On the Use and Reuse of Graphs for Network Security with Real-Time Edge Learning

Michael M. Jerge
Riverside Research, 2640 Hibiscus Way, Beavercreek, OH 45431

Virgil O. Barnard Iv
Riverside Research, 2640 Hibiscus Way, Beavercreek, OH 45431

Grant Fullenkamp, University of Cincinnati
Riverside Research, 2640 Hibiscus Way, Beavercreek, OH 45431

Andy Klawa, Penn State University
Riverside Research, 2640 Hibiscus Way, Beavercreek, OH 45431

ABSTRACT

We present a novel approach in network security using unsupervised online machine learning method at the edge, through graph learning. The proposed system takes advantage of an online learning paradigm, by collecting real network data to build a ground truth of a network's topology, using shallow graph neural networks (GNNs). Our proposed solution includes an edge-based infrastructure, through K3s and Kafka, which could then scale to match the needs of larger networks. We then perform simple cyber-attacks and show how visual analysis can identify malicious behaviors, without any prior labeled data. Our results against simple attacks show promise that improved graph analytics should capture even more complex attack vectors. We then conclude with some suggestions for improved edge deployment, against larger and more complex networks.

CCS CONCEPTS

• **Graph Neural Networks**; • **Cybersecurity**; • **Semi-Supervised Learning**;

ACM Reference Format:
Michael M. Jerge, Virgil O. Barnard Iv, Grant Fullenkamp, University of Cincinnati, and Andy Klawa, Penn State University. 2023. On the Use and Reuse of Graphs for Network Security with Real-Time Edge Learning. In *2023 the 12th International Conference on Networks, Communication and Computing (ICNCC) (ICNCC 2023), December 15–17, 2023, Osaka, Japan.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3638837.3638848

1 INTRODUCTION

The proliferation of Internet of Things (IoT) devices and the push for real-time, low latency computing has heralded a shift from centralized cloud computing to edge computing. One of the most significant threats faced in this new landscape is the increased exposure to cybersecurity breaches. The heterogeneity and sheer number of devices operating at the network periphery can create various entry points for cyber attackers. Moreover, the limited processing capabilities of some edge devices, the dynamic nature of edge networks, and the often less secure environments they operate in make traditional cybersecurity measures less effective. Additionally, with this proliferation of devices comes the increased vulnerabilities and attack surface related to the interconnection of edge devices. The recent incident of the Mirai malware attacks of 2023 [1]–[3] involved the targeting of a wide array of IoT devices and impacted devices from home appliances to industrial control systems. The attack, which relied on the increasing attack landscape, exploited a vulnerability in the firmware of millions of IoT devices. For example, once hackers breached the devices, they could control and manipulate the network control protocols and software within the devices, causing disruptions across the globe.

This recent incident not only underscored the risks involved in an increasing reliance on IoT devices, but also highlighted the intricacies of securing dynamic, ever-evolving network information and configuration. Since IoT networks are dynamic in nature, new devices are continuously being added, and continuous updates are constantly being rolled out to firmware and software. Perhaps one of the most common methods by which attackers gain access to networks is the process of lateral movement. Lateral movement is a key stage of advanced persistent threats (APTs) where an attacker tries to expand control over other machines in a network after compromising an endpoint, attempting to gain access to systems and credentials necessary to carry out their mission [4], [5]. Traditional lateral movement detection methodologies rely on tedious sifting of audit logs as recorded means to address anomalous behavior within networks. Government entities often require themselves, and their contractors, to follow standards and adhere to security guidelines, like the National Institute of Standards and Technology (NIST) documentation 800-53, with audit logs existing within its own control family. However, these controls are limited in their approach, insofar that networks are not all identical in their characteristics. To alleviate these concerns, security analysts create, compile, and revise audit trails as a means of consolidating applicable security logs, further appropriating them using frameworks such as MITRE ATT&CK [6]. Security information and event management tools currently exist for aggregating and structuring audit log data, but this approach is also intrinsically flawed. Most traditional anomaly detection methods look for triage patterns of known attacks, which means that any unexpected or undocumented control flow may go undetected. Therefore, anomalous activity may exist, but since the detected effect may not logically flow from the correct root cause, the detected anomaly transforms into a false positive.

ICNCC 2023, December 15–17, 2023, Osaka, Japan

ACM ISBN 979-8-4007-0926-5/23/12
https://doi.org/10.1145/3638837.3638848

Despite the failings of traditional based approaches, new methods have been developed to address the problem of detecting anomalies in cyber networks [4], [7], like detecting lateral movement. Attacks against authentication mechanisms themselves are varied, and the data associated with each are varied themselves. Nevertheless, standard industry authentication mechanisms, such as Kerberos and OAuth, maintain mechanisms to log authenticating clients and history of logging within a network. Furthermore, authentication log data can be enhanced through other various types of data, such as DNS queries and Cisco NetFlow, to gain more features and characteristics. Analyzing lateral movement through traditional rules-based detection is costly and time-consuming, and usually do not identify any useful results due to a lack of compromising indicators. Additionally, as has been stated, security frameworks are limiting in efficacy as well.

Recognizing these challenges, researchers and industry professionals have started to deploy machine learning (ML) models at the edge for enhanced threat detection and mitigation. ML-based anomaly detection systems can identify deviations from standard behaviors, thereby allowing for prompt detection and remediation of cyber threats. However, deploying such models on the edge is fraught with its unique set of problems, such as the lack of processing power of edge devices.

The collection of new works in this area show that graphs can be used to inform cyber-specific behaviors, and detect cyber-attacks, while being computationally efficient. By adopting an unsupervised learning model, analysts will be able to robustly estimate parameters by which different network variables are related, with an emphasis on establishing causal relationships in anomaly detections, without need pre-existing labeled data. Graph neural networks (GNNs) are an unsupervised learning algorithm, where nodes are connected by some edges to other nodes. In a network application, each node will represent some feature of the network, with their edge weights displaying their probability of connection [8]. Brian Powell of the Johns Hopkins Applied Physics Laboratory [9] has developed an unsupervised approach to look for behavior-based defense, to replace traditional rule-based frameworks. This approach first clustered systems according to their role, and then identified the patterns of communication between each system from which behavior can be identified. Although the framework itself addresses the need to adopt behavioral-based machine learning techniques, the use was restricted to enterprise networks that may utilize enterprise devices. Similarly, the Graph Computing Lab of the George Washington University [4, 10–12] developed an unsupervised graph machine learning technique that uses industry standard logging data for their models. Victor-Alexandru Dravariu, Stephen Hailes, and Mirco Musolesi of the University of College London have recently published multiple works [12], [13] in the areas of unsupervised graph network searching, and its improved performance over Reinforcement Learning (RL) models but are missing the application with real network data collections.

This works explores the complexities associated with deploying ML models for anomaly detection at the edge, the potential cybersecurity risks they can mitigate, and the new vulnerabilities they may introduce. This work presents an unsupervised, online learning, graphical machine learning technique to detect anomalous movement and activity movement in IoT networks. Our results show how one can utilize common network characteristics and construct a graphical representation of the network built from live network data, instead of expecting cleaned and transformed data captures. We then show how the system responds to real cyber- attacks, and present future work to build further into more complex anomaly detection.

2 MATERIALS AND METHODS

2.1 Infrastructure

Figure 1 represents our full proposed pipeline, all the subsystems, and their connections. The proposed infrastructure consists of a Raspberry Pi 4 Model B+ cluster, hereafter referred to as a bramble. The structure of the bramble design includes a principal Raspberry Pi (head node) and five auxiliary Raspberry Pis (worker nodes). While we limited our system to just five total worker nodes, due to cost and power consumption, the proposed solution is scalable towards a larger and more complex problem domain. Each of the nodes utilizes Ubuntu 22.10 OS with the head node using the desktop version and the worker nodes using the server version. The cluster is instantiated by running a bash script on the head node which searches the local area network using various, commonly used applications such as ipcalc, nmap, and itertools. The script looks for MAC addresses registered to Raspberry Pis on the network and adopts the IP addresses to an embedded system once discovered.

The nodes are connected using an edge-based Kubernetes variant cluster referred to as K3s. It is purpose- built for lightweight edge computing and for machines with constrained resources, such as Raspberry Pi's. Our proposed system allows Kubernetes to oversee an assemblage of Raspberry Pis as a singular unit. It facilitates equitable distribution of tasks, aptly manages system failures, and guarantees proficient resource utilization. When interconnected, the Raspberry Pis form a robust computational network that exploits the principles of distributed computing. The compatibility of K3s with Raspberry Pis stems from its inherent lightness. Conventional Kubernetes configurations encompass a vast array of components and services, a significant portion of which may not be necessary for less extensive or edge deployments. K3s eschews these superfluous elements, resulting in a binary of less than 50MB. Despite its compact nature, K3s offers full Kubernetes conformity, thereby ensuring seamless operations of regular Kubernetes applications and configurations. Upon configuration, the master node regulates the worker nodes, allotting tasks as required.

Once the bramble is instantiated with K3s, Docker [14] images and containers are used to stream data in real-time and store the data in a non-relational database. The pipeline utilizes Apache Kafka streaming containers [15] for real-time, online training, and MongoDB as a non-relational database as these tools are well suited to the demands of online real-time anomaly detection in cybersecurity due to their speed, scalability, and flexibility. Kafka, a high-throughput distributed messaging system, is ideal for managing streaming data and real-time analytics, which are critical for detecting anomalies in user behavior or network traffic [7]. It provides a fault-tolerant way to handle real-time data feeds, making it easier to identify unusual patterns as they occur. MongoDB complements Kafka's capabilities by providing a flexible schema for storing and analyzing data, which is essential given the diverse

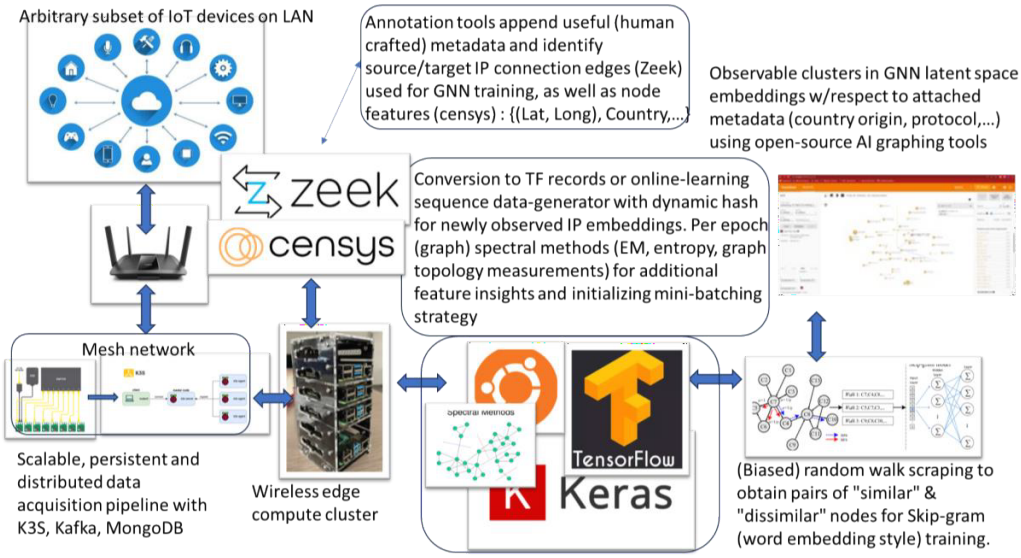

Figure 1: Proposed full system diagram.

range of data types and structures involved in cybersecurity. Its document-oriented structure allows for easy horizontal scaling, ensuring that data can be processed quickly even as the volume increases. In combination, Kafka's real-time data processing and MongoDB's flexible, scalable data storage provide a powerful infrastructure for identifying and responding to cybersecurity threats as they arise.

2.2 Data

Our infrastructure collects live data from a wireless network interface but is capable of additionally collecting data from other various sources, such as ethernet and Bluetooth modalities. TCP/IP network data is collected in the form of full packet captures, Zeek logs, Scapy [16] packets, and Censys third-party data information enrichment. Zeek is an open-source network security monitoring tool. Although every node within the bramble is designed to capture full packet captures that contain all layers of OSI data, from link layer to application layer, the intent is to allow the head node to primarily capture information, thus delegating parsing to other parts of the bramble. This is useful for deep traffic analysis and for capturing network anomalies that may indicate cyber threats. Zeek (formerly known as Bro) is a powerful network analysis framework that is much different from the typical IDS you may know. It places a high-level semantic layer on packet data, often simplifying the process of network traffic analysis and the development of custom security solutions.

Packet Capture (pcap) files represent raw data collected from the network layer and above, which include all the frames in their entirety. Information that can be gleaned from pcap files ranges from low-level network information to high-level data. On the lower levels, you can see data like source and destination IP addresses, source and destination ports, the protocols being used (TCP, UDP, ICMP, etc.), and MAC addresses. On the higher levels, pcap files can contain payload data from the application layer protocols like HTTP, SMTP, FTP, and others, which might include usernames, passwords, and other sensitive information if they are not encrypted.

However, pcaps only provide raw data. To interpret this data and make it meaningful, we need tools like Wireshark for packet analysis [17] or Zeek for generating high-level network logs. While focusing on network security monitoring, Zeek provides a comprehensive platform that allows more general traffic analysis as well. Zeek's powerful scripting language allows for the creation of more sophisticated analyses. Zeek logs are generated from pcap data, but they are much easier to read and interpret. Zeek logs are organized into different types, each with its own set of fields that are relevant to a particular protocol or type of network activity.

Some of the most used Zeek logs include:

conn.log: Provides a record of each connection seen on the network, with details like source and destination IPs, ports, connection duration, and amount of data transferred.

http.log: Contains details about HTTP transactions, such as the client and server, the method (GET, POST, etc.), the status code, user agent, etc.

dns.log: Logs all DNS requests and responses. It includes fields for the DNS query, the response, the DNS record type (A, MX, CNAME, etc.), and response codes.

files.log: Tracks file transfers over supported protocols (HTTP, FTP, SMTP, etc.). It includes information like the filename, size, source and destination, and a hash of the file.

ssl.log: Provides information about SSL/TLS connections, such as cipher suite, certificate issuer and subject, and certificate key length.

We also introduce Scapy collected packets as additional data ingested into the graph sequence generator. Scapy is a Python library for network packet manipulation and analysis. It can construct or decode packets of a wide number of protocols, send them over the wire, capture them, match requests and replies, and much more. Scapy deals with raw packets, similar to traditional packet

capture using systems such as Pyshark and Wireshark but includes additional information as well. This includes:

Layer information: Scapy lets you work with packets on every OSI layer from layer 2 (data link) up to layer 7 (application). This includes the Ethernet frame, IP header, TCP/UDP header, and application layer data.

Payload data: Scapy can dissect the payload to read the actual data being sent, provided it's not encrypted.

This can be HTTP data, DNS queries, or any other protocol that Scapy supports.

Packet metadata: Scapy also shows the metadata about the packet, like source and destination IP addresses and ports, protocol type, packet length, etc.

Lastly, we incorporate public, internet wide databases from Censys. Censys is a public search engine that enables researchers to quickly ask questions about the hosts and networks that compose the Internet. This is particularly useful when an external IP address is found within the pcaps, as Censys can provide rich contextual information about the IP.

Censys maintains three datasets:

Hosts: These are Internet hosts (IPv4 and IPv6) that have had one or more open ports when scanned. Information includes IP address, open ports, location, Autonomous System Numbers (ASN), operating system, services, and more.

Websites: This dataset contains websites, their location, server software, headers, certificates, cookies, third-party requests, and more.

Certificates: Information about all publicly observable TLS certificates, including issuance data, issuing CA, key details, and more.

In summary, we incorporate four types of data modalities into our graph sequence generator. Packet Capture (pcap) files constitute the raw data, capturing all the network layer and above information in their purest form. This encompasses data from low-level network details such as source and destination IP addresses, source and destination ports, protocols being used, and MAC addresses, up to higher-level application layer data like HTTP, SMTP, FTP payloads which could potentially contain sensitive information if not encrypted. Zeek logs serve to interpret and contextualize this raw pcap data, presenting it in a more digestible, high-level format. Zeek organizes data into a series of different logs, each dedicated to a particular type of network activity or protocol. This can include logs dedicated to connection records, HTTP transactions, DNS requests and responses, file transfers over supported protocols, and SSL/TLS connections. Scapy packets delve into the realm of network packet manipulation and analysis. Scapy data includes packet layer information, payload data, and packet metadata, providing a fine-grained inspection of network packets. Lastly, the Censys data adds another layer of context to the analysis by providing a macro-level view of the internet hosts and networks involved in the network traffic. When an external IP address is found within the pcaps, Censys can be used to glean rich metadata about the IP address, the domains it hosts, the network it's a part of, and the geographical location it's associated with. Altogether, this combination of Pyshark, Zeek, Scapy, and Censys data offers a comprehensive and detailed look at network activity. This allows for efficient network monitoring and anomaly detection, aids in diagnosing network issues, and grants a deeper understanding of overall network behavior.

2.3 Graph Neural Network

For online training of graph neural networks, we employ a version of the node2vec algorithm [18]. Node2vec is a machine learning algorithm that uses the structure of a network to generate embeddings for its nodes. It's designed to balance the exploration of local and global network structures and can be useful in a variety of applications, including anomaly detection in cybersecurity. In network defenses, one common task is the detection of anomalies in network traffic or user behavior, which could be indicative of a security threat such as a network intrusion, malware, or a malicious insider. These data can be modeled as a graph, where nodes represent entities and edges represent interactions between them. Using node2vec, each node in the graph is mapped to a vector in a high-dimensional space in such a way that nodes with similar neighborhood structures in the graph are close to each other in this space. For example, two devices that have similar patterns of network traffic could end up with similar embeddings. These embeddings can then be used as input to a machine learning model that is trained to distinguish normal behavior from anomalies.

The strength of node2vec in this application comes from its ability to capture the complex, higher-order relationships between entities in the graph. Traditional methods might treat each interaction in isolation, but node2vec understands that the relationships between nodes are important and can capture patterns that other methods might miss. For example, it can identify when a device is behaving unusually not just because of its individual actions, but because of the pattern of its interactions with other devices. This approach has several advantages. One is that it can detect anomalies even when they are subtle or complex since it understands the normal patterns of behavior in the network at a deep level. Another is that it can generalize from known types of threats to identify new ones. For example, if a new type of malware causes a device to behave similarly to known malware in the vein of network interactions, the model can potentially identify it as an anomaly, even though the type of malware has previously been unforeseen.

Being an unsupervised learning algorithm, node2vec doesn't require labeled data to operate, an advantage in cybersecurity where new threats constantly emerge and labeled data for these novel threats is not immediately available. Instead, it learns to represent nodes based on their network structure, independent of any specific task, allowing it to capture normal patterns of behavior and detect deviations from these patterns as potential anomalies. This significantly reduces the manual effort and time required to continuously update the model with new labeled data, making the model robust in identifying novel threats. Additionally, node2vec is scalable. As networks grow larger and more complex, scalable methods are required to handle the increasing volume of data. Node2vec uses efficient techniques for generating embeddings, making it capable of handling large-scale networks. By reducing the complex network structure into lower-dimensional embeddings, node2vec makes it feasible to apply complex detection algorithms that might not be practical to run directly on the original network data. Moreover,

node2vec is well-suited for deployment in edge computing environments. Edge computing pushes applications, data, and computing power away from centralized points to the logical extremes of a network, near the source of the data. This provides benefits in terms of latency and bandwidth usage. Due to the efficiency of node2vec, it can be run directly on a Raspberry Pi bramble, providing real-time anomaly detection at the edge of the network, close to where the data is generated. This approach can improve the speed and efficiency of anomaly detection since it reduces the need for data to be sent back and forth across the network.

3 SEMI-SUPERVISED ONLINE LEARNING

The proposed approach harnesses dynamic directional IP graphs to enhance online training of the GNN node2vec for real-time network traffic analysis. These evolving graphs, constructed in epochs from captured packet data, capture changing network topology and directional flow. Edge weights, determined using the Newman EM algorithm and Shannon entropy applied to packet data, represent the strength of connections between IP addresses. This dynamic adaptation ensures accurate node embeddings, facilitating prompt anomaly detection. While challenges such as computation and graph coherence exist, the integration shows potential for effective network management and security.

3.1 Ground Truth and Graph Expectation Maximization of Network Data

We define a graph representation of an information technology network as a heterogeneous, directed graph denoted by $G = (V, E)$where V is a set of nodes ($v1$, $v2$, $v3$, ...vn), where n is the number of nodes (dynamic in our case), and E is a set of edges. Furthermore, since the graph is heterogeneous, each relationship of type r between nodes can be represented as (vi, r, vj) where vi, vj ε V. In our case r is observed traffic between devices on the network. The different forms of network traffic gathered by our system give us the ability to reconstruct network topology in real-time, even as new devices join and exit the network. Multimodal network data is prone to insufficient authenticity, inaccuracy, limited traffic size and privacy security risks [19]. The ground truth of the network, which is the real devices and systems currently on the network, are often difficult to obtain especially in the context of small network topologies. When determining this baseline network structure, the true topology of the network is prone to errors when building graph structure.

To overcome this problem, we employ a type of an expectation-maximization (EM) algorithm, proposed by Newman, to best estimate the ground truth from noisy network data [20]. The EM algorithm is a well-known statistical technique used for finding the maximum likelihood estimates of parameters in statistical models when the data is incomplete or has missing values [21]. Newman's work on network structure analysis shows how the EM algorithm can be applied to infer hidden structures from observed data that might be noisy or incomplete. Given observed data that represents connections between nodes in a network, the goal is to infer the underlying structure of the network, including hidden or latent connections. The observed data is assumed to be noisy, containing false positives and false negatives, and the challenge is to recover the true network structure. We assume that there is an underlying true network represented by a matrix A, where $Aij = 1$ if there is a connection between nodes i and j, and $Aij = 0$ otherwise. The observed network is represented by a matrix B, where the entries are influenced by noise. The EM algorithm is used to iteratively estimate the true network matrix A from the observed matrix B. The algorithm consists of two main steps: the Expectation (E) step and the Maximization (M) step. In the first step, the expectation of the log-likelihood function is computed with respect to the current estimate of the true network. This involves calculating the probability distribution of the hidden variables given the observed data and the current estimate of the parameters. For example, the expectation of the connection Aij given the observed connection Bij and the current parameter estimates is calculated as seen in equation 1.

$$E\left(A_{ij} \middle| B_{ij}\right) = p\left(A_{ij} = 1 \middle| B_{ij}\ ; \theta\right) \tag{1}$$

θ represents the current estimates of the model parameters. In the second step, the parameters are updated to maximize the expected log-likelihood calculated in the first step. This involves finding the values of the parameters that maximize the expectation, typically through numerical optimization techniques. The E and M steps are iterated until convergence, resulting in the final estimates for the true network structure. Algorithm 1 shows this process in more detail.

Algorithm 1 EM Algorithm

```
# Initialize the parameters theta (e.g., probabilities related to noise)
initialize(theta)
# Initialize the estimated true network matrix A, possibly using the
observed network B initialize(A)
# Set a convergence criterion (e.g., change in log-likelihood or
parameters) convergence_criterion = 0.001
# Track changes to assess convergence change = float('inf')
# Iterate the EM algorithm until convergence while change >
convergence_criterion:
# E-Step: Compute the expected values of the true network A
given observed network B for each connection (i, j) in B:
E_A_ij = compute_expectation(B[i, j], theta) # E.g., p(A_{ij} = 1 |
B_{ij}; theta)
A[i, j] = E_A_ij
# M-Step: Update the parameters theta to maximize the expected
log-likelihood new_theta = maximize_log_likelihood(A, B)
# Calculate change to assess convergence (could be change in
parameters or log-likelihood) change = calculate_change(theta,
new_theta)
# Update theta for the next iteration theta = new_theta
# The final estimate of the true network is in A return A
```

3.2 Entropy Calculation

To further assess the ground truth of the network, we incorporate the calculation of Shannon ($H = \sum_{i=1}^{k} \text{pi} * \log2(\text{pi})$) and byte entropy [22] of packets being sent between IP addresses. The measured uncertainty of the packet captures can then be used to build

a better topology of the network. The model will then penalize abnormally high entropy connections in the final network topology.

3.3 Feature Representations

The core of the proposed approach lies in the utilization of dynamic directional IP graphs, which are continuously updated in real time using captured packets, Scapy packets, and Zeek logs. These dynamic graphs serve as the fundamental basis for generating feature representations during the online training of the GNN node2vec model. Unlike conventional static graphs, these dynamic graphs capture the real-time evolution of network topology and information flow directions. This dynamic aspect enriches the feature space by incorporating the latest network interactions and enables more accurate representations of IP address nodes and their contextual roles within the network.

Algorithm 2 Combining EM and Shannon/Byte Entropy Pseudocode:

```
# Perform measurements
EM algorithm (algorithm 1)
compute Shannon Entropy
compute Byte Entropy
combined score between EM and Shannon/Byte
return combined score
```

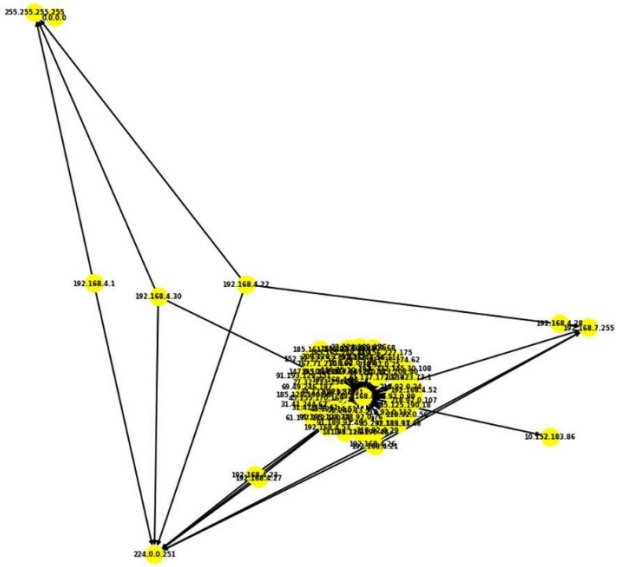

Figure 2: Directed IP graph at the end of a training session. Each yellow node is a real IP address on the network, with their connections shown in black. The clustered nodes, mostly in the center of the graph, show which captured packets best reflect the ground truth.

3.4 Graph Sequence Generator

The process commences with the construction of directional IP graphs based on captured network packets from Pyshark, Scapy, and Zeek, organized into time intervals known as epochs. Each epoch contributes to the evolving graph, with individual IP addresses acting as nodes and directional edges carrying weighted information flow. The edge weights are computed using the Newman EM algorithm and Shannon entropy, drawing from packet data to gauge the strength and significance of connections between nodes. This mechanism ensures that the graph synchronizes with real-time network activities and encapsulates changing communication patterns. Figure 2 shows an example of an IP graph at the end of generation.

3.5 Online Training

The dynamic nature of the directional IP graph aligns seamlessly with the concept of online training for the GNN node2vec model. Traditional node2vec algorithms are trained on static graphs, which fail to capture real-time changes. However, the proposed approach embraces these changes by continually updating the graph's nodes and edges with newly captured packet data from the most recent epoch. This continuous evolution of the graph enables the GNN to adapt to emerging patterns, detect anomalies promptly, and make informed decisions in real time.

To generate meaningful training examples for the GNN Node2Vec, the approach employs biased random walks. These walks simulate how information propagates through the network. Starting from a specific node (e.g., an IP address), the walk selects the next node probabilistically, considering edge weights as determined by the Newman EM algorithm and Shannon entropy. The introduction of return (p) and in-out (q) parameters allow control over exploration depth and local vs. distant node bias. This balanced exploration technique ensures that the walk captures both local and global network behaviors, aligning well with the GNN's learning needs.

In summary, the integration of biased random walks with the proposed approach creates a synergistic framework for real-time network traffic analysis. These walks enhance the training process by generating representative sequences of nodes that mirror actual network interactions, while the GNN node2vec model leverages these sequences to understand and adapt to the evolving network dynamics. This integrated approach holds potential for effective network management, security, and anomaly detection.

3.6 Model Performance

The integration of dynamic directional IP graphs and online training has substantial implications for the performance of the GNN node2vec model in real-time network traffic analysis. The adaptability introduced by the dynamic graph enables the model to maintain relevance in the face of evolving network behaviors. Directional context, often neglected in traditional GNN applications, enhances the accuracy of node embeddings, as the model can differentiate between incoming and outgoing interactions. As a result, the model becomes more adept at detecting anomalies, understanding network dynamics, and contributing to effective network management.

The latent space showed separation of device by rough use case, encoding temporal and physical features of the network. Internal and external IP addresses can be easily separated. Message details like byte entropy were also encoded by the network resulting in

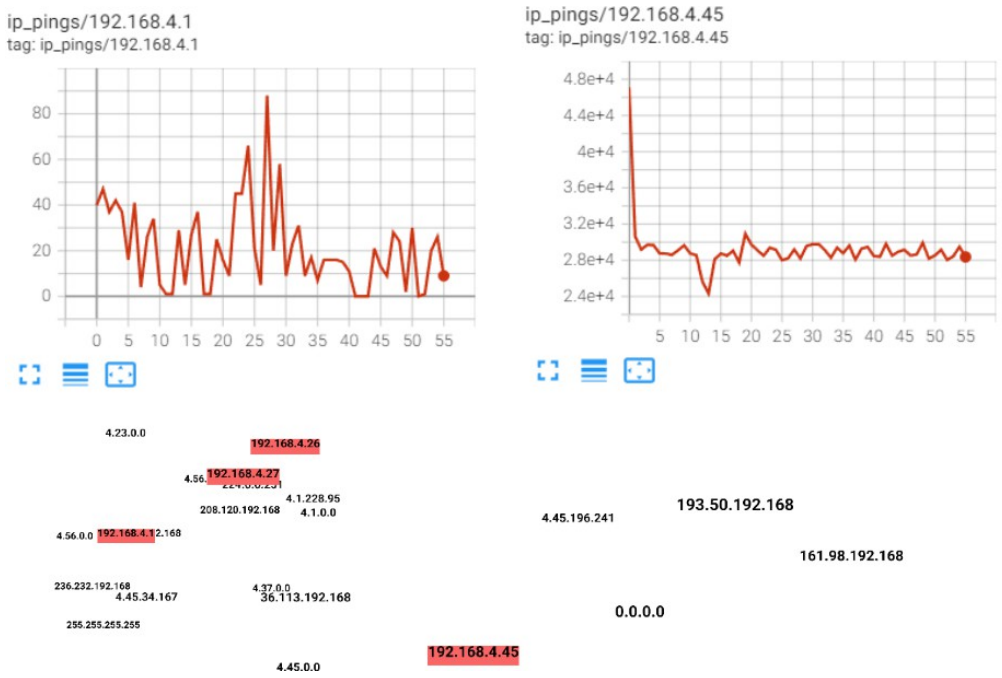

Figure 3: TensorBoard IP frequency per epoch scalars for the IP addresses 192.168.4.1 and 192.168.4.45 and TensorBoard embedding projector calculating PCA on the node2vec model embeddings of all seen IP addresses

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	185.125.190.48	192.168.4.45	ICMP	219	Echo (ping) request id=0x0000, seq=0/0, ttl=64 (reply in 2)
2	0.000232998	192.168.4.45	185.125.190.48	ICMP	219	Echo (ping) reply id=0x0000, seq=0/0, ttl=64 (request in 1)
3	0.000430051	185.125.190.49	192.168.4.45	ICMP	117	Echo (ping) request id=0x0000, seq=0/0, ttl=64 (reply in 4)
4	0.000549143	192.168.4.45	185.125.190.49	ICMP	117	Echo (ping) reply id=0x0000, seq=0/0, ttl=64 (request in 3)
5	0.000563106	185.125.190.58	192.168.4.45	ICMP	119	Echo (ping) request id=0x0000, seq=0/0, ttl=64 (reply in 6)
6	0.000619364	192.168.4.45	185.125.190.58	ICMP	119	Echo (ping) reply id=0x0000, seq=0/0, ttl=64 (request in 5)
7	0.001051508	180.169.85.126	192.168.4.45	ICMP	162	Echo (ping) request id=0x0000, seq=0/0, ttl=64 (reply in 8)
8	0.001194785	192.168.4.45	180.169.85.126	ICMP	162	Echo (ping) reply id=0x0000, seq=0/0, ttl=64 (request in 7)
9	0.010067274	185.125.190.48	192.168.4.45	ICMP	219	Echo (ping) request id=0x0000, seq=0/0, ttl=64 (reply in 10)
10	0.010253884	192.168.4.45	185.125.190.48	ICMP	219	Echo (ping) reply id=0x0000, seq=0/0, ttl=64 (request in 9)

Figure 4: A DDoS attack that attempts to make an online service unavailable by overwhelming it with traffic from multiple sources.

distinct separations in the latent space. Clusters emerged not only from the device's usage but also from the inherent characteristics of communication protocols, packet types (e.g. Scapy, Pyshark, Zeek), and even geographic locations (e.g. Censys), without direct incorporation of this data during training. The latent space managed to capture these patterns and variations solely through the input of IP addresses and their connections. The representation allowed for an intriguing exploration of the network's dynamics, suggesting that IP addresses alone carry a significant amount of information about the network's structure and behavior. Figure 3 shows the performance of the model over time for two separate IP addresses.

While the approach offers considerable advantages, challenges arise in terms of computational complexity, graph coherence during updates, and optimization strategies for real-time calculations. Future research could delve into refining algorithms for efficient edge weight recalculations, ensuring the model's stability during graph updates, and exploring techniques to minimize disruptions. Additionally, investigating the interplay between directionality, node2vec parameters, and GNN architecture could lead to further enhancements.

4 NETWORK ATTACKS

Herein we discuss our testing capabilities and the procedures that we took in order to simulate deviations from probabilistic network activity. Evidentiary capabilities are only as good as the tests that they purport to disprove, and in the testing phase of our operation, the formulated hypotheses are rigorously examined, insofar as the null hypothesis asserts that a given observation falls within the expected behavioral patterns of the system, whereas the alternative hypothesis claims the contrary. The process involves applying statistical tests to measure how the new or real-time data aligns with the established normal behavior model. Subsequently, each observation is assigned an anomaly score reflecting its deviation from the norm. Higher scores typically denote greater abnormality, so long as the range provides reference for greater or lesser degree of deviation. These scores are used in the thresholding and alerting phase. By setting carefully calibrated thresholds, the system can

No.	Time	Source	Destination	Protocol	Length	Info
52	0.413344829	192.168.4.45	192.168.7.255	ICMP	800	Echo (ping) request id=0x0000, seq=0/0, ttl=255 (no response fo…
53	0.413523364	192.168.4.45	192.168.7.255	ICMP	76	Echo (ping) request id=0x0000, seq=0/0, ttl=255 (no response fo…
54	0.414735559	192.168.4.45	192.168.7.255	ICMP	841	Echo (ping) request id=0x0000, seq=0/0, ttl=255 (no response fo…
55	0.415278536	192.168.4.45	192.168.7.255	ICMP	425	Echo (ping) request id=0x0000, seq=0/0, ttl=255 (no response fo…
56	0.417409224	192.168.4.30	192.168.4.45	ICMP	577	Echo (ping) reply id=0x0000, seq=0/0, ttl=64
57	0.417888906	192.168.4.23	192.168.4.45	ICMP	577	Echo (ping) reply id=0x0000, seq=0/0, ttl=64
58	0.417973350	192.168.4.23	192.168.4.45	ICMP	424	Echo (ping) reply id=0x0000, seq=0/0, ttl=64
59	0.417991905	192.168.4.23	192.168.4.45	ICMP	339	Echo (ping) reply id=0x0000, seq=0/0, ttl=64
60	0.418008794	192.168.4.23	192.168.4.45	ICMP	509	Echo (ping) reply id=0x0000, seq=0/0, ttl=64
61	0.418025108	192.168.4.23	192.168.4.45	ICMP	800	Echo (ping) reply id=0x0000, seq=0/0, ttl=64
62	0.418043238	192.168.4.23	192.168.4.45	ICMP	76	Echo (ping) reply id=0x0000, seq=0/0, ttl=64
63	0.418058386	192.168.4.23	192.168.4.45	ICMP	841	Echo (ping) reply id=0x0000, seq=0/0, ttl=64
64	0.421208788	192.168.4.27	192.168.4.45	ICMP	577	Echo (ping) reply id=0x0000, seq=0/0, ttl=64
65	0.422001171	192.168.4.27	192.168.4.45	ICMP	424	Echo (ping) reply id=0x0000, seq=0/0, ttl=64
66	0.422106985	192.168.4.27	192.168.4.45	ICMP	339	Echo (ping) reply id=0x0000, seq=0/0, ttl=64

No.	Time	Source	Destination	Protocol	Length	Info
52	0.413344829	192.168.4.45	192.168.7.255	ICMP	800	Echo (ping) request id=0x0000, seq=0/0, ttl=255 (no response fo…
53	0.413523364	192.168.4.45	192.168.7.255	ICMP	76	Echo (ping) request id=0x0000, seq=0/0, ttl=255 (no response fo…
54	0.414735559	192.168.4.45	192.168.7.255	ICMP	841	Echo (ping) request id=0x0000, seq=0/0, ttl=255 (no response fo…
55	0.415278536	192.168.4.45	192.168.7.255	ICMP	425	Echo (ping) request id=0x0000, seq=0/0, ttl=255 (no response fo…
56	0.417409224	192.168.4.30	192.168.4.45	ICMP	577	Echo (ping) reply id=0x0000, seq=0/0, ttl=64
57	0.417888906	192.168.4.23	192.168.4.45	ICMP	577	Echo (ping) reply id=0x0000, seq=0/0, ttl=64
58	0.417973350	192.168.4.23	192.168.4.45	ICMP	424	Echo (ping) reply id=0x0000, seq=0/0, ttl=64
59	0.417991905	192.168.4.23	192.168.4.45	ICMP	339	Echo (ping) reply id=0x0000, seq=0/0, ttl=64
60	0.418008794	192.168.4.23	192.168.4.45	ICMP	509	Echo (ping) reply id=0x0000, seq=0/0, ttl=64
61	0.418025108	192.168.4.23	192.168.4.45	ICMP	800	Echo (ping) reply id=0x0000, seq=0/0, ttl=64
62	0.418043238	192.168.4.23	192.168.4.45	ICMP	76	Echo (ping) reply id=0x0000, seq=0/0, ttl=64
63	0.418058386	192.168.4.23	192.168.4.45	ICMP	841	Echo (ping) reply id=0x0000, seq=0/0, ttl=64
64	0.421208788	192.168.4.27	192.168.4.45	ICMP	577	Echo (ping) reply id=0x0000, seq=0/0, ttl=64
65	0.422001171	192.168.4.27	192.168.4.45	ICMP	424	Echo (ping) reply id=0x0000, seq=0/0, ttl=64
66	0.422106985	192.168.4.27	192.168.4.45	ICMP	339	Echo (ping) reply id=0x0000, seq=0/0, ttl=64

Figure 5: A Smurf where large amounts of ICMP ping traffic is sent to a broadcast address, causing the devices in the network to reply to the victim's IP address, overwhelming it with responses.

effectively balance sensitivity, which prioritizes detection of true anomalies, against specificity, which minimizes the impact of false alarms. To mimic true anomalous, malicious intent, we designed scenarios that simulate traditional attack behavior in a modern network.

4.1 Attack Testing

To demonstrate our system's ability to detect malicious network activity, we staged a series of cyber-attacks, including Distributed Denial-of-Service (DDoS) [23], Smurf [24], and Man-in-the-Middle (MITM) [25] attacks.

DDoS Attack: The essence of a DDoS attack lies in its malevolent intent to disrupt the standard operation of a network, service, or server through an onslaught of excessive internet traffic. This traffic often exhibits a distributed origin, potentially emanating from an extensive network of sources, thereby distinguishing a DDoS attack from a more conventional denial-of-service (DoS) attack. The methodology adopted may span a variety of avenues, inclusive of, but not limited to, TCP/IP-based attacks (such as SYN flood and Smurf), application layer onslaughts, and exploitation of system-specific vulnerabilities. Figure 4 shows an example of this attack, and Algorithm 3 shows pseudocode of the generated attack.

Smurf Attack: This constitutes a variant of the distributed denial-of-service (DDoS) attack paradigm, capitalizing on IP spoofing in combination with ICMP echo requests to oversaturate the target network with extraneous traffic. The attacker orchestrates a deluge of ICMP echo requests directed towards IP broadcast addresses, while the source IP is manipulated to resemble that of the target. As an immediate response, each device interconnected within the broadcast network generates a response, culminating in a substantial surge of replies to the target's network. The multiplier effect that amplifies the impact of this attack emanates from the number of active devices within the broadcast network. Figure 4 shows an example Smurf attack, with its corresponding pseudocode in Algorithm 4

Algorithm 3 DDOS pseudocode:

```
FUNCTION launch_attack(target_ip, number_of_ips,
number_of_messages, interface, attack_type) CREATE a pool of
threads
GENERATE or SELECT the origin IP addresses FOR each origin_ip
in origin_ips
LAUNCH a thread WHILE attack is ongoing
CREATE a packet with specific characteristics based on
attack_type SEND the packet to target_ip from origin_ip
END WHILE END FOR
CALCULATE and PRINT attack statistics END FUNCTION
 CALL launch_attack with necessary parameters
```

Algorithm 4 Smurf pseudocode:

```
FUNCTION launch_attack(network, spoofed ip, number of
processes, number of packets)
CREATE a pool of processes
FOR each process in the pool
CREATE a new event loop FOR number of packets
CREATE and SEND a spoofed ICMP packet to the broadcast
address
END FOR
END FOR
WAIT for all async tasks to complete
END FUNCTION
CALL launch_smurf_attack with necessary parameters
```

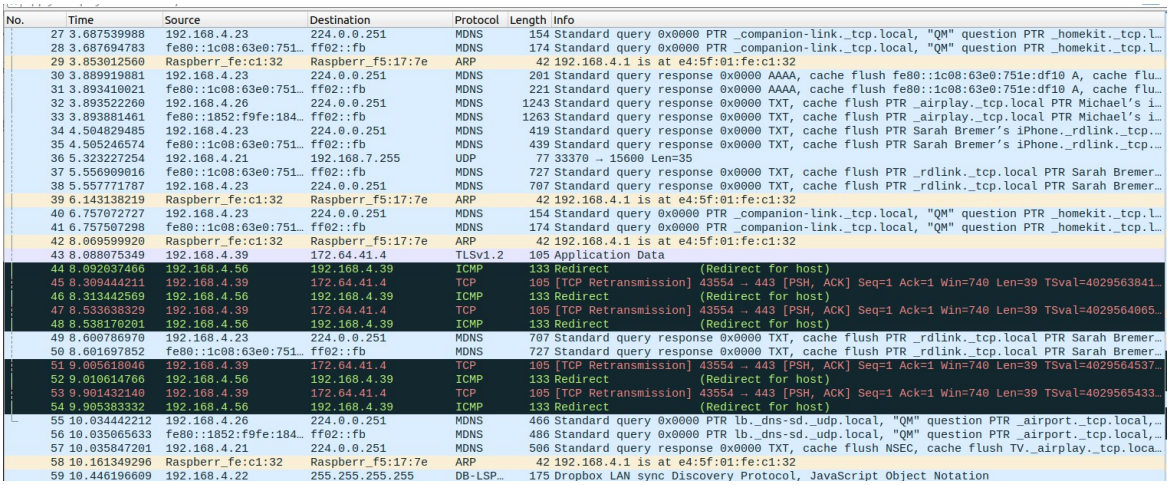

Figure 6: MITM attack where an unauthorized party intercepts and possibly alters communications between two parties while making it appear as if a normal conversation or data exchange is underway.

MITM Attack: This cyber threat is characterized by an attacker surreptitiously intercepting, and potentially modifying, the communication between two unsuspecting parties under the impression of being in a direct conversation. The attacker establishes individual connections with each victim and facilitates message transmission between them. This gives the victims the illusion of communicating over a private connection, while the entire conversation is manipulated by the attacker. The MITM attacks transcend the boundaries of wired and wireless environments, making various forms of communication (including but not limited to HTTP, HTTPS, SSL, TLS, and Wi-Fi) susceptible. Figure 5 shows an example of the occurring MITM attack between Alice and Bob, with its corresponding pseudocode in Algorithm 5

Algorithm 5 MITM pseudocode:

```
CREATE connection WITH alice
CREATE connection WITH bob
WHILE communication is ongoing
RECEIVE message FROM alice
FORWARD message TO bob
RECEIVE message FROM bob
FORWARD message TO alice
END WHILE
```

4.2 Results

We demonstrate the potential of the embedding through the application of simple algorithms applied to the latent space.

Additionally, we implemented callbacks to conduct IP ping count detection on a per-address basis during the online training of the node2vec GNN. This integration enhances the GNN's adaptability and anomaly detection capabilities by leveraging real-time IP ping data.

The process begins by establishing a reference baseline for normal network activity, derived from analyzing the initial 'n' epochs. This baseline data serves as a benchmark for subsequent anomaly detection. During the online training of the node2vec GNN, the count of IP pings per address is continuously monitored for each epoch.

A standard time series decomposition with seasonality, trend, and level is used to compensate for normal variation in network behavior. The residuals are checked against a threshold, and such instances are flagged as anomalies, indicating potential irregularities in network activity. This anomaly detection process seamlessly integrates with TensorBoard, allowing real-time visualization of detected anomalies.

Figure 7 shows the output IP scalars (orange) and anomaly scalars (blue) in TensorBoard during a Smurf attack at epoch 11. As seen in the TensorBoard image, the thresholding of the residual data effectively identified an instance of our Smurf attack (at epoch 11) within the IP ping count data. This integration allows for real-time visualization of the detected anomalies. Analysts and network administrators can observe the anomalies as they occur, facilitating prompt response and mitigation.

5 CONCLUDING REMARKS & CONTINUED RESEARCH

Our results show that our algorithm can identify common cyber-attacks in real-time, while learning the latent space of the network for visualization. Even without the inclusion of labeled training data, our solution can build the ground truth of a network's topology, by analyzing real network data. We then showed how network attacks can be visually identified through simple monitoring. We also developed preliminary techniques for detecting anomalous activities by observing drifting rates in cosine similarities between embeddings coupled with seasonal decomposition and plan to report those results in future work. Although the attacks were diminished in capability and are commonly used to demonstrate traditional malicious cyber activity, the fact that the algorithm was able to identify these simple attacks lends credence that even more complex attacks would be just as visible. Future work should look

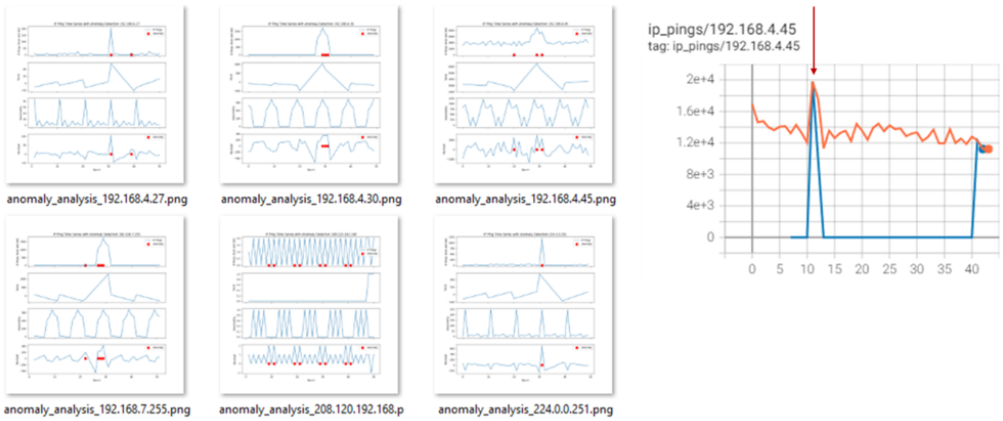

Figure 7: TensorBoard IP Scalars and Anomaly Scalars during a Smurf Attack.

to safely test more complex attacks, and look to follow existing campaigns, like those identified by MITRE [6]. We also suggest future work to focus on implementing a decentralized, peer-to-peer mesh network for real-time feature sharing amongst the worker nodes, as larger ground truth networks will require faster and more frequent data collection.

ACKNOWLEDGMENTS

The authors would like to thank Riverside Research and associated parties for their funding contributions. This work is the result of collaboration between parties and individuals and would not have been completed without respective due diligence. The authors would like to thank Dr. Bayley King, Dr. Joseph Salisbury, Mr. Ross Bobb, and Dr. Daniel Morton. Lastly, the authors would like to thank Riverside Research's Open Innovation Center for providing the opportunity to develop technologies directly related to problems in the domain of national security.

REFERENCES

[1] "What is the Mirai Botnet?," Cloudflare. https://www.cloudflare.com/learning/ddos/glossary/mirai-botnet/ (accessed Aug. 09, 2023).
[2] C. Kolias, G. Kambourakis, A. Stavrou, and J. Voas, "DDoS in the IoT: Mirai and other botnets,"
[3] Computer, vol. 50, no. 7, pp. 80–84, 2017.
[4] Avast Academy, "What Is the Mirai Botnet?," What Is the Mirai Botnet? https://www.avast.com/c-mirai (accessed Aug. 09, 2023).
[5] B. Bowman, C. Laprade, Y. Ji, and H. H. Huang, "Detecting Lateral Movement in Enterprise Computer Networks with Unsupervised Graph AI".
[6] A. Fawaz, A. Bohara, C. Cheh, and W. H. Sanders, "Lateral Movement Detection Using Distributed Data Fusion," in 2016 IEEE 35th Symposium on Reliable Distributed Systems (SRDS), Budapest, Hungary: IEEE, Sep. 2016, pp. 21–30. doi: 10.1109/SRDS.2016.014.
[7] "MITRE ATT&CK®," MITRE ATT&CK Framework v13.1, Apr. 01, 2023. https://attack.mitre.org/ (accessed May 19, 2023).
[8] M. T. Tun, D. E. Nyaung, and M. P. Phyu, "Performance Evaluation of Intrusion Detection Streaming Transactions Using Apache Kafka and Spark Streaming," in 2019 International Conference on Advanced Information Technologies (ICAIT), Nov. 2019, pp. 25–30. doi: 10.1109/AITC.2019.8920960.
[9] Y. Ji and H. Howie Huang, "NestedGNN: Detecting Malicious Network Activity with Nested Graph Neural Networks," in ICC 2022 - IEEE International Conference on Communications, May 2022, pp. 2694–2699. doi: 10.1109/ICC45855.2022.9838698.
[10] B. A. Powell, "Role-based lateral movement detection with unsupervised learning," Intelligent Systems with Applications, vol. 16, p. 200106, Nov. 2022, doi: 10.1016/j.iswa.2022.200106.
[11] "ProcAID: Process Anomaly-Based Intrusion Detection - ProQuest." https://www.proquest.com/openview/e4ce5ff777fc5943a8b4624677b3cad1/1?pq-origsite=gscholar&cbl=18750&diss=y (accessed May 18, 2023).
[12] H. He, Y. Ji, and H. H. Huang, "Illuminati: Towards Explaining Graph Neural Networks for Cybersecurity Analysis," in 2022 IEEE 7th European Symposium on Security and Privacy (EuroS&P), Genoa, Italy: IEEE, Jun. 2022, pp. 74–89. doi: 10.1109/EuroSP53844.2022.00013.
[13] V.-A. Darvariu, S. Hailes, and M. Musolesi, "Graph Neural Modeling of Network Flows." arXiv, May 12, 2023. Accessed: May 18, 2023. [Online]. Available: http://arxiv.org/abs/2209.05208
[14] V.-A. Darvariu, S. Hailes, and M. Musolesi, "Planning spatial networks with Monte Carlo tree search,"
[15] Proc. R. Soc. A., vol. 479, no. 2269, p. 20220383, Jan. 2023, doi: 10.1098/rspa.2022.0383.
[16] D. Merkel, "Docker: lightweight linux containers for consistent development and deployment," Linux Journal, vol. 2014, no. 239, p. 2, 2014.
[17] "Apache Kafka," Apache Kafka. https://kafka.apache.org/ (accessed Aug. 07, 2023).
[18] R. R. S, R. R, M. Moharir, and S. G, "SCAPY- A powerful interactive packet manipulation program," in
[19] 2018 International Conference on Networking, Embedded and Wireless Systems (ICNEWS), Dec. 2018,
[20] pp. 1–5. doi: 10.1109/ICNEWS.2018.8903954.
[21] C. Sanders, Practical Packet Analysis, 3rd Edition: Using Wireshark to Solve Real-World Network Problems. No Starch Press, 2017.
[22] A. Grover and J. Leskovec, "node2vec: Scalable Feature Learning for Networks," in Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, in KDD '16. New York, NY, USA: Association for Computing Machinery, Aug. 2016, pp. 855–864. doi: 10.1145/2939672.2939754.
[23] M. V. Gundy, D. Balzarotti, and G. Vigna, "Catch Me, If You Can: Evading Network Signatures with Web- based Polymorphic Worms," WOOT, 2007.
[24] M. E. J. Newman, "Network structure from rich but noisy data," Nature Phys, vol. 14, no. 6, Art. no. 6, Jun. 2018, doi: 10.1038/s41567-018-0076-1.
[25] T. K. Moon, "The expectation-maximization algorithm," IEEE Signal Processing Magazine, vol. 13, no. 6,
[26] pp. 47–60, Nov. 1996, doi: 10.1109/79.543975.
[27] C. E. Shannon, "A mathematical theory of communication," The Bell System Technical Journal, vol. 27, no. 3, pp. 379–423, Jul. 1948, doi: 10.1002/j.1538-7305.1948.tb01338.x.
[28] C. Douligeris and A. Mitrokotsa, "DDoS attacks and defense mechanisms: a classification," in Proceedings of the 3rd IEEE International Symposium on Signal Processing and Information Technology (IEEE Cat. No.03EX795), Darmstadt, Germany: IEEE, 2004, pp. 190–193. doi: 10.1109/ISSPIT.2003.1341092.
[29] S. Kumar, "Smurf-based Distributed Denial of Service (DDoS) Attack Amplification in Internet," in Second International Conference on Internet Monitoring and Protection (ICIMP 2007), San Jose, CA, USA: IEEE, Jul. 2007, pp. 25–25. doi: 10.1109/ICIMP.2007.42.
[30] F. Callegati, W. Cerroni, and M. Ramilli, "Man-in-the-Middle Attack to the HTTPS Protocol," *IEEE Secur. Privacy Mag.*, vol. 7, no. 1, pp. 78–81, Jan. 2009, doi: 10.1109/MSP.2009.12.