Graph Representation Learning for Cyber Threat Intelligence: Exploring Threat Similarity through Graph Embeddings

MARK KÖRNER

Supervisor: Prof. Dr. Jürgen Schönwälder
Supervisor: Prof. Dr. Stefan Kettemann
Associate Supervisor: Abhilash Hota

A thesis for the conferral of a Master of Science in Data Engineering

Data Engineering
Jacobs University
Bremen

Date of Submission: 14 August 2020

Statutory Declaration

Körner,Mark
30002281
Master Thesis

English: Declaration of Authorship

I hereby declare that the thesis submitted was created and written solely by myself without any external support. Any sources, direct or indirect, are marked as such. I am aware of the fact that the contents of the thesis in digital form may be revised with regard to usage of unauthorized aid as well as whether the whole or parts of it may be identified as plagiarism. I do agree my work to be entered into a database for it to be compared with existing sources, where it will remain in order to enable further comparisons with future theses. This does not grant any rights of reproduction and usage, however.

The Thesis has been written independently and has not been submitted at any other university for the conferral of a PhD degree; neither has the thesis been previously published in full.

German: Erklärung der Autorenschaft (Urheberschaft)

Ich erkläre hiermit, dass die vorliegende Arbeit ohne fremde Hilfe ausschließlich von mir erstellt und geschrieben worden ist. Jedwede verwendeten Quellen, direkter oder indirekter Art, sind als solche kenntlich gemacht worden. Mir ist die Tatsache bewusst, dass der Inhalt der Thesis in digitaler Form geprüft werden kann im Hinblick darauf, ob es sich ganz oder in Teilen um ein Plagiat handelt. Ich bin damit einverstanden, dass meine Arbeit in einer Datenbank eingegeben werden kann, um mit bereits bestehenden Quellen verglichen zu werden und dort auch verbleibt, um mit zukünftigen Arbeiten verglichen werden zu können. Dies berechtigt jedoch nicht zur Verwendung oder Vervielfältigung.

Diese Arbeit wurde in der vorliegenden Form weder einer anderen Prüfungsbehörde vorgelegt noch wurde das Gesamtdokument bisher veröffentlicht.

14 August 2020, 

.....

Date, Signature

Abstract

Graphs are used within cyber threat intelligence to visualize, store and share information. Leveraging graph-like storage formats such as STIX 2.1 [21], one can create graphs to visualize and analyze the relationships between different entities such as advanced persistent threats, malwares, attack patterns, indicators and intelligence reports. From these graphs, graph representation learning methods can create low-dimensional vector embeddings of each specific entity, or node in graph terms, which can then be decoded into similarity measures or used for further clustering or prediction tasks.

This project explores the use of a variety of graph representation learning algorithms on data gathered from several cyber threat intelligence data sources. The data formats used by these sources allow for the construction of several graphs with varying underlying structures, which are used to compare the similarity scores for APTs and threat reports to assess the graph embeddings in the context of the included data and graph structure.

Keywords: cyber threat intelligence, graph representation learning, node embeddings, advanced persistent threats, malware, attack pattern, DeepWalk, node2vec, graph auto encoders, LINE, HARP

Acknowledgements

I would like to acknowledge my supervisors Prof. Dr. Jürgen Schönwälder and Prof. Dr. Stefan Kettemann for their continuous guidance and feedback throughout the entirety of this project as well as previous projects, and Abhilash Hota for providing me with a clear project direction and necessary domain knowledge. This project could not have succeeded without their assistance.

I would also like to extend a special thank you to my colleagues Mimi Chindasook and Prateek Choudhary, whose encouragement, help and friendship continues to allow me to succeed in ways I otherwise would not have, as well as all other peers at Jacobs University that made our studies much more enjoyable and productive. Finally, I would like to thank my parents, Christina and Christoph, for their continued support throughout my entire academic journey.

Contents

List of Figures

List of Tables

Section 1

Introduction

Graph structures are a conventional format in which cyber threat intelligence (CTI) information is stored and visualized. As an example, graphs are used to visualize individual intrusion processes at Alienvault OTX[1], monitoring mitigation and detection efforts for persistent threat actors [18], or visualizing various forms of intelligence around such actors and attacks based on the STIX 2.1 standard[2]. In addition to visualizing cyberthreat information, graph-like structures are used in formats such as STIX 2.1, which aim to standardize the storing and sharing of cyber threat intelligence. Graph representation learning, which models feature representations from graphs and its contents, can be applied to these example graphs in order to aid analysts in assessing attacks by suggesting who is behind a particular attack, compare attack patterns and explore potential mitigation strategies. Thus, the goal for this project is to leverage the graph-like structures in which cyber threat intelligence is stored to create accurate feature-based representations of cyber threat intelligence graphs for further analysis.

As a result, this report will aim to answer the following question: **How can graph representation learning be used to gain insight from static cyber threat intelligence data stored in graph-like data structures?**

Additionally, three secondary research questions are proposed as a means to help answer the main research question:

(1) How can the graphs be constructed given the raw dataset structures?
(2) How can the graph structure be adapted for a particular modelling goal?
(3) What kind of data and what features should be included in the specific graphs?

In order to examine these questions, the report is structured in the following sections:

(2) **Background**

This section will give an introduction to cyber threat intelligence, graph theory and machine

[1]Example:https://otx.alienvault.com/malware/Trojan:Win32%2FDorv/samples
[2]Example:https://oasis-open.github.io/cti-documentation/examples/defining-campaign-ta-is

learning concepts related to graph theory. In addition, it will establish terms and abbreviations which will be used throughout the rest of the report.

(3) **Data Collection**

In this section, I will detail the data collection process. Specifically, I will take a look at the respective collection procedures, data structures and information contained in all 5 data sources. I will also share some initial exploratory analysis.

(4) **Methods**

In order to be able to build the graphs, I first needed to clean, filter and transform the data based on the requirements and goals for the graphing and modelling steps. This section will highlight the procedures used in order to prepare the data for the graphing step and discuss the structural choices made for the graphs. Finally, it will discuss the different algorithms used and the specific graph structures they were applied on as well as introducing several evaluation methods.

(5) **Implementation**

This section will detail the specific implementation of the methods described in Section 4. As a first step, it will feature some initial exploration of the datasets in the form of visualizations. Before being able to construct the graphs, I also needed to clean, filter and transform the data based on the requirements and goals for the graphing and modelling steps. After listing the model hyperparameters and implementation details, this section will also examine the embeddings by graph structure and discuss the differences between models.

(6) **Conclusion and Future Scope**

The Conclusion section will evaluate the success of the embedding results in the context of the research question. In addition, it will provide an overview of potential areas for further research, the limitations of the current approach, their causes and potential improvements to overcome these limitations in the future.

Section 2

Background

The report covers the intersection between cyber threat intelligence, graph theory and machine learning. Therefore, this section will introduce important basic concepts from each area that are fundamental to understanding the terminology and context for the rest of the report.

2.1 Cyber Threat Intelligence

Cyber Threat Intelligence (CTI) describes the analysis and sharing of information regarding the nefarious activities of (potential) adversaries [26]. In terms of CTI taxonomy, Menges and Günther [28], Bromiley [6] and Launius [26] provide comprehensive taxonomies and definitions, which allow the reader to connect various research papers and data sources with varying terminologies. In terms of threat strategic analysis and mitigation, the Diamond Model [7], ATT&CK Framework [41] and Kill Chain model [18] are some of the most influential works for assessing and mitigating cyber attacks.

Adversaries, also called threat actors or simply actors, are the entities behind a certain attack, for example criminal groups, activist groups, nation-states or state-sponsored groups [7] [26]. Adversaries with long-term strategies, long-term attack patterns and recurrent attacks are also called **Advanced Persistent Threats (APTs)**, which require more proactive mitigation strategies. **Attacks**, or intrusions, describe the act of infiltrating or attacking the **victim's** (cyber) systems in order to reach the attacker's goal, which could range from stealing information to shutting down the system to extortion.

Common attack terms, which will be used in the remainder of the report, are malware, trojans, exploits and phishing. **Malware** is an umbrella term used to describe malicious software, including exploits, trojans and viruses. An exploit abuses a software vulnerability to gain access to the particular system, extract information or insert their own malware into an otherwise trustworthy software. Trojans are often disguised as legitimate software to gain access to victims' systems. Phishing commonly refers to spam emails attempting to collect information about the recipient such as user passwords, bank account information or simply whether the email account is actively used [25].

CTI can describe a variety of information, such as malware files, emails used to distribute these files, common exploits used to gain access to a certain system or general activity patterns used by adversaries.

This information can be categorized into two types of information, **indicators of compromise (IoC)** and **tactics, techniques and procedures (TTP)**. Launius describes IoCs as "artifacts with the context pertinent to a cyberattack" [26], which can then be used to detect similar attacks. They can be classified further into atomic indicators such as IP or email addresses which cannot be broken down further and retain meaning in the context of an intrusion, computed indicators such as hash values of files, which are derived from data following an intrusion, and behavioral indicators, which combine atomic and computed indicators to describe patterns of behavior used in the attack [18]. TTPs are defined as "the actions, skills, methods, or modus operandi (MO) an adversary uses to accomplish their goals" by Launius [26]. Similar terms such as attack patterns in [18] all fall under TTP.

In order to comprehensively analyze, produce and store this information, a number of threat models and standards have emerged. Frameworks such as the Diamond Model [7], Cyber Kill Chain [18] and the MITRE ATT&CK Model [41] attempt to categorize the available information into separate phases, tactics or core categories and attempt to model attacks from the adversary's perspective or in the context of the adversary-victim relationship to derive actionable insights and more potent mitigation strategies. In addition, threat information standards such as Structured Threat Information Expression (STIX 2.1) [21] and Malware Information Sharing Platform (MISP) [45] provide machine-readable information standards to facilitate the sharing of cyber threat intelligence. These frameworks can be used in conjunction with other naming frameworks such as MITRE's Common Attack Pattern Enumeration and Classification (CAPEC) [3], which includes over 500 terms to describe all known attack patterns [26].

Finally, there are different levels of CTI analysis with different goals which organizations can employ, namely the strategic, operational and tactical levels. The operational and tactical levels of CTI analysis are focused on simple threat/intrusion tracking and sharing of the above-mentioned information, whereas analysis at the strategic level is concerned with the possible evolution of threats in the context of an organization's current and future strategy and is used to inform decision-making in terms of overall organizational strategy and resource allocation [26].

2.2 Graph Theory

In general, a graph is a collection of **vertices**, also called **nodes**, which are connected to each other through **edges**. Mathematically, a graph can be expressed as the pair of the sets of vertices and edges

$$G = (V, E), where\ V = \{v_1, ..., v_n\}, E = \{e_1, ..., e_m\},$$

where edges can be expressed as vertex pairs such that $e_k = (v_i, v_j)$ [8]. Given an example graph in which every node has at least one edge connected to it, the resulting graph can then be expressed as a collection of edges or node pairs, which will be referred to as an **edge list** going forward.

Another way to mathematically express a graph is the **adjacency matrix**, which can be thought of as an edge list in matrix form. The adjacency matrix is an nxn matrix, where entry $A_{i,j}$ in the matrix represents the (potential) edge between vertices v_i and v_j and n is the total number of nodes in the graph. As an example, if there is an edge between vertex 1 and vertex 3 in a simple graph, then $A_{1,3} = 1$, otherwise $A_{1,3} = 0$ [35].

A sequence of edges connecting multiple vertices form a **path**. In other words, edges between vertices 1-3 and 3-5 form a path from 1 to 3 to 5. The length of a path is given by the number of edges traversed, so the above example would have length 2. Paths are called self-avoiding if every node is only traversed once [35]. The **distance** between two nodes is defined as the length of the shortest path between them [8].

In terms of graph measures, the **degree** k_i of any vertex i is given by the number of edges connected to it. The average degree over the entire graph can be calculated as $\frac{\sum_i^n k_i}{n}$, or $\frac{\sum_i^n \sum_j^n A_{i,j}}{n}$ with respect to the adjacency matrix [35]. When evaluating the relationship between two nodes, one can evaluate it in terms of first- and second-order proximity. The **first-order proximity** between two nodes u and v is given by the edge weight $w_{u,v}$ or $A_{u,v}$ in adjacency matrix notation. Given the first-order proximities p_u and p_v between those nodes and all other nodes in the graph, the **second-order proximity** between two nodes can be found by evaluating the simularity between p_u and p_v. If they don't share any connected nodes, the second-order proximity between u and v is equal to 0 [42].

Finally, graphs in this report are usually **undirected**, **unweighted** and **heterogeneous**. A directed graph means that edges have a direction, i.e. from A to B but not necessarily from B to A, whereas undirected graphs imply that an edge from A to B can also be traversed from B to A [35]. Weighted graphs simply mean that each edge carries an edge weight parameter denoting the value of each edge,

whereas in unweighted graphs the implied weight for every edge is 1 and the same. A heterogeneous graph simply implies that the graphs contain different types of nodes and/or edges, for example a knowledge graph of a particular cyberattack, where the nodes are comprised of the threat actor, the attack pattern and the software used in the attack.

2.3 Machine Learning

Machine Learning broadly "refers to a set of tools for modeling and understanding complex datasets" [19, p. vii]. Generally, machine learning problems can be divided into two main categories: unsupervised vs supervised learning. Supervised learning requires a response variable for each data point, so that the particular model can relate the responses to the corresponding observation with the goal of predicting the response for future observations or better understanding the relationship between observation features and the response variable. Unsupervised learning is characterized by a lack of response variable, meaning that the aim of unsupervised learning is limited to understanding the relationship between observations or variables [19, p. 26-27].

In order to generate a model, often-used expressions are 'fitting the dataset' or 'training the model'. Generally, fitting or training a model on a dataset refers to optimizing a particular model according to some criterion, for example misclassification error in classification tasks or mean squared error in simple regression tasks. The optimization function is referred to as the **loss function**, and is utilized in both supervised and unsupervised learning settings [16, p. 18].

The task in this report is referred to as network embedding, graph embedding or broadly as graph representation learning [8] [44]. The goal of these embeddings is "to learn low-dimensional vector representations [...] for nodes in the graph [...] such that graph properties (local and global) are preserved" [8, p.4]. Essentially, network embeddings attempt to learn features from networks while preserving a semblance of the graph structure. Embeddings can also be extended to subgraphs, i.e. parts of graphs, instead of nodes and to include node features in the embedding.

Chami et al describe embeddings in the context of an encoder-decoder framework. Assuming a standard node-based embedding, the encoder produces the embedding, whereas the decoder uses the embedding to produce similarity scores for all node-pairs. The loss function usually compares the decoded similarity matrix to a transformed adjacency matrix of the initial graph [8]. Most embedding algorithms presented in this report follow a similar pattern.

2.4 Overview of Graph Representation Learning for CTI

Usman et al. provide a first overview of the applications and limitations of representation learning techniques and existing datasets in the cybersecurity domain, although not specifically using graph representations [44]. Holder et al introduce a graph-based learning approach based on EAGLE threat simulator data [17]. Finally, Böhm et al provide a simple visualization tool for the STIX2-based threat data to enable further manual threat analysis [4]. Their approach to building the graphs provided an interesting starting point for the analysis presented in this paper. Very recently, Al-Shaer et al also presented a clustering approach which could help predict future TTPs used by an attacker by the previously used ones [39].

In terms of deep learning graph-based approaches that have been applied in the CTI field, several embedding techniques have been developed as an application of the famous word2vec algorithm, which is primarily used for working with natural language. These approaches have mostly been applied to the Android Malware Dataset, which features application programming interface (API) dependency graphs of malicious Android apps [32]. Specifically, graph2vec [32] and the specialized subgraph approaches called sub2vec [1] and subgraph2vec [33] have all been applied to this particular dataset.

In general, Chami et al provide an overview on the current major approaches for applying machine learning to graph-structured data [8]. The paper categorizes algorithms in terms of the way graph information is encoded and what information is included and creates a taxonomy framework which highlights major algorithms in each category. From this paper, we applied the DeepWalk [36], node2vec [13], LINE [42], HARP [9] and GAE [24] algorithms in addition to the previously mentioned subgraph2vec [33] to the graph datasets.

Section 3

Data Collection

Cyber threat intelligence data was collected from five sources: the Pulsedive API [1] [2], the Alienvault API [3] [4], the ThreatTracking resource [5], the MITRE CTI repository [6] and the CTIMiner dataset [7]. The sources express information in different formats and terminology, but with a degree of similarity that allowed us to combine some of them into joint graphs. All sources include information about APTs or adversaries and contain direct or indirect information on the tools and malwares employed by these groups. Sources also sometimes include IOCs, TTPs, geolocations, names of major attack campaigns and various other metadata.

3.1 Pulsedive

Pulsedive provides an API[1] that can be accessed for free with a limit on the number of API calls per user in specific time frames. Use cases range from personal website analysis, to threat tracking and APT research. The Python API[2] allows queries for Indicators, Threats or Feeds. Feeds combine data from several open-source intelligence feeds, which is aggregated with user submissions to generate the full dataset of Indicators and Threats. Indicators represent IoCs such as URLs, domains, IP addresses or file artifacts which include an automatically determined risk level and tags containing more information about the specific indicator. Risk levels range from Unknown to Very Low, Low, Medium, High, Critical and Retired. Threats include a wide variety of categories from general ones such as attacks and abuse, to malware, to threat actors such as groups and APTs, to tactics such as reconnaissance.

In order to access the dataset, Pulsedive provides an open-source Python client, which provides the query results in Python dictionary format albeit at the cost of less available parameters as compared to the HTML-based examples. I first queried the Threats API for all threat information, followed by querying the Indicator API by threat and risk level. Querying individually by threat was necessary

[1] Pulsedive API: https://pulsedive.com/api/
[2] Pulsedive Python Implementation: https://github.com/pberba/pulsedive-py
[3] Alienvault API: https://otx.alienvault.com/api
[4] Alienvault Python Implementation: https://github.com/AlienVault-OTX/OTX-Python-SDK
[5] Document: https://apt.threattracking.com
[6] Repository: https://github.com/mitre/cti
[7] DOI for Download: http://dx.doi.org/10.21227/dpat-qd69

because the API only returns the most recent 15,000 results per query, and the final dataset included more than 350,000 indicators. Thus, I needed parameters such as threat and risk level to return the maximum amount of indicators overall. The responses were then transformed into Pandas dataframe format for further analysis. Since the dataset includes indicators submitted for personal analysis from users, whose links to threats are oftentimes unreliable, only indicators with a risk level of medium, high, critical or unknown were included in the final version. Due to the nature of the query process, a representation of the dataset structure including a few select variables can be seen in Figure 3.1.

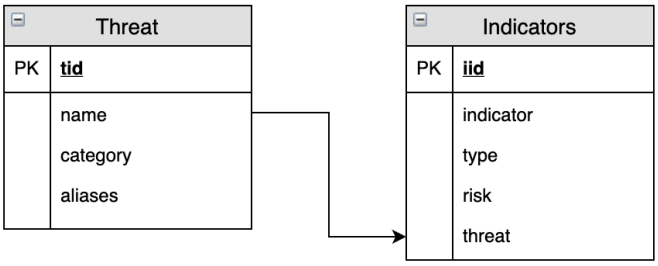


Figure 3.1. The diagram shows a schema representation of the Pulsedive dataset structure. The Indicators table includes indicators, indicator metadata and the associated threat, which is used as a link to the Threat object to include further threat-related information.

Through the threat name, I was also able to include the threat category in the indicator data. Some indicators are present multiple times in the dataset, as the same indicator can be attributed to multiple threats. These duplicates were useful for attempting to link different threats to each other, such as linking APTs to the specific malwares they used.

In terms of reliability of the data, IOCs are sourced from both threat intelligence feeds and user submissions. All user submissions for website safety analysis are included in the dataset as IOCs, which thus includes a lot of safe websites that ideally need to be filtered out. Threat intelligence feeds are usually much more reliable in only including relevant IOCs. In addition, the IOCs are also "vetted by Pulsedive's risk assessment" and "corrected by contributors and customers" [37]. Judging by the size of the dataset and the inclusion of user submissions for analysis, however, the dataset should be regarded as fairly noisy.

3.2 Alienvault

The Alienvault Open Threat Exchange API contains similar data and use cases as Pulsedive, albeit with a different structure. On Alienvault, registered users provide threat intelligence information in

so-called Pulses, which are essentially collections of threat information and can contain a variety of relevant information such as IoCs, attributed APTs/threat groups and related malware families. Alienvault does not allow the user to query the entire dataset through the API, but instead forces the user to subscribe to individual pulses which can only be accessed and filtered through the website. As such, the dataset I extracted can only be reconstructed using my OTX Key or the list of pulses I subscribed to. APT and malware information is included in the Pulses object, while all the indicator information is contained in a sub-object at the individual indicator level as displayed in Figure 3.2.

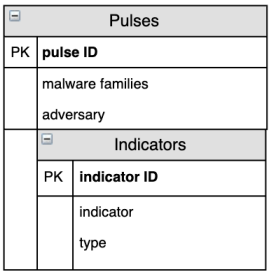


Figure 3.2. The diagram shows a simplified schema representation of the Alienvault dataset structure. The Pulses object includes information on the malware families and the adversaries connected to a certain incident, campaign or otherwise collected bundle of indicators as well as the indicator sub-object. The Indicator object is embedded into each Pulse, and contains a varying amount of indicators and related information.

The retrieved dataset consisted of 887 pulses related to APTs in JSON-Format. These pulses included around 7000 rows of indicator-level data which provided information about the malware family used in the pulse's attack or campaign.

As the extracted pulses are user-submitted, the indicator quality depends heavily on the user that submitted them. Pulses are submitted through extraction from text or other sources, and are automatically screened for potential false positives upon submission through an algorithm implemented by Alienvault. It is up to the submitter to then include or exclude the extracted indicators or even manually add further indicators [2]. This submission process is susceptible to human error and noise, and thus the quality of the dataset highly depends on the users who created the pulses.

3.3 Threat Tracking

The Threat Tracking resource consists of high-level analysis of APT group's activities. The information is stored in a read-only, downloadable Google Sheet[5], with APTs arranged by geographical location in separate sheets. Sheets follow a similar yet slightly varying structure, and include group name aliases,

major operations, toolset, targets and modus operandi. The resource also includes a sheet with aliases for commonly used malwares.

While it is very comprehensive, the data format is not directly conducive to analysis due to format differences between the sheets and columns themselves. Namely, the "Toolset / Malware" column consists mostly of comma-separated lists, but also includes full text fields or a mix of both. Thus, processing the file requires either advanced text mining techniques or manual cleaning. The Google Sheet document can be downloaded as an Excel file and then transformed into a separate Pandas dataframe for every individual sheet. This resource was mostly used for linking the other datasets and as a conversion reference for aliases of malwares and APTs.

In terms of reliability, the README sheet lists its contributors with name and Twitter handle. Non-contributors can comment on cells, which is then subject to review. Finally, it also includes a disclaimer stating that it should not be seen as a reliable source as the information could quickly become outdated or is only based on a single incident report. Finally, individual sources are not provided for every entry but rather the most prominent sources are listed in a separate sheet, making it difficult to ascertain every piece of information.

3.4 MITRE Cyber Threat Intelligence Repository

The MITRE repository is hosted on GitHub[6] and structured as a STIX2 JSON based file system. It is divided into Enterprise- (Windows, MacOs, Linux etc.), Mobile- (Android, iOS etc.) and Pre-Attack file systems. STIX2 is essentially a JSON wrapper with a pre-defined structure and keys for several data categories sorted into the corresponding objects. Figure 3.3 shows a mapping table between STIX2 objects and MITRE ATT&CK categories.

These individual objects have several common properties, such as an ID, name, a list of aliases, external references such as MITRE ATT&CK or CAPEC IDs, created and last modified dates and kill chain phases. For the graphs, the Technique, Group and Software objects are the most relevant as they contain similar information to the other data sources. The objects are linked by Relationship object, which describes the type of relationship and provides a source and target reference ID as well as external references which led to the creation of said relationship. A simplified version of the underlying schema is shown in Figure 3.4 below.

ATT&CK concept	STIX Object type
Technique	`attack-pattern`
Group	`intrusion-set`
Software	`malware` or `tool`
Mitigation	`course-of-action`
Tactic	`x-mitre-tactic`
Matrix	`x-mitre-matrix`

Figure 3.3. Mapping table of ATT&CK framework concept to object types in STIX 2.0, from [11].

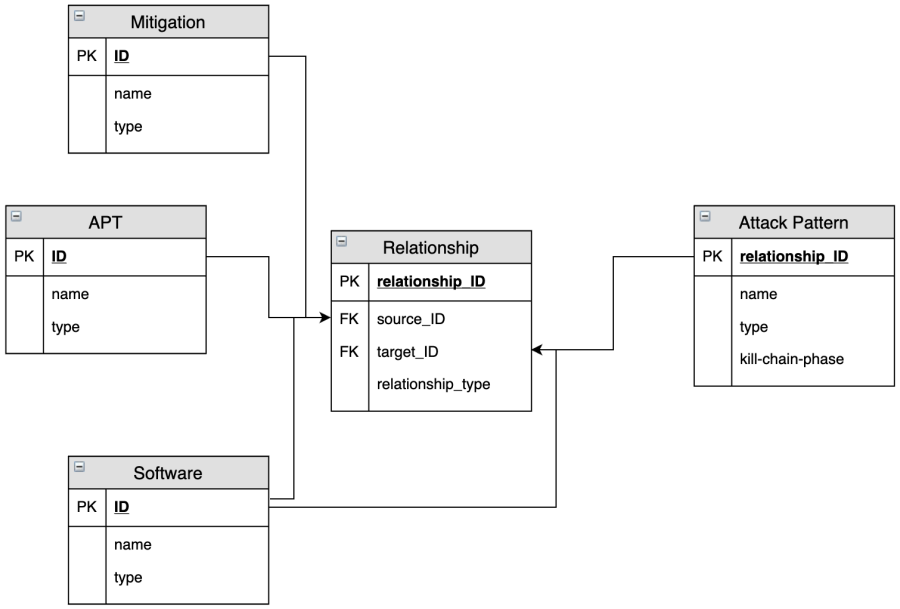


Figure 3.4. The diagram shows a schema representation of the MITRE dataset structure. The Relationship object lists the connections between entries in the APT, Software, Mitigation and Attack Pattern objects. Due to the nature of the underlying relationships, the source_ID field consists of IDs from the APT, Software and Mitigation objects, whereas the target_ID field consists of IDs from just the Attack Pattern and Software objects.

In essence, the APT, Software, Mitigation and Attack Pattern object IDs are used as source and target ID in the Relationship object, with the arrows showing the overall direction of the relationship. The relationships can be simplified to four cases: APT “uses” Software, APT “uses” Attack Pattern, Software “uses” Attack Pattern, Mitigation "mitigates" Attack Pattern.

To create an object that contains all available information, one can use the source and target reference IDs to join the Software, APT, Group, Mitigation and Technique objects to the Relationship objects, creating a dataset at a relationship level - or edge level in graph terms. MITRE contains well-curated

information, which does not require cleaning besides some minor reformatting. Compared to the other datasets it only lacks IoC information, yet provides much more reliable links between Groups and Software.

The repository is based on the ATT&CK knowledge base curated by the MITRE Corporation, which allows referenced contributions that will be reviewed by periodically. The content is updated every three to six months [10]. The dataset used for this report is based on the most recent commit on March 9, 2020.

3.5 CTI Miner

This dataset is hosted on IEEE Dataport[7], and is the result of parsing 600 CTI reports for Indicators of Compromise. It contains some meta information on the reports themselves as well as indicators, the type of indicator (for example, filehash, url etc.) and what category of analysis led to their uncovering (Network activity, External analysis etc.). The data is provided in XML format and split up into Report and Malware Events across years. The Report objects contain information about the parsed report and the associated indicators. In order to gather more information, the malware hashes that were extracted from the reports were then cross-checked with another data source for more available information, and the additional indicators are saved in the respective Malware Event objects [34]. The underlying data structure is displayed in Figure 3.5, with the info field in the Malware object filled with malware hash values in the Indicator value field in the Indicator Reports sub-object, which allows us to connect the objects. After cleaning, the dataset contains 71910 unique indicators in the Report and Malware objects combined.

The quality of this dataset depends on the quality of the parser used, the quality of the indicators present in the reports that were sourced from APTnotes[8] and APT CyberCriminal Campaign Collection[9] as well as the quality of the additional data source used to gather more information on the malware file hashes[10] [34]. Overall, the information seems fairly well-curated overall, yet the scope of the reports varies quite a bit from individual campaigns to multi-year overviews.

[8]Github repository:https://github.com/aptnotes/data
[9]Github repository:https://github.com/CyberMonitor/APT_CyberCriminal_Campagin_Collections
[10]API:https://www.malwares.com/about/api

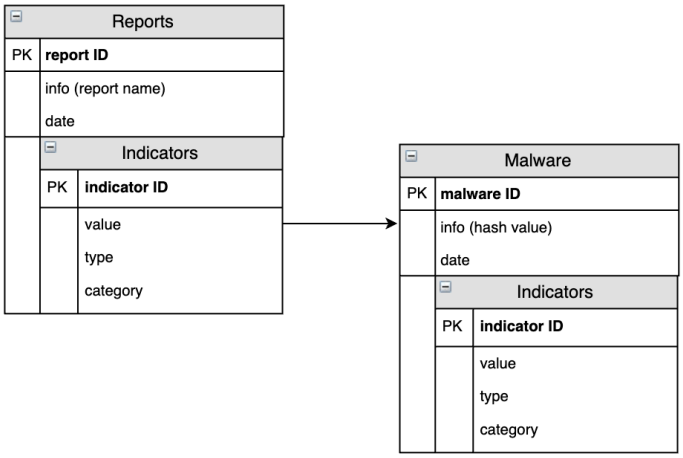


Figure 3.5. The diagram shows a schema representation of the CTI Miner dataset structure with all yearly objects combined into one Reports and Malware object each. Both the Reports and Malware object contain meta information about the specific report or malware hash in addition to an Indicators sub-object with IOCs contained in the report or related to the specific malware hash.

Section 4

Methods

Although some cyber threat intelligence is stored in graph-like structures, building graphs from CTI data usually requires a number of cleaning and preprocessing steps in order to generate a graph structure via an edge list or adjacency matrix as decribed in Section 2.2. This section will detail the preprocessing steps necessary to prepare for graph creation (Section 4.1), the different structure choices made for graphing given the data (Section 4.2) as well as the algorithm choices depending on graph structure and modelling goal (Section 4.3).

4.1 Preprocessing

Considering the differences in dataset structure, format and information present for each source, the preprocessing steps varied for the different graphs. The ultimate goal was to build a graph from one or a combination of sources, thus the dataset structure had to be transformed into either an edge list or an adjacency matrix to represent a graph. For this project, we assumed that IOCs, TTPs, APTs, malwares and other information are only useful given their specific context, meaning it was necessary to connect them to at least one other piece of information. Thus, in graph terms, I made the assumption that every node had to have at least one edge, making it viable to use an edge list to represent and create the graphs. Therefore, the main preprocessing goal was to transform the datasets into edge lists and clean them to remove synonymous names between datasets.

Transforming traditional table-structured data into an edge list is fairly elementary, as it generally just involves filtering and extracting two of the columns. Most of the dataset was obtained in JSON- and XML-based formats, however, which is more akin to a tree-like structure [40]. XML works with tags, which have to be opened and closed and include values or more tags in between. JSON functions on key-value pairs, where the value can also be a collection of further key-value pairs. Examples of both JSON- and XML-based CTI data structures are shown in Figure A.1 in the appendix.

Given these nested, tree-like data structures, the resulting graphs should feature edges between values in the top level and all values in the lower levels of the structure in addition to same-level one-to-one edges. As an example, the Kwampirs Trojan mentioned in Figure A.1a should be connected to both file

hash indicators below as well as the adversary Orangeworm, which is cut off in the figure as it is below all the associated indicators. The different levels of data in the tree-structured data can be likened to different levels of data granularity, and building an edge list between top-level and lower-level variables requires a transformation to the lower-level granularity. Specifically, the observation in Figure A.1a would be split into two separate observations with the rest of the variables included in both observations. This type of transformation was used with both the CTIMiner and Alienvault datasets.

Moreover, a clean dataset is very important for modelling purposes. Since all variables in the raw dataset are categorical, the main goal for these particular datasets was standardizing names within and between datasets. Standardizing in this case means setting a name for a particular entity and renaming all observations using synonymous or misspelled versions of the same name. The chosen name is fairly arbitrary, as the text content of the name will not be directly analyzed in the modelling process but rather just used as a label representing a distinct entity. In terms of misspellings, the limited size of the labels present in the dataset allowed me to assess each label manually. Misspellings could also be detected automatically using word similarities between labels or node similarity in graphs in larger projects.

Finally, many algorithms require numerical inputs even for categorical variables. Thus, the name for every node was assigned to an ID in $\{0, n\}$, which was then used in the modelling task. Some algorithms also support node-level numerical features. A commonly used way to encode categorical variables as features is via one-hot encoding, which refers to the use of a vector with length k for a categorical variable with k different terms. For example, encoding a variable with the labels 'indicator', 'malware' and 'apt' would result in a vector of length three, with 'indicator' represented as $[1, 0, 0]$ and 'apt' represented as $[0, 0, 1]$.

4.2 Graph Structures

The final preprocessing step can be reduced to finalizing the graphs. In order to create the graphs, I first had to consider the desired structure in light of the modelling goal and the available data. Fortunately, the use of edge lists to build graphs simplifies the process of adding variables to the graph, as the additional edge lists can just be concatenated to represent the expanded graph. This interaction makes it fairly effortless to mix and match variables and datasets from multiple sources, provided all datasets were cleaned prior to being added.

The first limitation to all possible graph structures is the available data. Figure 4.1 shows the possible connections between different entities for all different data sources. Noticeably, Pulsedive and Alienvault allow for similar connections between different entity categories and could be connected to the MITRE CTI dataset by matching APTs and Malwares to include Attack Patterns and Mitigation Techniques. CTI Miner only shares the IoC category with other data sources and only adds the associated report. Finally, the MITRE CTI dataset features directed edges, as the relationships between entities are directly defined in the Relationships object unlike the other sources. Overall, this means the graph structure is largely limited to connecting individual IoCs to APTs and Malwares, and then connecting APTs and Malwares to Attack Patterns along with Mitigation Techniques. CTIMiner will largely be seen as a separate dataset with individual modelling objectives.

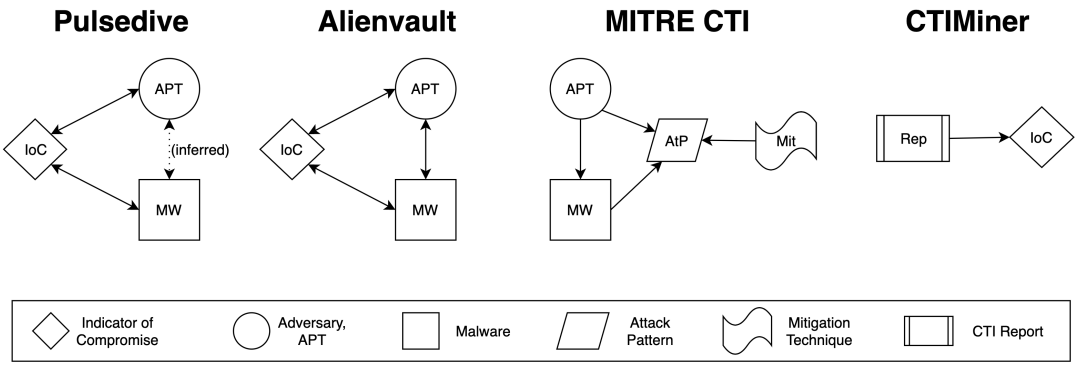


Figure 4.1. Diagram showing the possible edge structures between different entities for four of the data sources, namely Pulsedive, Alienvault, MITRE and CTIMiner.

At this point, it is also important to point out that all the mentioned graph structures involve including different node categories in the same graph. This approach is modelled loosely after STIX and other CTI visualizations, which generally feature multiple categories in order to visualize the relationships between the different pieces of information. In contrast, classical graph theory datasets such as Zachary's Karate Club social network, collaboration networks and biological networks only feature one type of node [12]. In this case, single category graphs could be created by only connecting nodes from one category if they share connections to the same entity from another category. Yet, the aim of this project is to exploit the graph structure of STIX-like or similar graphs, and therefore the rest of the report will use graphs which include multiple node categories.

Based on these existing structures, I decided to examine APT similarity with a variety of different structures and approaches as well as Report similarity with the CTIMiner dataset as my modeling

goals. In terms of APT similarity, the evaluated structures included combinations of the Pulsedive and Alienvault datasets with the MITRE dataset to include IoCs in the MITRE structure and just the MITRE dataset structure with and without Mitigation Techniques. For Report similarity, the underlying CTI Miner structure is used. It is important to note, however, that the CTI Miner structure contains edges between reports and IoCs contained in the report, and edges from some of the IoCs to more IoC values as those values were found when researching the original IoC.

Finally, besides the node category structure, the graphs can also be built as one graph which encompasses all data points or as individual subgraphs around an entity like a particular APT. In the APT example, a subgraph for every APT in the dataset is created, which only includes the connected malwares and attack patterns. Most evaluated graphs in this report are full graphs, but some algorithms require subgraphs or created subgraphs themselves, in which case they include every node up to a certain distance.

4.3 Modelling Options for different Graph Structures

Given the different graph structures, there are different ways to model these graphs and create embeddings. Some algorithms are better suited for certain graph structures and modelling goals, so this section will describe the different algorithms and their use case. A lot of the algorithm use the Skip-Gram technique, which originated from the field of natural language processing, and I will thus describe this technique at first before delving into the particular algorithms [30].

4.3.1 Skip-gram

At its core, the Skip-gram model is generally used in natural language processing in the classification of words in a sentence by looking at the surrounding words. Specifically, the objective is to find the conditional probabilities for the occurrence of other words in a sequence of words $(w_1, ..., w_t)$ given each target word w_k using the loss function

$$\mathcal{L} = - \sum_{-K \leq i \leq K, i \neq 0} \log \mathbb{P}(w_{k-i} \mid w_k),$$

where K corresponds to the number of surrounding words before and after the target word [8]. During training, the surrounding words are often weighted according to the distance to the target word, the closer it is the higher its weight. Skip-gram is a word embedding model, and while it tries to predict

context words, target words with similar context words will have similar embedding values, meaning that it can be used to find similar words or synonyms.

In the graph context, these word sequences are not inherently available, but can be generated using random walks across the graph's nodes and edges. As an example, the DeepWalk algorithm follows the two-step process of walk, i.e. node sequence generation for each node of length t, followed by the application of the Skip-gram algorithm. The walks are generated by randomly selecting an outbound edge from the specific node and subsequently adding the destination node to the walk sequence [36]. Thus, the graph application of the skip-gram algorithm embeds nodes in the context of its neighboring nodes when using random walks to generate context.

4.3.2 DeepWalk

After discussing the Skip-gram algorithm, it is only fitting to discuss its first application for graph embeddings: DeepWalk [8]. As previously stated, DeepWalk functions as a two-step process of random walk generation and utilization of the Skip-gram algorithm to generate embeddings. The random walks can be adjusted by setting the walk length t which represent the number of nodes traversed on each walk, window size of nodes w considered when calculating the loss and updating the node probabilities for the target node, and walks per vertex γ used in the training set[36]. In the case of DeepWalk, the γ parameter can be seen as synonymous with the term epochs, which is commonly used in the context of neural networks to set how many times the algorithm runs through the entire training dataset to fit the model.

To further illustrate, let us consider a random walk with $t = 7$ and and $w = 2$. The walk is sampled from node 5 and results in the sequence $[5, 2, 4, 3, 6, 9, 8]$. Now, to update the probabilities for the third node in the sequence (node 4), we consider the probabilities of nodes 5,2,3 and 6 given its embedding $Z(4)$ and update it to maximize the probability of co-occurence [36]. For the next node, we would consider nodes 2,4,6,9 given embedding $Z(3)$.

Besides the application to graphs, DeepWalk uses an optimization to the algorithm called hierarchical softmax to calculate co-occurrence probabilities. The normal softmax formula calculates the probability of one node in the context of all other nodes, i.e. $softmax(n_i) = \frac{e^{n_i}}{\sum_j e^{n_j}}$. Hierarchical softmax instead builds a binary tree and assigns nodes to the leaves of the tree. Instead of updating the probabilities for each graph node directly, the probabilities for each graph node are given by the path from the tree's

root node to the particular leaf node and only the tree node binary probabilities are updated. Instead of a calculation with the complexity of the number of nodes in the graph, this reduces the complexity to the number of tree nodes in the particular path to the graph node [31].

As DeepWalk looks at nodes in the context of its neighboring nodes, I theorized that the algorithm would be a good fit for both modelling tasks. Ideally, it would be able to model APT similarity in the context of which malwares, IoCs and attack patterns they share as well as reports in the context of shared IoCs. Since it is also able to capture higher-order relationships, it should even be able to pick up more indirect relationships between APTs such as a shared Indicator between an APT and a Malware, where is also used by a different APT which itself has no connection to the specific Indicator. Overall, I applied this algorithm to all structures except for the smaller MITRE graph as that particular structure was only created for work with subgraphs.

4.3.3 node2vec

Node2vec is similar to DeepWalk in that it also employs random walks to model the underlying graph structure and then applies the Skip-gram model to generate embeddings. It uses a different technique when creating the random walks, however, which can be tuned to focus on breadth-first sampling (BFS) and depth-first sampling (DFS). BFS focuses on sampling the local neighborhood of the starting node, whereas DFS focuses on sampling nodes at increasing distance from the starting node [13]. These approaches can be adapted through the hyperparameters p and q, which affect the likelihood of selecting a particular node x to be the next node v_i in the walk:

$$\pi_{v_{i-1}x} = \alpha_{p,q}(v_{i-2}, x) \cdot w_{v_{i-1}x},$$

where w is the edge weight and alpha is the search bias term given by

$$\alpha_{p,q}(v_{i-2}, x) = \begin{cases} \frac{1}{p} & \text{if } d_{v_{i-2}x} = 0 \\ 1 & \text{if } d_{v_{i-2}x} = 1 \\ \frac{1}{q} & \text{if } d_{v_{i-2}x} = 2 \end{cases},$$

where d is the shortest-path distance between the two nodes. Thus, p regulates how likely the random walk is to return to the previous node from the current node and q regulates how likely it is to visit a node which is the maximum distance away from the previous node. A high value for p indicates that the next node will likely not be the previous node, whereas a low value for p will lead to a large amount

of redundancy in the sampled walks as it is likely the walk will return to the previous node. A high value for the q parameter biases the edge selection towards nodes that are equidistant from the previous node as the current node or towards the previous node itself, whereas a low value promotes nodes that are the maximum distance from the previous node. As such, both parameters can be used individually or in conjunction with each other to bias the random walk sampling towards a breadth-first-like or depth-first-like approach.

Apart from node sampling for random walks, node2vec also uses a technique called negative sampling to lessen computational complexity [13]. Specifically, the objective function contains a per-node partition function which in part sums over the embedding of every single node in the graph, which is computationally expensive for very large graphs. Negative sampling instead only samples a pre-defined small number of these nodes and updates the co-occurrence probabilities only for these nodes [29].

In terms of use cases, the algorithm is applicable in similar scenarios as DeepWalk, albeit it being more adaptable to a variety of problem settings due to its BFS and DFS modes. After directly comparing its results with DeepWalk on the full MITRE graph, however, I decided to mostly go with DeepWalk as a continuous reference point for most graphs due to more straightforward implementation and better results overall.

4.3.4 LINE

LINE, or Large-scale Information Network Embedding, is a model designed to preserve first- and second-order node proximity (compare Section 2.2) as well as being scalable and working with undirected, directed, weighted and unweighted graphs. In order to preserve both first- and second-order proximity, LINE fits two separate model for both measures and then suggests to concatenate both embedding feature vectors and re-weighting them according to the training data. The original paper suggests re-weighting only when the weights can be learned from the training data in a supervised learning setting [42], which is thus not applicable to our use case. The algorithm is only used together with HARP, which only uses the first-order proximity part of LINE [9].

As the first-order proximity algorithm is only applicable to undirected graphs, it models the joint probability between two vertices u and v as

$$p_1(u, v) = \frac{1}{1 + e^{-Z_u \cdot Z_v^T}},$$

where Z_u is the low-dimensional vector embedding of node u. In order to fit the algorithm, the loss function minimizes the negative sum of the graph's edge weight $w_{u,v}$ and the log-transform of the joint probability $p_1(u, v)$ as following:

$$L_1 = -\sum_{(u,v)\in E} w_{u,v} \log p_1(u, v) \quad [42].$$

First-order proximity does not seem to be a very good measure to find similarity between APTs or Reports within the provided graph structures as APTs are usually separated by at least two edges. Therefore, this algorithm was mostly intended as a reference point within HARP to compare its performance to node2vec and DeepWalk.

4.3.5 HARP

Hierarchical Representation Learning (HARP) is a graph representation learning algorithm, which combines a graph coarsening pre-processing framework with popular embedding algorithms, namely LINE, node2vec and DeepWalk. Graph coarsening refers to iteratively grouping nodes together and therefore reducing the size of the input graph before embedding. The coarsening can be achieved through the edge collapsing and star collapsing techniques [9]. Edge collapsing refers to creating a joint supernode from joining two nodes which are connected through an edge. The edges are sampled from the edge list without overlap, meaning that at each coarsening step each node can only be selected once. Star collapsing refers to collapsing nodes into supernodes if they share the same neighbors. The star structure features a center, high-degree node, also called hub, whose neighboring nodes form a star around it. These two techniques are combined into a hybrid coarsening scheme by applying star collapsing first before applying edge collapsing to the reduced graph.

Iterative application of the coarsening scheme to the initial graph results in increasingly smaller graphs $G_0, G_1, ..., G_L$. Starting with the smallest graph G_L, the chosen embedding algorithm is applied to create a representation of the remaining super nodes. This embedding is then used to initialize the embedding algorithm for the next smallest graph in a technique called embedding prolongation. Super nodes separated in the smaller graph retain the node embedding from the super node they were a part of. This procedure is repeated until the embedding for the initial graph G_0 is calculated [9]. This technique produces more stable training and convergence leading to more stable embeddings [8].

Overall, this technique is applicable to both graph structures but seems very promising particularly for the CTIMiner graph due to the large clusters of IoCs for every report. Report nodes are exemplary of the hub nodes at the center of star structures, and should thus profit immensely from the star collapsing technique. Therefore, I used the HARP framework as the main technique for the CTIMiner graph and applied all three originally mentionated algorithms - LINE, node2vec and DeepWalk - with it to compare results.

4.3.6 subgraph2vec

Subgraph2vec utilizes a radial adaptation of the Skip-gram model to learn representations of subgraphs for one or multiple large graphs. Subgraphs in this context refer to smaller substructures of larger graphs, which are sometimes used as a means to express the full graph structure and compare between graphs. Subgraphs can be rooted around a node, and are then referred to by the root node and the distance from the root node it encompasses ("subgraph degree") [33]. In previous subgraph modelling approaches, subgraphs were seen as independent structures even if they exhibited a high degree of similarity in terms of the nodes present within them, which lead to increasingly high dimensionality of the feature space due to overlap and graphs only being similar to themselves.

The algorithm follows a two-step process similar to other skip-gram based models by first extracting rooted subgraphs of a chosen degree and then applying the radial skip-gram to learn embeddings. Finding the rooted subgraph $sg_v^{(d)}$ of node v of degree $d > 1$ is a recursive operation of first finding the neighbors of v, and then using the same operation for all the neighbor nodes at $d - 1$. In order to apply the skip-gram algorithm, the context used to embed the similarity of the target subgraph $sg_v^{(d)}$ is the set of $d + 1$, d and $d - 1$ subgraphs rooted at each of the neighbors of v such that

$$J(\Phi) = -\log \Pr(sg_{cont} \mid \Phi(sg_v^{(d)}))$$

is maximized, where Φ represents the embedding. Subgraph2vec also uses negative sampling to calculate and update the co-occurence probabilities [33].

Since subgraph2vec can take multiple graphs as inputs, I decided to examine the use of single APT subgraphs to determine APT similarity. In other words, I decided to create a different graph for every APT, which only features the APT and the malwares, attack patterns and Indicators it is directly connected to.

4.3.7 GAE

Convolutional graph auto-encoders (GAE) combine graph convolutional networks (GCN) [23] with variational auto-encoders (VAE) [22] to create an unsupervised learning algorithm which can incorporate node features into the embeddings. The node embeddings are calculated as $Z = \text{GCN}(X, A)$, where X is the feature matrix and A is the adjacency matrix [24]. The GCN can be generalized to

$$Z = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}X\Theta,$$

where $\tilde{D}$ is a diagonal matrix given by $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$ with $\tilde{A} = A + I_N$ and Θ is a matrix of filter parameters. The decoder function reconstructs the adjacency matrix as follows: $= \sigma(ZZ^T)$, where σ is the logistic sigmoid function [23]. GAE also uses negative sampling to overcome computational complexity [8].

Since this method can incorporate node features, I first thought about which available features could be useful in creating a graph embedding. Specifically in the APT similarity case, incorporating the node type, i.e. APT, malware, IoC etc., could prove beneficial, whereas it might not be as useful for the Report similarity task as it has a very defined structure and less node types. Thus, I used this algorithm only for the APT structured data.

4.3.8 SimRank

SimRank is not a graph representation learning algorithm but rather just a node similarity algorithm. The basic premise behind SimRank is that “two objects are similar if they are referenced by similar objects" [20]. As such, it is mainly based on second-order proximity. The basic SimRank equation is a recursive comparison of nodes’ neighborhoods’ similarity.

$$s(u, v) = \frac{C}{\mid I(u) \mid\mid I(v) \mid} \sum_{i=1}^{|I(u)|} \sum_{j=1}^{|I(v)|} s(I_i(u), I_j(v)),$$

where $I(v)$ is the set of in-neighbors for a particular node v and $I_i(u)$ is a particular member of that set. If u and v are the same node x, $s(x, x) = 1$. C in this case can be seen as either a confidence level or a decay rate. As an example, if two APTs a and b use the same malware m, we can agree that $s(m, m) = 1$ but $s(a, b)$ probably should not equal 1. Thus, C can be used as a confidence level in the fact that two nodes with the same neighbor are similar, or a decay rate of the same neighbor similarity score across the network.

As SimRank is not a graph embedding method, it is simply used as a reference point when evaluating embedding similarity scores for both similarity tasks.

4.4 Embedding Evaluation

Since the presented algorithms are mostly unsupervised learning algorithms, the methods of evaluating them are generally more complicated than for supervised learning models. In supervised learning, the way to evaluate a model is to compute some error measures for the predictions based on the response values, meaning that there are some ground truth values available which allow for comparison which is not the case with datasets used for this report. In constrast, unsupervised learning methods attempt to find lower-dimensional representation of the underlying dataset, which can then be used explain certain characteristics of the underlying distribution and to group certain observations together. For graph embeddings, however, it is difficult to find a direct link between the embeddings and the underlying cause for a certain embedding vector, as the embedding process is generally rather complicated and non-linear.

Embedding outputs can be evaluated in terms of two measures, the embedding vectors themselves and the embedding similarity. The embedding vectors can be used for clustering and a variety of other prediction tasks, but prediction tasks are again not possible due to a lack of consistent response variable. In terms of clustering, Al-Shaer et al provide an interesting example on attack pattern clustering in the context of tactics and evaluated the clusters by surveying industry experts [39]. Unfortunately, surveying industry experts on embedding-based clusters was not possible due to the increasing complexity of my thesis but would be a logical extension of the method.

Other evaluation frameworks propose to assess embeddings in terms of intrinsic measure categories such as relatedness [38] or by attempting to recover topological features such as node degree from the embeddings [5]. Unfortunately, [5] concludes that it is not consistently possible to recover node or graph features from embeddings. Although the original paper largely requires datasets with response variables, the evaluation category of relatedness might be applicable to CTI graph embeddings. Namely, relatedness refers to using the high correlation between cosine similarity and human relatedness scores to evaluate similarity scores [38]. The MITRE dataset refers to some relationships between APTs in the APT description fields, and the CTIMiner dataset includes some reports in the sub-object that also contains Indicators, which could be used as evidence for strong relatedness between entities

mentioned in both of these scenarios. Cosine similarity leverages the dot product of two vectors to create similarity scores similar to the Skipgram methods, but normalizes the scores by the magnitude of each vector as follows:

$$sim(v_1, v_2) = \frac{v_1 \cdot v_2}{\mid v_1 \mid\mid v_2 \mid},$$

where $\mid v \mid$ is the magnitude of the embedding vector v given by $\mid v \mid = \sqrt{v \cdot v^T}$ [27].

Thus, I decided to evaluate the graph embeddings based on cosine similarity scores for known node relationships and in the context of the scores for the rest of the embedding. In addition, I also included SimRank scores for some graphs in order to get a sense of node similarity depending on the graph structure.

Section 5

Implementation

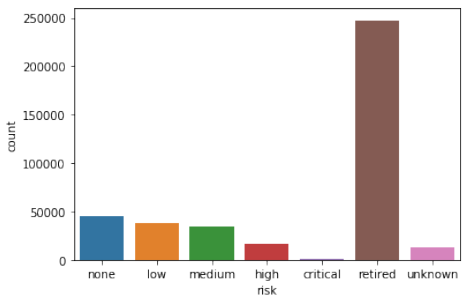
After introducing the theory behind the methods used, this section will detail their implementation. Specifically, I will first discuss the initial exploration of the four datasets in order of access in Section 5.1, which led to implementing the preprocessing steps discussed in Section 5.2. The modelling approach is detailed in Section 5.3.

5.1 Initial Dataset Exploration

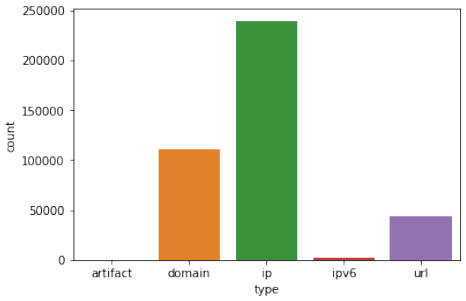
The first dataset accessed was Pulsedive. In total, I scraped 396,094 unique indicator-threat pairs from the Pulsedive API. Figure 5.1 shows an initial exploration of the distribution of indicators relative to risk level, indicator type and threat category. As evidenced in Figure 5.1a, most of the indicators are classified as retired, and only a minority are either high or critical risk. Still, a little over 18,000 indicators are rated as high or critical risk. In terms of indicator type, almost all indicators are web-based with only 11 being artifacts. In terms of category, note that the group and APT categories are also at the lower end of the spectrum at 4703 and 4141, respectively, while the malware, attack and abuse categories are attributed to most of the indicators.

Finally, Figure 5.1d shows the proportion of indicators which are related to multiple threats and at least one APT with multi-threat indicators, all unique indicators and all observations. One of the first graphing attempts was based on the idea of linking malwares and APTs through shared indicators based on just the Pulsedive dataset, but the lack of observations ultimately stifled the idea.

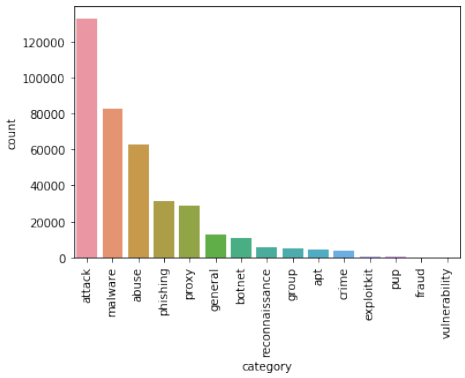
Alienvault Open Threat Exchange contains similar data to Pulsedive collected in Pulses. Similar to Pulsedive, some Pulses contain data on both associated APTs and Malwares, while some only feature APT information. Figure 5.2a shows that the share of Pulses with Malware information is rather small. Of the Pulses with both Malware and APT information, Figure 5.2b shows the distribution of indicators per Pulse in log-scale. More than half the Pulses contain less than 20 indicators, yet the largest Pulse contains more than 2,500. An explanation for this disparity is that some Pulses are related to one specific attack, whereas some Pulses cover multiple campaigns or operations over multiple years.



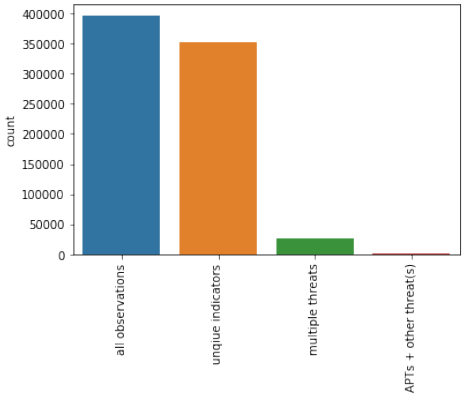
(a) Bar chart of indicator count by risk level.



(b) Bar chart of indicator count by indicator type.

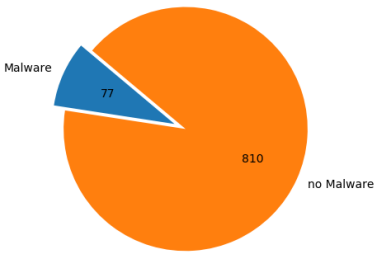


(c) Bar chart of indicator count by threat category.

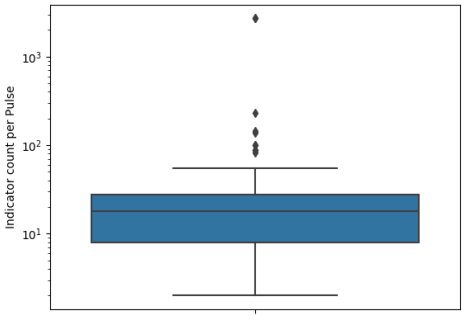


(d) Bar chart of indicator count for all, unique and multi-threat indicators as well as indicators associated with APTs and at least one other threat.

Figure 5.1. The figures show indicator count by several features, such as risk level pertaining to the specific indicator, indicator type, and underlying threat category in the Pulsedive dataset.

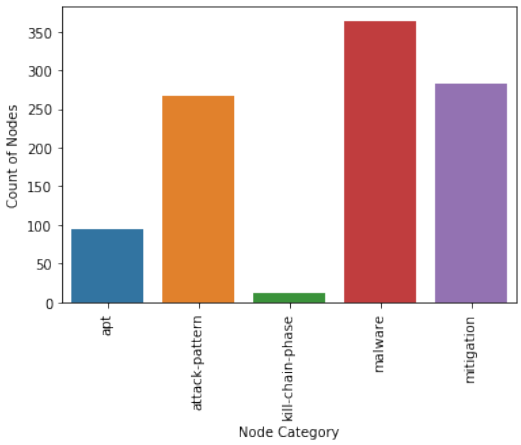


(a) Pie chart of Pulses with and without malware information.

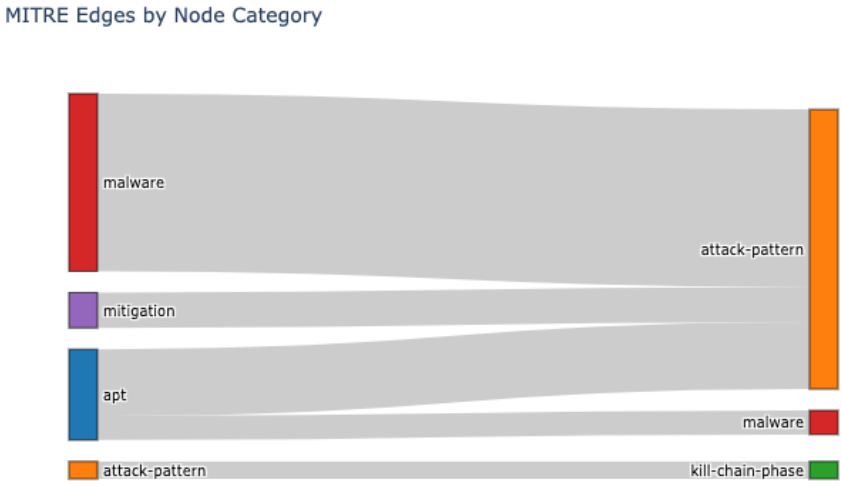


(b) Boxplot of Indicators per Pulse in log-scale.

Figure 5.2. The figures show indicator count by several features, such as risk level pertaining to the specific indicator, indicator type, and underlying threat category in the Pulsedive dataset.



(a) Bar chart of the count of nodes per category.



(b) Sankey chart of the count of edges by node category.

Figure 5.3. The figures the nodes and edges in the MITRE dataset. Specifically, the bar chart shows the count of individual nodes by category and the Sankey chart shows the count of all possible edges by source and target node category.

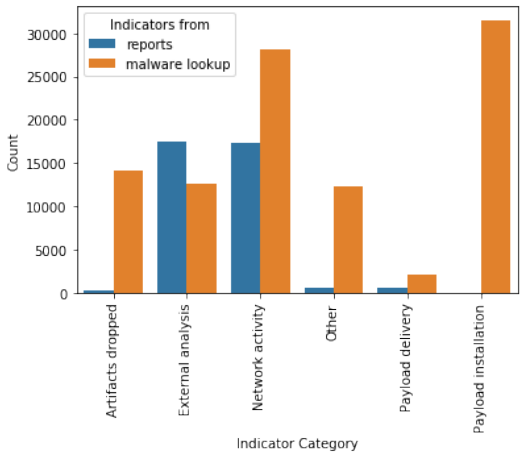
For the MITRE dataset, the underlying dataset structure is much more akin to a traditional database structure with a main table (Relationships) which links the other tables as shown in Figure 3.4, with the APT, software, attack pattern and mitigation objects being referenced by the source and target ID in the Relationship object. From the individual object for each information category, we can get the distribution of potential nodes seen in Figure 5.3a, whereas the Relationship object allows us to display the distribution of possible edges seen in 5.3b. It is important to note that the relationship to the kill chain phases is actually in the attack pattern object and needs some pre-processing in the

form of a granularity transform as one attack pattern can belong to multiple phases. This difference is also evidenced in Figure 5.3b, as this particular relationship is isolated from the others. Aside from the kill chain phases, most of the relationships have attack patterns as the target, with the only other non-attack-pattern-related relationship linking APTs and malwares. The largest share of edges is between malwares and attack patterns, which makes sense given that these two also make up a large portion of the individual nodes. Finally, although mitigation techniques is the second largest node category, only a fraction of the edges references mitigation techniques. This occurrence is caused by the fact that mitigation techniques are usually fairly specific, meaning that most of the techniques only mitigate one or a small number of attack patterns, which is different for all other categories.

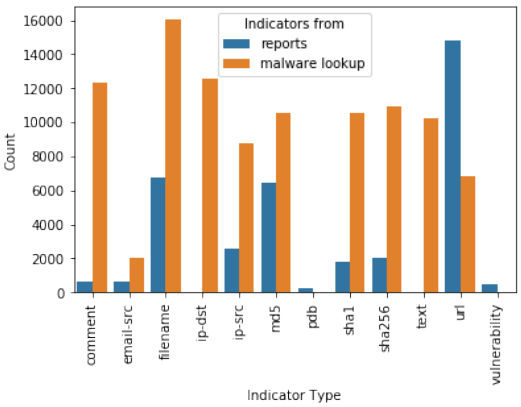
Finally, the CTI Miner dataset is structured into the Reports and Malware objects as shown in Figure 3.5. The Reports objects contain the reports as well as Indicators mentioned in the reports. The Malware objects contain both the researched malware hash as well as the related Indicators, which were found as a result of the search. Figure 5.4 shows the distribution of the Indicators in terms of their specific category and type variables, which were taken directly from the data, as well as the relationships and possible edges present in the dataset. Figures 5.4a and 5.4b show that there are not only more Indicators present in the Malware objects, but the categories and types are also fairly different. Figure 5.4c shows that most of the relationships in the dataset are actually between the searched hash values and the resulting related Indicators instead of the Indicators from reports. In addition, there are some relationships between reports in the dataset, which can later be used to validate the report similarities if these relationships are removed from the training dataset prior to modelling.

5.2 Preprocessing

The preprocessing stage can be condensed into two tasks: cleaning and transforming the data. Since some of the proposed graphs involve multiple datasets, a significant part of the cleaning process involved unifying the terminology between different sources. Since MITRE seemed to provide the most consistent naming convention and framework, the APT and malware names from other sources were adapted to the MITRE ones. The name changes were done mostly manually by checking the list of unique malwares and APTs from both Pulsedive and Alienvault OTX against a lists of MITRE names as well as synonyms, and changing the synonyms into the main names. Then, APTs and malwares not matching any existing names were manually researched for potential overlaps. Ultimately, all non-matching observation were discarded as they would have resulted in an unbalanced graph.



(a) Bar chart of the number of Indicators per category.



(b) Bar chart of the number of Indicators per category.

(c) Sankey chart of the Edges by Node Category.

Figure 5.4. The figures show the nodes and edges in the CTIMiner dataset. Specifically, the bar charts shows the count of Indicators by category and type and the Sankey chart shows the count of all possible edges by source and target node category.

A lot of indicator cleaning efforts were carried out and evaluated. For example, we tried to only include indicators of a certain risk from Pulsedive, which resulted in too small of a dataset. We also had to decide whether to include indicators that are only linked to either a malware or an APT, and ultimately decided to only include the ones with links to both in order to ensure a better connectivity for indicators. Overall, nodes that are only linked to one another node and do not provide much context information provide little value to an embedding algorithm and lead to a much larger training set and much higher

computational complexity. One cleaning step that was always present was removing duplicate entries from a dataset. This step was really only necessary for the indicator-based datasets, and was carried out by looking at column-pairs or -combinations to see if there are duplicate values in all in order to only remove duplicates which do not add additional context.

In terms of dataset structure transformations, the two types used in the report were transforming the data format from JSON- and XML-based data into a more traditional tabular format and granularity transforms from Pulse- or Report-level to IoC-level. JSON and XML data can be transformed into column format by taking the highest layer which contains useful information and taking the keys or tags for said layer as column headers. This transformation necessitates the granularity transformation as data from lower granularity layers is saved in a specific field as a dictionary or list of dictionaries, making it largely unusable for further analysis. Thus, the granularity transform expands these fields and copies the information from the other fields. These transformations were necessary for the Alienvault OTX and CTIMiner datasets.

After most cleaning and transformation steps, I also looked at the number of indicators per malware and APT pair to see if there were any imbalances. The pair of the Orangeworm APT and Kwampirs malware had more than 5,000 indicators associated with it, whereas the next highest pair had only 229. In order to create a more balanced dataset, I decided to randomly sample 250 indicators for this pair and remove the rest. An imbalance in the distribution of APT-malware pairs and indicators could lead to algorithms neglecting pairs with a low number of associated indicators in favor of the high-frequency ones, resulting in an embedding with a large bias towards the pairs with the most observations.

The final preprocessing step was creating the actual graphs, or just providing edge lists or adjacency matrices in the required formats for each particular algorithm. Again, the edge lists were created by simply concatening edge lists between several categories. For datasets which only include some information present in the dataset, I decided to only include nodes which had relationships with all possible node categories. As an example, I created a combined graph between the Alienvault, Pulsedive and MITRE datasets, and decided to exclude APTs and malwares without Indicator connections and attack patterns without connections to the remaining APTs and malwares. I also created a truncated MITRE graph with only APTs, malwares and attack patterns, and decided to exclude attack patterns with no connections to those two categories. Finally, the nodes in the edge lists were encoded into IDs as required by some of the embedding algorithms, and both the edge lists and a nodes dataframe, which

allowed the association between IDs, node names and node categories, were saved. If an adjacency matrix was required, I used the NetworkX package[1] to create graphs using edge lists, and then used the built-in function to extract the adjacency matrix from the graph. Figure 5.5 shows an example visualization of the combined graph.

Figure 5.5. The figure shows an example visualization of the combined graph with Alienvault and Pulsedive Indicator information and MITRE CTI malware, APT and attack pattern data. Indicators are in green, APTs in blue, malwares in red and attack patterns in purple.

Finally, some algorithms also allow for node features, which are usually provided in a separate node-based data frame format similar to the node ID data frame mentioned above. Specifically, I used the features to describe what category a node belongs to and for Indicator nodes, what type of Indicator it presents, for example URL, IP address etc. These categorical variables were encoded as dummy variables and submitted separately, with the row index corresponding to the node ID in the graph.

5.3 Modelling

The preprocessing and graphing measures pipeline resulted in four graphs with the following characteristics listed in Tables 5.1 and 5.2. As shown in Table 5.1, graphs 1-3 depend on the MITRE dataset, with graph 3 including indicators from the Alienvault and Pulsedive sources. Graph 2 includes

[1]Documentation: https://networkx.github.io/documentation/stable/

less attack patterns than graph 1, as only attack patterns with connections to APTs and Malwares were included, which means that some attack patterns from graph 1 are only connected to mitigation techniques and not malwares and attack patterns. As stated above, graph 3 includes only APTs and malwares found in the IoC datasets, and attack patterns connected to those specific nodes. Finally, CTIMiner is a completely separate dataset with only reports and IoCs used to model report similarity.

Graph	Name	APTs	MW	AtP	Mit	KCP	IoC	Rep
1	MITRE full	94	364	266	282	12	-	-
2	MITRE truncated	94	364	227	-	-	-	-
3	MITRE IoC combined	20	33	161	-	-	1074	-
4	CTIMiner	-	-	-	-	-	71910	600

Table 5.1. Node category comparison between the different graphs. Legend:
MW: malware, AtP: attack pattern, Mit: mitigation technique, KCP: kill-chain phase, Rep: report

Graph	Nodes	Edges	Avg. Deg.	Algorithms
1	1018	6344	12.46	DeepWalk, node2vec
2	681	5297	15.56	subgraph2vec
3	1288	3308	5.14	GAE, DeepWalk
4	72510	107393	2.96	HARP, DeepWalk, LINE, node2vec

Table 5.2. Graph measures comparison and list of algorithms used on each graph.

In terms of graph measures, the disparity of average degrees initially stands out. The differences can be explained in terms of the type of data included in the graphs, however. Specifically, Indicators usually have a very low degree in these types of graphs, as they are usually only connected to a couple of malwares and indicators, with one of each being the most common case. Thus, part of the reason why graphs 3 and 4 are lowest in terms of average degree is the inclusion of IoCs, as graph 4 is made up almost exclusively of Indicators. The difference between graph 1 and 2 can be explained by the inclusion of mitigation techniques, which are usually specific to one attack pattern and therefore usually have a degree of one, thus reducing the average degree of the graph. Finally, CTIMiner is a much larger dataset than the other three due to only necessitating minimal filtering for the Indicators.

Table 5.2 also lists the embedding algorithms used for each dataset. The reasons for choosing each algorithm for the specific graphs are outlined in Section 4.3. All algorithms embed all nodes in the graph, as well as subgraphs in the case of subgraph2vec. As for the hyperparameters, I mostly consulted the original papers for suggestions and ran some algorithms such as node2vec, subgraph2vec and GAE with different parameters and varying inputs to compare results. In terms of embedding vector size d, I used multiples of 32 for comparison sake, but mostly went with 32 because computational limitations.

Thus, most embedding outputs have the shape $n \times d = n \times 32$, where n is the number of nodes in the graph. In addition, I used a stable learning rate of 0.025 for all embeddings.

For graph 1, both node2vec and DeepWalk accept edge lists as inputs. Both node2vec[2] and DeepWalk[3] have package implementations, which allow the user to provide custom edgelists and output the embedding. I used the same parameters for node2vec and DeepWalk as much as possible, and tried three iterations of p and q node2vec values for comparison' sake. Specifically, I used a embedding dimension of 32, walk length of 60, window size of 10 and number of walks of 80. For the p and q parameters, I used combinations of $(p, q) = \{(1, 1), (0.5, 2), (2, 0.5)\}$.

For subgraph2vec, I used two different graph input configurations with the same parameters to compare approaches. Subgraph2vec has a Tensorflow implementation available on GitHub[4], which was developed by one of the authors of the original paper [33]. Unfortunately, it is written in Python 2, and I decided to slightly adapt the code to work with Python 3. The first embedding uses custom subgraphs for each APT as input, which only include the immediate neighbors and the connections between them. The second embedding is an embedding of the full graph with subgraphs found automatically by the algorithm. Both embeddings feature a size of 64, number of negative samples of 20, validation size of 5. The first embedding was trained with a batch size of 32 and a number of epochs of 100, whereas the second one was trained with a batch size of 64 and a number of epochs of 500, as it seemed less computationally expensive than expected.

For graph 3, I used both the DeepWalk and the GAE algorithm. The hyperparameters used were the same as for graph 1 for number of walk and window size, but I used a larger embedding size of 64 and a shorter walk length of 40. The GAE algorithm also allows node features to be included in the model as described at the end of Section 5.2. In total, I ran 4 different iterations for GAE with all possible combinations of no dropout, dropout of 0.2, including features and not including features. All GAE embeddings were trained for 500 epochs and with a hidden layer size of 128 and an embedding size of 64. GAE also offers an official implementation on GitHub[5].

Lastly, the CTIMiner graph was embedded using the HARP framework in combination with DeepWalk, LINE and node2vec. In addition, I also used DeepWalk without HARP as a point of reference to be

[2]Node2vec: https://pypi.org/project/node2vec/0.3.2/
[3]DeepWalk: https://pypi.org/project/deepwalk/1.0.3/
[4]Subgraph2vec: https://github.com/MLDroid/subgraph2vec_tf
[5]GAE: https://github.com/tkipf/gae

able to directly compare it to other graphs. I used the GitHub implementation [6] of the original paper to create the embeddings with their default parameters. The embedding size for all algorithm was 128, except for Line with an embedding size of 64. LINE also uses a window-size of 1. DeepWalk and node2vec both feature 40 for the number of walks, a walk length of 10 and window size of 10. For the DeepWalk reference model I used larger hyperparameters in relation to the embedding size of 128 and more akin to my other choices of hyperparameters for other graphs. Namely, I used a walk length of 80, window size of 10 and 80 for the number of walks.

5.4 Embedding Evaluation

As described in Section 4.4, evaluating embeddings according to any single metric is difficult as there is no known ground truth and embeddings do not directly encode topological features such as node degree. Thus, I decided to evaluate embeddings by comparing them to the SimRank results and evaluating the embedding similarity scores for known relationships in the context of the rest of the similarity scores. In the MITRE dataset, the description fields for APTs contain comments on other APTs which might be related or similar to the given APT. I manually extracted all of these relationships and decided to evaluate the similarity scores for these pairs in the context of all other APT similarity scores, embedding similarity scores and the shortest path distance in the graph. For CTIMiner, the dataset contains some relationships between reports in the dataset; linked reports are listed with the Indicators in the sub-object. Similar to the other sections, I will evaluate the embeddings in the order of the graphs they are based on.

5.4.1 MITRE full

This graph was the first graph structure chosen to examine APT similarity based on embeddings, and includes the full MITRE dataset with node categories such as attack patterns, malwares, mitigation techniques, kill-chain-phases and APTs. Table 5.3 shows the cosine similarity scores for the APTs with known similarity based on embeddings from the listed algorithms. In terms of the individual similarity scores, the algorithms seem fairly consistent in determining which nodes are more or less similar, i.e. a high similarity score from one algorithm means other algorithms also assign a relatively high similarity score and vice versa. Overall, DeepWalk outputs the highest mean similarity score for all the APT

[6] HARP: https://github.com/GTmac/HARP

pairs listed in the table, followed by node2vec(2,0.5) and node2vec(1,1). The node2vec(0.5,2) model seems to perform the worst in this assessment.

When comparing the table pairs to all APT similarities, however, it becomes obvious that the similarity scores for known similar pairs are actually lower than the APT mean, meaning that on average these pairs are less similar than a standard APT pair. While that is certainly concerning in terms of the validity of the embedding, it is important to consider the kind of information that is included in this embedding. APTs are only connected to attack patterns and malwares, resulting in embeddings that will likely see APTs as very similar that show similarity in these categories. An example of this pattern would be the pair of Moafee and DragonOK, which is known to use overlapping TTP, custom tools and, in the case of this particular graph, share an edge with PoisonIvy [15]. On the other hand, the connection between PROMETHEUM and NEODYMIUM is based on twin zero-day attacks with different malwares and victims [43], which would unfortunately not be reflected in the graph. Thus, it seems that the embeddings reflect the similarity which is evident in the graph and therefore possible to be encoded. Finally, the mean APT similarity is much higher than the overall embedding mean, meaning that all algorithms potentially pick up on the structural similarity between APT nodes versus other nodes.

Name	**Name**	**DeepWalk**	**node2vec(1,1)**	**node2vec(0.5,2)**	**node2vec(2,0.5)**
APT19	Deep Panda	0.7625	0.7025	0.5355	0.6844
Carbanak	FIN7	0.6692	0.5193	0.3719	0.4888
APT30	Naikon	0.4839	-0.1910	-0.2540	-0.1296
APT37	APT38	0.8042	0.6535	0.4015	0.7093
APT37	Lazarus Group	0.7567	0.6665	0.2290	0.6608
APT38	Lazarus Group	0.8375	0.7807	0.5359	0.8299
BlackOasis	NEODYMIUM	0.5137	-0.3037	-0.2586	-0.1399
Charming Kitten	Magic Hound	0.5546	0.1526	-0.1047	-0.1938
Cobalt Group	Carbanak	0.8350	0.8678	0.6956	0.8740
Dragonfly	Dragonfly2.0	0.5624	-0.2934	-0.4236	0.0067
DragonOK	Moafee	0.9122	0.8240	0.6358	0.8491
PROMETHEUM	NEODYMIUM	0.5177	0.0351	0.1565	-0.0930
Putter Panda	Scarlet Mimic	0.6284	0.3572	0.1567	0.2276
Winnti Group	Axiom	0.4574	-0.2066	-0.2225	0.0500
Winnti Group	APT17	0.4788	0.1650	0.4122	0.2636
Winnti Group	Ke3chang	0.5552	-0.2090	-0.0715	0.2206
Known APTs Mean		**0.6456**	**0.2825**	**0.1747**	**0.3318**
All APTs Mean		**0.6523**	**0.3844**	**0.2189**	**0.3864**
All Nodes Mean		0.2151	0.0149	0.0186	0.0130

Table 5.3. MITRE full graph cosine similarity comparison for known similar APTs. The parameters in parentheses for node2vec are (p,q).

Name	Name	Description	Distance
APT19	Deep Panda	sometimes tracked as same group	2
Carbanak	FIN7	both use Carbanak malware	2
APT30	Naikon	share some characteristics	4
APT37	APT38	sometimes tracked as same group	2
APT37	Lazarus Group	sometimes tracked as same group	2
APT38	Lazarus Group	sometimes tracked as same group	2
BlackOasis	NEODYMIUM	reportedly associated	5
Charming Kitten	Magic Hound	TTPs overlap	3
Cobalt Group	Carbanak	Carbanak malware, reportedly linked	2
Dragonfly	Dragonfly2.0	extent of actual overlap debated	3
DragonOK	Moafee	same TTPs, custom tools	2
PROMETHEUM	NEODYMIUM	victim and campaign characteristics	6
Putter Panda	Scarlet Mimic	use same IP addresses	4
Winnti Group	Axiom	both use Winnti malware[7]	3
Winnti Group	APT17	reportedly closely linked	3
Winnti Group	Ke3chang	reportedly closely linked	2

Table 5.4. Description of the relationship and shortest path distance based on the full MITRE graph between similar APTs according to MITRE APT descriptions.

In order to further assess the differences in similarities between the pairs in Table 5.3, I decided to look at the actual description of the relationship from the APT objects as well as look at the shortest path distance from the actual graph, i.e. the minimum amount of edges to be traversed to get from node A to node B. Table 5.4 lists the description of the relationship between the APT pairs as well the distance. Generally, groups which are 'sometimes tracked as same group' or share characteristics that are included in the graph have a lower distance between the two nodes. When comparing the two tables, it becomes clear that nodes with a lower shortest path distance were also found more similar by the embeddings. Thus, the embeddings seem to follow node distance as an indicator of similarity, which could be caused by the random walk sampling present in both node2vec and DeepWalk. Overall, it seems both embedding algorithms are fairly successful in embedding APTs judging by similarity and based on the information contained in the graph, with DeepWalk outperforming node2vec.

5.4.2 MITRE truncated

The second graph only included nodes in the APT, malware and attack pattern categories based on the MITRE dataset. This particular structure was chosen to explore subgraph embeddings based on the subgraph2vec algorithm, and to assess the effect of this technique on APT similarity scores. Similar to the previous section, we will assess the embeddings based on the known relationships between APTs.

Table 5.5 shows the similarity scores for both subgraph2vec embeddings as well as the SimRank similarity.

Name	Name	sg2v APT Subg.	sg2v full	SimRank
APT19	Deep Panda	0.9062	0.8623	0.0444
Carbanak	FIN7	0.9206	0.9408	0.0461
APT30	Naikon	0.8572	0.9368	0.0349
APT37	APT38	0.9289	0.9399	0.0450
APT37	Lazarus Group	0.8944	0.9721	0.0423
APT38	Lazarus Group	0.9224	0.9359	0.0449
BlackOasis	NEODYMIUM	0.4137	0.9829	0.0188
Charming Kitten	Magic Hound	0.4508	0.9554	0.0206
Cobalt Group	Carbanak	0.8766	0.9220	0.0496
Dragonfly	Dragonfly2.0	0.7223	0.9278	0.0204
DragonOK	Moafee	0.5296	0.9803	0.2489
PROMETHEUM	NEODYMIUM	0.2758	0.9751	0.0381
Putter Panda	Scarlet Mimic	0.8303	0.9173	0.0419
Winnti Group	Axiom	0.8559	0.9517	0.0301
Winnti Group	APT17	0.5768	0.9757	0.0287
Winnti Group	Ke3chang	0.7951	0.9308	0.0399
Known APTs Mean		**0.7845**	**0.9427**	**0.0497**
All APTs Mean		**0.7822**	**0.9421**	**0.0503**
All Nodes Mean		**0.6944**	**0.9352**	**-**
Embedding Mean		**0.5223**	**0.9220**	**-**

Table 5.5. MITRE truncated graph cosine similarity comparison of graph 2 embeddings for similar APTs based on APT descriptions. "APT Subg." means using subgraphs around APTs as multiple inputs, whereas "full" uses the full graph as the one input.

For both subgraph2vec techniques, it is immediately obvious that the similarity scores are much higher. Unfortunately, the scores are higher not just for the particular APTs, but also for all nodes and the entire embedding. This sentiment is especially true for the full graph embedding, where most similarity scores are above 0.9. Overall, since the averages and individual scores are all fairly close and I cannot discern a clear pattern from the scores, I do not have much confidence in this particular embedding. For the subgraph embeddings, the similarity scores follow a similar pattern as the full graph embeddings. In addition, the listed pairs' mean similarity is actually slightly higher than the APT average, although not by much. In terms of the node and full embedding mean, however, the APT mean is not that much higher when compared to the DeepWalk and node2vec embeddings. Finally, the SimRank similarity seems to suggest a low similarity overall between the all the pairs. The cause for this is probably the highly connected graph, as graph 2 has the highest average node degree between all four graphs at 15.56, which results in the low similarity values even at a decay rate of 0.9.

5.4.3 Combined Graph

The combined graph is the only APT-centered graph which includes Indicators in addition to APTs, malwares and attack patterns. Since only APT and malware pairs with Indicator connections were considered, only 19 APT nodes are present in the dataset as compared to the 94 in both MITRE graphs. Due to the low number of APTs, only two pairs with known similarity are included in the graph, rendering the previously used method of comparison rather ineffective. Nonetheless, Table 5.6 shows the embedding similarity results for DeepWalk and all four iterations of the GAE algorithm. At first glance, it seems that including features has a positive effect on overall APT similarity; GAE1 and GAE2 have a higher similarity score for the two pairs as well as the overall APT mean as compared to the rest of the models. Yet, GAE3 has the pairs' similarity score well above the APT Mean as well and all GAE models have an APT similarity mean which is higher than the overall node similarity mean, which means they seem to be able to detect the common structure among APT nodes. On the other hand, the same cannot be said for DeepWalk, unfortunately.

Name	Name	GAE1	GAE2	GAE3	GAE4	DeepWalk
APT37	Lazarus Group	1.0000	0.9370	0.8838	-0.0353	0.3387
Cobalt Group	Carbanak	0.9415	1.0000	0.7309	0.0063	0.2487
All APTs Mean		**0.6457**	**0.6720**	**0.5154**	**0.2050**	**0.2438**
All Nodes Mean		**0.0024**	**0.0090**	**0.0108**	**0.0035**	**0.2596**

Table 5.6. Cosine similarity comparison of graph 3 embeddings for similar APTs based on APT descriptions as well as descriptive statistics for the whole embedding. Only two APT pairs with a known similarity were present in this dataset.
GAE1: features, no dropout; GAE2: features, 0.2 dropout; GAE3: no features, 0.2 dropout; GAE4: no features, no dropout;

In order to further differentiate and explore the GAE models, I decided to visualize the cosine similarity scores in the form of a heatmap. Figure 5.6 shows a heatmap representation of the cosine similarity matrix of all APT nodes in the combined graph. For the first three models, it seems that there exist two clusters within the APTs: the Gamaredon Group, Silence, MuddyWater and APT34 cluster and all other APT nodes. The difference mostly lies in the similarity scores of the nodes within the cluster, as GAE1, GAE2 and GAE3 have a high similarity score between everyone in the big cluster, whereas the scores for GAE4 especially range from -0.5 to 1. In addition, GAE1, GAE2 and GAE3's relationships for a particular node within the cluster resemble lines, meaning that one node is mostly equally similar to all other nodes in the cluster, whereas the relationships in GAE3 seem much more independent. The GAE4 model, on the other hand, seems to form different clusters. Specifically, DarkHydrus, Orangeworm, APT41, APT37 and to a lesser extent FIN6 form one cluster, and Cobalt Group, Threat

Group-3390, Lazarus Group and TA505 can be grouped based on high similarity scores between all possible combinations of pairs. Although the clusters seem fairly obvious in this context, they could also be an artifact of limiting the graph to a small number of APTs and thereby excluding APTs which are similar to APTs from both clusters or arbitrary Indicator overlap due to limited data.

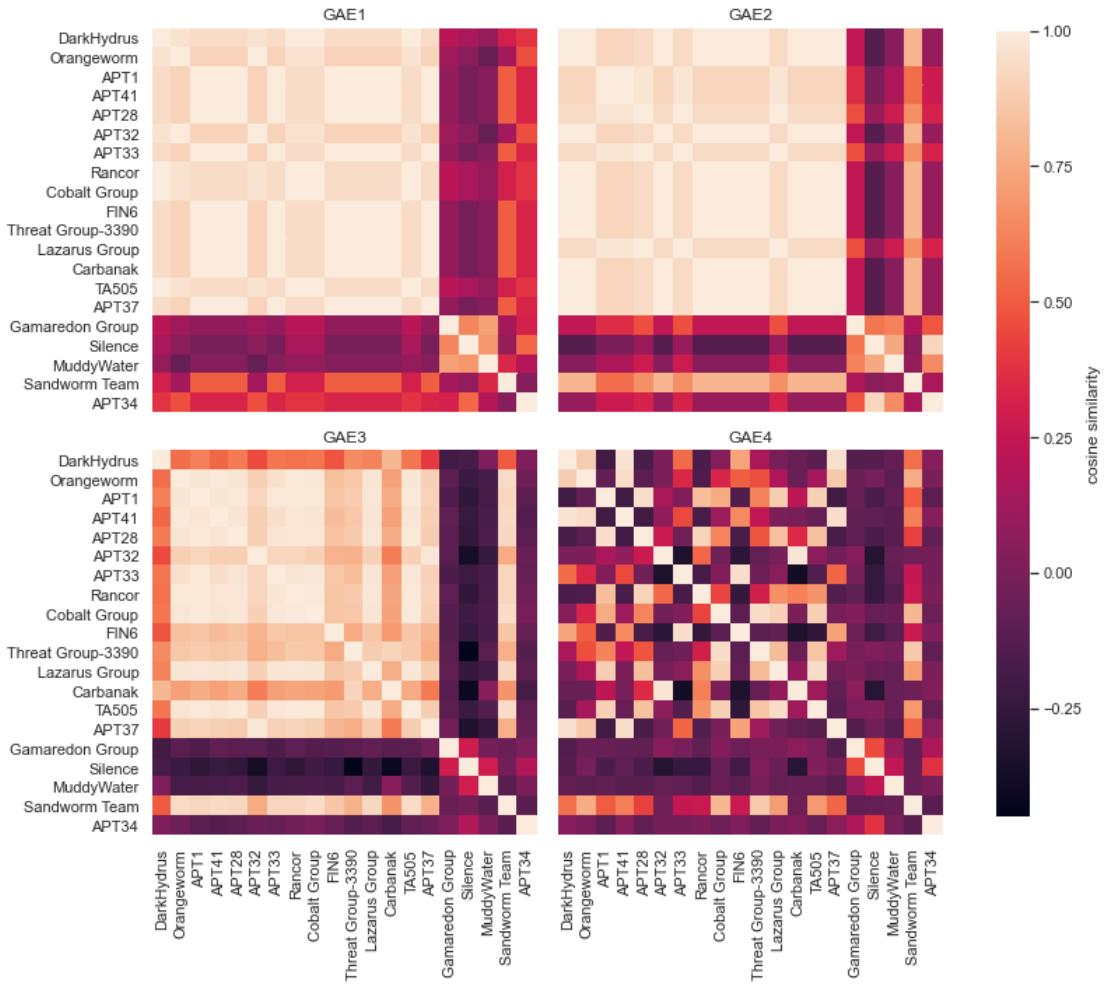


Figure 5.6. Heatmaps showing the cosine similarity for all APT nodes and for all four GAE variants: GAE1: features, no dropout; GAE2: features, 0.2 dropout; GAE3: no features, 0.2 dropout; GAE4: no features, no dropout;

Overall, it is difficult to choose the best model configuration based on this context, as it mostly depends on preference and the difference in similarity scores are mostly arbitrary as they all follow a similar pattern. GAE performs much better than DeepWalk on this graph even without features, and should be applied on similarly structured graphs with more or different data in the future.

5.4.4 APT Graph Comparison

Based on the discussion in Sections 5.4.1-5.4.3, it is inherently difficult to determine an optimal model given a certain graph structure, which could render choosing a best graph structure difficult given the available models. In general, both Indicator-less graph structures had models struggle to separate APTs from other nodes in terms of similarity. While this could be a reflection on graph structure, it could also be caused by Indicators inherently being low-degree nodes and as such vastly different from the generally high-degree malware and APT nodes. Regardless of cause, including Indicators seemed to have a positive effect on APT similarity overall, and GAE seemed to produce the best performing models in terms of APT similarity vs node similarity albeit on a small dataset.

With that being said, the difference in included nodes also has an adverse effect on the node similarities the models are able to calculate. In Section 5.4.3, I discussed the apparent cluster of Gamaredon Group, Silence, MuddyWater and APT34. In embeddings for both other graph, however, the similarity between these APTs were not significantly different from the APT mean and all four had different APTs with higher similarity. Thus, it seems that the cluster was caused by an inherently incomplete dataset as it excluded a lot of the other APTs, malwares and attack patterns included in MITRE, and thus created an embedding which was inherently biased due to the small dataset. It seems that including the rest of the MITRE dataset nodes without Indicator information might have been beneficial, although that might have introduced another bias in the form of missing Indicator information.

5.4.5 CTIMiner

For the CTIMiner graph, the embeddings are evaluated based on report similarity instead of APT similarity, and the evaluated report pairs are actually included in the dataset. As there are 82 report pairs, I will only list the results measures instead of individual pairs. Table 5.7 displays the mean, maximum and minimum similarity scores between the given report pairs for the different embedding algorithms. I also attempted to calculate the SimRank similarity, but the graph was too large making the calculation too computationally expensive. Based on the table, it seems that HARP with node2vec and LINE produced the best results. Although node2vec's average similarity is higher than LINE's, the range of scores is also the highest across all embeddings, meaning that the embedding might be a somewhat unstable. DeepWalk without HARP performed decently albeit worse than the two mentioned embeddings, and HARP(DeepWalk) performed the worst in terms of both mean and maximum score. The fact that DeepWalk with HARP performed much worse than DeepWalk by itself can reflect

on the efficiency of HARP, but it is fairly likely that the lack of performance is due to the default hyperparameters used for HARP(DeepWalk). Either way, DeepWalk seems to be outperformed by node2vec and HARP in this case.

Measure	**HARP(node2vec)**	**HARP(LINE)**	**HARP(DeepWalk)**	**DeepWalk**
Known Reports Max	0.9756	0.9838	0.8440	0.9245
Known Reports Mean	**0.9133**	**0.8750**	**0.3005**	**0.6051**
Known Reports Min	-0.2462	0.5323	-0.1074	0.3646
All Reports Mean	**0.2742**	**0.3850**	**0.0476**	**0.1042**
Node Mean	0.3177	0.4922	0.0181	0.0859

Table 5.7. CTIMiner cosine similarity comparison for included report edges.

Section 6

Conclusion and Future Scope

Section 5.4 shows that graph representation learning can be used to gain insight from static cyber threat intelligence data with a variety of different graph structures and algorithms. Graphs can easily be constructed from Indicator- and STIX-based data formats by creating and concatenating edge lists. In terms of modelling techniques, traditional approaches such as DeepWalk and node2vec, subgraph-based approaches such as subgraph2vec and feature-inclusive approaches such as GAE seem to be able to distinguish APT nodes from other nodes based on node embedding cosine similarity. In addition, heatmap visualizations of similarity results allow for closer analysis of similarity patterns and potential clusters of similar APTs. While only shown for graph 3 due to size constraints, this technique is applicable to all embedding evaluations based on similarity. Furthermore, representation learning can be used to assess report similarity, which could potentially help automate research tasks such as finding similar reports based on report or indicator information.

In terms of graph structure, it seems that the proposed structures were all ostensibly viable, and the similarity results and embeddings largely depend on the context of included information and graph structure. As such, choosing a graph structure for a particular task largely depends on the context in which one wants to compare certain entities. For APT similarity, both the full graph and the subgraph approach adaptation resulted in similar scores, with the scores for the combined graph being affected by a more limited set of APTs due to Indicator constraints. In addition to the modelling goals, it is also necessary to take the respective biases into account which come from certain structural choices as evident in the APT clusters from the combined graph. Thus, the graph structure, included data and features all shape the context in which the modelling goal should be evaluated.

Overall, this report serves as a starting point for the application of graph representation learning to cyber threat intelligence data. With access to more complete datasets as well as expert knowledge on the validity of the embedding results, the methods and process detailed in this report could be extended to build products and automate previously manual CTI analysis workflows.

6.1 Limitations and Future Scope

This project was completed with two limitations: the available hardware and data. In terms of hardware, the computational complexity for some of the embedding methods is rather high and thus did not allow for much hyperparameter optimization, which is why some of the models were created using the default parameters or intuition. Moreover, assessing the embedding results, especially of large graphs such as CTIMiner, was also computationally expensive, which meant I was unable to assess the embeddings and then improve the methodology based on the results. As a result, most of the modelling choices are based on instinct rather than due process. Therefore, the modelling process introduced in this project could be replicated with more intense model selection and hyperparameter tuning techniques, which would likely result in more accurate embeddings.

In terms of the data, the scope of relationships that are possible to include in a graph were ultimately very limited. As an example, the data does not include campaign or victim characteristics, which could have been helpful to assess APT similarity. Likewise, the temporal context of relationships is mostly missing. As an example, an APT that used a certain malware once in a small campaign has the same edge to the malware as an APT that created the malware and has used it with adaptation for years as their primary means of operation. When it comes to indicators, the data is also missing context. If two indicators are employed with the same malware by the same APT, it is not clear if they occurred together or in separate attacks, or both. Thus, it is impossible to group indicators other than doing it arbitrarily. Attack-specific indicator data coupled with APT, malware and possibly attack pattern information would allow someone to create attack-based subgraphs to model attack similarly and potentially predict the likely attacker and malware used. In general, adding more context or building graphs from different contexts would allow one to create a more holistic model of the similarity between different APTs or other categories such as different malwares. One recent example of different context is the clustering of attack patterns based on tactics for the MITRE dataset [39], which could be replicated using embeddings in order to further prove the validity of these methods.

The most severe limitation of the data, however, is that it is based on potentially incomplete data. Besides excluding APTs or malwares, it might also exclude relationships between APTs, malwares or other entities, which are already present in the graphs. This exclusion could be harmful in that it skews similarity without my knowledge and thus invalidates the results. Without a lot of manual research or a different data gathering process, this limitation is also fairly difficult to eliminate.

Finally, in order to build a generalizable embedding model from these graphs for previously unseen nodes, one would have to use inductive embedding algorithms, meaning that the algorithm learns an embedding function independent from the full graph [8]. Algorithms like GraphSage [14] learn embedding functions based on node text features similar to GAE and calculated node neighborhood features to construct embeddings, allowing it to apply this embedding function to previously unseen nodes. Using inductive embedding models would allow for the application of the embedding to new attacks without creating a new embedding, which would greatly reduce time and computational constraints and potentially increase model stability depending on the quality of the initial training dataset.

Bibliography

[1] Bijaya Adhikari et al. *Distributed Representation of Subgraphs*. 2017. arXiv: 1702.06921 [cs.SI].

[2] Alienvault. *Create New Pulse*. 2020. URL: https://otx.alienvault.com/pulse/create (visited on 23/06/2020).

[3] Sean Barnum and Amit Sethi. *Attack Patterns as a Knowledge Resource for Building Secure Software*. Tech. rep. Cigital Inc., 2007. URL: https://capec.mitre.org/documents/Attack_Patterns-Knowing_Your_Enemies_in_Order_to_Defeat_Them-Paper.pdf.

[4] Fabian Böhm, Florian Menges and Günther Pernul. 'Graph-based visual analytics for cyber threat intelligence'. In: *Cybersecurity* 1.1 (2018). ISSN: 2523-3246. DOI: 10.1186/s42400-018-0017-4.

[5] S. Bonner et al. 'Evaluating the quality of graph embeddings via topological feature reconstruction'. In: *2017 IEEE International Conference on Big Data (Big Data)*. 2017, pp. 2691–2700.

[6] Matt Bromiley. *Threat Intelligence: What It Is, and How to Use It Effectively*. Tech. rep. SANS Institute, 2016.

[7] Sergio Caltagirone, Andrew Pendergast and Christopher Betz. 'The Diamond Model of Intrusion Analysis'. In: 2013. URL: http://www.activeresponse.org/wp-content/uploads/2013/07/diamond.pdf.

[8] Ines Chami et al. *Machine Learning on Graphs: A Model and Comprehensive Taxonomy*. 2020. arXiv: 2005.03675 [cs.LG].

[9] Haochen Chen et al. *HARP: Hierarchical Representation Learning for Networks*. 2017. arXiv: 1706.07845 [cs.SI].

[10] The MITRE Corporation. *Contribute*. 2020. URL: https://attack.mitre.org/resources/contribute/ (visited on 23/06/2020).

[11] The MITRE Corporation. *USAGE*. 2020. URL: https://github.com/mitre/cti/blob/master/USAGE.md (visited on 23/06/2020).

[12] M. Girvan and M. E. J. Newman. 'Community structure in social and biological networks'. In: *Proceedings of the National Academy of Sciences* 99.12 (2002), pp. 7821–7826. DOI: 10.1073/pnas.122653799.

[13] Aditya Grover and Jure Leskovec. 'Node2vec: Scalable Feature Learning for Networks'. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '16. San Francisco, California, USA: Association for Computing Machinery, 2016, pp. 855–864. ISBN: 9781450342322. DOI: 10.1145/2939672.2939754.

[14] William L. Hamilton, Rex Ying and Jure Leskovec. *Inductive Representation Learning on Large Graphs*. 2017. arXiv: 1706.02216 [cs.SI].

[15] Thoufique Haq et al. *OPERATION QUANTUM ENTANGLEMENT*. Tech. rep. FireEye Labs, 2014. URL: https://www.fireeye.com/content/dam/fireeye-www/global/en/current-threats/pdfs/wp-operation-quantum-entanglement.pdf.

[16] Trevor Hastie, Robert Tibshirani and Jerome Friedman. *The elements of statistical learning: data mining, inference and prediction*. 2nd ed. Springer, 2017. URL: https://web.stanford.edu/~hastie/ElemStatLearn/printings/ESLII_print12.pdf.

[17] Lawrence Holder et al. 'Graph-Based Relational Learning with Application to Security'. In: *Fundam. Inf.* 66.1–2 (2004), pp. 83–101. ISSN: 0169-2968.

[18] Eric Hutchins, Michael Cloppert and Rohan Amin. 'Intelligence-Driven Computer Network Defense Informed by Analysis of Adversary Campaigns and Intrusion Kill Chains'. In: *Leading Issues in Information Warfare Security Research* 1 (Jan. 2011).

[19] Gareth James et al. *An Introduction to Statistical Learning: With Applications in R*. Springer Publishing Company, Incorporated, 2014. ISBN: 1461471370.

[20] Glen Jeh and Jennifer Widom. 'SimRank: A Measure of Structural-Context Similarity'. In: *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '02. Edmonton, Alberta, Canada: Association for Computing Machinery, 2002, pp. 538–543. ISBN: 158113567X. DOI: 10.1145/775047.775126.

[21] Bret Jordan, Rich Piazza and Trey Darley. *STIX™ Version 2.1*. Tech. rep. OASIS, Mar. 2020. URL: https://docs.oasis-open.org/cti/stix/v2.1/cs01/stix-v2.1-cs01.pdf.

[22] Diederik P Kingma and Max Welling. *Auto-Encoding Variational Bayes*. 2013. arXiv: 1312.6114 [stat.ML].

[23] Thomas N. Kipf and Max Welling. *Semi-Supervised Classification with Graph Convolutional Networks*. 2016. arXiv: 1609.02907 [cs.LG].

[24] Thomas N. Kipf and Max Welling. *Variational Graph Auto-Encoders*. 2016. arXiv: 1611.07308 [stat.ML].

[25] Information Technology Laboratory. *malware - Glossary*. 2020. URL: https://csrc.nist.gov/glossary/term/malware.

[26] Steven Launius. *Evaluation of Comprehensive Taxonomies for Information Technology Threats*. Tech. rep. SANS Institute, 2018.

[27] Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze. *Introduction to Information Retrieval*. USA: Cambridge University Press, 2008. ISBN: 0521865719.

[28] Florian Menges and Günther Pernul. 'A comparative analysis of incident reporting formats'. In: *Computers Security* 73 (2017), pp. 87–101. DOI: 10.1016/j.cose.2017.10.009.

[29] Tomas Mikolov et al. 'Distributed Representations of Words and Phrases and Their Compositionality'. In: *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*. NIPS'13. Lake Tahoe, Nevada: Curran Associates Inc., 2013, pp. 3111–3119. URL: https://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf.

[30] Tomas Mikolov et al. *Efficient Estimation of Word Representations in Vector Space*. 2013. arXiv: 1301.3781 [cs.CL].

[31] Frederic Morin and Yoshua Bengio. 'Hierarchical probabilistic neural network language model'. In: *AISTATS'05*. 2005, pp. 246–252. URL: http://www.iro.umontreal.ca/~lisa/pointeurs/hierarchical-nnlm-aistats05.pdf.

[32] Annamalai Narayanan et al. 'graph2vec: Learning Distributed Representations of Graphs'. In: *CoRR* abs/1707.05005 (2017). arXiv: 1707.05005. URL: http://arxiv.org/abs/1707.05005.

[33] Annamalai Narayanan et al. *subgraph2vec: Learning Distributed Representations of Rooted Sub-graphs from Large Graphs*. 2016. arXiv: 1606.08928 [cs.LG].

[34] Surya Nepal, Daegeon Kim and Huy Kang Kim. 'Automated Dataset Generation System for Collaborative Research of Cyber Threat Analysis'. In: *Security and Communication Networks* (2019). DOI: 10.1155/2019/6268476.

[35] M. E. J. Newman. *Networks: An Introduction*. Oxford; New York: Oxford University Press, 2010. ISBN: 9780199206650 0199206651.

[36] Bryan Perozzi, Rami Al-Rfou and Steven Skiena. 'DeepWalk'. In: *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '14* (2014). DOI: 10.1145/2623330.2623732.

[37] Pulsedive. *Feed*. 2020. URL: https://pulsedive.com/about/?q=feed (visited on 23/06/2020).

[38] Tobias Schnabel et al. 'Evaluation methods for unsupervised word embeddings'. In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Lisbon, Portugal: Association for Computational Linguistics, Sept. 2015, pp. 298–307. DOI: 10.18653/v1/D15-1036. URL: https://www.aclweb.org/anthology/D15-1036.

[39] Rawan Al-Shaer, Jonathan M. Spring and Eliana Christou. *Learning the Associations of MITRE ATTCK Adversarial Techniques*. 2020. arXiv: 2005.01654 [cs.CR].

[40] Alen Šimec and Magdalena Magličić. 'Comparison of JSON and XML Data Formats'. In: 2014.

[41] Blake E. Strom et al. *MITRE ATTCK^{TM}: Design and Philosophy*. Tech. rep. McLean, VA: The MITRE Corporation, 2018. URL: https://attack.mitre.org/docs/ATTACK_Design_and_Philosophy_March_2020.pdf.

[42] Jian Tang et al. 'LINE: Large-scale Information Network Embedding'. In: *Proceedings of the 24th International Conference on World Wide Web - WWW '15* (2015). DOI: 10.1145/2736277.2741093.

[43] Microsoft Defender ATP Research Team. *Twin zero-day attacks: PROMETHIUM and NEODYMIUM target individuals in Europe*. 2016. URL: https://www.microsoft.com/security/blog/2016/12/14/twin-zero-day-attacks-promethium-and-neodymium-target-individuals-in-europe/?source=mmpc.

[44] Muhammad Usman et al. 'A Survey on Representation Learning Efforts in Cybersecurity Domain'. In: *ACM Comput. Surv.* 52.6 (Oct. 2019). ISSN: 0360-0300. DOI: 10.1145/3331174.

[45] Cynthia Wagner et al. 'MISP: The Design and Implementation of a Collaborative Threat Intelligence Sharing Platform'. In: *Proceedings of the 2016 ACM on Workshop on Information*

Sharing and Collaborative Security. WISCS '16. Vienna, Austria: Association for Computing Machinery, 2016, pp. 49–56. ISBN: 9781450345651. DOI: 10.1145/2994539.2994542.

Section A

Appendix A

```
[{'industries': ['Manufacturing', 'Healthcare'],
  'tlp': 'white',
  'description': '',
  'created': '2020-03-30T18:54:28.683000',
  'tags': ['Kwampirs', 'RAT', 'malware', 'ICS'],
  'malware_families': ['Kwampirs'],
  'modified': '2020-03-30T18:54:28.683000',
  'author_name': 'Cyber_Hat',
  'public': 1,
  'extract_source': [],
  'references': ['https://blog.reversinglabs.com/blog/unpacking-kwampirs-rat'],
  'targeted_countries': ['United States'],
  'indicators': [{'indicator': 'bc34ba385fa15b469a8e0d10c2985d66f868b530',
    'description': '',
    'title': '',
    'created': '2020-03-30T18:54:28',
    'is_active': 1,
    'content': '',
    'expiration': None,
    'type': 'FileHash-SHA1',
    'id': 312446352},
   {'indicator': '93657729f5c2e5310fa6d3c27bd40e158505ad4c',
    'description': '',
    'title': '',
    'created': '2020-03-30T18:54:28',
    'is_active': 1,
    'content': '',
    'expiration': None,
    'type': 'FileHash-SHA1',
    'id': 324540662},
```

(a) Alienvault OTX Pulse in JSON-like Python dictionary.

```
<?xml version="1.0" ?>
<CTIMinerDataset>

        <Event>

                <id>4161</id>

                <date>2014-01-16</date>

                <info>FTA 1001 FINAL 1.15.14.pdf</info>

                <Attribute>

                        <item>

                                <category>Other</category>

                                <comment/>

                                <value>FTA 1001 FINAL 1.15.14.pdf</value>

                                <type>comment</type>

                                <id>50724</id>

                        </item>

                        <item>

                                <category>External analysis</category>

                                <comment/>

                                <value>netsat.exe</value>

                                <type>filename</type>

                                <id>50725</id>

                        </item>
```

(b) Sample CTIMiner report data in XML format.

Figure A.1. The images show examples of JSON-like and XML data structures from the Alienvault OTX and CTIMiner datasets.