow 1\}$, and the following relation mappings might be created: $\{\text{isFatherOf} \rightarrow 0, \text{isSonOf} \rightarrow 1\}$. The mappings for entities and relations are stored in separate files so that translations can be made when necessary. This considerably reduces the size on disk of the knowledge graph.

Another popular form of storage for knowledge graphs is using the Resource Description Framework (RDF) [17], a standard developed by the World Wide Web Consortium (W3C) to support interchanging data over the Internet. RDF uses Extensible Markup Language (XML) syntax to represent triples, and each entity and relation in the triple is a Uniform Resource Identifier (URI). RDF was not purpose-built to represent knowledge graphs; it was made for the more general purpose of representing structured and semi-structured information on the Internet. However, knowledge graphs are a major application of RDF.

One of the major advantages of using the RDF language for representing knowledge graphs is the ability to link to external knowledge sources. Since many sources use RDF to represent knowledge, it is easy to incorporate this information into a knowledge graph. In a cybersecurity context, a local knowledge graph may be enhanced by linking to external facts about known attacks and threats. One disadvantage of RDF is that the XML syntax is difficult for humans to parse and understand. To address this, W3C introduced the Turtle dialect of RDF [18], which offers a more

human-readable syntax. Note that the use of ID mappings to reduce storage cost is also easily applicable to the Turtle format. W3C also maintains a standard query language for finding and manipulating RDF data. This standard is known as SPARQL, which is short for SPARQL Protocol and RDF Query Language [19].

When using RDF and other Semantic Web technologies, knowledge is often represented using ontologies, or sets of concepts and the possible relations between those concepts [20]. W3C also maintains a standard, known as the W3C Web Ontology Language (OWL), for representing ontologies [21]. Ontologies are typically domain specific. For example, the friend of a friend (FOAF) ontology [22] contains concepts and relations to describe knowledge about people and social networks, while the Structured Threat Information Extension (STIX) [23] and Unified Cybersecurity Ontology (UCO) [24] contain cybersecurity concepts and relations.

II. Knowledge Graph Completion

Organizing knowledge into a graph structure enables machines to more easily reason over the knowledge compared to unstructured text. This reasoning includes inferring new facts based on the ones already present. Since knowledge graphs do not contain all knowledge in their domain, they are considered incomplete [25]. Knowledge graph completion, or link prediction, is the task of filling in missing facts or inferring new facts and adding them to the knowledge graph. This task is equivalent to predicting the probability that a particular edge would be correct if added to the graph [25]. Knowledge graph completion is most often performed with machine learning techniques. In these types of machine learning models, a scoring function $f(x_{s,r,o}, \theta)$ is used to determine the probability that a given triple exists in the knowledge graph, where $x_{s,r,o}$ represents the embeddings of the subject, relation, and object of the triple in question, and θ represents the model's parameters.

A. *Open and Closed World Assumptions*

During training, KGEMs can choose between a few different assumptions to work under about triples not contained in the knowledge graph. The first such assumption is the open world assumption (OWA). In the OWA, triples that are not contained in the KG are considered unknown, i.e., it is unclear whether any such triple is correct or incorrect. If the OWA is used, then unknown entities are included in the testing and validation sets. Learning under this assumption leads to model over-generalization and under-fitting [25].

Another assumption is the closed world assumption (CWA). Under this assumption, any triple not contained in the KG is considered incorrect, implying that the KG has complete knowledge of the domain. This assumption is similarly unhelpful, especially since the KGC problem assumes that the KG is incomplete. The CWA also leads to model over-fitting [25]. When the CWA is used, any triples involving unknown entities are removed from the testing and validation sets.

There are a couple variations of the CWA, which are useful for training KGEMs. One variation of this assumption is the local closed world assumption (LCWA), proposed in [26], which assumes that only a subset of the triples not contained in the knowledge graph are incorrect. Note that some KGEMs require the creation of negative, or false, triples.

The LCWA can also be used to generate sets of negative triples. This is usually done by corrupting, or changing, one of the entities or the relation. Specifically, consider the positive (or correct) triple $(s, r, o) \in \mathcal{K}$. A negative triple can be generated by corrupting any element. The traditional method is to corrupt the object entity, so that the set of negative triples $\mathcal{T}_o^-$ looks like

$$\mathcal{T}_o^-(s, r) = \{(s, r, o) \mid o \in \mathcal{E} \wedge (s, r, o) \notin \mathcal{K}\}$$

In other words, such a negative triple is one where the object entity exists in the KG, but the corrupted triple is not in the KG and is considered false. However, similar strategies can be used to generate sets of negative triples by corrupting the subject entity or the relation.

Another similar assumption is the stochastic local closed world assumption (sLCWA), which is similar to the LCWA, but differs in the negative triple generation strategy. Here, the subject and object negative triple generation strategies are used, but only a random sample of the union of these sets are considered as negative samples [27]. So, instead of considering all corruptions as false, only random subsets of them are considered false.

B. Models

The most common approach to knowledge graph completion is to train knowledge graph embedding models, where the model learns vector representations of the entities and relations of a knowledge graph (the training dataset) [27]. These vectors are learned in such a way that when a given triple is fed through the model's scoring function, the scoring function should return a high probability if the triple is correct and a low probability if the triple is incorrect. There are several approaches to learn such vector representations, and this section examines several of them.

TransE

TransE [28] is an early KGEM that uses a translational approach. Its basic idea is that if a triple (s, r, o) is true, then the sum of the embedding vectors $\mathbf{s}$ and $\mathbf{r}$ should be very close to the embedding vector $\mathbf{o}$, so that $\mathbf{s} + \mathbf{r} \approx \mathbf{o}$. Therefore, the score of a triple can be taken as some measure of distance from the origin to $\mathbf{s} + \mathbf{r} - \mathbf{o}$. The original paper considers the L1 norm (Taxicab distance) and the L2 norm (Euclidean distance), so that the scoring function is given by

$$f(s, r, o) = -\|\mathbf{s} + \mathbf{r} - \mathbf{o}\|_k.$$

where $k \in \{1, 2\}$.

The simplicity of the scoring function allows this model to scale well to large knowledge graphs. However, a major drawback of this approach is that it implicitly prefers one-to-one relations and has difficultly considering one-to-many, many-to-one, and many-to-many relations [27]. For

example, consider two correct triples with the same subject entity and relation, but a different object entity, say (s, r, o_1) and (s, r, o_2). Then, under the assumptions of TransE, the result is $\mathbf{s} + \mathbf{r} \approx \mathbf{o_1} \approx \mathbf{o_2}$. This implies that o_1 and o_2 are the same entity or extremely similar, which may not be the case. For example, if the KG represents a social network, it is certainly possible for one person to have multiple friends.

RESCAL

RESCAL [29] models entities as vectors and relations as matrices $\mathbf{W}_r \in \mathbb{R}^{d \times d}$, where d is the embedding dimension hyperparameter. The relation matrices contain weights that capture the interactions between each pair of subject and object entities $\mathbf{s} \in \mathbb{R}^d$ and $\mathbf{o} \in \mathbb{R}^d$. The scoring function for RESCAL is given by

$$f(s, r, o) = \mathbf{s}^T \mathbf{W}_r \mathbf{o}.$$

DistMult

DistMult [30] is quite similar to RESCAL [29]. The only difference is that $\mathbf{W}_r$ is specified to be diagonal, or equivalently, there is a vector $r \in \mathbb{R}^d$ that can be converted to a diagonal matrix so that $\text{diag}(\mathbf{r}) = \mathbf{W}_r$. The scoring function can be characterized either by a dot product, or as a similar product to RESCAL:

$$f(s, r, o) =< \mathbf{s}, \mathbf{r}, \mathbf{o} >= \mathbf{s}^T \cdot \text{diag}(\mathbf{r}) \cdot \mathbf{o} = \mathbf{s}^T \mathbf{W}_r \mathbf{o}.$$

Clearly, this leads to increased computational efficiency. However, this comes at the cost of not being able to represent anti-symmetric relations [27], which are those where, for instance, the subject has a certain relation to the object, but the object does not have the same relation to the subject. This is lost with the diagonal matrix restriction since $f(s, r, o) = \mathbf{s}^T \mathbf{W}_r \mathbf{o} = \mathbf{o}^T \mathbf{W}_r \mathbf{s} = f(o, r, s)$.

ComplEx

ComplEx [31] is in turn an extension of DistMult [30]. In this case, the embedding dimension is extended to complex numbers, and the scoring function involves the Hermitian product, or dot product in complex space, of the entities and relation. In particular, $\mathbf{s}, \mathbf{r}, \mathbf{o} \in \mathbb{C}^d$, where d, again, is the embedding dimension hyperparameter. The scoring function is given by

$$f(s, r, o) = Re(< \mathbf{s}, \mathbf{r}, \overline{\mathbf{o}} >) = \mathbf{s}^T \cdot \text{diag}(\mathbf{r}) \cdot \overline{\mathbf{o}} = \mathbf{s}^T \mathbf{W}_r \overline{\mathbf{o}}$$

where $Re(x)$ is the real part of a vector $x \in \mathbb{C}^d$ and $\overline{\mathbf{x}} = Re(x) - iIm(x)$ is the complex conjugate of a vector $x \in \mathbb{C}^d$. Since the Hermitian product is not commutative, ComplEx is able to model anti-symmetric relations, where DistMult is not [27].

III. Knowledge Graph Generation From Log Files

Log files are semi-structured documents that record certain activities on a computer, including status messages, software application events, and user and network activity. Information in log files is invaluable to security analysts since they often contain information vital to identifying attacks on computer systems. However, their semi-structured data and varying formats makes it difficult for computers to parse. Therefore, reviewing log files is often a manual task performed by humans. However, it is vital that attacks and intrusions are identified as quickly as possible to reduce response time and the damage caused by an attack. This has led to much research on processing log files to automatically detect intrusions, and these techniques normally include a preprocessing step to convert the raw log files into more structured formats. Some of these works involve converting log files into knowledge graphs. This section will review these works and discuss their strengths and weaknesses.

A. TABEL

Mulwad, Finn, and Joshi introduced the Tables Extracted as Linked Data (TABEL) framework [6–8], which is a process of inferring the semantics of tables and writing those semantics as RDF linked data. This framework was designed to be domain-indpendent, operating on arbitrary table data, so it was not created specifically to operate on log files. However, Nimbalkar *et al.* [9] used this approach to process log files by first converting the log files to table data, then applying the TABEL framework . This approach is meant to operate on arbitrary log data, where the format of the log file may be unfamiliar or specific to a certain application, generating RDF linked data that describes both the structure and content of the log data.

The first step of the process is tabulating the log data, which is done by dividing the data into rows and columns. Each column represents a similar type of data, while each row represents a separate entry. The log data is divided into columns using a number of "specialists", which are regular expressions, dictionaries, and other classifiers that can recognize data types like IP addresses, HTTP status messages, etc [9]. Each column is linked to a semantic class from an ontology, and each pair of columns is associated with a semantic relation from an ontology [9]. From this, a set of RDF triples can be generated to form a knowledge graph. In this framework, the knowledge graph triples are organized around the concept of log entries. For example, a triple may state that a log entry with a certain identifier has a certain timestep. Since such a linked data format is used, other external data can be linked to these RDF triples to connect the log information to cybersecurity concepts. Another related work describes extracting such cybersecurity information from semi-structured databases as well as unstructured texts such as cybersecurity bulletins [32]. The resulting knowledge graph can better facilitate search and reasoning than the raw log data.

One of the major strengths of applying the TABEL framework to log file knowledge graph generation (KGG) is that it can process arbitrary log files and is completely automatic in splitting the raw log data into columns, identifying their probable semantic classes, creating RDF triples, and linking to external data. One of the primary motivations of this work was that there were existing tools that could parse certain sets of log files well, but could not support types of log files at all.

While handling a fixed set of log files allows a more concrete understanding of those particular formats, in many cases, the convenience of being able to infer the semantics of any type of log file may outweigh the potential inaccuracies. However, While this approach may work well for log files that have a mostly consistent format, some log files have several log messages with very different formats. For these types of log files, it would be difficult to separate the log file content into consistent columns. In this case, a line-by-line approach may yield better results (i.e., generating a separate one-row table for each line). Additionally, while this organization of data is well-suited for querying with simple search or query languages such as SPARQL, it is not well-suited for a downstream task of knowledge graph completion for reasons that will be discussed below.

B. SLOGERT

Semantic LOG ExRaction Templating (SLOGERT) [11, 12] is a framework for generating structured RDF knowledge graphs from unstructured log files of various formats. SLOGERT relies on several established log processing tools to build its RDF conversion pipeline. First, a template is created for the log file, which includes the structure of a log line and the locations of parameters (i.e., variable parts such as IP addresses). Next, semantics are introduced to the log templates by annotating parameters with semantic types and extracting certain keywords. Then, the templates are transformed to RDF templates for generating knowledge graph triples, and these templates are combined with the extracted parameters to generate the final knowledge graph. Additionally, the authors of SLOGERT have made a log ontology to complement this work and provide some of the necessary semantic classes and relations.

Similarly to how to the TABEL framework is applied to log file KGG [9], SLOGERT organizes the generated KG around log events representing one log line, each of which is given a unique identifier. Then, each parameter, as well as some metadata, is related back to this event. Parameters include data provided explicitly in the log line, such as user names and IP addresses,

while metadata includes information which may not be explicitly provided, such as which log file and the host from which the event originates.

This organization and especially the inclusion of metadata lends itself well to the intended downstream task of manual querying and forensic analysis. For example, an analyst could set up SPARQL queries to find data pertaining to the activities of a certain user over a time period. Also, using the Turtle RDF dialect to represent log data as linked data allows enriching local data with external knowledge (e.g., with the related work of the SEPSES knowledge graph [33]), which could help give analysts a deeper understanding of the data at a glance.

However, when considering KGC as the downstream task, SLOGERT has some of the same weaknesses as TABEL, which will be elaborated upon later. Additionally, SLOGERT only supports a few types of log files, such as some UNIX logs and a few other types, such as suricata logs, out of the box. Supporting an additional log file format requires a considerable amount of configuration across multiple files, which is not ideal. However, while the TABEL framework allows arbitrary log formats to be processed, having specific configuration knowledge as SLOGERT requires may increase the quality of the final KG since the structure of log files does not have to be inferred completely. Finally, SLOGERT does not take full advantage of the information-rich message fields of log files. In fact, these fields often contain the bulk of information within a log file. The entire message field is treated as a single entity; however, several additional entities and relations could be extracted from these message fields, using either natural language processing or templates.

C. LEKG

Wang *et al.* [10] presented the LEKG system for extracting knowledge graphs from log files, which aims to make advancements and improvements over previous methods such as SLOGERT [11, 12] and TABEL [9]. The underlying goal, however, is aligned with that of SLOGERT: to create a semantic representation of log file data using a linked data format (RDF) and to complement this data with background domain knowledge to enrich the raw log data. This results in improved

comprehension for analysts in a short time, as well as making the normally unstructured log data more understandable to machines.

The first improvement offered by LEKG [10] involves using named entity recognition (NER) in addition to the more traditional methods based on regular expressions, dictionaries, and other "specialists", as termed in TABEL. In particular, a domain-specific NER model is trained on a text corpus extracted from parsed log files. When a group of log files is being processed, the entities and relations form a local knowledge graph, and then these entities and relations are linked to the global knowledge graph. Finally, rule-based validation is applied to the newly added triples, and any contradictory triples are removed from the global knowledge graph.

Another improvement offered by LEKG is that in addition to generating explicit triples from log files (i.e., triples constructed from data explicitly listed in the log files), LEKG also infers implicit triples from the extracted entities and relations. Here, additional triples are inferred based on log data and background knowledge. For instance, each triple in the local knowledge graphs generated from the raw log files is created using information given explicitly in the raw log files. Then, these local knowledge graph triples are integrated into the global knowledge graph. Now, new implicit triples may be learned on the global knowledge graph through a combination of existing triples and rules [10].

It should be noted that the resulting knowledge graph structure of LEKG differs from TABEL [9] and SLOGERT [11, 12]. Instead of organizing triples around log event identifiers, LEKG aims to model the actual entities within the computer system and their interactions, as well as enriching this data with outside background knowledge. This organization lends itself better to a downstream task of knowledge graph completion, as will be discussed in the following section.

Wang *et al.* [10] gave a root cause analysis as a potential use case for LEKG and give an example of finding what caused an alarm to go off by inferring an implicit triple from the global knowledge graph using a rule based on a combination of triple templates. These rules are manually built, so a user of this system would need to know what sorts of triple combinations to look for. This requires the user to already have extensive knowledge of their system, such as what can go wrong,

and what the root causes of certain issues are. Another limitation of this work is that no metrics are provided to determine the effectiveness of a system. Instead, a simple example is given.

D. Summary

A common limitation of TABEL [9] and SLOGERT [11, 12] is that the structure of the generated knowledge graph is not ideal for KGC as the downstream task. Specifically, assigning IDs to log events results in isolating information from each log line. As an example, a small subset of the AIT dataset KG generated by SLOGERT is shown in Figure 2.1. This figure shows that information in this KG is only loosely connected. Much of the information in the KG is only connected to the log event entity, and these log events are generally only connected by ideas such as both events being from the same log source, or that both log events contain the same IP address or user. Because of this loose connection of information, the data are isolated into "islands" or separated by "hubs", where it takes multiple hops to connect some entities, even if, intuitively, they are directly related, since the central "hubs" (log event IDs) of these data islands must be traversed. The small dataset generated for the figure is constructed from two log events, whose nodes are highlighted in orange.

This type of KG is not ideal for KGC for a few reasons. First, it is difficult to relate triples to each other because of the unique log event identifiers separating the actual entities of the graph. These seem like an unnecessary abstraction for this information, at least considering KGC as the downstream task. Additionally, these unique identifiers vastly increase the number of entities present, while the identifiers themselves should not necessarily be considered entities since they are abstract concepts.

The addition of all these extra entities also makes it nearly impossible to train a KGEM based on the closed world assumption. In particular, in the closed world assumption of knowledge graph completion, the testing and validation sets should include entities from the same pool as the training set [26]. Using unique identifiers for each log event or log line with the closed world assumption, the only log events that could exist in the testing and validation sets are those that exist in the training

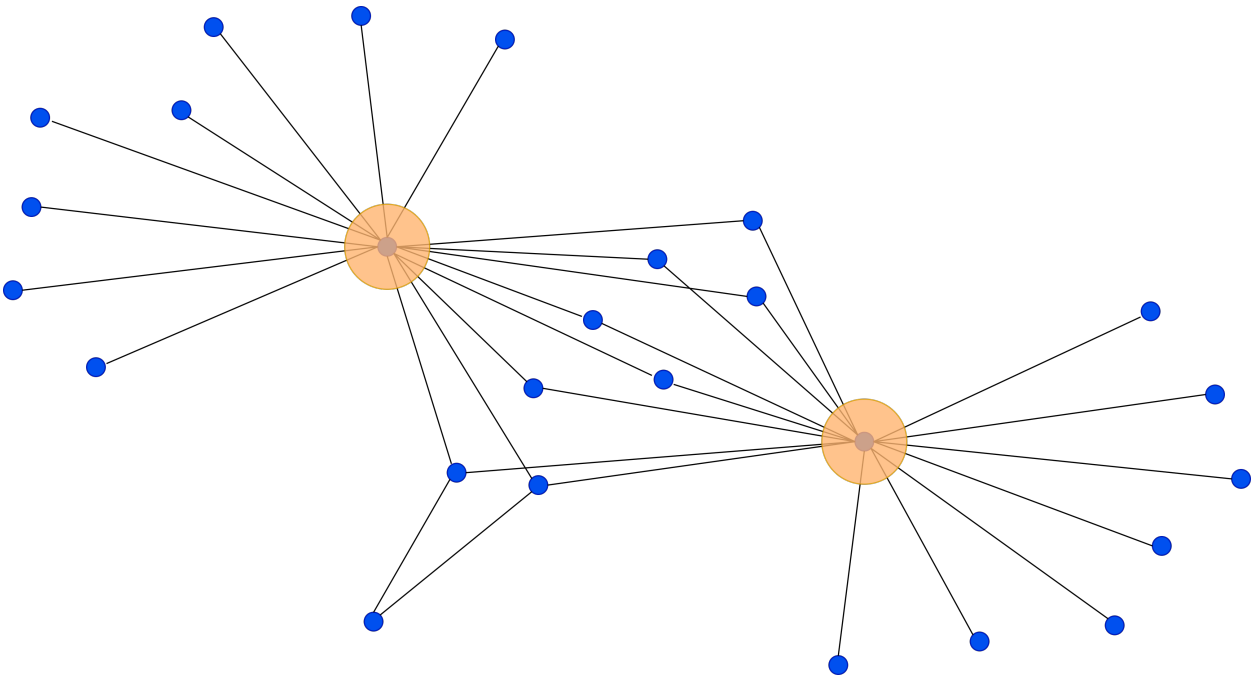

Figure 2.1 Small Subset of the AIT Dataset Knowledge Graph Generated by SLOGERT

set. Clearly, since the testing and validation sets should have different triples than the training set, this KG organization simply does not work for a downstream task of KGC.

While TABEL and SLOGERT may be useful frameworks for manual querying, they are not well-suited for KGC because of the organization of triples around unique log event IDs. A more useful organization is to directly model the actual entities of the computer system and their interactions, which is how the KGs of LEKG [10] are organized. This organization obeys the closed world assumption. The only triples that would have to be discarded in this case are those in the testing and validation sets that contain entities (e.g., users, IP addresses, etc.) or relations that were not seen in the training set, which can be a very small percentage of the overall dataset, depending on how the testing and validation sets are generated. Moreover, this organization does not abstract the structure of the computer system, which vastly decreases the number of entities present on the graph, decreases the distance between entities on the graph, and increases the degree of the average node on the graph.

IV. Log File Anomaly Detection

Lo *et al.* [34] introduced the E-GraphSAGE model, a GNN model for network intrusion detection in the Internet of Things. In [34], network flow data is represented in a graph format, where nodes are network endpoints and edges are flow traffic. While other studies focus on individual flow records, this paper considers the connection between them. E-GraphSAGE uses edge classification to detect malicious network flows.

Similarly, Protogerou *et al.* [35] characterized an IoT network activity as graph data, where nodes are network entities and edges represent message passing over the network. Rather than having one centralized intrusion detection system (IDS), each active node in the network performs anomaly detection considering only itself and its neighboring nodes.

Deng and Hooi [36] proposed the Graph Deviation Network (GDN), which learns the graph data along with the machine learning model. This approach was developed for time series data such as sensor data. Graph nodes are sensors and edges are relationships between sensors. Additionally, graph attention is used to help explain the anomalies that are detected.

Garrido, Dold, and Frank [4] proposed that intrusion detection systems can be improved by integrating machine learning on a domain-specific knowledge graph. The researchers constructed a knowledge graph from several sources of application, system, server, and network logs in an industrial automation system. The authors believe that including knowledge from several sources in a single graph can help add context to cybersecurity observations. Two KGs were created: one contains only information from normal operations of the system, while the other contains both normal and malicious activities. The former is used for training KGC machine learning models to learn from normal operations, while the latter is used for testing whether the model can distinguish between normal and malicious behaviors. This work was evaluated using both the RESCAL [29] KGEM and the proposed energy-based learning method [4, 5]. The knowledge graphs generated in this work will be referred to as the CyberML dataset. The research from [4] focuses on the quality of generated alerts, but it does not evaluate the effectiveness of the KGEMs using standard metrics of KGC such as Hits@k, mean rank, and mean reciprocal rank or standard metrics of classification

such as accuracy and F1-score. The lack of numerical data makes it difficult to accurately determine the effectiveness of models and relate the performance to previous KGC research.

CyberML also presents the idea of having a range of labels for different levels of suspicion for events. The labels correspond to events that have been observed or are expected, unexpected, suspicious, or highly suspicious. Since a KGC model outputs a confidence score that a given triple belongs in a KG, these labels can be mapped to confidence ranges. This means that the output confidence and suspicion of the triple have an inverse relationship (i.e., lower confidence corresponds to higher suspicion, while higher confidence corresponds to lower suspicion). For example, if the set of labels is given as {0: Observed, 1: Expected, 2: Unexpected, 3: Suspicious, 4: Highly Suspicious}, there could be a mapping from confidence ranges to labels given as $\{[0, 0.2) \rightarrow 4,\ [0.2, 0.4) \rightarrow 3,\ [0.4, 0.6) \rightarrow 2,\ [0.6, 0.8) \rightarrow 1,\ [0.8, 1] \rightarrow 0\}$. These confidence ranges could also be thought of as classes, which suggests the idea of using link prediction for classification.

V. Summary

Until recently, log file anomaly detection has been approached primarily with statistical and traditional ML methods. There have been several recent works which characterize network activity as graph data. However, using KGC to detect anomalies has only begun to be explored. There is a wealth of research on KGs and KGC which can be applied to this task. While some methods for log file have been proposed, they have mainly considered querying rather than KGC as the downstream task, and many of them have properties which make it difficult to apply KGC. KGC has recently been proposed for log file anomaly detection, but this area of research needs considerably more attention. For instance, standard KGC metrics need to be introduced to this task for easier comparison moving forward. Finally, additional KGC methods need to be considered for this task, and the effects of adding GNN components to these models should be studied.

CHAPTER 3

METHODOLOGY

This chapter describes in detail the methods used to detect anomalies in log files with knowledge graph completion. First, an overview of the datasets used is provided in Section I. Next, Section II, explains the process for generating knowledge graphs from raw log files. Then, Section III describes the methods and models used for knowledge graph completion. Section IV describes how the output of the knowledge graph completion step is used for binary classification. Then, the metrics used in evaluating the models, both in terms of KGC and classification, are introduced in Section V. Finally, the chapter ends with a summary in Section VI.

I. Datasets

The first step of the anomaly detection pipeline is to collect raw log data. Logs can come from several different sources (e.g., different hosts or different types of logs). It is vital to an anomaly detection system to collect quality training and testing data. All of the logs together should provide a holistic view of network activity. Intuitively, the more context a KGEM has of normal network activity, the greater its ability should be to differentiate normal from anomalous behavior. Thus, all hosts on the network should provide log information from several different log sources. The logs can be system logs, server logs, intrusion detection system (IDS) logs, etc. The log data also needs to be collected separately for training data and testing and validation data, the differences of which will be discussed below.

Two preconstructed datasets are used in this work. One dataset consists of raw log files collected from testbeds simulating both normal user activity and attacks, and the other dataset

consists of log information that has already been transformed into a KG in Turtle format using a KGG system different to that presented in this work. These datasets and their properties will be explored in the following sections.

A. Dataset Generation

The training data should represent the baseline behavior of the network (i.e., it should provide a holistic model of network behavior under normal circumstances). It is the deviation from this baseline behavior that the link prediction step should flag as anomalous. The first step of collecting training data is deciding on all the sources of data in the network: the hosts, users, applications, equipment, etc., and the types of logs they will produce. In addition to the normal system and server logs, other sources such as IDS logs (e.g., from Zeek, suricata, etc.) can be included. Once all log sources have been identified, the network should operate under normal circumstances for a period of time. Dold and Garrido [5] collected data over approximately 50 minutes, which generated 37,000 triples on the resulting knowledge graph, but clearly this time will vary depending on the network and the types of components and interactions it has. In general, it should be run long enough to generate a meaningful view of normal network activity on a typical timescale.

While the training data represents the baseline behavior of the network, where no suspicious activity takes place, the testing data should introduce a variety of suspicious behavior. Testing data can be generated by real network traffic [37, 38] or through synthetically generated traffic or logs [39–42]. While the real traffic approach may generate higher quality data that more accurately reflects the actual network activity, it may be too costly to generate or may interrupt normal operation [41]. The generated suspicious activity should follow established attack patterns [4]. Simulating these established patterns, such as brute force login attempts, DoS attacks, etc., gives an idea of the system's effectiveness during actual attacks. It should be noted that a small portion of normal behavior should be left out of the training set and instead be placed in the testing set. This way, the

testing set will contain both normal and anomalous behavior. The normal behavior can be used to test the model's susceptibility to reporting false positives.

Two datasets were used for training and testing. The first, which will be referred to as CyberML [4, 5], consists of log data that has already been processed into a KG in Turtle format. This dataset does not contain the raw log data used to generate the KG, so it will be used mainly to test the KGC step. The second dataset, the AIT Log Dataset v1.1 [41], consists only of raw log data, so this dataset will be used to test both the KGG and KGC steps. The following sections describe these datasets in greater detail.

B. CyberML

[4] and [5] present the application of KGC to log file anomaly detection. To test their models, the authors created a log file dataset and converted the log data to KG triples. The data were collected from an industrial automation demonstrator system. The training dataset consists of normal network traffic communicating with the Internet, automation devices, and applications, and the testing dataset consists of both normal network traffic and potentially suspicious or malicious behavior. The training dataset was collected over a period of 50 minutes, and about 37441 triples were extracted from the resulting log data, where 4347 entities interact using 38 different relations [5]. The data sources for this dataset include OPC-UA server logs [43] and Zeek connection logs [4], and custom ontologies are used to represent the data from these logs.

C. AIT

The other dataset used in this work is the AIT Log Dataset [41], which is a raw log file dataset generated from several testbeds which simulate user activity on mail servers and online shops. The dataset contains log data from a period of several days. Data from some days contain only benign behavior, which will be used to construct a training knowledge graph, while other days contain a mix of benign behavior and multi-step attacks, which include steps such as port and vulnerability

scans, user enumeration, bruteforce login, uploading a reverse shell, and remote command execution. Labels are automatically applied to each log line generated to distinguish lines that contain benign and malicious behavior.

D. Dataset Extraction

The datasets described above do not come in the organization desired for this work. Therefore, some preprocessing is necessary to fulfill these requirements. The organization required for this work is for a dataset to be divided into a training set, testing set, and validation set, where the training sets contains only normal behavior, and the testing and validation sets contains a mix of normal and malicious behavior.

The original CyberML dataset only contains a training set and testing set, and the testing set is divided into several test cases. In the first step of dataset extraction, the several test cases are combined into one test set. Next, the validation set is constructed by first randomly permuting the training set, keeping a certain percentage of the triples in the testing set, and creating the validation set using the remaining triples. In this case, the split ratio is 0.5 or 50%.

The original AIT dataset does not contain train, test, or validation sets. Instead, it is only divided into normal and malicious behavior based on the labels assigned to each log line. Therefore, each set needs to be extracted. First, the training set is extracted by taking all log lines from days where no malicious activity occurred. Next, the testing set is generated by first retrieving all log lines labeled malicious, then adding several additional log lines containing benign behavior. Finally, the validation set is generated by splitting off a portion of the testing set according to a given ratio in the same way mentioned above. In this case, the split ratio is 0.5 or 50%.

II. Knowledge Graph Generation

Some methods for generating knowledge graphs from log files use a template-based approach [10–12], while others use an ad hoc approach [9]. This work uses the template-based approach.

While this limits the types of logs that can be processed without additional configuration, it allows for more specific knowledge of the formats used. Since each line in a single log file may have different formats, templates are constructed for each type of log line that should be processed. This also works as a sort of implicit filter, since templates are only constructed for the types of log lines that should be processed.

This KGG method, which is depicted in Figure 3.1, consists of several steps. First, templates are extracted from each line of each log file based on user-supplied configuration files. The log files are parsed, and these results are compared to the user-supplied templates. These templates are used along with regular expressions to recognize entities and their types, as well as the relations connecting the entities to construct triples. Once the triples have been added to the KG, each entity can be mapped to an entity ID, and each relation can be mapped to a relation ID. The KG can then be reconstructed using these IDs for data compression. The test and validation sets should also be pruned of any triples containing entities not found in the training set in order to follow the LCWA. The implementation of KGG is described in more detail in the following sections.

A. Log Parsing

Template extraction is implemented using an updated version of the Drain package [44] called Drain3[1]. Drain3 is an updated version of the original program with Python 3 compatibility and additional features, such as masking parameters with regular expressions and the ability to stream log information. This package identifies entities in a log line using pairs of regular expressions and masks.

Drain3 maintains a list of paired regular expressions and masks. For example, the regular expression in the listing below matches an IPv4 address by starting with either a non-alphanumeric character or the beginning of a line, followed by four sets of one to three digits separated by periods, followed by either another non-alphanumeric character or the end of a line. Note that in the actual configuration file containing these expressions, backslashes must themselves be escaped to prevent

[1]https://github.com/logpai/Drain3

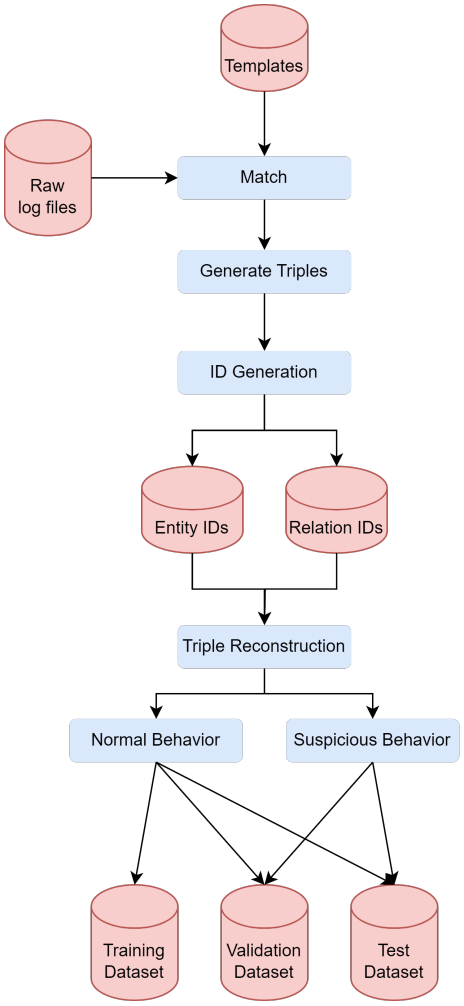

Figure 3.1 Knowledge Graph Generation

them being consumed during regular expression processing. This regular expression is paired with the mask "IP". The mask prefix is specified as "<:", and the mask suffix is specified as ":>". So, when Drain3 encounters a string that matches this regular expression, that string will be replaced by (masked with) "<:IP:>".

```
((?<=[^A-Za-z0-9])|^)(\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})((?=[^A-Za-z0-9])|$)
```

For each log line, Drain3 will mask out any expressions it recognizes and will store two main pieces of information. First is the `template_mined`, which is the log line processed with any recognized strings replaced by their masks. The other piece of information is a list of pairs called

`params`. This stores as pairs each string that was extracted along with its mask. This can be used to identify the type of each entity recognized, which will be discussed below.

B. Entity Extraction

Drain3 works well out of the box, but it can be extended in a few ways. One of the main extensions is providing additional regular expression masking. For example, regular expressions can be added to recognize usernames, MAC addresses, IPv6 addresses, etc. Note that a dictionary-based specialist can also be defined as a regular expression. For instance, the regular expression `(SSH)|(FTP)|(HTTP)` will match a small selection of Internet protocols. An additional consideration to make in creating specialists is that there could be a parent category and several subcategories. For example, it is useful to have a category `NUM` to recognize numbers in log files, but a number could mean several things in different contexts. It is more useful to link numbers to more specific classes, such as `PORT`, `UID`, or `PID`, since this gives more information and context and can generate more useful relations in later steps. However, it is useful to keep the `NUM` category, at least for determining how effective specialists are. For regular expression specialists, subcategories can be told apart in most cases by specifying different (usually positive) lookaheads. For instance, a port number can be told apart from other numbers if the log contains "`port <number>`" or "`port=<number>`". Also note that some natural language processing (NLP) methods such as named entity recognition (NER) can be used for pulling out entities, but it was found unnecessary in this prototypical implementation, where regular expressions sufficed to recognize all desired entities. However, this technique will likely become useful in future work.

A couple examples will follow to illustrate the process of extracting templates and recognizing entities. Take the following log line from an `auth.log` file:

```
Feb 29 00:09:01 mail-0 CRON[32002]: pam_unix(cron:session): session opened for user root
    by (uid=0).
```

The extracted template would be as follows:

```
<:PROCESS:>(cron:session): session opened for user <:USER:> by (uid=<:UID:>)
```

Note that the structured information (up to `CRON[32002]:`) is discarded. Further processing later may utilize this information, but in most cases, the majority of the important information within a log line will be held in the unstructured portion of a log message. There are some exceptions such as Zeek connection logs, which only contain structured information, but these are challenges that can be addressed later or separately. Now, the parameters extracted would be `pam_unix`, recognized as `PROCESS`, `root`, recognized as `USER`, and `0`, recognized as `UID`. Now, take another example from `auth.log`:

```
Mar  3 13:35:36 mail auth: pam_unix(dovecot:auth): authentication failure; logname=
    uid=0 euid=0 tty=dovecot ruser=irwin rhost=127.0.0.1  user=irwin
```

The extracted template would be as follows:

```
<:PROCESS:>(dovecot:auth): authentication failure; logname= uid=<:UID:> euid=<:UID:>
    tty=dovecot ruser=<:USER:> rhost=<:IP:> user=<:USER:>"
```

The extracted parameters would be `pam_unix`, recognized as `PROCESS`, `0`, recognized as `UID`, `irwin`, recognized as `USER`, and `127.0.0.1`, recognized as `IP`.

C. Relation Extraction

Relation extraction is the next step in the KGG process, and it involves searching for connections between each pair of extracted entities. Relation extraction can be performed either by considering each pair of entities and inferring relations between them or through using templates to construct a predetermined set of triples. The former approach may be preferred for very large datasets or for datasets that have no consistent format. While the template approach requires more manual configuration, it also gives more control and accuracy. The template approach is used here, and its implementation is described below.

Templates are stored as JSON objects in a file called `templates.json`, and they specify a template extracted from a log line and a list of triples that should be constructed based on that log line. The listing below shows an example template. In this template, the string following `template_mined` specifies a template extracted by Drain3 to match against. The next item, "`"relations"`", is a list

that specifies triples to be extracted from the template. Here, entities are listed as numbers because they are used to index into the list of parameters called `params` extracted by Drain3. So, in the example, the entity referred to by `0` is the parameter masked out by `PROCESS`. This will be explored further below.

```
{
    "template_mined": "<:PROCESS:>(dovecot:auth): authentication failure; logname=
    uid=<:UID:> euid=<:UID:> tty=dovecot ruser=<:USER:> rhost=<:IP:> user=<:USER:>",
    "relations": [
        {"subject": 0, "relation_label": "has_user", "object": 5},
        {"subject": 5, "relation_label": "has_ip", "object": 4},
        {"subject": 5, "relation_label": "authentication_failure", "object": 0}
    ]
}
```

Drain3 writes its results from parsing the specified log files and extracting templates to a JSON file. Each line specifies a `template_mined` and a list of parameters. Each parameter is a pair where one item is the actual parameter (i.e., the value that was masked out) and the other item is the mask. For instance, one such pair may be `["irwin", "USER"]`. These pairs will always generate a triple that states the types of the parameter, so in this example, `irwin a user` would be generated, where the relation "`a`" means "has the type".

After Drain3 has processed the log files and written results to files, the anomaly detection system will read each of these files. For each line, the `template_mined` from the Drain3 result will be compared with the `template_mined` from `templates.json`. If the templates match, then a triple will be created for each item in the "`relations`" list for that template. Here, the numbers listed for the `subject` and `object` items are used to index into the `params` list for the Drain3 template. For example, take the following example log line and extracted template from the previous section:

```
Mar  3 13:35:36 mail auth: pam_unix(dovecot:auth): authentication failure; logname=
    uid=0 euid=0 tty=dovecot ruser=irwin rhost=127.0.0.1  user=irwin
<:PROCESS:>(dovecot:auth): authentication failure; logname= uid=<:UID:> euid=<:UID:>
    tty=dovecot ruser=<:USER:> rhost=<:IP:> user=<:USER:>"
```

Here, the `params` list would be

```
[["pam_unix", "PROCESS"], ["0", "UID"], ["0", "UID"], ["irwin", "USER"], ["127.0.0.1",
    "IP"], ["irwin", "USER"]]
```

Using the list of relations given above, with the subject and object numbers used to index into the `params` list and retrieve the first element, the following triples could be generated:

```
pam_unix    has_user    irwin
irwin   has_ip  127.0.0.1
irwin   authentication_failure  pam_unix
```

Additionally, each pair in the the `params` list can be used to create additional triples that tell the type of each entity:

```
pam_unix    a   process
0   a   uid
irwin   a   user
127.0.0.1   a   ip_address
```

Note that each mask used in template extraction is mapped to a type. For example, the mask `IP` is mapped to the type `ip_address`. It follows that these simple type mappings can be replaced with mappings to types from appropriate ontologies. Additionally, the simple tab-separated value format used for storing triples can be replaced by, for example, the Turtle format for storing RDF triples. This would allow linking external data to the local knowledge graph. While these techniques would be useful, they are not explored here and are left for future work.

D. ID Generation and Triple Reconstruction

Once the KG has been constructed, another useful technique is to generate IDs for each entity and relation and reconstruct each triple set using these IDs. This can greatly reduce the size of the KG and in some cases can lead to improved performance. This section will give an overview of the ID generation and KG reconstruction process.

The ID generation process starts with empty mappings for entities and relations, and the entity ID and relation ID both start at 0. For each triple in the KG, the entities and relation are considered. If any of them do not exist in the mapping yet, they are given the current entity or relation ID, then that ID is incremented. This process is shown in Algorithm 1. For convenience, the resulting ID mappings are saved to files. Note when IDs are generated, only the training triple set is

Algorithm 1 ID Generation

Input: training knowledge graph $\mathcal{K}_{train}$

ent_ids ←empty dictionary
rel_ids ←empty dictionary
ent_count ←0
rel_count ←0
for each triple (s, r, o) in $\mathcal{K}_{train}$ **do**
    if s not in ent_ids **then**
        ent_ids[s] ←ent_count
        ent_count ←ent_count + 1
    end if
    if r not in rel_ids **then**
        rel_ids[r] ←rel_count
        rel_count ←rel_count + 1
    end if
    if o not in ent_ids **then**
        ent_ids[o] ←ent_count
        ent_count ←ent_count + 1
    end if
end for

Output: ent_ids and rel_ids

considered. This is because the training set represents the knowledge graph that will be learned by the model, and it represents the local closed world. Therefore, including entities that may be in the testing set but not the training set would violate the LCWA.

Once the IDs have been generated, they can be used to reconstruct the KG by replacing each entity or relation with its corresponding ID. This will create a new triple set. Note that while the ID generation step is only applied to the training set, this reconstruction step applies to the training, testing, and validation sets. For each triple in each set, if an entity or relation does not have an ID mapping, the triple is discarded to obey the LCWA. Otherwise, each element of the triple is replaced by its corresponding ID, a new triple is constructed using these IDs, and the new triple is added to the new triple set. This process is shown is Algorithm 2, and an example walking through the KGG process is shown in Figure 3.2.

Algorithm 2 KG Reconstruction

Input: triple set $\mathcal{T}_{in}$, ID mappings ent_ids and rel_ids

$\mathcal{T}_{out} = \emptyset$
for each triple $(s, r, o) \in \mathcal{T}_{in}$ **do**
  if $s \notin$ ent_ids **or** $r \notin$ rel_ids **or** $o \notin$ ent_ids **then**
    continue
  end if
  $s_{out} \leftarrow$ent_ids[s]
  $r_{out} \leftarrow$rel_ids[r]
  $o_{out} \leftarrow$ent_ids[o]
  $\mathcal{T}_{out} \cup \{(s_{out}, r_{out}, o_{out})\}$
end for

Output: triple set $\mathcal{T}_{out}$

III. Knowledge Graph Completion

Knowledge graph completion, or link prediction, is the final step of this anomaly detection pipeline. For each triple evaluated, this step will output the confidence that the triple belongs in the knowledge graph based on the training dataset. In the context of intrusion detection, this is the confidence that the behavior modeled by the triple is normal. Here, a high confidence corresponds to normal behavior, while a low confidence corresponds to anomalous behavior. To increase understanding, [4] and [5] also used labels such as Observed, Expected, and Suspicious.

A. Models

Any KGEM can be used here; several such models were discussed in Chapter 2. For this study, DistMult [30] and ComplEx [31] are applied. These are shallow models and can be thought of as having only one layer, which is a scoring function, between the input and output. These models are also used as a base to construct other KGEMs, each with three layers. For these extended models, the first two layers are graph neural network (GNN) layers, and the third layer is the original KGEM (the scoring function). The GNN layers can be either graph convolutional network (GCN) layers or gated graph neural network (GGNN) layers. Thus, constructed a total of six models were constructed, including the original KGEMs and two variants each: DistMult, GCNDistMult,

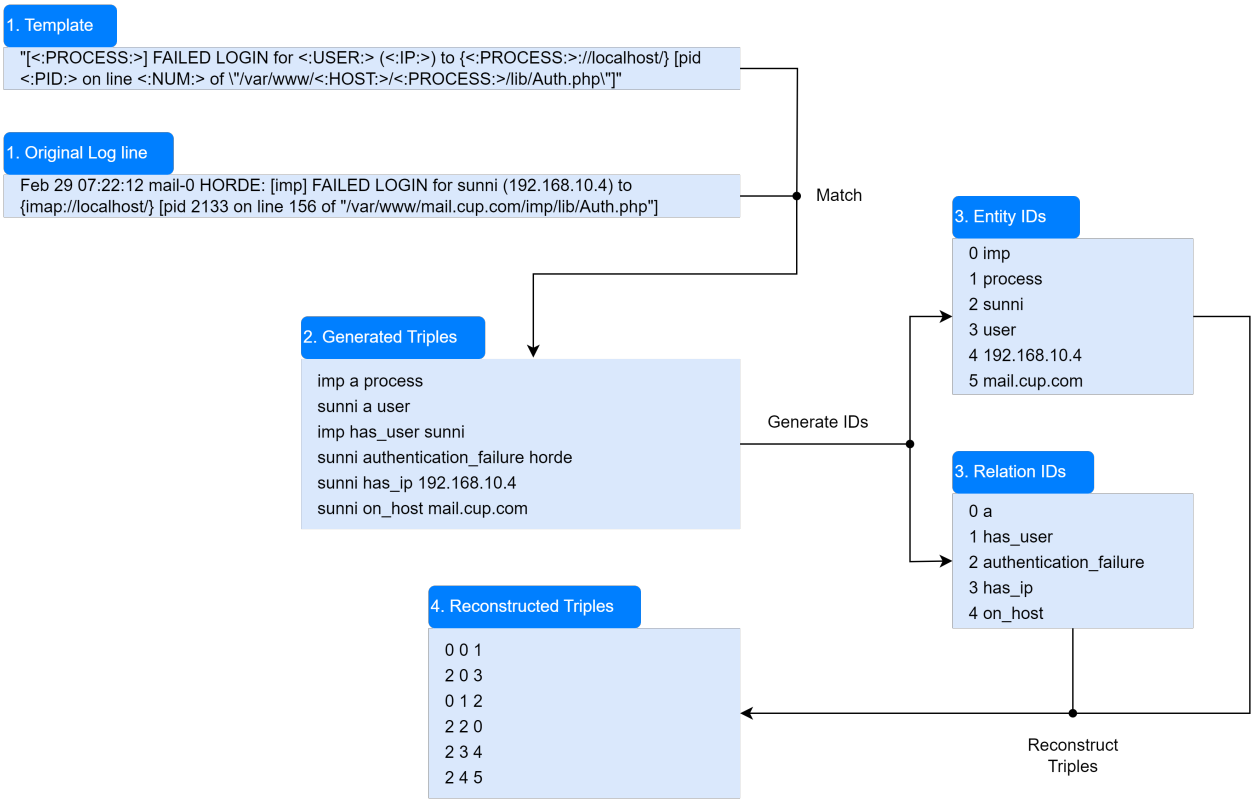

Figure 3.2 Knowledge Graph Generation Example

GGNNDistMult, ComplEx, GCNComplEx, and GGNNComplEx. Here, the GNN layers act as an alternative method to embed graph information and are anticipated to better preserve graph properties such as information about neighboring nodes. The implementation for each model is based on the Graph4NLP[2] library. The KGEM architecture is shown in Figure 3.3.

IV. Classification Using Link Prediction

When a triple is passed through the KGEM, the output of the link prediction step is a confidence score between 0 and 1 that represents the likelihood that the triple belongs in the training knowledge graph (i.e., the behavior represented by the triple aligns with normal behavior within the context of the computing environment). Certain ranges of confidence can be mapped to labels, which enables classification through link prediction. Binary classification is used in this study.

[2]https://github.com/graph4ai/graph4nlp/tree/master/examples/pytorch/kg_completion

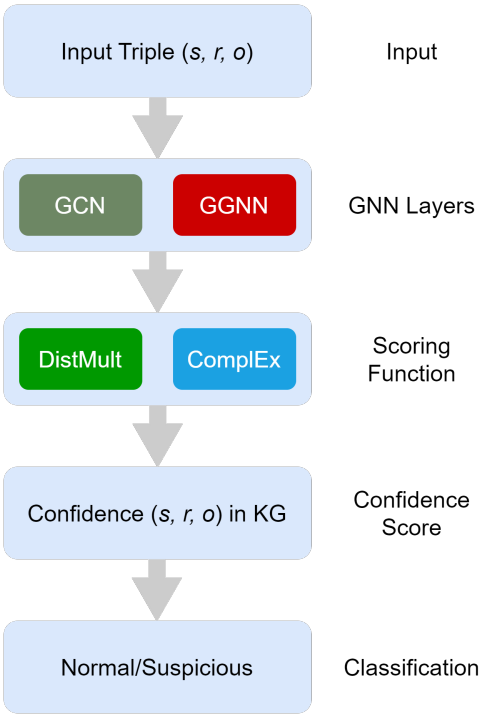

Figure 3.3 Knowledge Graph Embedding Model Architecture

Here, the confidence scores can be divided into two ranges: $[0, 0.5)$ and $[0.5, 1]$. Recall that a low confidence score corresponds to suspicious behavior, while a high confidence score corresponds to normal behavior. However, it is conventional for anomalous behavior to have the positive label. Therefore, the top range can map to the label 0 or "normal", and the bottom range can map to the label 1 or "suspicious". This creates the mapping $[0, 0.5) \rightarrow 1$, $[0.5, 1] \rightarrow 0$. For example, if a model calculates a confidence score of 0.2, it would label the triple as 1 or "suspicious", and if the model calculates a confidence score of 0.7, it would label the triple as 0 or "normal". This idea can be expanded beyond binary classification into multi-class classification by dividing the range of possible confidence values into smaller sections. This could provide more granularity to labels, such as labeling data as "highly suspicious", "suspicious", "unexpected", or "normal".

A major advantage of the KGC approach to classification is the granularity of triples compared to log lines. Since each log line is composed of multiple triples, each triple is evaluated and labeled separately. Note that even a suspcious log line may not contain only suspicious triples; there may

only be one or two triples that make the entire log line suspicious, while the rest of the triples that make up the log line may contain normal behavior. This allows per-triple labels to be more precise than per-line labels and gives KGC-based classification the potential to pinpoint the suspicious elements of a log line. On the other hand, this advantage makes the labeling process difficult, both in terms of volume and complexity. By definition of the problem, there are many more triples than log lines, which would already make the labeling process more time consuming. Additionally, it would be more difficult to assign rules to determine which triples should be labeled as suspicious and which triples should be labeled as normal.

The CyberML dataset comes with per-triple labels. The authors assign one of five possible labels to each triple, which represent different levels of suspicion: “highly suspicious”, “suspicious”, “unexpected”, “expected”, and “observed”. For the purposes of this study, these five labels were mapped into the binary labels “suspicious” and “normal”. Since the AIT dataset consists only of raw log data, only the log lines are labeled, and no per-triple labels are provided. Therefore, classification was performed only on the CyberML dataset, and only KGC metrics are reported for model performance on the AIT dataset.

V. Evaluation Metrics

Each model’s performance can be evaluated using a range of metrics from both KGC and classificaiton. these metrics are introduced in this section. First, the KGC metrics are explained before defining the metrics used for evaluating classification performance.

Given an entity and relation pair, link prediction calculates the likelihood of every entity in the set of all entities being the missing value. In other words, given (x, r, o) or (s, r, x), the link predictor calculates the probability that each entity could fill in for x. Since the training dataset only contains facts and does not contain any false statements, false statements or corruptions must be generated, where each subject or object entity is replaced by each other entity in the set. Then, link prediction tests the model’s ability to tell true statements from corruptions.

Three main metrics are used to describe the performance of the link prediction task: Hits@k, mean rank (MR), and mean reciprocal rank (MRR). Each of these metrics focuses on the rank of the true triple compared to its corruptions (e.g., if the link prediction results in 4 corruptions having a higher probability than the true triple, the true triple's rank is 5).

In the following equations, $\mathcal{K}$ is the knowledge graph, and $rank(t)$ is the rank of a triple. MR is the average rank of all test triples and is given in [27] by

$$\text{MR} = \frac{1}{|\mathcal{K}_{test}|} \sum_{t \in \mathcal{K}_{test}} rank(t). \tag{3.1}$$

Since $rank(t) = 1$ is desired, lower values of MR are better. MRR is the average of the reciprocals of test triple ranks and returns a value between 0 and 1, where values closer to 1 are better. It is given in [27] by

$$\text{MRR} = \frac{1}{|\mathcal{K}_{test}|} \sum_{t \in \mathcal{K}_{test}} \frac{1}{rank(t)}. \tag{3.2}$$

Finally, Hits@k gives the ratio of triples that appear in the top k ranks, and it is given in [27] by

$$\text{Hits@}k = \frac{|\{t \in \mathcal{K}_{test} \mid rank(t) \leq k\}|}{|\mathcal{K}_{test}|}. \tag{3.3}$$

Hits@k gives a value between 0 and 1, where values closer to 1 are better.

Left and right variants of each metric can also be defined based on the prediction task. For example, left Hits@k is the Hits@k performance for predicting missing subject entities (i.e., the task of predicting $(?, r, o)$), while right Hits@k is the Hits@k performance for predicting missing object entities (i.e., the task of predicting $(s, r, ?)$). When using these left and right variants, Hits@k can be defined as the arithmetic mean of the left and right variants. This work refers to this value as simply Hits@k. The other metrics defined above can be defined in the same way.

Classification metrics measure how often the model guesses the correct label of a given data item, and they are most understandable in the case of binary classification, which is used here. Labels in a binary classification problem can be positive or negative. For example, in this case, the

label “normal” is the negative case, while the label “suspicious” is the positive case. Classification metrics are based on the concepts of true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN). A true positive occurs when the model correctly guesses a positive label, and a false positive occurs when the model guesses a positive label, but the true label is negative. Similarly, a true negative occurs when the model correctly guesses that the label is negative, while a false negative occurs when the model guesses a negative label, but the true label is positive. Note that all of the metrics below are numbers between 0 and 1.

Accuracy is the most recognizable of all classification metrics, and it is defined by the number of correct guesses (true positives and true negatives) divided by all guesses. It is defined by

$$\text{Accuracy} = \frac{TN + TP}{TP + FP + TN + FN}. \tag{3.4}$$

Precision measures how many of the positive label predictions are correct. In other words, it is the ratio of true positives to both true and false positives. Precision is defined by

$$\text{Precision} = \frac{TP}{TP + FP}. \tag{3.5}$$

Recall, which is also known as sensitivy or true positive rate, measures how many positive guesses were correct out of all positive labels, or the ratio of true positives to true positives and false negatives. It is defined by

$$\text{Recall} = \frac{TP}{TP + FN}. \tag{3.6}$$

F1-Score is the harmonic mean of precision and recall. Its purpose is to summarize both metrics of precision and recall into one number. It is defined by

$$\text{F1-Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}. \tag{3.7}$$

True positive rate (TPR) is identical to recall and is the ratio of true positives to all positive labels. It is defined by

$$\text{TPR} = \frac{TP}{TP + FN}. \tag{3.8}$$

False positive rate (FPR) is the ratio of false positives to all negative labels. It is defined by

$$\text{FPR} = \frac{FP}{FP + TN}. \tag{3.9}$$

True negative rate (TNR) is the ratio of true negatives to all negative labels. It is defined by

$$\text{TNR} = \frac{TN}{TN + FP}. \tag{3.10}$$

Finally, false negative rate (FNR) is the ratio of false negatives to all positive labels. It is defined by

$$\text{FNR} = \frac{FN}{FN + TP}. \tag{3.11}$$

Note that in the case of anomaly detection, the false negative rate can be more damaging than the false positive rate. False positives in anomaly detection may lead to annoying alerts for cybersecurity staff, and enough may lead them to disregard ADS alerts in general. However, false negatives can be more damaging since they imply that some undetected malicious activity is occurring or has occurred.

VI. Summary

This chapter described the methods used to detect anomalies in log files using knowledge graph completion. First, a set of log files is collected and a dataset is extracted from them. The training dataset contains only normal behavior of the system, while the testing set should contain a mix of normal and suspicious or malicious behavior. These datasets of raw log data are then used to construct a training knowledge graph and testing and validation triple sets. The approach presented

here is template-based, where a template is defined for each type of log line to be included in the dataset. The templates define where in the log line entities appear and what types they are, as well as the relations to be drawn between the entities. An optional step is to map the entities and relations to those from a set of ontologies, but that is not performed in this prototypical implementation. To complete the knowledge graph generation step, each entity is mapped to an entity ID and each relation to a relation ID, and each triple set is reconstructed using these IDs rather than the string values. Next, a KGEM is used to learn the structure of the training knowledge graph. Any existing model can be used in this step, including the ones discussed in Chapter 2. The DistMult and ComplEx models are used here, and several variants are made of each by placing two GNN layers before the scoring function layer. Link prediction can be performed on new triples, and the output confidence represents the likelihood that the triple represents normal behavior in the system. Finally, this confidence score can be translated to a label for classification (in this case, binary labels of "normal" or "suspicious"). The effectiveness of each model is evaluated based on a range of both KGC and classification metrics, and the results of this evaluation will be given in the following chapter.

CHAPTER 4

RESULTS

This chapter presents an evaluation of the proposed methods that employ the KGC task for detecting anomalies in log files. First, Section I describes the setup for the experiments performed. Then, the experimental findings are presented in Section II.

I. Experiment Setup

The previous chapter describes the methods used to perform KGG from log files and various KGEMs models. The size of each preprocessed dataset is described in Table 4.1. Note that while the AIT dataset contains more triples overall, the CyberML dataset is more complex in terms of the number of entities and relations. Now, the most effective versions of the models need to be found through training and hyperparameter tuning. Several hyperparameters are considered, and several values are tested for each one. One of the hyperparameters are specific to the GNN portions of the models, while others are only concerned with the scoring function portions of the models.

Table 4.1 Dataset Sizes

	AIT	**CyberML**
Entities	79	4,346
Relations	7	37
Training Triples	98,502	37,519
Validation Triples	2,636	683
Testing Triples	2,636	684
Total Triples	103,704	38,886

The GNN-specific hyperparameter is the direction option, which specifies what type of graph to use for node embeddings. This hyperparameter has three options, one for an undirected graph, and the other two for a bidirectional graph. The two bidirectional options are referred to as bi-sep and bi-fuse. The bi-sep option specifies that node embeddings should remain separate, while the bi-fuse option specifies that node embeddings should be combined in both directions.

The models in general have hyperparameters, some of which were left constant in this experiment, while some varied within a range. The batch size is left at a constant of 128, and the number of epochs is kept constant at 200. Note that the best results were usually attained before reaching this epoch before starting to deteriorate, and the best results are the ones reported here. Additionally, the learning rate is kept at a constant initial value of 0.0005. The Adam optimizer is used for each model, which adapts the learning rate throughout the training process. Each model evaluated in this work is optimized based on binary cross entropy loss. Additionally, the embedding dimension for entity and relation vectors is kept at a constant 200.

The models can also make use of various regularization techniques in both the GNN and scoring function layers to reduce potential overfitting. The three primary forms of regularization are L2 regularization, input dropout, and label smoothing. L2 regularization effectively removes some number of weights on each iteration to make the model simpler. Similarly, input dropout removes a portion of the model's inputs. Label smoothing introduces noise to the ground-truth labels in order to decrease overconfidence and improve generalization.

Hyperparameters were tuned using a grid search over certain hyperparameters. In particular, L2 regularization uses the range $[0.0, 0.1]$, label smoothing uses the range $[0.0, 0.1]$, and input dropout uses the range $[0.0, 0.5]$. For each model that is trained, these hyperparameters were set by sampling a value from the corresponding range. The direction option hyperparameter is also tested on each of its three possible values: undirected, bi-sep, and bi-fuse. Note that this hyperparameter only applies to the models that include GNN layers. For completeness, Table 4.2 shows the hyperparameter values that are held constant between all the models, and Table 4.3 shows the direction option hyperparameter values corresponding to each best-performing GNN-augmented

Table 4.2 Constant Hyperparameter Values

Hyperparameter	**Value**
Epochs	200
Batch Size	128
Learning Rate	0.0005
Embedding Dimension	200
L2	0.0
Input Dropout	0.2
Label Smoothing	0.1

Table 4.3 Direction Option Hyperparameter Values for Each Dataset

Model	**AIT**	**CyberML**
GCNDistMult	undirected	bi-sep
GGNNDistMult	bi-sep	bi-sep
GCNComplEx	undirected	undirected
GGNNComplEx	bi-sep	bi-fuse

DistMult and ComplEx model variant. Recall that the direction option only applies to the GNN layers, so this hyperparameter is meaningless to the base DistMult and ComplEx models. The best models are selected based on the highest MRR value. Notice that the bi-fuse option only shows the best performance in one model, while the bi-sep and undirected options have nearly equal representation. Indeed, the performance difference between the latter two options is often miniscule.

II. Findings

The experimental results are shown in the following tables. Each model is evaluated using left, right, and average Hits@1, Hits@10, MR, and MRR. Results are rounded to four significant digits, and the best result for each metric and model is in bold. Table 4.4 summarizes the link prediction performance of each of the DistMult model variants on the AIT dataset, and Table 4.5 summarizes the link prediction performance of each ComplEx model variant on the AIT dataset.

Table 4.4 DistMult Model Link Prediction Performance on the AIT Dataset

Metric	DistMult	GCNDistMult	GGNNDistMult
Left Hits@1	0.2353	**0.3579**	**0.3579**
Right Hits@1	0.4361	**0.4827**	0.4793
Hits@1	0.3357	**0.4203**	0.4186
Left Hits@10	**0.6740**	0.6558	0.6531
Right Hits@10	0.9002	**0.9028**	0.9006
Hits@10	**0.7871**	0.7793	0.7769
Left MR	**13.07**	14.66	14.06
Right MR	4.545	4.493	**4.240**
MR	**8.802**	9.576	9.150
Left MRR	0.4077	**0.4736**	0.4573
Right MRR	0.5574	**0.6018**	0.5985
MRR	0.4825	**0.5377**	0.5279

Likewise, Tables 4.6 and 4.7 summarize the performance of each DistMult and ComplEx model, respectively, on the CyberML dataset. Tables 4.8 and 4.9 show the classification performance of the DistMult and ComplEx models on the CyberML dataset.

First note the classification performance on the CyberML dataset, which is high overall. The largest noticeable difference is between the DistMult and ComplEx models in general. The highest accuracy achieved in a DistMult model is 77.27% (GCNDistMult and GGNNDistMult), while all ComplEx models achieved 93.18% accuracy, a 15.91% improvement. For the DistMult models, the GNN variants show noticeably better performance, with both variants improving the accuracy from 72.73% to 77.27%, a 4.54% improvement. However, the ComplEx models exhibit identical classification performance between the base model and its GNN variants.

The F1-Score, precision, and recall metrics follow the same general patterns as the accuracy. The largest difference is between the DistMult and ComplEx models. The highest suspicious label F1-Score for DistMult models is 0.5455 (GCNDistMult and GGNNDistMult), while all CompleX models have a suspicious label F1-Score of 0.80000, a 25.45% improvement. Similarly, the highest normal label F1-Score for DistMult models is 0.8485, while all ComplEx models have a normal

Table 4.5 ComplEx Model Link Prediction Performance on the AIT Dataset

Metric	ComplEx	GCNComplEx	GGNNComplEx
Left Hits@1	**0.6455**	0.6281	0.6281
Right Hits@1	**0.4641**	**0.4641**	0.4315
Hits@1	**0.5548**	0.5461	0.5298
Left Hits@10	**0.9074**	**0.9074**	0.9066
Right Hits@10	0.8190	**0.9059**	0.8231
Hits@10	0.8632	**0.9066**	0.8649
Left MR	3.694	**3.677**	3.744
Right MR	4.607	**4.179**	4.549
MR	4.150	**3.928**	4.146
Left MRR	**0.6969**	0.6900	0.6872
Right MRR	**0.5854**	0.5778	0.5746
MRR	**0.6411**	0.6339	0.6309

Table 4.6 DistMult Model Link Prediction Performance on the CyberML Dataset

Metric	DistMult	GCNDistMult	GGNNDistMult
Left Hits@1	0.8041	0.8085	**0.8319**
Right Hits@1	0.8026	0.7939	**0.8027**
Hits@1	0.8034	0.8012	**0.8173**
Left Hits@10	0.9474	0.9751	**0.9737**
Right Hits@10	**0.8158**	0.8129	**0.8158**
Hits@10	0.8816	0.8940	**0.8947**
Left MR	30.03	10.56	**10.20**
Right MR	173.2	**119.1**	122.6
MR	101.6	**64.80**	66.40
Left MRR	0.8496	0.8573	**0.8710**
Right MRR	0.8076	0.8029	**0.8083**
MRR	0.8286	0.8301	**0.8396**

Table 4.7 ComplEx Model Link Prediction Performance on the CyberML Dataset

Metric	ComplEx	GCNComplEx	GGNNComplEx
Left Hits@1	**0.8904**	0.8860	0.8787
Right Hits@1	0.8041	**0.8070**	0.8056
Hits@1	**0.8472**	0.8465	0.8421
Left Hits@10	0.9532	**0.9576**	0.9488
Right Hits@10	**0.8173**	0.8158	0.8158
Hits@10	0.8852	**0.8867**	0.8823
Left MR	**9.475**	31.55	23.78
Right MR	**123.1**	128.4	133.9
MR	**67.26**	79.97	78.83
Left MRR	**0.9104**	0.9071	0.8987
Right MRR	0.8096	**0.8111**	0.8104
MRR	**0.8600**	0.8591	0.8545

Table 4.8 DistMult Model Classification Performance on the CyberML Dataset

Metric	DistMult	GCNDistMult	GGNNDistMult
Accuracy	0.7273	**0.7727**	**0.7727**
F1-Score	0.5000	**0.5455**	**0.5455**
Precision	0.3333	**0.3750**	**0.3750**
Recall	**1.0000**	**1.0000**	**1.0000**
TPR	**1.0000**	**1.0000**	**1.0000**
FPR	0.3158	**0.2632**	**0.2632**
TNR	0.6842	**0.7368**	**0.7368**
FNR	**0.0000**	**0.0000**	**0.0000**

Table 4.9 ComplEx Model Classification Performance on the CyberML Dataset

Metric	ComplEx	GCNComplEx	GGNNComplEx
Accuracy	**0.9318**	**0.9318**	**0.9318**
F1-Score	**0.8000**	**0.8000**	**0.8000**
Precision	**0.6667**	**0.6667**	**0.6667**
Recall	**1.0000**	**1.0000**	**1.0000**
TPR	**1.0000**	**1.0000**	**1.0000**
FPR	**0.0789**	**0.0789**	**0.0789**
TNR	**0.9211**	**0.9211**	**0.9211**
FNR	**0.0000**	**0.0000**	**0.0000**

label F1-Score of 0.9589, an 11.04% improvement. The GNN DistMult model variants again exhibit noticeably better performance than the base versions, but there is no performance difference between the GCN and GGNN variants. The base DistMult model has a suspicious label F1-Score of 0.5000, while the GNN variants score 0.5455, a 4.55% difference. Likewise, the base DistMult model has a normal label F1-Score of 0.8125, while the GNN variants score 0.8585, a 3.6% improvement. Additionally, the performance of DistMult models is much greater for normal labels than suspicious labels. This is an expected result since each model is trained to learn normal behavior. Again, the base and GNN ComplEx model variants showed identical classification performance.

For the KGC evaluation, first notice that GNN models generally show better or similar performance in all metrics compared to the base models. This difference is noticeable in most cases, and in some cases, such as the difference between the Hits@10 performance of the base DistMult model and that of its GNN variants on the AIT dataset, the performance difference is significant. This suggests that including GNN layers to encode graph information into a KG embedding model is worthwhile and should continue to be explored in future research.

While GCN model variants appear to have better performance than GNN model variants, the difference in performance between GCN and GGNN models is quite small, suggesting that while encoding graph information into the KGEM tends to lead to better performance, the manner of encoding the information makes little difference. However, other types of GNNs such as graph attention networks [45] and GraphSAGE [46] should still be investigated in the future.

The ComplEx models generally show better performance across both datasets compared to the DistMult models. This is an expected result due to the ability of ComplEx to model anti-symmetric relations and its complex-space embeddings. Indeed, these datasets contain mostly anti-symmetric relations. For example, most relations in each dataset list properties of the various entities, such as `has_ip`. Only some relations, such as `sent_message` could connect two entities from both directions, and this is not necessary. For instance, if `user1` sent a message to `user2`, it does not necessarily mean that `user2` sent a message to `user1`.

Performance in terms of Hits@k and MRR on the CyberML dataset is much higher overall compared to the AIT dataset. This is especially apparent when comparing the Hits@1 performance of the DistMult models in the AIT and CyberML datasets. This is a surprising result considering CyberML has a greater variety of entities and relations, making it a more complicated dataset. Note that the performance in terms of MR looks much better on the AIT dataset. However, this metric is not normalized like all the others. Thus, since CyberML has a much larger amount of entities, the MR scores will naturally be higher. However, the performance in terms of the normalized metrics is consistently better on the CyberML dataset.

Note that with the exception of the DistMult models on the AIT dataset, the left metrics are almost universally higher than the right metrics, and often by a significant margin. This implies that most of the models predict missing subject entities better than missing object entities. That is, the models are better at answering $(?, r, o)$ than $(s, r, ?)$. This is likely due to the trend that most object entities in these datasets are qualities of the subject entities, and it might be easier to predict a subject given a quality than it is to predict a quality given a subject.

The results on the CyberML dataset show that classification performance is positively correlated with KGC performance. This implies that employing more effective KGC metrics will lead to improved classification performance. A link between these metrics makes sense because the KGC task is directly transformed into the classification task by mapping the confidence score output from the KGC task onto a classification label.

Based on the results shown and discussed in this chapter, the approach of applying the KG link prediction task to anomaly detection is promising. There is a noticeable improvement in performance when adding GNN layers to the DistMult model, but the improvement is more substantial when moving from a DistMult model to a ComplEx model. This implies that the effectiveness of the base KGEM may be a more important factor for classification performance. Even without considering methods such as using RDF format for KGs and linking to external data, the results are strong, especially for the ComplEx models on the CyberML dataset. Future improvements to both KGG and link prediction methods will further increase the feasibility of using link prediction for anomaly detection.

CHAPTER 5

DISCUSSION AND CONCLUSION

This chapter discusses the results of the study and give a conclusion. Section I reiterates the objectives of this study. Next, the limitations of the study are explained in Section II, and Section III summarizes this study's findings and relate them to the objectives. Section IV lists several potential areas of future study on this topic, and this thesis is concluded in Section V.

I. Objectives of the Study

There is a wealth of research on knowledge graph completion using traditional and GNN models, as well as generating knowledge graphs from log files and other resources. The focus of log file KGG research is on querying, and current methods are not suited for KG completion. Using link prediction for anomaly detection is a relatively new topic, and existing work only focuses on the link prediction part and not the KGG part. Additionally, performance was not measured with standard KGC or classification metrics.

The objectives of the study are to combine these ideas and present a full end-to-end anomaly detection system, from generating a KG from raw log files to training KGEMs, some of which are augmented with GNN components, and performing the link prediction task to detect anomalies, as well as measuring and comparing the performance of the resulting KGEMs on both the dataset generated in this study (AIT) and a previously generated dataset (CyberML). The results are used to determine the feasibility of this approach and determine whether it is worthwhile to include GNN component in the KGEMs.

II. Limitations of the Study

Using template-based KGG introduces some inherent limitations to the study. First, templates cannot easily handle variable-length lists that occur in logs (e.g., a log line describing a user sending a message to any number of other users). There are workarounds for this limitation. For instance, if an upper limit of the list's length is known, separate templates can be constructed for each case. This is the approach used here. However, this method can quickly become cumbersome. Templates also have difficulty in the case of highly unstructured messages, such as log lines that display HTTP requests.

Additionally, understandability of results is not a focus of this study. That is, the only output of the models is a level of confidence that the fact belongs in the KG. The models do not justify this probability or attempt to classify the type of suspicious/malicious behavior. Other works, such as xNIDS [47], focus primarily on the problem of understanding intrusion detection system results.

Finally, the proposed system is not meant to process new log data streamed in real time. Instead, this study lays the groundwork for future systems to expand on the features provided here.

III. Summary of the Findings

This study has accomplished its objectives. A prototypical end-to-end anomaly detection system was successfully built, including modules to generate a KG from a set of log files and to train KGEMs on this KG in order to use the link prediction task to detect anomalies in new data. Both traditional KGEMs and models using GNN layers were tested. The models were trained on both the AIT dataset generated by this work as well as the CyberML dataset from [4, 5].

The results presented in the previous chapter demonstrate that the models that include GNN layers generally have better performance than the traditional models, suggesting that it is worthwhile to include GNN layers before the scoring function in these models. The performance levels between the two types of GNNs are generally very close across all metrics. However, a larger difference in performance is seen when moving to more effective base KG embedding models. The ComplEx

models perform better overall across both datasets compared to the DistMult models. The results also show that, surprisingly, the models perform better on the more complicated CyberML dataset. Generally, the models also perform better in predicting missing subject entities than object entities, which is likely due to most object entities being properties of the subject entities. Additionally, classification metrics show that the models have better performance for normal labels than suspicious labels, which is expected since they are trained on only normal behavior.

IV. Recommendations for Further Study

There are several areas of future work to be explored within this topic, both within the KGG and KG completion components. Template-based generation can work well for semi-structured log messages since they give complete control what triples will be extracted from a given message. However, it can be cumbersome to create templates for very large datasets, and they cannot be applied as effectively to messages that contain highly unstructured data, such as HTTP requests, which can still contain valuable information. Thus, one area of future work within KGG for this task is to use machine learning methods for entity and relation extraction to augment or replace template-based generation for these more difficult tasks. Another area for future work is to link entities and relations to ontologies and output the final KG using RDF format. This will also allow linking to external data, such as data extracted from vulnerability databases or cybersecurity reports or blogs. Several other considerations, such as entity linking, named entity recognition, etc. may be applied to improve the KGG process.

Future work within the KGC component of this topic may include the use of several different types of KGEMs. Just as the properties of the ComplEx models, such as the ability to represent anti-symmetric relations, resulted in improved performance compared to the DistMult models, other newer models (many of which are listed in [48]) may similarly have properties beneficial to cybersecurity-related data and its structure. These models may also be tested when augmented with GNN component, as the results from this study suggest that using GNN layers to encode graph

information can lead to significant performance improvements. Additionally, different types of GNNs, such as GAT and GraphSAGE, can be tested in conjunction with additional KGEMs.

It would be helpful to develop an effective per-triple labeling strategy in order to apply metrics such as accuracy. One strategy would be to simply label all triples from a suspicious log line as suspicious. However, this could greatly overestimate the number of suspicious triples and skew the results significantly. The strategy should also be at least mostly automatic, since a knowledge graph used in a production environment could contain hundreds of thousands or millions of triples, and validation and test set sizes will need to grow with the training KG in order to be effective.

One promising direction that requires new methods in both components is the use of temporal knowledge graphs as described in [49]. Instead of representing facts using triples, temporal KGs use quadruples where the additional component is a timestamp associated with the fact. This allows the general awareness of timescales of events and allows facts to change over time. This seems particularly relevant to cybersecurity applications due to the time-sensitive nature of detecting attacks. As a simple example, it may be suspicious for a normal KGEM to receive several failed login attempts, but if timestamp information is included with these events and reveals that several login attempts failed within one second, this points to a brute force login attempt with high probability.

Other areas of future work can focus on making a KG-based system usable in a production environment, where new log data is streamed into the system and processed in real time, and network operators must be able to interpret the results quickly and accurately in order to formulate a response. Some modifications to the pipeline described in this study would be necessary for real-time processing. Drain3, which is the tool used in the proposed system for parsing raw log files, supports streaming data. If unknown entities are encountered while streaming in new data, they would need to be handled, which could be accomplished simply by adding them to the mapping, or assigning all unknown entities the same ID and treating them with increased suspicion. Additionally, the system should allow for changes in the computing environment, including new users, programs, configurations, etc. that appear in normal operation. To accomplish this, new training data should be acquired from log files once changes are made, new ID mappings should be created for any new

entities or relations, and the KGEMs should allow incremental training using this new training data. Finally, some methods should be applied to increase the interpretability of results that would give network operators actionable steps to take to contain possible threats. This should pinpoint the suspicious elements or interactions that caused the alert and possibly reccommend containment actions.

V. Conclusions

This study has developed a prototypical system for applying the knowledge graph completion or link prediction task to detect anomalies in log files. There are two main components in this system. The first component uses a template-based approach to generate a knowledge graph from a set of raw log files. This knowledge graph constitutes the training dataset for the next component, and this training data is composed of only typical behavior and should represent all types of normal behavior performed within the computing environment. Two more sets of triples, which do not themselves constitute knowledge graphs, are extracted to construct the validation and testing sets, and these data can contain both normal and malicious behavior. The second component of this system applies knowledge graph embedding models to learn a representation of the training knowledge graph. The previously established DistMult and ComplEx models are used, and they are extended by placing graph neural network layers before the scoring function layer to encode the graph information. Once the model learns a representation of the training knowledge graph, it applies the link prediction task to determine whether new information is suspicious. The output of the model is a probability or level of confidence that the fact belongs in the knowledge graph (i.e., the level of confidence that behavior represented by the fact is typical). Thus, a low confidence corresponds to a high level of suspicion. This confidence score is mapped onto a binary classification label so that triples are definitively labeled as either normal or suspicious.

The KG approach to performing anomaly detection has several advantages over more traditional machine learning methods. For example, organizing log data into multiple triples per log line

allows anomalies to be pinpointed within a log line. KGs, when represented in a linked data format such as RDF, also allow linking to external linked data sources, such as vulnerability databases, which can provide the KGEM with additional cybersecurity context to complement local data and improve downstream classification. Moreover, no attack data is needed in the training process since the KGEM only learns on normal behavior. Attack data is only used in model evaluation. Anomalies are detected as deviations from normal activity, so a KGC-based ADS should also be able to adapt to any new attacks.

There is limited prior research in applying link prediction to detecting anomalies in log files, and this work fills in the gaps from that prior research. One gap filled is that there is no work known to the author that describes a method of generating knowledge graphs from log files with the knowledge graph completion task considered as the downstream task. Previous work in this area considered manual querying, and the resulting structure of those knowledge graphs is not suited for knowledge graph completion, which was demonstrated in this study. Additionally, previous research on knowledge graph completion for anomaly detection did not report results in the standard metrics for that task, namely Hits@k, mean rank, and mean reciprocal rank. This study reports its results using these metrics for easy comparison to the knowledge graph completion task performance in fields other than cybersecurity. This study also uses standard classification metrics such as accuracy and F1-Score to provide more intuitive results that fit in with broader anomaly detection research. The results reported in this study make clear the feasibility of this approach, which will only increase as future work improves both the knowledge graph generation and knowledge graph completion components of this method.

REFERENCES

[1] A. Oprea, Z. Li, T.-F. Yen, S. H. Chin, and S. Alrwais, "Detection of early-stage enterprise infection by mining large-scale log data," in *Proc. 45th IEEE/IFIP DSN*, 2015, pp. 45–56.

[2] S. He, J. Zhu, P. He, and M. R. Lyu, "Experience report: System log analysis for anomaly detection," in *Proc. 27th International Symposium on Software Reliability Engineering (ISSRE)*, 2016, pp. 207–218.

[3] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly Detection and Diagnosis from System Logs Through Deep Learning," in *Proc. 2017 ACM CCS*, 2017, pp. 1285–1298.

[4] J. S. Garrido, D. Dold, and J. Frank, "Machine learning on knowledge graphs for context-aware security monitoring," in *2021 IEEE International Conference on Cyber Security and Resilience (CSR)*, 2021, pp. 55–60.

[5] D. Dold and J. S. Garrido, "An energy-based model for neuro-symbolic reasoning on knowledge graphs," in *The 20th IEEE International Conference on Machine Learning and Applications (ICMLA)*, 2021, pp. 916–921.

[6] V. Mulwad, T. Finin, and A. Joshi, "A domain independent framework for extracting linked semantic data from tables," in *Search Computing: Broadening Web Search*, S. Ceri and M. Brambilla, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 16–33, ISBN: 978-3-642-34213-4. DOI: 10.1007/978-3-642-34213-4_2. [Online]. Available: https://doi.org/10.1007/978-3-642-34213-4_2.

[7] V. Mulwad, T. Finin, and A. Joshi, "Semantic message passing for generating linked data from tables," in *The Semantic Web – ISWC 2013*, H. Alani *et al.*, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 363–378, ISBN: 978-3-642-41335-3. DOI: 10.1007/978-3-642-41335-3_23. [Online]. Available: https://doi.org/10.1007/978-3-642-41335-3_23.

[8] V. Mulwad, "Tabel - a domain independent and extensible framework for inferring the semantics of tables," Ph.D. dissertation, University of Maryland, Baltimore County, Jan. 2015.

[9] P. Nimbalkar, V. Mulwad, N. Puranik, A. Joshi, and T. W. Finin, "Semantic interpretation of structured log files," *2016 IEEE 17th International Conference on Information Reuse and Integration (IRI)*, pp. 549–555, 2016.

[10] F. Wang *et al.*, "Lekg: A system for constructing knowledge graphs from log extraction," in *The 10th International Joint Conference on Knowledge Graphs*, ser. IJCKG'21, Virtual Event, Thailand: Association for Computing Machinery, 2021, pp. 181–185, ISBN: 9781450395656. DOI: 10.1145/3502223.3502250. [Online]. Available: https://doi.org/10.1145/3502223.3502250.

[11] A. Ekelhart, E. Kiesling, and K. Kurniawan, "Taming the logs - vocabularies for semantic security analysis," *Procedia Computer Science*, vol. 137, pp. 109–119, 2018, Proceedings of the 14th International Conference on Semantic Systems 10th - 13th of September 2018 Vienna, Austria, ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2018.09.011. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050918316156.

[12] A. Ekelhart, F. J. Ekaputra, and E. Kiesling, "The slogert framework for automated log knowledge graph construction," in *The Semantic Web*, 2021, pp. 631–646.

[13] A. Singhal, *Introducing the knowledge graph: Things, not strings*, 2012. [Online]. Available: https://www.blog.google/products/search/introducing-knowledge-graph-things-not/.

[14] A. Krishnan, *Making search easier*, 2018. [Online]. Available: https://www.aboutamazon.com/news/innovation-at-amazon/making-search-easier.

[15] N. Noy, Y. Gao, A. Jain, A. Narayanan, A. Patterson, and J. Taylor, "Industry-scale knowledge graphs: Lessons and challenges," *Commun. ACM*, vol. 62, no. 8, pp. 36–43, Jul. 2019, ISSN: 0001-0782. DOI: 10.1145/3331166. [Online]. Available: https://doi-org.proxy.lib.utc.edu/10.1145/3331166.

[16] S. Shrivastava, *Bring rich knowledge of people, places, things and local businesses to your apps*, 2017. [Online]. Available: https://blogs.bing.com/search-quality-insights/2017-07/bring-rich-knowledge-of-people-places-things-and-local-businesses-to-your-apps.

[17] RDF Working Group, *Resource description framework (rdf)*, 2014. [Online]. Available: https://www.w3.org/RDF/.

[18] D. Beckett, T. Berners-Lee, E. Prud'hommeaux, and G. Carothers, *Rdf 1.1 turtle*, 2014. [Online]. Available: https://www.w3.org/TR/turtle/.

[19] SPARQL Working Group, *Sparql 1.1 overview*, 2013. [Online]. Available: https://www.w3.org/TR/sparql11-overview/.

[20] A. Maedche, "Ontology — definition & overview," in *Ontology Learning for the Semantic Web*. Boston, MA: Springer US, 2002, pp. 11–27, ISBN: 978-1-4615-0925-7. DOI: 10.1007/978-1-4615-0925-7_2. [Online]. Available: https://doi.org/10.1007/978-1-4615-0925-7_2.

[21] OWL Working Group, *Owl 2 web ontology language document overview (second edition)*, 2012. [Online]. Available: https://www.w3.org/TR/2012/REC-owl2-overview-20121211/.

[22] D. Brickley and L. Miller, *Foaf vocabulary specification*, 2004. [Online]. Available: http://xmlns.com/foaf/0.1/.

[23] R. Struse and T. Darley, *Stix version 2.1*, 2021. [Online]. Available: https://docs.oasis-open.org/cti/stix/v2.1/os/stix-v2.1-os.html.

[24] Z. Syed, A. Padia, T. W. Finin, M. L. Mathews, and A. Joshi, "Uco: A unified cybersecurity ontology," in *AAAI Workshop: Artificial Intelligence for Cyber Security*, 2016.

[25] M. Nickel, K. Murphy, V. Tresp, and E. Gabrilovich, "A review of relational machine learning for knowledge graphs," *Proceedings of the IEEE*, vol. 104, no. 1, pp. 11–33, 2016. DOI: 10.1109/JPROC.2015.2483592.

[26] X. Dong *et al.*, "Knowledge vault: A web-scale approach to probabilistic knowledge fusion," in *KDD '14*, New York, New York, USA: Association for Computing Machinery, 2014, pp. 601–610, ISBN: 9781450329569. DOI: 10.1145/2623330.2623623. [Online]. Available: https://doi-org/10.1145/2623330.2623623.

[27] M. Ali *et al.*, "Bringing light into the dark: A large-scale evaluation of knowledge graph embedding models under a unified framework," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021. DOI: 10.1109/TPAMI.2021.3124805.

[28] A. Bordes, N. Usunier, A. Garcia-Durán, J. Weston, and O. Yakhnenko, "Translating embeddings for modeling multi-relational data," in *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, ser. NIPS'13, Lake Tahoe, Nevada: Curran Associates Inc., 2013, pp. 2787–2795.

[29] M. Nickel, V. Tresp, and H.-P. Kriegel, "A three-way model for collective learning on multi-relational data," in *Proc. 28th ICML*, 2011, pp. 809–816.

[30] B. Yang, W. Yih, X. He, J. Gao, and L. Deng, "Embedding entities and relations for learning and inference in knowledge bases," *CoRR*, vol. abs/1412.6575, 2015.

[31] T. Trouillon, J. Welbl, S. Riedel, É. Gaussier, and G. Bouchard, "Complex embeddings for simple link prediction," in *Proc. 33rd ICML*, 2016, pp. 2071–2080.

[32] A. Joshi, R. Lal, T. Finin, and A. Joshi, "Extracting cybersecurity related linked data from text," in *2013 IEEE Seventh International Conference on Semantic Computing*, 2013, pp. 252–259. DOI: 10.1109/ICSC.2013.50. [Online]. Available: https://doi.org/10.1109/ICSC.2013.50.

[33] E. Kiesling, A. Ekelhart, K. Kurniawan, and F. Ekaputra, "The sepses knowledge graph: An integrated resource for cybersecurity," in *The Semantic Web – ISWC 2019*, C. Ghidini *et al.*, Eds., 2019, pp. 198–214.

[34] W. W. Lo, S. Layeghy, M. Sarhan, M. R. Gallagher, and M. Portmann, "E-graphsage: A graph neural network based intrusion detection system," *ArXiv*, vol. abs/2103.16329, 2021.

[35] A. Protogerou, S. Papadopoulos, A. Drosou, D. Tzovaras, and I. Refanidis, "A graph neural network method for distributed anomaly detection in iot," *Evolving Systems*, vol. 12, pp. 19–36, 2021.

[36] A. Deng and B. Hooi, "Graph neural network-based anomaly detection in multivariate time series," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 5, pp. 4027–4035, 2021.

[37] S. García, M. Grill, J. Stiborek, and A. Zunino, "An empirical comparison of botnet detection methods," *Computers & Security*, vol. 45, pp. 100–123, 2014, ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2014.05.011. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404814000923.

[38] S. Garcia, A. Parmisano, and M. J. Erquiaga, *Iot-23: A labeled dataset with malicious and benign iot network traffic*, version 1.0.0, Zenodo, Jan. 2020. DOI: 10.5281/zenodo.4743746. [Online]. Available: https://doi.org/10.5281/zenodo.4743746.

[39] F. Skopik, G. Settanni, R. Fiedler, and I. Friedberg, "Semi-synthetic data set generation for security software evaluation," in *2014 Twelfth Annual International Conference on Privacy, Security and Trust*, 2014, pp. 156–163. DOI: 10.1109/PST.2014.6890935.

[40] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," in *International Conference on Information Systems Security and Privacy*, 2018.

[41] M. Landauer, F. Skopik, M. Wurzenberger, W. Hotwagner, and A. Rauber, "Have it your way: Generating customized log datasets with a model-driven simulation testbed," *IEEE Transactions on Reliability*, vol. 70, no. 1, pp. 402–415, 2021. DOI: 10.1109/TR.2020.3031317.

[42] M. Landauer, F. Skopik, M. Frank, W. Hotwagner, M. Wurzenberger, and A. Rauber, "Maintainable log datasets for evaluation of intrusion detection systems," *ArXiv*, vol. abs/2203.08580, 2022.

[43] M. Damm, S.-H. Leitner, and W. Mahnke, *OPC Unified Architecture*. Springer Berlin, Heidelberg, Apr. 2009, p. 339, ISBN: 978-3-540-68899-0. DOI: 10.1007/978-3-540-68899-0. [Online]. Available: https://doi.org/10.1007/978-3-540-68899-0.

[44] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, "Drain: An online log parsing approach with fixed depth tree," in *2017 IEEE International Conference on Web Services (ICWS)*, 2017, pp. 33–40. DOI: 10.1109/ICWS.2017.13.

[45] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lió, and Y. Bengio, "Graph attention networks," *ArXiv*, vol. abs/1710.10903, 2017.

[46] W. L. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *NIPS*, 2017.

[47] F. Wei, H. Li, Z. Zhao, and H. Hu, "Xnids: Explaining deep learning-based network intrusion detection systems for active intrusion responses," in *USENIX Security Symposium (SECURITY)*, 2023.

[48] S. Ji, S. Pan, E. Cambria, P. Marttinen, and P. S. Yu, "A survey on knowledge graphs: Representation, acquisition, and applications," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 2, pp. 494–514, 2022. DOI: 10.1109/TNNLS.2021.3070843.

[49] B. Cai, Y. Xiang, L. Gao, H. Zhang, Y. Li, and J. Li, "Temporal knowledge graph completion: A survey," *ArXiv*, vol. abs/2201.08236, 2022.

VITA

Lucas Payne was born in Tazewell, TN and has one sibling, an older sister. He attended Tazewell-New Tazewell Elementary School in New Tazewell, TN, H.Y. Livesay Middle School in Harrogate, TN, and continued to J. Frank White Academy in Harrogate, TN for high school. In 2021, he received the Bachelor of Science degree in Computer Science from Lincoln Memorial University in Harrogate, TN, where he also worked as an IS helpdesk technician. The same year, Lucas accepted a graduate research assistant position at the University of Tennessee at Chattanooga and has since been pursuing the Master of Science degree in Computer Science.