CARLETON UNIVERSITY

SCHOOL OF
MATHEMATICS AND STATISTICS

HONOURS PROJECT

TITLE: Network Alert Graph Construction and Community Detection

AUTHOR: Ching-wen Wang

SUPERVISOR: Shirley Mills

DATE: April 29, 2020

Network Alert Graph Construction and Community Detection

Math 4905 - Honours Project

Ching-wen Wang (100928606)

Carleton University
Under the supervision of Dr. Shirley Mills
April - 2020

Acknowledgements

I would like to thank my supervisor Dr. Shirley Mills for her support and guidance. I would also like to acknowledge that this honours project is a part of an industry project collaboration with a Toronto-based company and I worked closely with scientists from the company as well as Master student Benjamin Burr as part of his thesis.

Contents

1. Introduction

1.1 Motivation

Intrusion Detection Systems (IDS) are software applications that organizations install in their network to monitor their network and systems for malicious activity. The output from an IDS is typically collected and processed by a Security Information Management and Event Management (SIEM) System [32]. These SIEM systems are usually handled by a human operator who will use the information provided by the system to detect malicious campaigns.

Within an IDS there are usually multiple anomaly and rule based detectors that generate alerts. With the prevalence of Advance Persistent Threats (APTs), network attacks are usually complex and penetrated the system through multiple stages from initial access to exfiltration. This means that a single detector cannot capture the entire attack campaign. Another problem with IDS is that they generate a massive amount of alerts, often overwhelming both their system and the human operators handling them.

An active area of research is to develop algorithms and systems that can deal with this problem, whether it is through alert aggregation and synthesis or malicious alert modelling and classification.

The goal of this project is to be able to effectively use the alerts generated by intrusion detection systems and translate these alerts into a format that makes it easier for human operators to action upon cyber events.

1.2 Background

1.2.1 Intrusion Detection Systems

Similarly to antivirus software, firewall, and access control schemes, IDSs are another layer of defense used to protect computer systems from hackers and external parties. The goal of IDSs is to identify any kind of malicious activity that can cause damage to a network system.

The literature on IDSs can be roughly broken down into three different groups:

- **Signature-based IDS**: This type of IDS uses a signature database of known attacks.
- **Anomaly-based IDS**: This type of IDS uses different techniques to model benign and malicious behaviours.
- **Alert Correlation**: Alert correlation techniques are modules built on top of anomaly-based and signature-based IDSs to analyze alerts in the context they appear in.

There are numerous reviews and surveys that try to capture the whole intrusion detection literature such as [47], [19], [10], and [25].

Some reviews focus solely on anomaly-based IDSs: Lazarevic et al. [22] compared several anomaly detection techniques and provide evaluations based on the DARPA 1998 dataset [28] of network connections; Ahmed et al. [2] presented an in-depth analysis of different categories of anomaly detection techniques and discussed the research challenges of the datasets used for network intrusion detection; Agrawal and Agrawal [1] focused on anomaly detection that employs data mining techniques.

1.2.1.1 Signature-based

A signature is a pattern created by an IDS that corresponds to a known attack or threat. To detect intrusion attempts, signature-based IDSs inspect contents of a binary file or network packets to find matches to the intrusion signatures [40].

Multiple papers study techniques for signature matching and content inspection. Pao et al. [40] presented a memory-based N-finite automata (NFA) architecture for regex matching. Lin et al. [26] combined two commonly used string-matching methods for virus detection, namely the Aho Corasick algorithm and the backward hashing algorithm.

Early signature creation requires experts to manually process malicious events and to create signatures based on their knowledge of the system and of the threat. This manual process is inefficient and costly. As a result, techniques for automatic signature generation were developed. Automatic attack signatures can be generated through text pattern matching techniques [21], [46]. Statistical methods such as the latent Dirichlet allocation-algorithm [23] and hidden Markov models [20] are also used for automatic rule generation. There are also automatic signature generation methods for specific types of attack: [20] and [50] both presented methods for network worm detection and automatic signature generation.

Signature-based IDS are employed in numerous network monitoring applications such as Snort[44], NetSTAT[52], Bro[41], and Suricata[1].

Signature-based IDSs detection are often precise and accurate for capturing known attacks [40]. However, there are still areas of weaknesses in this type of system [10]:

- Signature-based IDSs use a massive amount of resources because they have to compare each event against many signatures in the database;
- Signature-based IDSs can generate a high percentage of false positives alerts from poorly formed signatures;
- Signature-based IDSs can only identify a single type of attack when modern attacks often involve multiple attack steps;
- Signature-based IDSs cannot detect unknown attacks or attacks built as a variant of known attacks.

1.2.1.2 Anomaly-based

Anomaly-based IDSs detect deviations from known behaviours. This type of IDS requires training a model of normal behaviours and detects deviations from the normal model in the observed data [22].

[1] https://suricata-ids.org/

Anomaly detection algorithms can be broken down into two different categories:

- Statistical Methods
- Machine Learning Methods
  - Supervised
  - Unsupervised

Statistical Methods

Statistical methods typically utilize outlier detection in different contexts. In multiple papers, normal data were modelled over a period of time before time-series analysis was applied to said data to find anomalies [56], [53]. Ye et al. [59] presented a multivariable statistical process to detect host-based intrusions. Another approach for finding anomalous events is through finding rare combinations of features through conditional probability, like shown in [48] and [8].

Machine Learning Methods

As machine learning techniques become more popular, more modern anomaly-based IDS utilizes machine learning techniques to find anomalies in data.

Supervised To employ supervised machine learning technique in IDSs, it is assumed that the user has access to labeled data of the normal and of the attack behaviours.

Popular machine learning techniques such as Support Vector Machine (SVM) and Decision Tree (DT) have been applied to anomaly detection. Li et al. [24] used SVM for anomaly detection and improved its performance by custom feature removal methods. SVM has also been used in combination with other methods. Khan et al. [17] presented a method using hierarchical clustering that improved iteratively with SVM. Lin et al. [27] propose a method that combines SVM and DT. Hu et al. [13] proposed a framework to construct a model using Particle Swarm Optimization and SVM and then utilizes the Adaboost algorithm to improve classification results. Other methods used for anomaly detection are: DT/C5 classification [18], artificial neural network [54], Kohonenen net [45], genetic fuzzy system [9], using naive Bayes classifier [20], hierarchical hidden Markov model [15], and Boltzmann machine [11].

Unsupervised Unsupervised machine learning techniques for anomaly detection do not use labeled attack data. It is assumed in most papers that normal traffic data is available and it can be used to train models for normal behaviours.

Different types of clustering algorithms are presented in different papers for anomaly detection. Liu et al. [30] proposed using nearest-neighbour clustering with genetic optimization for detecting anomalies. Hyun Oh et al. [39] proposed using a density-based clustering algorithm (DBSCAN) for anomaly detection.

One-class SVM is a special type of unsupervised SVM, which Wang et al. [55] use. Tao at el. [51] and Kang et al. [16] proposed using Support Vector Data Description (SVDD), which is a variant of SVM, using a spherical class boundary for classification.

Lastly, Liu et al. [29] proposed using the isolation forest algorithm for anomaly detection. This is a tree-based algorithm that builds multiple trees to represent normal data. Sun et al. [49] proposed a method to extend isolation forest algorithm for categorical data.

Anomaly-based IDSs are generally very effective for detecting unknown attacks. However, it is difficult to keep an accurate profile on changing behaviours of users and attackers [47]. Several research challenges are still present for anomaly-based IDS:

- Anomaly-based IDSs often work with network data that are complex, imbalanced, and with large number of features [47];
- Anomaly-based IDSs generate huge amounts of alerts with a high percentage of false positives [58];
- Anomaly-based IDSs have to adapt to the ever changing behaviour of network traffic data [47];
- Anamaly-based IDSs' decision boundary between normal and malicious events is difficult to establish, given the distribution of the data being highly imbalanced as anomalies are rare occurrences [10].

1.2.1.3 Alert Correlation

As new attacks are constantly evolving and network systems are becoming more complicated, IDSs with a single detector have limited capabilities in capturing malicious attacks.

Alert correlation techniques are proposed to fix two issues encountered by IDSs: 1) the enormous amount of alerts they created and 2) their incapability to detect multi-step attack scenarios. Alert correlation techniques focus on discovering relationships between different alerts to provide security analysts with aggregated alerts and their contexts.

Existing techniques can be roughly divided into four categories [58]:

- Similarity based correlation
- Predefined attack scenarios
- Prerequisites and consequences
- Integrated technique

Similarity Based Correlation

Alerts generated by IDSs usually contain several attributes and a similarity score can be calculated between two alerts based on their attributes. This similarity score can then be used for alert clustering and aggregation. Similarity based correlation methods utilize different clustering and aggregation techniques to effectively reduce the total number of alerts.

Clustering techniques for alert correlation are shown in multiple papers. Maggi et al. [31] aggregated alerts together using timestamps of alerts and with a fuzzy clustering technique. Almamory and Zhang's approach [3] focused on identifying the root cause of large groups of alerts through clustering. With this approach, a generalized alert is created for each cluster and a new alert is assigned to a cluster using the

nearest-neighbour clustering method. Pei et al. [42] proposed a correlation module that constructs a uniform vector representation of the connections between alerts. Then a graph is constructed through these alert connections and a community clustering technique is applied to the graph. Haas and Fischer [12] utilized IPs and ports associated with each alert in order to cluster alerts. An alert graph is built based on these alerts and community clustering method is applied to the alert graph.

Predefined Attack Scenarios

An attack scenario can be user-defined or learned from a training dataset. Alert correlation techniques based on predefined attack scenarios map alerts to different steps in an attack scenario to detect multi-step attack campaigns.

Jan et al. [14] proposed a classification module that utilizes human constructed rule classes to classify alert patterns into malicious or benign behaviours. Millajerdi et al. [34] proposed an IDS named HOLMES that tracks the information flow of low-level entities by representing them in a graph. These graphs are mapped to a high-level attack scenario graph representing the information flow of an attack. Noel et al.[38] proposed building attack scenarios by explicitly defining network vulnerability relationships into an attack graph. This attack graph model can be manually generated by an expert or automatically by external software.

Prerequisites and Consequences

Prerequisites and consequences-based correlation methods rely on predefined logical connections between alerts. A correlation graph can be constructed through these relations to model attack scenarios.

Ning and Xu [37] presented a method that creates prerequisites and consequences logic units to correlate alerts and construct known attacks through alert graphs. Zhou et al. [60] proposed a method that constructs logical representations of requirements for attacks to correlate attacks. Alerhani et al. [5] presented an IDS that combines a knowledge-base of prerequisites and consequences for attacks and statistical correlation.

Integrated Technique

An Alert correlation method that uses integrated technique automatically correlates alerts to data from different sources in the system. Porras et al. [43] proposed a mission-impact-based approach that utilizes knowledge databases and a sequence of alert processing steps to automatically correlate alerts from different system sources such as multiple IDSs and firewalls. Xu and Ning [57] presented a system that utilized trigger events from several security systems and utilized pre-conditions and post-conditions to build attack scenarios.

1.2.2 Community Detection

Community detection algorithms focus on finding a partition of a graph such that each partition is densely connected within the partition and sparsely connected to the other parts of the graph. Community detection algorithms are often used as a part of alert correlation methods for finding clusters of alerts within an alert graph.

Modularity is a metric that is used to measure the quality of such partition of the graph. Formally, modularity is defined by [35],

$$Q = \frac{1}{2m}\sum_{i,j}[A_{ij} - \frac{k_i, k_j}{2m}\delta(c_i, c_j)]$$

where A_{ij} represents the weight of the edge between i and j, $k_i = \sum_j A_{ij}$ represents the sum of all the weight of edges adjacent to node i, $m = \sum_{ij} A_{ij}$ represents the total weight of the graph, c_i represents the community that vertex i is assigned to, and where $\delta(c_i, c_j) = 1$ if $c_i = c_j$ or 0 otherwise.

While finding the partition of the graph with the optimal modularity is known to be computationally intractable, numerous heuristic techniques are invented to solve this problem. I will explore two community detection algorithms for this project: the Louvain algorithm, and the Clauset-Newman-Moore greedy modularity maximization algorithm.

Louvain Algorithm [6]

The Louvain algorithm was developed by a group of researchers from the University of Louvain. This is a heuristic algorithm that optimizes the modularity of the overall graph.

This algorithm is divided into two phases and these two phases are repeated iteratively until the mudolarity of the graph can no longer be improved.

Algorithm 1: Phase 1 of Louvain algorithm

Result: A weighted graph with each node assigned to a community.
Data: A weighted graph $G = (V, E)$.
Initialization: Every node in graph G belongs to its own community.

```
modularity_change ⟵ 1;
while modularity_change > 0 do
    graph_modularity ⟵ Evaluate modularity of the current graph;
    for Node i in Graph do
        maximum_change ⟵ 0;
        community ⟵ community of node i;
        for Neighbour node j of i do
            change ⟵ gain of modularity by removing node i
              from its community and placing it in node j's community;
            if change > maximum_change then
                community ⟵ community of node j;
                maximum_change ⟵ change;
            end
            if maximum_change > 0 then
                community of node i ⟵ community;
            end
            modularity_change ⟵ graph_modularity - modularity of the current graph
        end
    end
end
```

For phase 2 of the Louvain algorithm, we build a new graph G' with the communities created from phase 1 as nodes in the new graph and the edges between two community nodes are calculated by the total weight of the edges going from one community to the other community.

Algorithm 2: Phase 2 of Louvain algorithm

Result: New Graph $G' = (V, E)$

Data: A weighted graph with community labels $G = (V, E)$ that is the output from phase 1 of the Louvain algorithm.

Initialization: Create new graph G' such that each community in G is a node in G'

```
for community c in G do
    for community d in G do
        for uv and edge in G such that u ∈ community c and v ∈ community d do
            if edge cd ∉ G' then
                Add edge cd to G';
                weight(cd) ⟵ 0;
            end
            weight(cd) ⟵ weight(cd) + weight(uv) ;
        end
    end
end
```

The new graph G' that is the output of phase 2 can go through phase 1 of the Louvain algorithm again. This process continues iteratively until we cannot get any more increase to the modularity of the graph.

Clauset-Newman-Moore Greedy Modularity Maximization [7]

This greedy modularity maximization algorithm can be classified to be a type of agglomerative hierarchical clustering method, where the communities are iteratively joined together to maximize modularity.

The initial algorithm was first proposed in [36] then the current version of the greedy modularity maximization was improved by using more efficient data structures [7]. For the purpose of this project, I will describe the general algorithm described in [36].

Algorithm 3: Clauset-Newman-Moore greedy modularity maximization algorithm

Result: Graph $G = (V, E)$ with each node belonging to a community

Data: A weighted graph $G = (V, E)$

Initialization: Each node in graph G belongs to its own community.

```
while TRUE do
    max_change ⟵ 0 ;
    community_pair ⟵ nil;
    for every unique pairs of communities c,d in G where c ≠ d do
        change ⟵ change in modularity of the entire graph by combining community c and d;
        if change > max_change then
            max_change ⟵ change;
            community_pair ⟵ (c, d)
        end
    end
    if community_pair ≠ nil then
        G ⟵ change all nodes in communities of community_pair to be in the same community;
    else
        break;
    end
end
```

2. Methodology

2.1 Problem Statement

For this project, a Toronto-based cyber security company provided the network connection data and network alert data for two months. This Toronto-based company developed a IDS software product that is currently deployed in multiple companies' network system. This IDS uses multiple anomaly detectors and rule-based detectors that raise several alerts to potentially problematic traffic within the network. One problem with IDSs is that it generates a massive amount of data that is not possible for human security analysts to review and action upon all the alerts that are generated.

The goal of this project is to automatically group and synthesize network alerts and network connections data together in a meaningful way so that security analysts can investigate a reasonable amount of possible security incidents.

2.2 Data Set

As mentioned in the problem statement, the data used in this work was provided by a Toronto-based cyber security company. The data comes from one of their client company's network traffic for the two months of September and October 2019. The monitored company has around 50 employees using primarily Windows hosts, and its network includes several internal servers. Network taps capture traffic between the internal gateway, as well as traffic between internal hosts. The traffic is analyzed by Zeek [1], an open-source traffic analyzer tool, to produce connection events. These events, as well as other events produced by Zeek, host-based events, and events from other sources are ingested by the Toronto company's own system, which produces the alerts.

The data set has approximately 67.5 million events from the two month period and has raised approximately 37,000 alerts.

Although no known, successful attacks occurred during the monitored period, there was frequent scan and brute-force attempts from external hosts as well as non-malicious anomalous behaviour.

2.2.1 Connection Events

Each connection event produced by Zeek represents a bi-directional TCP, UDP, or ICMP connection. For UDP, a connection represents a flow of packets between two IP/port pairs. For ICMP, a connection is a sequence of packets between two IPs, with the same ICMP message type and code or packets representing an ICMP exchange, like an echo request (ping).

[1] https://www.zeek.org/

2.2.2 Network Alerts

The alerts come from a mixture of sources. Some are produced by Suricata[2], a well-known IDS. Others are produced by the Toronto company's rules engine and anomaly detectors. Suricata monitors the same network traffic that produced the connection event dataset, while the Toronto company's system produces alerts based on both network events and host-based events, such as standard Windows Sysmon[3] events.

Alerts can have a large number of fields that vary by alert type and source. This project focuses on the IPs associated with alerts.

2.3 Approach

2.3.1 Overview

The overall approach of this project is to aggregate related alerts together and to present a security analyst with a group of alerts describing an attack or malicious incident. A graph is a useful representation for attacks, as it allows the visualization of the flow of malicious activities through different assets and allows additional information to be encoded into its topology. A rich set of well-known graph partition algorithms can be applied to the graph to get automatic grouping of alerts.

The construction of the graph has three main parts, each building upon the previous one:

- IP embedding: using the network log data to embed numerical representations capturing the underlying relationships between each IP addresses.
- Graph Construction: combining alerts and network hosts into a single graph representation
- Community Detection: partitioning the graph constructed in the previous step to find groups of related alerts and hosts within the graph.

2.3.2 Graph Construction

To create an effective visualization that groups alerts together and describes possible attacks and malicious incidents, an alert graph is created. As previously mentioned, a graph is a useful representation for attacks, as the flow of malicious alerts can be shown through the edge connections in the graph and additional information can be encoded in the graph's topology. Graph partition algorithms can be applied to this graph to isolate sub-graphs that highlight potentially meaningful clusters of alerts and assets.

The graph construction portion of this project is broken down into smaller modules performing each task separately. An overview of the tasks breakdown and data flow can be found in Figure 2.1.

IP Embedding

Network connection events provide important contextual information relating IPs to each other. Such information can be encoded in a Word2Vec embedding that provided numerical representation of the IPs. To

[2]https://suricata-ids.org/
[3]https://docs.microsoft.com/en-us/sysinternals/downloads/sysmon

supplement this information into the graph model, edges and edge weights between IPs are created using this embedding.

Word2Vec embeddings are numerical representations of words learned by unsupervised neural networks, where words, phrases, sentences, or entire documents are mapped onto real vectors. Mikolov et al. [33] introduced a series of Word2Vec models that generate embeddings based on the context words most likely to appear around a given word.

We extracted the following words from each network connection event:

- source IP
- destination IP
- transport protocol + destination port, e.g. TCP/80, UDP/53

These words were used to train the IP embedding used for the construction of the attack graph. As all IPs in the dataset are used as words to train this IP embedding, each will have an IP embedding that is the numerical vector representation of itself.

This IP embedding is used to provide context as to how "similar" two given IPs are. Because this IP embedding is trained on the network connection event data, if two IPs frequently appear in the same network connection events, then their IP embeddings would also be close in distance to each other.

For later steps in the graph construction, a method for comparing two IPs is needed. To define how similar or dissimilar two IPs are, a similarity score is calculated by taking the IP embedding of the two IPs and using the following formula:

$$\text{Similarity\_Score} = \frac{1}{1 + \text{distance}}$$

The distance refers to the Euclidean distance between the IP embedding of the two IPs. This function will output a similarity score that is between 0 and 1; it is decreasing with respect to the distance between the IPs.

Further work and analysis of the IP embeddings will be presented by Benjamin Burr for his Master's thesis at Carleton University.

Data Processing

The input alert and connection event data are in two formats: 1) Elasticsearch logs which are JSON objects, and 2) Parquet files which are structured data exported from the Hadoop ecosystem. Different python modules were written to process these initial data.

- Alert Processor: The tasks of processing each individual alert are broken down into three separate modules, as follows:
    1. Alert Node Extraction - This module creates the Alert Nodes file in csv format. Information such as alert ID, alert label, description and Cyber Killchain[4] stage is extracted from each network alert.

[4]https://www.lockheedmartin.com/en-us/capabilities/cyber/cyber-kill-chain.html

2. Alert to IP Edge Creation - This module appends to the Edges file that is in csv format. For IP addresses that are associated with an alert, an edge is created using an alert ID to IP address pair. A weight of 1 is also given to each of the alert edges.
3. Alert IP Node Extraction - This module extracts all IPs that are associated with the alerts and creates an IP Nodes file that is in csv format. The Alert IP Node Extraction module also checks the Internal Nodes file for each IP. If an IP appears in the Internal Nodes file, then that IP address is labeled as "int-ip". Else the IP is labeled as "ip".

- Internal IP Extraction - This module uses the connection events labels for each of the IP addresses that are internal to the company. Note that internal IP addresses include both the private IP addresses and the company-owned IP addresses. These internal IP addresses are written to the Internal Nodes file.
- Internal IP Edge Creation - This module uses the IP embedding and the Internal Nodes file. For every possible pair of internal IP nodes there is an edge and the edge weight is the similarity score that is calculated using the method mentioned in the IP Embedding section.

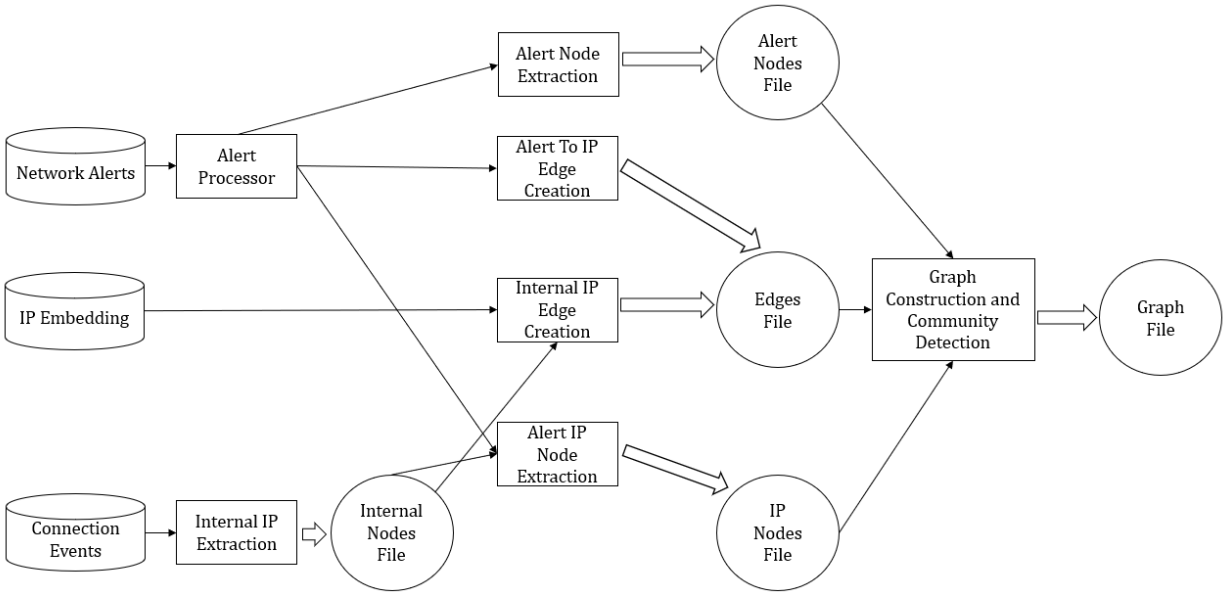


Figure 2.1: An overview of the data flow and the steps to construct the graph.

Graph Creation

After the data processing steps, all the required files are created for the graph creation step. The graph creation module takes input from all the files created in the last step and creates an undirected weighted graph using these inputs. This construction of the graph object in Python is done through the NetworkX[5] package.

[5]https://networkx.github.io/

After the graph is constructed, community detection algorithms were used to divide the alert graph into separate sub-graphs, the goal being to identify malicious attack campaigns from the topology of each of the sub-graphs.

For this project, two community detection algorithms were used: the Louvain, and the Clauset-Newman-Moore greedy modularity maximization algorithm to partition our network graph. The implementation of these two algorithms can be found in the NetworkX package.

After the community detection algorithms are run on the alert graph, the graph goes through some transformation steps:

1. Any community with only a single IP node is filtered out in order to find communities with more complex structure.
2. Any single node communities are removed from the graph. The nodes belonging to a community with more than one node are labeled by their community number.

The output of all of these modules is a graph file in the gexf format[6] which is an XML for complex network structures.

Visualizations of the alert graph are created using Gephi[7], an open source graph visualization software.

2.4 Experiments

2.4.1 IP Similarity Distribution

To supplement contextual information from connection events data, IP embedding was used to calculate similarity scores between all pairs of internal IP addresses.

The IP similarity scores translate into weighted edges connecting internal IP addresses. Edges of lower weights were excluded from the graph because it indicated that there were no strong relations between the two IP addresses from connection events data.

To study the distribution of the similarity scores between all internal IP pairs, a histogram was created (figure 2.2). The mean of the similarity scores was about 0.49 and the maximum was about 0.93.

2.4.2 Testing Thresholds

To find a suitable threshold for filtering internal IP edges with a lower similarity score, different alert graphs with different thresholds were created.

Since the mean of the internal IP similarity scores was around 0.5, different graphs were created using thresholds 0.5, 0.6, 0.7, 0.8, and 0.9. At a threshold of 0.9, the graph was almost the same as the graph without any internal IP edges. Visualization of the alert graph at each threshold was created so that the effects of thresholds on the communities can be studied.

[6] https://gephi.org/gexf/format/schema.html
[7] https://gephi.org/

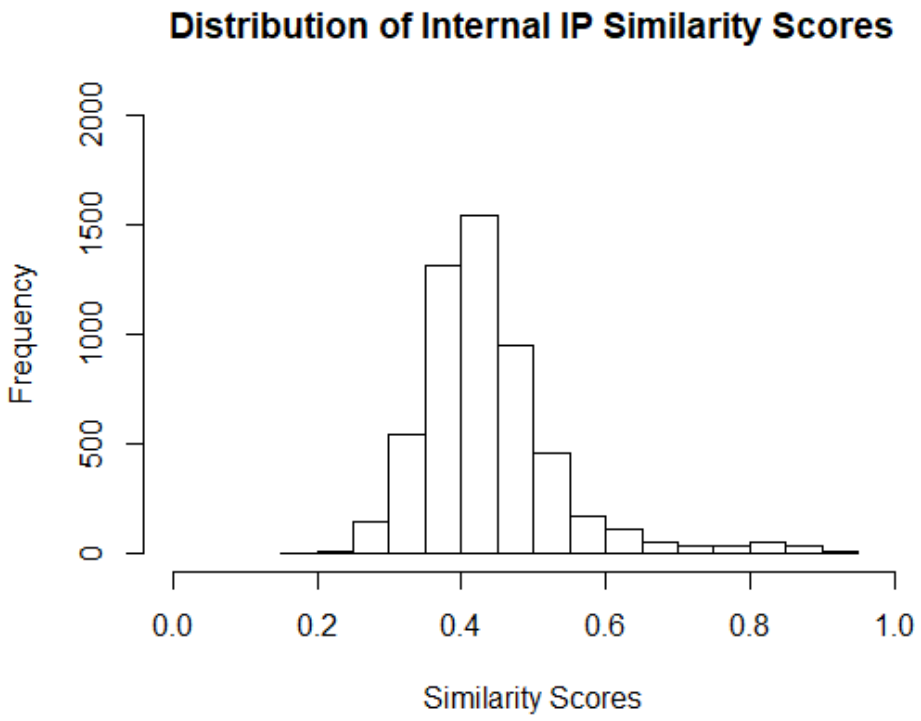


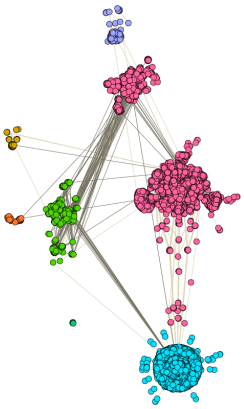
Figure 2.2: The distribution of internal IP similarity scores.

Table 2.1: The number of communities generated by each community detection algorithm at each threshold

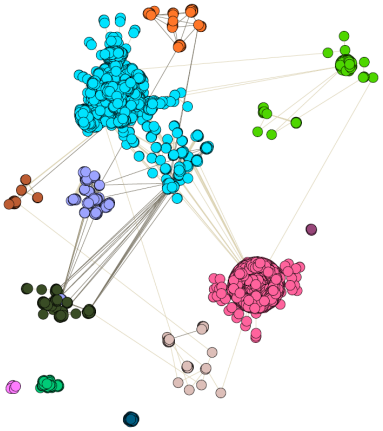
Threshold	0.5	0.6	0.7	0.8	0.9
Louvain	7	12	15	10	8
Clauset-Newman-Moore	6	10	14	9	8

The partition of the alert graph using the specified community detection algorithm were shown in figures 2.3 and 2.4 and the number of communities created by each algorithm at each threshold can be found in 2.1. From the visualization of the clusters at each threshold, one can see that each densely connected component of the graph is in different communities.

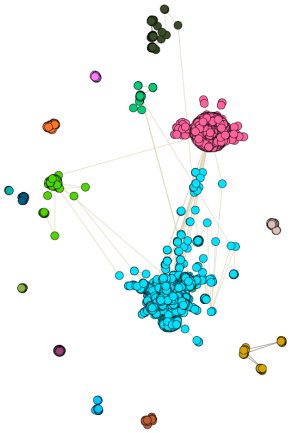
As the number of the threshold increases, more (IP) edges are filtered out of the graph. As a result, the graphs with the lower thresholds are more densely connected. Even though the smaller communities differ at each threshold, the two biggest communities in both Louvain and Clauset-Newman-Moore greedy modularity maximization algorithm remain unchanged. The number of small communities changes as the threshold changes. This is because the density of the smaller communities is more sensitive to the edges added and removed at each threshold. From the table, we can see that the number of communities peaked at a threshold of 0.7 for both Louvain and Clauset-Newman-Moore greedy modularity maximization algorithms.



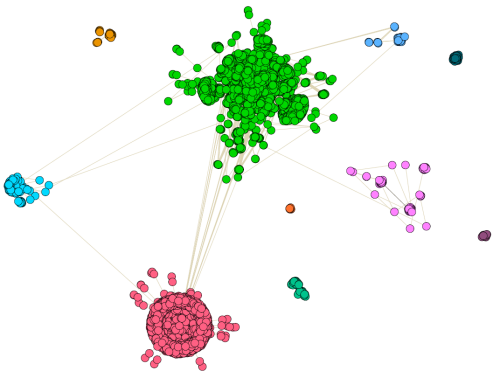
(a) Communities at threshold 0.5.



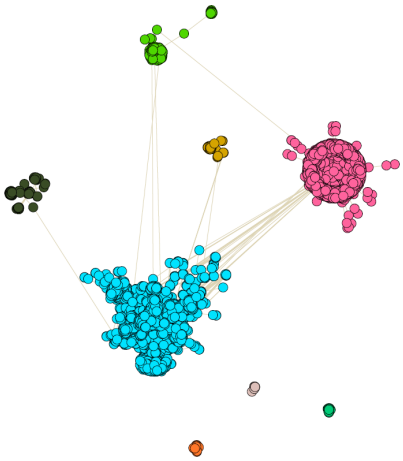
(b) Communities at threshold 0.6.



(c) Communities at threshold 0.7.

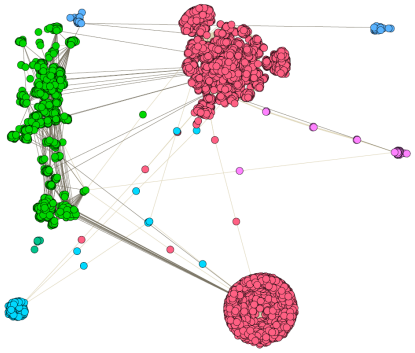


(d) Communities at threshold 0.8.

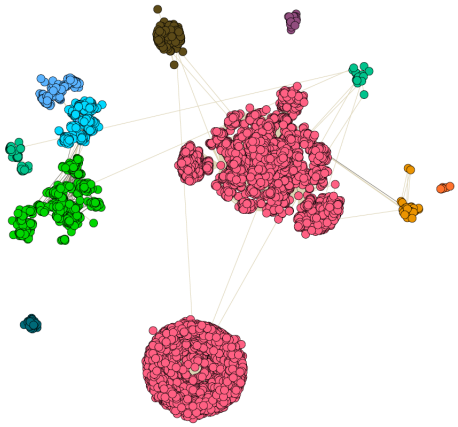


(e) Communities at threshold 0.9.

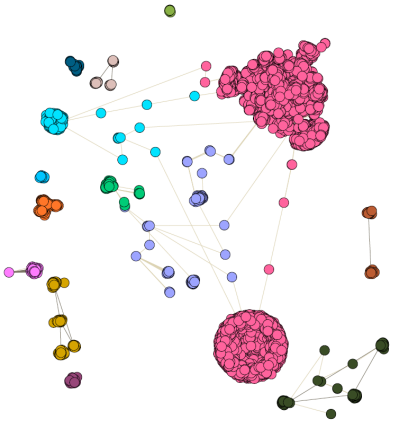
Figure 2.3: Plots of communities generated by the Louvain algorithm at each threshold level. The beige edges are alert edges and the grey edges are IP edges, colour of the nodes are by communities.



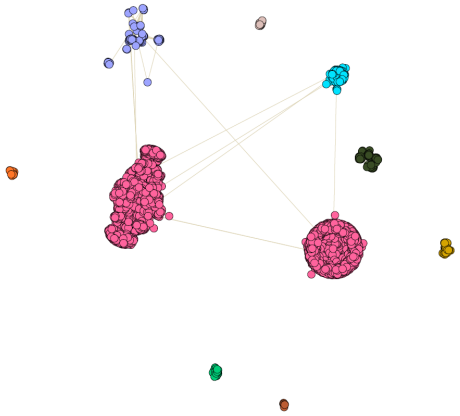
(a) Communities at threshold 0.5.



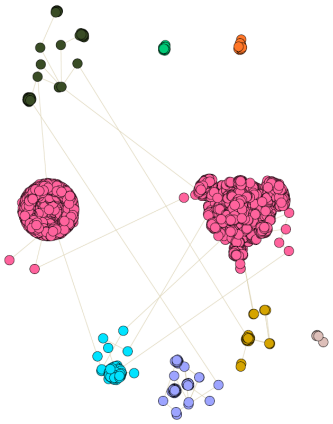
(b) Communities at threshold 0.6.



(c) Communities at threshold 0.7.



(d) Communities at threshold 0.8.



(e) Communities at threshold 0.9.

Figure 2.4: Plots of communities generated by the Clauset-Newman-Moore greedy modularity maximization algorithm at each threshold level. The beige edges are alert edges and the grey edges are IP edges, colour of the nodes are by communities.

2.4.3 Result

In this section, different communities created from the community detection algorithms at a threshold of 0.7 are highlighted. For each community, there will be a description of the graph presented and a discussion of how it could be used by security analysts to identify security incidents.

Louvain 2: Analyze Alerts In Context

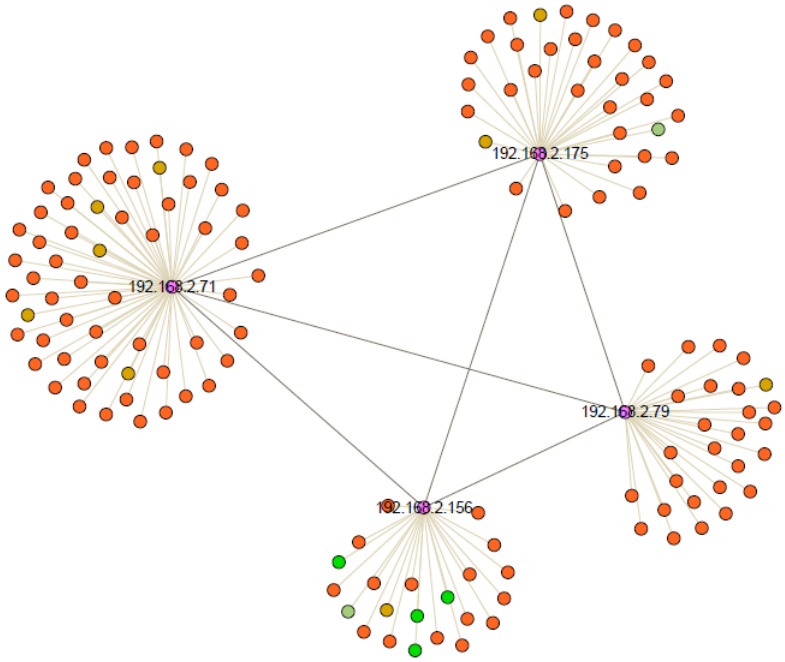


Figure 2.5: Community 2 partitioned by the Louvain algorithm.

Figure 2.5 represents an example of a sub-graph where similar internal IPs are generating a lot of anomaly alerts.

There are only internal IP addresses in this sub-graph (192.168.2.71, 192.168.2.156, 192.168.2.175, 192.168.2.79). Each IP address generated multiple anomaly alerts and bad HTTP connections alerts. The only connections between these IPs are from the grey edges, which suggests that these IP addresses behave similarly in terms of network connection events.

A possible conclusion we can draw from this graph is that these similar internal IPs are behaving anomalously together during this time period. Instead of looking at one single IP and the alerts generated by it, a security analyst can investigate all four IPs together and it may provide more context as to why so many alerts were raised.

Louvain 13: Remove Potential False Positives

In contrast to Figure 2.5, Figure 2.6 depicts a densely connected sub-graph of internal IPs that are similar to each other while each IP itself raised very few alerts. Because of the low number of alerts in this graph, it would be reasonable to believe that this graph does not point to a malicious attack step. This example

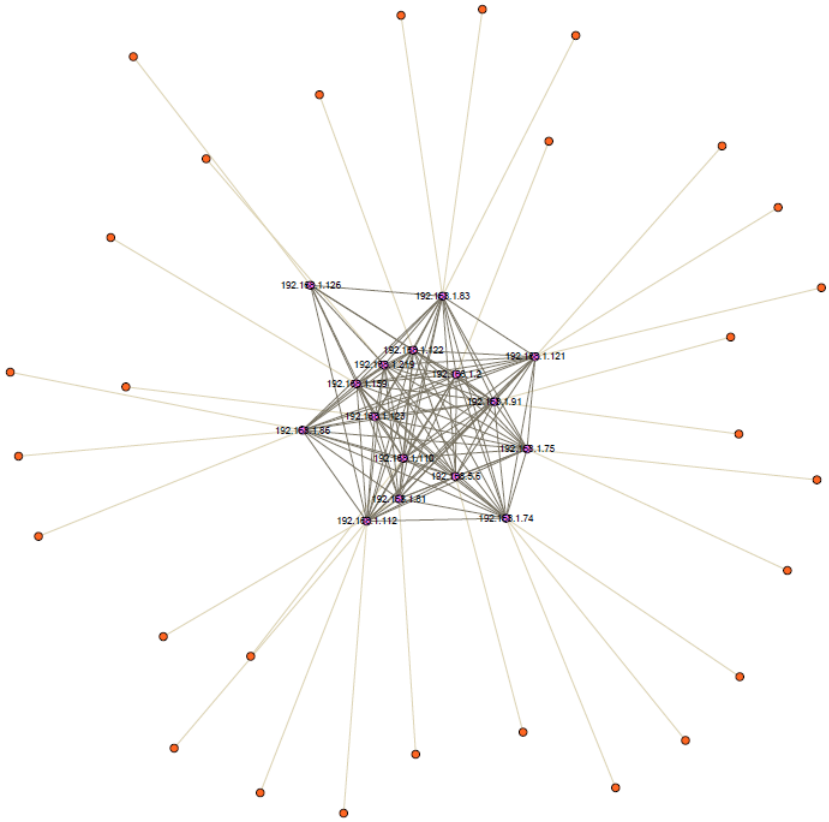


Figure 2.6: Community 13 partitioned by the Louvain algorithm.

shows that not all communities created from the alert graph point to malicious activities. However, since these alerts are grouped together, it would be easy for security analysts to discard a large number of alerts like these and focus on other alerts.

Louvain 1: Identify Potential Malicious Hosts

Figure 2.7 represents community number 1 as partitioned by the Louvain algorithm. The IP nodes are in purple and labeled by their IP addresses. The other coloured nodes are alert nodes, as different colours represent different types of alerts. The internal IP addresses are connected by a grey edges, which means that these IPs are similar according to the IP embedding.

The majority of the alerts in this sub-graph are anomaly alerts (in orange). Each of the internal IP nodes (192.168.2.50, 192.168.2.96, 192.168.2.45) raised several anomaly alerts that come from anomaly detection algorithms such as random-forest, time series, and rule-based anomalies.

The rule-based alerts are raised because of "connections to hosts with a bad reputation" and "suspicious HTTP connections". Notably, IP 192.168.2.96 also triggered an alert for *sysmon.cve-2019-0708-exploit* which is a potential Windows vulnerability. This suggests that this IP asset may also be infected by malware. There are two external IP addresses (209.251.75.37, 50.28.16.195) that triggered IDS alerts (green) and brute force attempt alerts (blue) to each of the internal IPs in this graph.

This graph can be used as a guide for how to investigate the alerts generated by the three internal IPs in

this graph. As the external IP (209.251.75.37) is connected to the three internal IPs through RDP brute force alerts, security analysts could investigate this external IP as a potential cause for all the anomaly alerts generated by the three internal IPs shown. Based on analysis of the causes of anomalies in this graph, one could further determine whether or not this external IP is a malicious host and whether or not to add this IP to the network's blacklist.

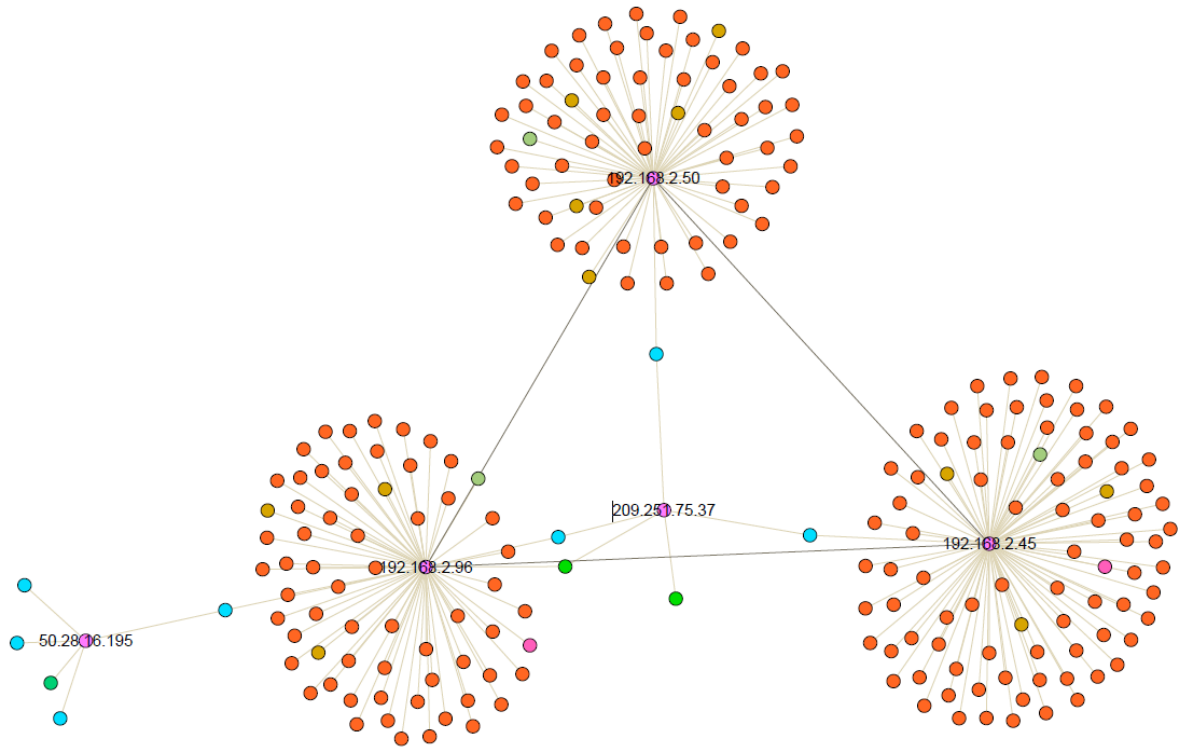


Figure 2.7: Community 1 partitioned by the Louvain algorithm.

Louvain 3: Identify Potential Vulnerability

Numerous external IPs triggering IDS alerts and brute force alerts to a single internal IP (192.168.2.81) are shown in Figure 2.8. The internal IP (192.168.2.81) also raised numerous anomaly alerts.

This sub-graph suggests that the internal IP 192.168.2.81 is easily accessible for external connections. This means that this internal IP asset is a potential vulnerability within the system. A security analyst could use this sub-graph to identify potential vulnerabilities within the system.

Summary

The communities presented in Section 2.4.3 are representative of the types of communities that are present in the data set. There are some variations on the number of internal or external IPs in a sub-graph, but the overall structure is the same.

A graph with multiple alerts raised by several IPs that have similar network traffic is shown in Figure 2.5. Such a graph could be used by an analyst to investigate these alerts in the context of several IPs together

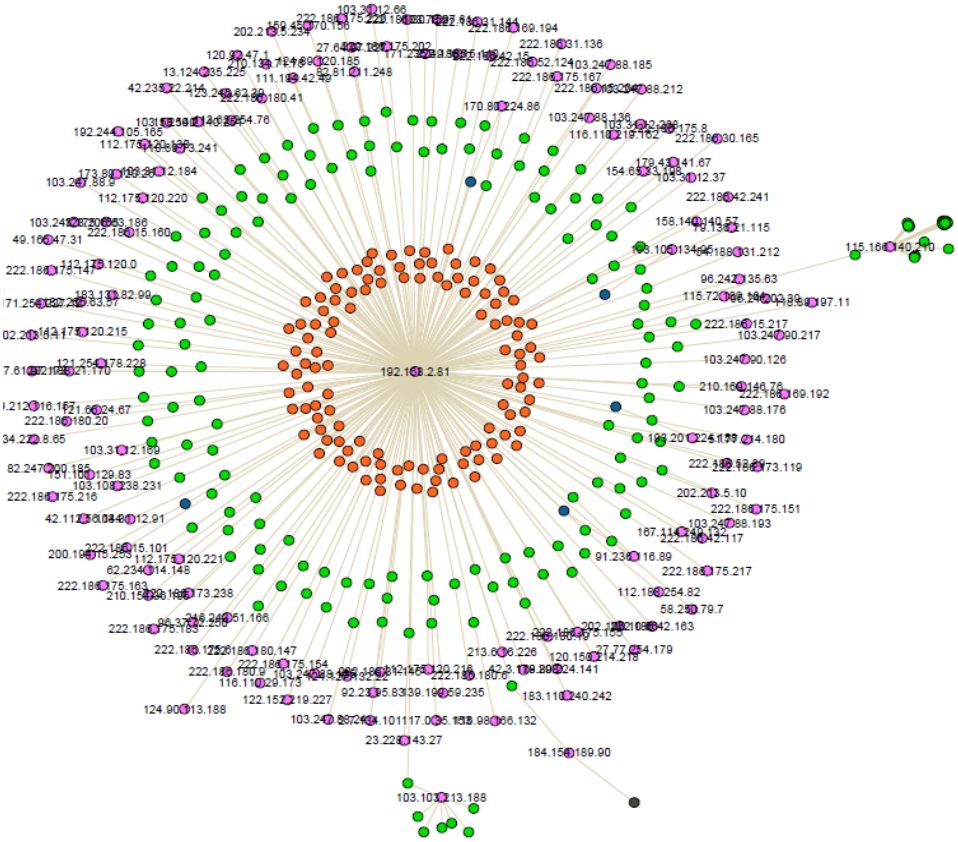


Figure 2.8: Community 3 partitioned by the Louvain algorithm.

instead of looking at each alert individually. Figure 2.7 leads to a potentially malicious IP from an outside source that targets multiple IPs that are similar in the network. The external IP is a possible candidate for the blacklist of the firewall. Figure 2.8 depicts an internal IP that has numerous alerts generated by multiple external IPs, which is potentially a vulnerable asset in the system. This type of graph can provide easy visuals for security analysts to discover vulnerable assets that requires extra security measures.

Although each of the sub-graphs themselves do not suggest malicious campaigns from malicious actors, the graphs are useful for analyzing alerts in groups instead of individually. These figures are examples of ways that the alert graph could be used to assist security analysts with their efforts in finding security incidents from IDS alerts.

3. Discussion

3.1 Conclusion

This project presents a method to group alerts together and visualize them through alert graphs. Information from network connection events are incorporated through IP similarity scores calculated using the IP embedding. Overall, this alert graph presents the alert-to-IP relationship and the IP-to-IP relationship in a way that is easily interpreted by security analysts.

This alert graph could be used to assist security analysts with their efforts in finding security incidents and vulnerabilities in their system. In Section 2.4.3, multiple sub-graphs of the alert graphs are presented and its potential usage to the security analysts are presented. Although each of the communities themselves do not suggest malicious campaigns from malicious actors, the graphs are useful for analyzing alerts in groups rather than individually.

3.2 Future Work

Multiple future directions can be taken from different sections of this project.

IP Embedding The current approach of this project uses only the source IP, destination IP, and port to train the IP embedding. The network event data provided has other useful information such as packet size, country, timestamp, etc. Training the IP embedding using different features would encode different contextual information into the embedding and this would have a significant impact on the alert graph topology generated in the later stage. A possible future direction is to experiment using different combinations of features for IP embedding generation and create a metric to evaluate each IP embedding.

Alert Graph Generation Currently, the alert graph is constructed using all of the alerts generated by the IDS. As a result of this, the alert graph is extremely dense with a large number of vertices. It would be useful to apply alert aggregation techniques before the graph creation step so that the resulting graph visualization is easier to interpret.

Community Detection This project uses Louvain and Clauset-Newman-Moore greedy modularity maximization algorithm for community detection. These two community detection methods use modularity to partition the graph into different clusters. Other metrics can be used to partition a graph, such as silhoutte index, conductance, coverage, and performance [4]. Overlapping community detection algorithms should be considered as well. With a good alert graph, it would be useful to study different clusters created by different community detection algorithms.

Dataset As mentioned in the Methodology section, there are no known attacks present in the dataset used for this project. This means that all the results discussed in this paper are only speculations.

Since the experiment is on the alert correlation part of an IDS, this project uses IDS alerts instead of raw network traffic data. This means publicly available datasets cannot be applied directly to this project since most public datasets only contain network traffic data. Instead, the scope of this project is limited to the IDS alerts generated by the Toronto company's IDS that is based on a real company's network traffic.

To overcome this problem in the future, one can utilize existing IDS software such as Bro and Suricata to create IDS alerts on top of existing public datasets. This would provide a better evaluation of the graph visualization since the graph partitions can be compared to the attack campaigns present in the dataset.

4. Appendix

Code and implementation of the project can be downloaded from:

`https://drive.google.com/open?id=1NcuoJVpVh8BwrtAAe6i_bq6L9EgYoW1m`

If you wish to gain access to the private GitHub repository, please contact me at shellywang@cmail.carleton.ca.

Bibliography

[1] Shikha Agrawal and Jitendra Agrawal. Survey on anomaly detection using data mining techniques. *Procedia Computer Science*, 60:708–713, 2015.

[2] Mohiuddin Ahmed, Abdun Naser Mahmood, and Jiankun Hu. A survey of network anomaly detection techniques. *Journal of Network and Computer Applications*, 60:19–31, 2016.

[3] Safaa O Al-Mamory and Hongli Zhang. Intrusion detection alarms reduction using root cause analysis and clustering. *Computer Communications*, 32(2):419–430, 2009.

[4] Hélio Almeida, Dorgival Guedes, Wagner Meira, and Mohammed J Zaki. Is there a best quality metric for graph clusters? In *Joint European conference on machine learning and knowledge discovery in databases*, pages 44–59. Springer, 2011.

[5] Faeiz Alserhani, Monis Akhlaq, Irfan U Awan, Andrea J Cullen, and Pravin Mirchandani. Mars: multi-stage attack recognition system. In *2010 24th IEEE International Conference on Advanced Information Networking and Applications*, pages 753–759. IEEE, 2010.

[6] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment*, 2008(10):P10008, 2008.

[7] Aaron Clauset, Mark EJ Newman, and Cristopher Moore. Finding community structure in very large networks. *Physical review E*, 70(6):066111, 2004.

[8] Kaustav Das and Jeff Schneider. Detecting anomalous records in categorical datasets. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 220–229, 2007.

[9] Salma Elhag, Alberto Fernández, Abdullah Bawakid, Saleh Alshomrani, and Francisco Herrera. On the combination of genetic fuzzy systems and pairwise learning for improving detection rates on intrusion detection systems. *Expert Systems with Applications*, 42(1):193–202, 2015.

[10] Huwaida Tagelsir Elshoush and Izzeldin Mohamed Osman. Alert correlation in collaborative intelligent intrusion detection systems—a survey. *Applied Soft Computing*, 11(7):4349–4365, 2011.

[11] Ugo Fiore, Francesco Palmieri, Aniello Castiglione, and Alfredo De Santis. Network anomaly detection with the restricted boltzmann machine. *Neurocomputing*, 122:13–23, 2013.

[12] Steffen Haas and Mathias Fischer. On the alert correlation process for the detection of multi-step attacks and a graph-based realization. *ACM SIGAPP Applied Computing Review*, 19(1):5–19, 2019.

[13] Weiming Hu, Jun Gao, Yanguo Wang, Ou Wu, and Stephen Maybank. Online adaboost-based parameterized methods for dynamic distributed network intrusion detection. *IEEE Transactions on Cybernetics*, 44(1):66–82, 2013.

[14] Nien-Yi Jan, Shun-Chieh Lin, Shian-Shyong Tseng, and Nancy P Lin. A decision support system for constructing an alert classification model. *Expert Systems with Applications*, 36(8):11145–11155, 2009.

[15] Chunfu Jia and Feng Yang. An intrusion detection method based on hierarchical hidden markov models. *Wuhan University Journal of Natural Sciences*, 12(1):135–138, 2007.

[16] Inho Kang, Myong K Jeong, and Dongjoon Kong. A differentiated one-class classification method with applications to intrusion detection. *Expert Systems with Applications*, 39(4):3899–3905, 2012.

[17] Latifur Khan, Mamoun Awad, and Bhavani Thuraisingham. A new intrusion detection system using support vector machines and hierarchical clustering. *The VLDB journal*, 16(4):507–521, 2007.

[18] Ansam Khraisat, Iqbal Gondal, and Peter Vamplew. An anomaly intrusion detection system using c5 decision tree classifier. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 149–155. Springer, 2018.

[19] Ansam Khraisat, Iqbal Gondal, Peter Vamplew, and Joarder Kamruzzaman. Survey of intrusion detection systems: techniques, datasets and challenges. *Cybersecurity*, 2(1):20, 2019.

[20] Deguang Kong, Yoon-Chan Jhi, Tao Gong, Sencun Zhu, Peng Liu, and Hongsheng Xi. Sas: semantics aware signature generation for polymorphic worm detection. *International Journal of Information Security*, 10(5):269–283, 2011.

[21] Christian Kreibich and Jon Crowcroft. Honeycomb: creating intrusion detection signatures using honeypots. *ACM SIGCOMM computer communication review*, 34(1):51–56, 2004.

[22] Aleksandar Lazarevic, Levent Ertoz, Vipin Kumar, Aysel Ozgur, and Jaideep Srivastava. A comparative study of anomaly detection schemes in network intrusion detection. In *Proceedings of the 2003 SIAM international conference on data mining*, pages 25–36. SIAM, 2003.

[23] Suchul Lee, Sungho Kim, Sungil Lee, Jaehyuk Choi, Hanjun Yoon, Dohoon Lee, and Jun-Rak Lee. Largen: automatic signature generation for malwares using latent dirichlet allocation. *IEEE Transactions on Dependable and Secure Computing*, 15(5):771–783, 2016.

[24] Yinhui Li, Jingbo Xia, Silan Zhang, Jiakai Yan, Xiaochuan Ai, and Kuobin Dai. An efficient intrusion detection system based on support vector machines and gradually feature removal method. *Expert Systems with Applications*, 39(1):424–430, 2012.

[25] Hung-Jen Liao, Chun-Hung Richard Lin, Ying-Chih Lin, and Kuang-Yuan Tung. Intrusion detection system: A comprehensive review. *Journal of Network and Computer Applications*, 36(1):16–24, 2013.

[26] Po-Ching Lin, Ying-Dar Lin, and Yuan-Cheng Lai. A hybrid algorithm of backward hashing and automaton tracking for virus scanning. *IEEE transactions on computers*, 60(4):594–601, 2010.

[27] Shih-Wei Lin, Kuo-Ching Ying, Chou-Yuan Lee, and Zne-Jung Lee. An intelligent algorithm with feature selection and decision rules applied to anomaly intrusion detection. *Applied Soft Computing*, 12(10):3285–3290, 2012.

[28] Richard Lippmann, Joshua W Haines, David J Fried, Jonathan Korba, and Kumar Das. The 1999 darpa off-line intrusion detection evaluation. *Computer networks*, 34(4):579–595, 2000.

[29] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation-based anomaly detection. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 6(1):1–39, 2012.

[30] Yongguo Liu, Kefei Chen, Xiaofeng Liao, and Wei Zhang. A genetic clustering method for intrusion detection. *Pattern Recognition*, 37(5):927–942, 2004.

[31] Federico Maggi, Matteo Matteucci, and Stefano Zanero. Reducing false positives in anomaly detectors through fuzzy alert aggregation. *Information Fusion*, 10(4):300–311, 2009.

[32] Maurizio Martellini and Andrea Malizia. *Cyber and chemical, biological, radiological, nuclear, explosives challenges*. Springer, 2017.

[33] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[34] Sadegh M Milajerdi, Rigel Gjomemo, Birhanu Eshete, R Sekar, and VN Venkatakrishnan. Holmes: real-time apt detection through correlation of suspicious information flows. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 1137–1152. IEEE, 2019.

[35] Mark EJ Newman. Analysis of weighted networks. *Physical review E*, 70(5):056131, 2004.

[36] Mark EJ Newman. Fast algorithm for detecting community structure in networks. *Physical review E*, 69(6):066133, 2004.

[37] Peng Ning and Dingbang Xu. Learning attack strategies from intrusion alerts. In *Proceedings of the 10th ACM conference on Computer and communications security*, pages 200–209, 2003.

[38] Steven Noel, Eric Robertson, and Sushil Jajodia. Correlating intrusion events and building attack scenarios through attack graph distances. In *20th Annual Computer Security Applications Conference*, pages 350–359. IEEE, 2004.

[39] Sang Hyun Oh and Won Suk Lee. An anomaly intrusion detection method by clustering normal user behavior. *Computers & Security*, 22(7):596–612, 2003.

[40] Derek Pao, Nga Lam Or, and Ray CC Cheung. A memory-based nfa regular expression match engine for signature-based intrusion detection. *Computer communications*, 36(10-11):1255–1267, 2013.

[41] Vern Paxson. Bro: a system for detecting network intruders in real-time. *Computer networks*, 31(23-24):2435–2463, 1999.

[42] Kexin Pei, Zhongshu Gu, Brendan Saltaformaggio, Shiqing Ma, Fei Wang, Zhiwei Zhang, Luo Si, Xiangyu Zhang, and Dongyan Xu. Hercule: Attack story reconstruction via community discovery on correlated log graph. In *Proceedings of the 32Nd Annual Conference on Computer Security Applications*, pages 583–595, 2016.

[43] Phillip A Porras, Martin W Fong, and Alfonso Valdes. A mission-impact-based approach to infosec alarm correlation. In *International Workshop on Recent Advances in Intrusion Detection*, pages 95–114. Springer, 2002.

[44] Martin Roesch et al. Snort: Lightweight intrusion detection for networks. In *Lisa*, volume 99, pages 229–238, 1999.

[45] Suseela T Sarasamma, Qiuming A Zhu, and Julie Huff. Hierarchical kohonenen net for anomaly detection in network security. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 35(2):302–312, 2005.

[46] Asaf Shabtai, Eitan Menahem, and Yuval Elovici. F-sign: Automatic, function-based signature generation for malware. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 41(4):494–508, 2010.

[47] Raman Singh, Harish Kumar, Ravinder Kumar Singla, and Ramachandran Ramkumar Ketti. Internet attacks and intrusion detection system. *Online Information Review*, 2017.

[48] Xiuyao Song, Mingxi Wu, Christopher Jermaine, and Sanjay Ranka. Conditional anomaly detection. *IEEE Transactions on knowledge and Data Engineering*, 19(5):631–645, 2007.

[49] Li Sun, Steven Versteeg, Serdar Boztas, and Asha Rao. Detecting anomalous user behavior using an extended isolation forest algorithm: an enterprise case study. *arXiv preprint arXiv:1609.06676*, 2016.

[50] Yong Tang, Bin Xiao, and Xicheng Lu. Signature tree generation for polymorphic worms. *IEEE transactions on computers*, 60(4):565–579, 2010.

[51] Xinmin Tao, Furong Liu, and Tingxian Zhou. A novel approach to intrusion detection based on support vector data description. In *30th Annual Conference of IEEE Industrial Electronics Society, 2004. IECON 2004*, volume 3, pages 2016–2021. IEEE, 2004.

[52] Giovanni Vigna and Richard A Kemmerer. Netstat: A network-based intrusion detection system. *Journal of computer security*, 7(1):37–71, 1999.

[53] Jouni Viinikka, Hervé Debar, Ludovic Mé, Anssi Lehikoinen, and Mika Tarvainen. Processing intrusion detection alert aggregates with time series modeling. *Information Fusion*, 10(4):312–324, 2009.

[54] Gang Wang, Jinxing Hao, Jian Ma, and Lihua Huang. A new approach to intrusion detection using artificial neural networks and fuzzy clustering. *Expert systems with applications*, 37(9):6225–6232, 2010.

[55] Yanxin Wang, Johnny Wong, and Andrew Miner. Anomaly intrusion detection using one class svm. In *Proceedings from the Fifth Annual IEEE SMC Information Assurance Workshop, 2004.*, pages 358–364. IEEE, 2004.

[56] Qingtao Wu and Zhiqing Shao. Network anomaly detection using time series analysis. In *Joint international conference on autonomic and autonomous systems and international conference on networking and services-(icas-isns' 05)*, pages 42–42. IEEE, 2005.

[57] Dingbang Xu and Peng Ning. Alert correlation through triggering events and common resources. In *20th Annual Computer Security Applications Conference*, pages 360–369. IEEE, 2004.

[58] Dingbang Xu and Peng Ning. *Correlation analysis of intrusion alerts.* Springer, 2006.

[59] Nong Ye, Syed Masum Emran, Qiang Chen, and Sean Vilbert. Multivariate statistical analysis of audit trails for host-based intrusion detection. *IEEE Transactions on computers*, 51(7):810–820, 2002.

[60] Jingmin Zhou, Mark Heckman, Brennen Reynolds, Adam Carlson, and Matt Bishop. Modeling network intrusion detection alerts for correlation. *ACM Transactions on Information and System Security (TISSEC)*, 10(1):4–es, 2007.