On the Detection of Persistent Attacks using Alert Graphs and Event Feature Embeddings

Benjamin Burr and Shelly Wang
Dept. of Mathematics and Statistics
Carleton University
Ottawa, Canada
{benburr,shellywang}@cmail.carleton.ca

Geoff Salmon and Hazem Soliman
RANK Software
Toronto, Canada
{geoff.salmon,hazem.soliman}@ranksoftwareinc.com

***Abstract*—Intrusion Detection Systems (IDS) generate a high volume of alerts that security analysts do not have the resources to explore fully. Modelling attacks, especially the coordinated campaigns of Advanced Persistent Threats (APTs), in a visually-interpretable way is a useful approach for network security. Graph models combine multiple alerts and are well suited for visualization and interpretation, increasing security effectiveness. In this paper, we use feature embeddings, learned from network event logs, and community detection to construct and segment alert graphs of related alerts and networks hosts. We posit that such graphs can aid security analysts in investigating alerts and may capture multiple aspects of an APT attack. The eventual goal of this approach is to construct interpretable attack graphs and extract causality information to identify coordinated attacks.**

***Index Terms*—word embedding, IP address, IP2Vec, graph model, unsupervised clustering, community detection**

I. Introduction

Security monitoring systems consume logged network traffic events (events) and trigger alerts based on signatures or anomalies (alerts) in the traffic using a variety of detectors. An unknown complex multi-stage attack by an Advanced Persistent Threat (APT) may trigger multiple types of alerts across multiple detectors because each detector is sensitive to different behaviours. These alerts may not immediately appear related. For previously-unknown attacks, we cannot determine in advance which detectors will be important. Increasing the number of detectors increases the odds of catching aspects of such attacks, but also increases the overall number of alerts. Ultimately, a human security analyst can investigate only a limited number of alerts. An overwhelming number of alerts results in alert fatigue for the analyst, especially in the presence of false-positives, undermining the effectiveness of the entire security system.

By automatically aggregating related alerts, we hope to avoid alert fatigue without sacrificing the system's ability to detect unknown attacks by APTs. Ideally, we would present a correlated group of these aggregated alerts describing a single malicious incident [1] (incident). A graph model (graph) is a useful representation for such incidents [2], as additional information can be encoded in the graph's topology and a rich set of well-known algorithms can be applied to the graph, such as community detection and graph layout algorithms. These algorithms are useful both during the process of aggregating alerts into single incidents and for the analyst to apply manually when viewing the resulting graph visualizations of correlated incidents. Establishing causality between alerts is an active area of research, with previous work including the development of techniques to automatically extract causal relationships among alerts in order to build attack scenarios [3] and to find previously-unknown relationships between incidents [4].

Due to the difficulty of measuring causality between events, some researchers have explored building alert graphs using correlation measures instead. In [5], several similarity metrics have been proposed for the key network event entities, e.g., IP addresses (IPs) and port numbers (ports). In [6], alerts are correlated according to common combinations of attributes, then links between hosts and correlated single attacks are established to build multi-step attack paths. [7] describes a model for multi-stage intrusions. Unweighted and undirected graphs are first built using correlated log entries; then feature selection is used to capture potential causality along the graph edges and the Louvain algorithm, which is efficient for large networks, is used for community detection.

In this work, we leverage recent advancements from the natural language processing (NLP) field to estimate numerical embeddings for the categorical entities in the events. As in the NLP domain, this approach has the advantage of measuring the similarity between entities from the raw data without the need for an expensive and error-prone manual design. Hence, the focus of our work is on host and alert correlation as a way to establish relationships between disjointed sources of information. We aim to present structured incident graphs of correlated alerts to the analyst, while looking for opportunities to identify causation within the graph.

II. System Architecture

Our proposed incident detection system is summarized in Fig. 1. The system comprises three main parts, each building upon the previous one:

Funding for this research was generously provided by the Fields Institute Centre for Quantitative Analysis and Modelling in Toronto, Canada.

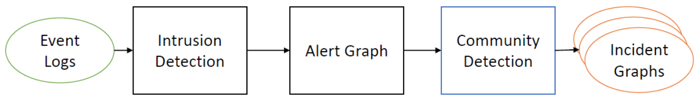

Fig. 1. Incident Detection Architecture.

- Intrusion Detection: We use the set of raw network and host event logs as input; we obtain the set of alerts representing anomalous or suspicious behaviour as output.
- Alert Graph Construction: Given the set of alerts, we group alerts into an alert graph comprising potential incidents. Two important steps are involved:
  - Feature embedding: Alerts contain information about event log fields (IPs, protocols, etc.) but lack the context needed to compute correlations between those fields. We use the underlying event logs to embed the field values into a numerical representation, which helps capture hidden relationships between alert field values.
  - Graph construction: We combine both alerts and network hosts involved in the alerts into a single graph representing potential incidents. Embeddings from the first step are used to add weighted edges between similar internal hosts.
- Community Detection: We use the Louvain and Greedy community detection algorithms to find groups of alerts and hosts that have a high probability of being related within the graph. The resulting incident graphs contain sets of alerts that are related by the network hosts involved, incorporating information learned during the feature embedding phase to link similar hosts. Our hope for these graphs is two-fold: 1) an analyst can triage multiple alerts presented in a graph using host similarity information more efficiently compared to a list or table of alerts, and 2) a complex attack by an APT that involves multiple hosts and triggers multiple alerts will be more easily recognized when those alerts are presented together.

III. Description

In this section, we briefly describe important details for each step in our proposal and highlight how they are connected to other works in the literature.

A. Event Dataset

The data used in this work comes from a real company network monitored for two months, September and October 2019, and consists of network connection events and alerts. The monitored company has roughly 50 employees using primarily Windows hosts, and its network includes several internal servers. Network taps capture traffic between the internal network and the internet gateway, as well as traffic between internal hosts. The traffic is analyzed by Zeek[1] to produce connection event logs[2]. These and other events produced by Zeek and other network monitoring systems, along with host-based events, are ingested by RANK Software's system, which produces the alerts based on pre-defined rules and typical intrusion detection methods previously implemented.

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). Each event from Zeek provides information such as the timestamp, the amount of traffic in each direction, and connection state[3].

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

B. Intrusion Detection

The alerts come from a mixture of sources. Some are produced by Suricata[4], a well-known Intrusion Detection System (IDS). Others are produced by RANK Software's rules engine and anomaly detectors. Suricata monitors the same network traffic that produces the connection events dataset, while the RANK system produces alerts based on both network events and host-based events, such as standard Windows events and Sysmon[5] events.

Alerts can have a large number of fields that vary by alert type and source. In this work we focus on the IP addresses associated with alerts, and leave exploration of the other fields for future work.

C. Feature Embedding

Word embeddings are numerical representations of words learned by unsupervised neural networks, where words, phrases, sentences, or entire documents are mapped onto real vectors. [8] introduced Word2Vec models, which generate embeddings using the context words most likely to appear around a given word. Such contextual associations may be used to cluster words using the embedding vectors to measure distance between words. [9] extended this approach by generating embeddings for uni-directional network flow data, calling it IP2Vec.

We used Gensim version 3.8.1[6] for the implementation of Word2Vec in Python 3.7. The set of words extracted from each event is treated as a Word2Vec "sentence." We extracted Source and Destination IPs, and Transport Protocol + Destination Port (e.g. TCP/80, UDP/53) as the vocabulary "words" from each connection event to form the training sentences,

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

[2]Example Zeek log files: https://docs.zeek.org/en/current/examples/logs/index.html

[3]https://docs.zeek.org/en/current/scripts/base/protocols/conn/main.zeek.html#type-Conn::Info

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

[5]https://docs.microsoft.com/en-us/sysinternals/downloads/sysmon

[6]https://radimrehurek.com/gensim/

with roughly 67.5 million such sentences used during model training producing a vocabulary size of around 141,000. The connection events we used are bi-directional, so training input did not need to account for the existence of another event corresponding to the flow in the reverse direction, as in the IP2Vec work. We excluded the source port from the vocabulary since the source port in a connection is often ephemeral and not meaningful due to being chosen by the OS, while inclusion in the model dramatically increases the vocabulary size.

Parameter settings followed [9]. We used a skip-gram model with negative sampling because we were interested in examining the connection traffic around host IP addresses and potential threat traffic. Minimum count for a vocabulary word was 1, with no maximum vocabulary size. The maximum window size was set to 3. Ten epochs were enough for convergence. We continue to analyze the effects of embedding size and other model parameters on the embeddings. The choice of some model parameters has been found to impact the model's quality [10], but model quality assessment with Word2Vec relies on heuristics rather than concrete methods. In the absence of such benchmarks, we compared the IP addresses of hosts whose behaviour is known to be similar to evaluate whether the embedding provides a reasonable distance metric. For example, two internal name servers primarily receive DNS requests from internal hosts and send DNS requests to external name servers, so we expect the name server IPs will have a similar embedding.

D. Alert Graph Construction

The Word2Vec embeddings were used to supplement contextual information relating IPs to each other in the creation of the edges of the graph model. There are two different types of IPs, internal and external. The internal IPs are either private IPs within the network or IPs that are owned by the monitored company.

Graph construction involved two types of nodes (alert nodes and IP nodes) and two types of edges (alert edges and internal IP edges). Each IP that appears in an alert is an IP node and each individual alert is an alert node. Alert edges were used to connect alert nodes to IPs that are associated with the alert by the detector that produced it. Internal IP edges connect internal IPs together. These edges are created using the similarity between the Word2Vec embeddings. Each IP has a vector, between two of which we may compute the Euclidean distance. The similarity between two vectors is defined as $\mathbf{1/(1+distance)}$. Fig. 2 shows the resulting distribution of similarities.

E. Community Detection

After the alert graph was constructed, community detection algorithms were used to divide the graph into separate incident graphs, with the goal of identifying malicious attack campaigns from the topology of each of the subgraphs. We used the Louvain and Clauset-Newman-Moore Greedy Modularization algorithms from the NetworkX package[7] in Python.

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

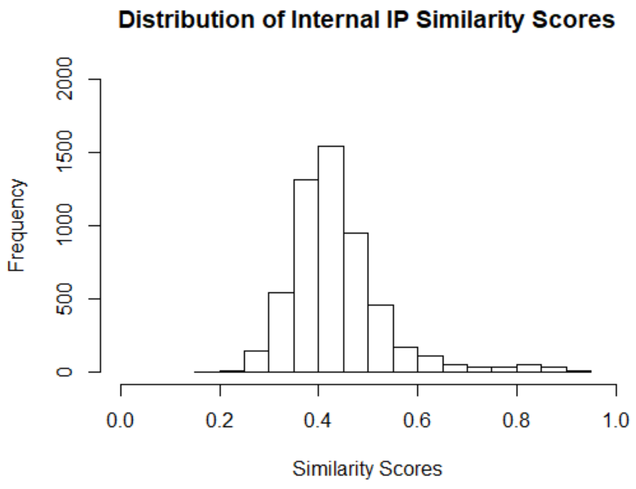

Fig. 2. Represents the distribution of similarities across all pairs of internal IPs. Distribution is approximately Normal and centered around 0.45. Mild asymmetry suggests higher similarity scores to be slightly more common, which is not surprising in a small network with mostly-similar devices.

The Louvain algorithm [11] extracts hierarchical community structures from networks, while the Clauset-Newman-Moore algorithm [12] starts with each node as a community and joins pairs of communities together iteratively to maximize modularity, with a similar goal of extracting hierarchical community structures.

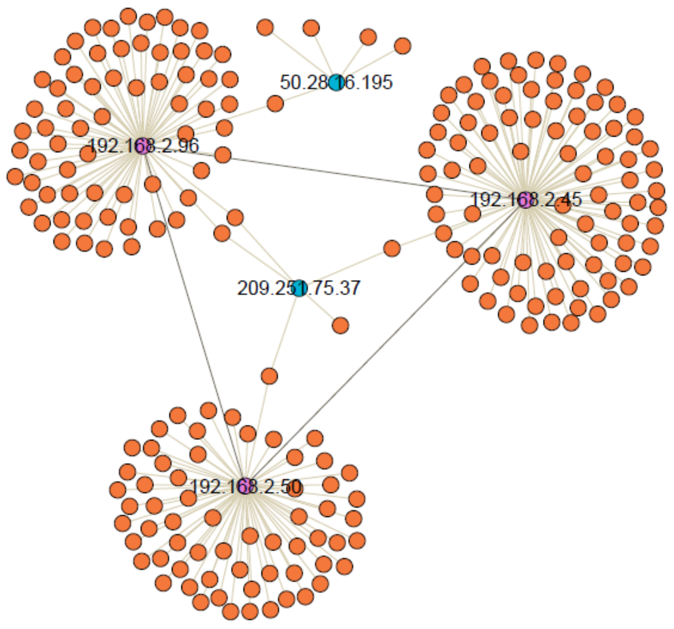

Fig. 3. Example of alert sub-graph community. Orange nodes represent alerts. Blue nodes represent external (non-client) IPs. Purple nodes represent internal (client) IPs. Black edges represent embedding similarity between internal IP addresses above the set threshold of 0.7. Yellow edges connect alert nodes to IP nodes.

IV. Preliminary Results

The dataset we used has approximately 67.5 million events for a period of two months. The IDS stage has raised approximately 37,000 alerts for the same period. The community detection algorithm partitioned the alert graph into 15 communities. These communities were then manually investigated for malicious indications.

One example of such suspicious behaviour is shown in Fig. 3. Three internal IPs (192.168.2.45, 192.168.2.50, 192.168.2.96) were connected through graph edges based on pairwise similarity scores from their word embeddings. An external IP (209.251.75.37) was associated with RDP brute force attacks targeting all three internal IPs, which raised separate alerts from IDS, anomaly detectors, and rules-based alerts. Two internal IPs captured attempts to exploit a known Windows OS vulnerability. Internal IP 192.168.2.96 also raised alerts for incoming executable shell code originating from the same external IP and a brute force attack by external IP 50.28.16.195. Since each pair of internal IPs was highly similar, they were connected by edges in the graph. Since all three experienced alerts of a similar nature and from similar sources, these events could then be linked and could eventually be connected to other events that indicate a campaign of attack.

V. Future Work

A. Feature Embeddings

A natural extension of the feature embeddings discussed in this work is to train one or more models using input from additional event fields and event types. Other event types contain categorical fields that appear in some alert types and could affect alert correlations. For example, http events contain URIs, and host-based alerts contain executable file paths.

There are also alternate methods for producing feature embeddings that are worth exploring for this application. Many of the available event types can be viewed as forming graphs themselves, e.g. network connections between hosts and parent-child process trees. Algorithms like Deepwalk [13] and node2vec [14] learn continuous feature representations of vertices and their neighbourhoods within a graph.

B. Alert Correlation

In this work we calculate similarity between IP addresses appearing in alerts. Given feature embeddings for more categorical features of alerts, we can calculate a more nuanced similarity between the alerts themselves. The Word Mover's Distance [15] function leverages similarities between pairs of words to calculate similarities between documents. By treating an alert as a structured document, expert knowledge of the alert types could be incorporated by penalizing the distance calculated for word movement between certain parts of the documents.

C. Graph Construction and Analysis

To reduce the noise of repeated similar alerts and similar graph structures, similar alerts could be combined into one graph vertex. During construction, an iterative process could further collapse similar alerts with similar graph neighbourhoods together using a graph complexity metric to decide when alerts have been sufficiently combined, with the goal of producing a graph that highlights structure between alerts, or groups of alerts, that aren't naively similar but are still related.

Another area worth pursuing is focusing on attack evolution. When available, the Cyber Kill Chain stage or MITRE ATT&CK tactic of an alert could influence either the graph edge creation and clustering or the scoring of resulting graphs. Security analysts are familiar with these models, and the models exist to represent real attacks, so an alert graph containing alerts in multiple stages and following a sensible path through the kill chain is more likely to be interpreted as an attack by an analyst.

References

[1] Eshete, Birhanu and Gjomemo, Rigel and Hossain, Md Nahid and Momeni, Sadegh and Sekar, R and Stoller, Scott and Venkatakrishnan, VN and Wang, Junao, "Attack Analysis Results for Adversarial Engagement 1 of the DARPA Transparent Computing Program" arXiv preprint arXiv:1610.06936, 2016.

[2] J. Lambert, "Defenders think in lists. Attackers think in graphs. As long as this is true, attackers win." Defender Mindset Blog: https://docs.microsoft.com/en-us/archive/blogs/johnla/defenders-think-in-lists-attackers-think-in-graphs-as-long-as-this-is-true-attackers-win Apr. 2015

[3] H. Ren, "Statistical Causality Analysis of Alert Correlation Feature Selection," M.Sc. Thesis, Faculty of Computer Science, The University of New Brunswick, April 2010. Accessed on October 6, 2019

[4] W. Lee, X. Qin, "Statistical Causality Analysis of Infosec Alert Data," in Managing Cyber Threats: Issues, Approaches, and Challenges, Springer 2005.

[5] Shittu, Riyanat O, "Mining intrusion detection alert logs to minimise false positives & gain attack insight" City University London, 2016.

[6] S. Haas, M. Fischer, "On the Alert Correlation Process for the Detection of Multi-step Attacks and a Graph-based Realization," Applied Computing Review, vol. 19, no. 1, pp. 5-19, Mar. 2019.

[7] K. Pei, Z. Gu, B. Saltaformaggio, S. Ma, F. Wang, Z. Zhang, L. Si, et al. "HERCULE: Attack Story Reconstruction via Community Discovery on Correlated Log Graph," ACSAC '16, Dec. 2016.

[8] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, J. Dean, "Distributed Representations of Words and Phrases and their Compositionality," in Proceedings of the 26th International Conference on Neural Information Processing Systems, vol. 2, pp. 3111-3119, Dec. 2013.

[9] M. Ring, A. Dallmann, D. Landes and A. Hotho, "IP2Vec: Learning Similarities Between IP Addresses," 2017 IEEE International Conference on Data Mining Workshops (ICDMW), New Orleans, LA, 2017, pp. 657-666.

[10] H. Caselles-Dupré, F. Lesaint, J. Royo-Letelier, "Word2vec applied to recommendation: hyperparameters matter," RecSys '18: Proceedings of the 12th ACM Conference on Recommender Systems, pp. 352–356, Sept. 2018.

[11] V. Blondel, J.-L. Guillaume, R. Lambiotte, E. Lefebvre, "Fast unfolding of communities in large networks," Journal of Statistical Mechanics: Theory and Experiment, 2008.

[12] A. Clauset, M.E.J. Newman, C. Moore, "Finding community structure in very large networks,“ arXiv:cond-mat/0408187v2. [cond-mat.stat-mech]

[13] B. Perozzi, R. Al-Rfou, S. Skiena, "Deepwalk: Online learning of social representations," KDD '14: Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 701–710, Aug. 2014

[14] A. Grover, J. Leskovec, "node2vec: Scalable Feature Learning for Networks," KDD '16: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 855-864, Aug. 2016.

[15] M. Kusner, Y. Sun, N. Kolkin, K. Weinberger, "From word embeddings to document distances" ICML'15: Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37, pp. 957–966, Jul. 2015.