Detecting Lateral Movement in Enterprise Computer Networks with Unsupervised Graph AI

Benjamin Bowman Craig Laprade Yuede Ji H. Howie Huang
Graph Computing Lab
George Washington University

Abstract

In this paper we present a technique for detecting lateral movement of Advanced Persistent Threats inside enterprise-level computer networks using unsupervised graph learning. Our detection technique utilizes information derived from industry standard logging practices, rendering it immediately deployable to real-world enterprise networks. Importantly, this technique is fully unsupervised, not requiring any labeled training data, making it highly generalizable to different environments. The approach consists of two core components: an authentication graph, and an unsupervised graph-based machine learning pipeline which learns latent representations of the authenticating entities, and subsequently performs anomaly detection by identifying low-probability authentication events via a learned logistic regression link predictor. We apply this technique to authentication data derived from two contrasting data sources: a small-scale simulated environment, and a large-scale real-world environment. We are able to detect malicious authentication events associated with lateral movement with a true positive rate of 85% and false positive rate of 0.9%, compared to 72% and 4.4% by traditional rule-based heuristics and non-graph anomaly detection algorithms. In addition, we have designed several filters to further reduce the false positive rate by nearly 40%, while reducing true positives by less than 1%.

1 Introduction

According to the 2019 FireEye M-Trends report [5], the median time to detection of a network intrusion was 78 days. While this is an impressive improvement from the 418 days reported in 2011, this still means an adversary would have over 2 months inside an environment to accomplish their mission prior to detection. Additionally, nearly half of all compromises are detected via external sources, indicating that the tools currently employed by enterprise-level cyber defenders are insufficient for detecting the highly sophisticated modern-day adversaries.

Existing systems and techniques for detecting network intrusions rely heavily on signatures of known-bad events [25], such as file hashes of malware, or byte streams of malicious network traffic. While these techniques are able to detect relatively unskilled adversaries who use known malware and common exploitation frameworks, they provide almost no utility for detecting advanced adversaries, coined Advanced Persistent Threats (APTs), who will use zero-day exploits, novel malware, and stealthy procedures.

Similarly, the state-of-the-art behavioral analytics [26] in use today by network defenders utilize relatively rudimentary statistical features such as the number of bytes sent over a specific port, number of packets, ratio of TCP flags, etc. Not only are these types of analytics relatively noisy in terms of false positives, but they are also challenging to investigate due to their limited information and scope. For example, the fact that a particular host sent 50% more network packets in a given day could be indicative of many different events, ranging from data exfiltration, botnet command & control, to a myriad of other possibilities, most of which would not indicate a compromise, such as streaming a video.

To address these challenges, our approach is to build an abstract, behavior-based, graph data model, with key elements related to the particular behavior of interest we are trying to detect. Specifically, we model a computer network using a graph of authenticating entities, and the target behavior we detect is anomalous authentication between entities indicative of lateral movement within the network. Lateral movement is a key stage of APT campaigns when an attacker will authenticate to new resources and traverse through the network in order to gain access to systems and credentials necessary to carry out their mission [17, 21]. This is very challenging to detect as attackers will often use legitimate authentication channels with valid credentials as opposed to noisy exploitation procedures.

In order to effectively detect lateral movement, we first convert our input data, which is the in form of industry standard authentication logs, into a representation which will allow for not only learning about individual authentication events, but also the authentication behavior of the network as a whole. To that end, we construct an authentication graph, where nodes represent authenticating entities which can either be machines or users, and edges represent authentication events. Next, we utilize an unsupervised node embedding technique where latent representations are generated for each vertex in the graph. Finally, we train a link predictor algorithm on these vertex embeddings, and utilize this link predictor to identify low-probability links in new authentication events.

We apply our technique on two distinct datasets representing two contrasting computer networks. The PicoDomain dataset is a small simulated environment we developed in-house with only a few hosts, and spanning only 3 days. The second dataset is from Los Alamos National Labs (LANL) [11] and is a real-world network capture from their internal enterprise computer network spanning 58 days with over 12,000 users and 17,000 computers. In both cases, there is labeled malicious authentication events associated with APT-style activity which were used as ground truth for evaluation purposes. We were able to detect the malicious authentication events in the real-world dataset with a true positive rate of 85% and a false positive rate of only 0.9%. In comparison, traditional heuristics, and non-graph based machine learning methods, were able to achieve at best 72% true positive rate and 4.4% false positive rate. Understanding that modern day cyber defenders are frequently receiving far too many false positives, we spent additional time building simple filters that allowed us to further reduce our false-positive rate by nearly 40% on the LANL dataset, while reducing true positives by less than 1%.

In summary, our contributions of this work are as follows:

- A graph data structure for modeling authentication behavior within enterprise-level computer networks based on information available in industry standard log files.

- An unsupervised graph-learning technique for identifying anomalous authentication events which are highly indicative of malicious lateral movement.

- Experiments on two datasets showing the strength of graph learning for this application domain.

The remaining of this paper will be laid out as follows. Section 2 will provide some background into authentication protocols, the graph structure, and define the problem of lateral movement. Section 3 will discuss our proposed method and explain the learning algorithm. Section 4 will discuss our experimental evaluation and results. Section 5 will discuss the related work. Section 6 will discuss some limitations of our approach and our planned future work, and Section 7 will conclude.

2 Background & Problem Definition

In this section we will discuss some background on authentication in enterprise networks, how we build our graph structure, and define the problem of lateral movement.

2.1 Authentication

Modern enterprise computer networks rely on the ability to manage the permissions and privileges of users in order to maintain a safe and secure network. Users in the enterprise network will be given explicit permissions to access resources within the environment ranging from folders and network share drives, to applications and services. To make this possible, there have been many network authentication protocols developed through the years, which allow users to authenticate to resources in the network in order to verify that they have the privileges necessary to perform a certain action.

Common authentication protocols in today's enterprise computer networks include protocols such as Kerberos, NTLM, SAML, and others. Each one is designed to be a secure way to authenticate users inside an environment, and each has the ability to be abused. APT-level adversaries are well-versed in the workings of these authentication protocols, and they are often abused during an attack campaign. For example, the well-known "Pass the Hash" attack is a weakness in the NTLM implementation where the hash of a user's password, which can often be harvested from system memory, is used to authenticate to additional resources by the attacker.

Because hackers often abuse existing authentication channels, logs related to these critical protocols are valuable to the security analyst and detection algorithms. Typically these logs capture key information such as the account that is requesting to authenticate, the origin of the request, what they are attempting to authenticate to, as well as the result of that authentication request. Additionally, as authentication in the environment is network activity, we have the ability to capture this critical information from centralized network taps, rather than requiring expensive host-based log collection.

2.2 Graph Structure

There were two main considerations in how we chose to build our graph data structure. First, we wanted the input data to be highly accessible to our network defenders. This means utilizing data that is likely already being collected at the enterprise scale. While some smaller enterprises may have the luxury of collecting verbose system logs from all endpoints, larger enterprises are limited to coarse feeds from centralized resources such as network sensors or domain controllers. Second, we wanted the data to provide clear and concise information related to our target detection of lateral movement. Therefore, we design our algorithm to utilize network-level authentication logs generated from Zeek sensors [31] (formerly

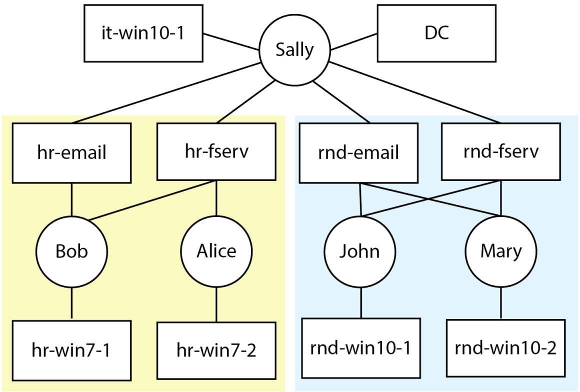

Figure 1: Example of an authentication graph for a small simulated network.

Bro). Specifically, we utilize the Kerberos logging capability, which generates protocol specific logging on the Kerberos authentication protocol which is utilized in the majority of Microsoft Windows domains. The technique is easily adaptable, however, to other authentication logs such as host-based authentication logs, NTLM logs, Active Directory logs, or others, providing they can uniquely identify authentication events between user and system identities in the network.

For Kerberos logs, we extract the client and service principals, which are unique identifiers associated with users and services in the network, as well as the source IP address of the requesting entity, which will uniquely identify the machine from which the client is operating. The destination IP address will always be the IP of the Kerberos server itself, and thus does not add valuable information to our graph. Here is an example of content we extract from the Kerberos logs with their respective Zeek column headings:

client	id_orig_h	service
jdoe/G.LAB	10.1.1.152	host/hr-1.g.lab

This record shows that the user *jdoe* of domain *G.LAB* authenticated to service *host/hr-1.g.lab*, which is a host in the network, from IP address *10.1.1.152*.

Definition 1 *An* ***authentication graph (AG)*** *is defined as a graph $G = (V,E)$ with a node type mapping ϕ: $V \rightarrow A$ and an edge type mapping ψ: $E \rightarrow R$, where V denotes the node set and E denotes the edge set, $A = \{IP, user, service\}$ and $R = \{authentication\}$.*

A simple authentication graph generated from a small simulated computer network is shown in Figure 1. We can infer from this graph that there are two separate organizational units in our enterprise: the *hr* unit and the *rnd* unit, each with two user nodes (*Bob* and *Alice*, *John* and *Mary*) interacting with user workstations represented as service nodes (*hr-win7-1*, *hr-win7-2*, *rnd-win10-1*, *rnd-win10-2*), as well as some email servers and file servers (*hr-email*, *hr-fserv*, *rnd-email*, *rnd-fserv*). We can see that user *Sally* is a network administrator, as she has authentication activity to the Domain Controller service node (*DC*) in the environment, the email and file server nodes, as well as her own workstation node (*it-win10-1*). Note that for display purposes, the IP nodes have been collapsed into their representative service nodes.

2.3 Lateral Movement

Lateral movement is a key stage of APT-level attack campaigns as seen in various attack taxonomies such as the Lockheed Martin Cyber Kill Chain [17], and the MITRE ATT@CK framework [21]. Figure 2 provides a simplified version of an APT-style campaign. After some initial compromise, and prior to domain ownership by the adversary, there is a cycle of lateral movement through the network. In most cases, the system that is initially compromised will be a low privileged account, typically a user workstation. This is due to the prevalence of client-side attacks (e.g., phishing), which are much more effective on typical, low-privilege users, as opposed to high-privilege IT professionals. Thus, the attacker almost always gains a foothold on a low privilege system and is thus required to move laterally through the network to achieve their goals.

Definition 2 ***Lateral movement*** *is defined as a malicious path $< u,v >$ conducted by an attacker in an organization's network characterized by the authentication graph, where u,v belong to entity set $\{IP, user, service\}$.*

For example, in Figure 1, if the user Alice fell victim to a phishing email and downloaded malware, the attacker would gain their initial foothold as account *Alice* on *hr-win7-2*. As *Alice* is a low-privilege account, it is unlikely that the attacker would be able to do much harm to the enterprise at large, such as installing ransomware on all the systems in the network, or exfiltrating highly sensitive business data. Therefore, the attacker would be required to move laterally to systems and

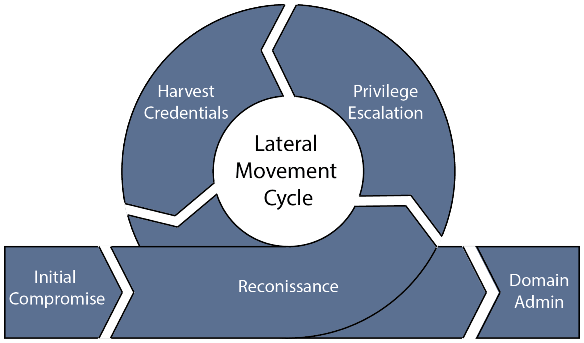

Figure 2: An APT-style campaign showing the cycle of lateral movement after initial compromise and prior to full domain ownership.

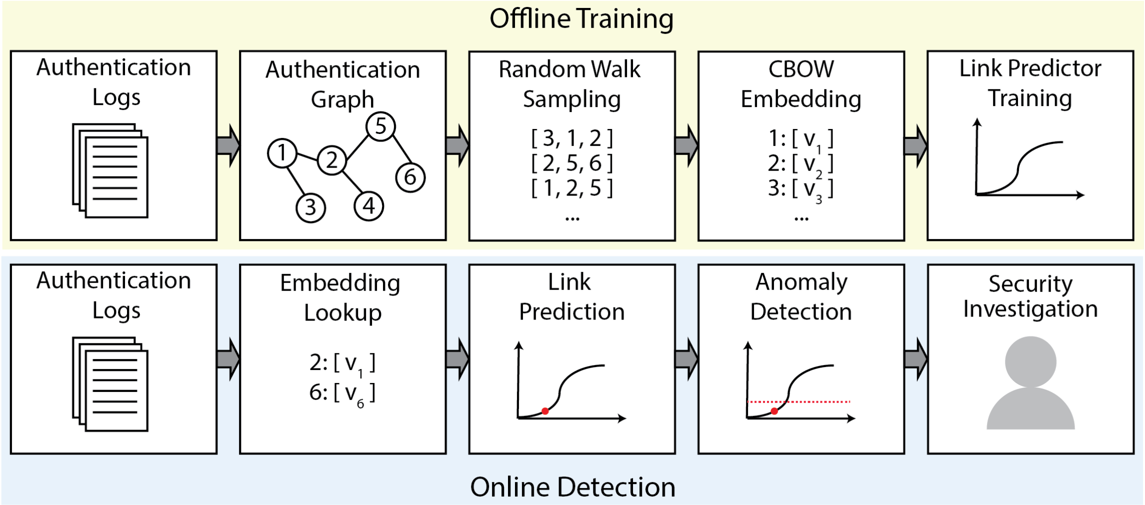

Figure 3: Full algorithm pipeline including offline training of node embeddings and logistic regression link predictor, as well as online detection via an embedding lookup, link prediction, and threshold-based anomaly detection.

accounts that have higher permissions in the environment. This can be done by exploitation of vulnerabilities, however, this is often a noisy and error prone process. More often, adversaries will harvest and abuse legitimate credentials from the set of compromised systems. In the case of our example, Alice could harvest the domain admin *Sally*'s credentials from the file server *hr-fserv* which *Sally* had previously authenticated to, and *Alice* has privileges to access. Now, with *Sally*'s credentials, *Alice* can authenticate from *hr-win7-2* to the *Domain Controller (DC)*. This attack could be characterized by the lateral movement path: <*hr-win7-2*, *Sally*, *DC* >.

Existing techniques are not well suited to detect lateral movement within enterprise-scale environments. Most Intrusion Detection Systems (IDSs) are placed at the border of a network, and will fail to detect attacker actions after an initial foothold has been established. Even if the IDS had total visibility, an attacker using legitimate authentication channels would likely not trigger any alerts. Host-based security software relies almost exclusively on identifying signatures of known malware, and thus will prove ineffective at detecting APT-level adversaries who will move laterally through a network using novel malware or legitimate authentication mechanisms. Some environments may implement a Security Information Events Management (SIEM) System, which would allow for more complex log analytics. However, SIEMs are typically standard row or columnar data stores such as Splunk [26] which only allow for relatively basic statistical analysis of the data. Behavioral analytics implemented in SIEMs are typically simple aggregate trends of low level features such as bytes over particular ports and protocols.

3 Proposed Method

In this section we will discuss our proposed method for detecting lateral movement in enterprise computer networks. We will provide an overview of our machine learning pipeline, followed by detailed discussions of the node embedding process, the link predictor training, and the anomaly detection.

3.1 Overview

In order to detect lateral movement in enterprise computer networks, we generate authentication graphs as discussed previously and apply an unsupervised graph learning process to identify low probability links. Figure 3 shows the algorithm pipeline. During the offline training stage (the top half of the figure), we start by generating authentication graphs, then create node embeddings via a random walk sampling and embedding process, and finally train a logistic regression link predictor using the node embeddings and ground-truth edge information from the authentication graph.

During the online detection stage (the bottom half of the figure), new authentication events are processed resulting in new edges between authenticating entities. Embeddings for these entities are generated via an embedding lookup, and link prediction is performed using the trained logistic regression link predictor. Anomaly detection is performed via a (configurable) threshold value, where links below a particular probability threshold will be forwarded to security experts for investigation.

3.2 Node Embedding Generation

Node embedding generation is the process by which a d-dimensional vector is learned for each node in a graph. The goal of these approaches is to generate a vector representation for each node which captures some degree of behavior within the network as a whole.

For the authentication graph, we use H to denote the set of node embeddings, $H = \{h_1, h_2, ..., h_n\}$, where h_i denotes the node embedding for the ith node, and n denotes the number of nodes in the graph. In the beginning, nodes do not have embeddings, which means $h_i = \emptyset$.

In order to extract latent node representations from the graph, we utilize an unsupervised node embedding technique similar to *DeepWalk* [22], and *node2vec* [7]. We first sample our authentication graph via unbiased, fixed-length random walks. Specifically, for any node v in the graph, we will explore r random walks with a fixed-length l. For a random walk starting from node v, let v_i denote the ith node in the walk, the node sequence for this walk is generated with the following probability distribution:

$$P(v_i = x | v_{i-1} = y) = \begin{cases} \frac{1}{d_y}, & if(x,y) \in E \\ 0, & otherwise \end{cases} \tag{1}$$

where E denotes the edge set in the graph, and d_y is the degree of node y. This results in a set of random walk sequences $S = \{S_1, S_2, ..., S_m\}$, where S_i denotes the ith random walk sequence, and m denotes the total number of sequences.

With the sequence set of the random walks, we then tune node embeddings via a Continuous-Bag-of-Words (CBOW) model with negative sampling as proposed in [18]. In the CBOW model, we predict the target node provided context nodes from the random walk sequence. We utilize negative sampling such that we only update the vectors of a subset of nodes that were not found in the particular context window of the target node.

We use the Noise Contrastive Estimation (NCE) loss as defined in Equation 2:

$$L = -[logp(y = 1 | h_T, h_I) + \sum_{h_U \in N(h_I)} logp(y = 0 | h_U, h_I)] \tag{2}$$

where y denotes the label, h_T denotes the embedding of the target node, h_I denotes the embedding of the input node which is the average of the context nodes, h_U denotes the embedding of a noise node, and $N(\cdot)$ denotes the set of noise node embeddings for that input. This loss function differentiates the target sample from noise samples using logistic regression [8].

Further, the probability for different labels of negative sampling is defined in Equation 3,

$$\begin{aligned} p(y = 1 | h_T, h_I) &= \sigma(h_T'^{\top} h_I) \\ p(y = 0 | h_T, h_I) &= \sigma(-h_T'^{\top} h_I) \end{aligned} \tag{3}$$

where $\sigma(\cdot)$ denotes the sigmoid function, and h'_T denotes the column vector for h_T. Therefore, the final loss value is calculated by Equation 4.

$$L = -[log\sigma(h_T'^{\top} h_I) + \sum_{h_U \in N(h_I)} log\sigma(-h_T'^{\top} h_I)] \tag{4}$$

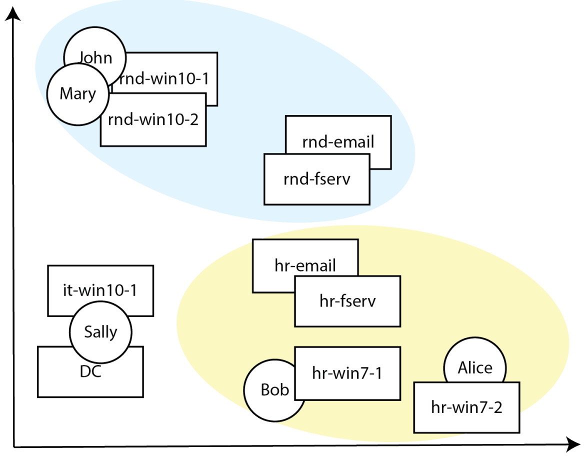

Figure 4: Example embedding space generated from a random-walk based node-embedding process.

By minimizing the loss value from Equation 4, we are able to tune our node embeddings such that we are more likely to predict our target node embedding h_T given the context node embeddings h_I, while simultaneously less likely to predict the negative sample node embeddings h_U given the same context h_I. We use Stochastic Gradient Descent (SGD) to minimize the loss function. In the end, we generate the output node embedding set $H' = \{h'_1, h'_2, ..., h'_n\}$, where $h_i{}'$ is the d-dimension embedding for node i.

In the context of the authentication graph, this process equates to predicting a user based on the machines and users found within at-most l-hops away. This will result in node embeddings where users who often authenticate to similar entities will be embedded in a similar region. Similarly, systems which share a user base will be found embedded in a similar region. This provides us the ability to then look at authentication events as events between two abstract vectors, as opposed to between distinct users and machines.

Figure 4 provides a 2-dimensional embedding space generated for the graph in Figure 1 using this node embedding process. We can see that the embedding of the graph corresponds nicely to the organizational units of the various users and systems. Additionally we see that the servers are clearly separated from the users and their workstations. Also, the network administrator is clearly separated from both organizational units. In addition, notice that the user *Alice* does not have an edge to the *hr-email* server in the authentication graph, despite clearly being a member of the *hr* organization. Even though this is the case, we can see that *Alice* is co-located in the embedding space with other *hr* users and systems. This fact will be crucial during the link prediction process, as even though there is no explicit link between *Alice* and the *hr-email* server, we would like our link prediction algorithm to predict a high probability for the authentication event between *Alice* and *hr-email*, considering it is perfectly reasonable that *Alice* authenticates to the *hr-email* server.

3.3 Link Prediction

Next, we utilize a traditional logistic regression (LR) algorithm to provide us with a probability estimate that a particular authentication event occurs between two nodes a and b. Formally, our LR algorithm models:

$$P(y=1|h') = \sigma(h') = \frac{1}{1+e^{-w^{\top}h'}} \tag{5}$$

where y is the binary label indicating if an edge exists or not, the weight vector w contains the learned parameters, and h' is the element-wise multiplication of the node embeddings h_a and h_b defined in Equation 6, also known as the Hadamard product.

$$h_a \circ h_b = (h_a)_{ij} \cdot (h_b)_{ij} \tag{6}$$

We train the above model by generating a dataset of true and false edge embeddings from the ground truth authentication graph. The true edge set consists of all edges in the authentication graph:

$$E_T = h_a \circ h_b \; \forall (a,b) \in E \tag{7}$$

with each edge embedding receiving a binary label of 1. On the contrary, the false edge set consists of all edges that do not exist in the authentication graph:

$$E_F = h_a \circ h_b \; \forall (a,b) \notin E \tag{8}$$

with each edge embedding receiving a binary label of 0. Training on these two sets of data would cause significant over fitting as E_F contains every possible edge not in the original edge set E. Therefore, we down sample E_F via a random sampling process, and only train on the same number of false edges as found in E_T.

3.4 Anomaly Detection

Anomaly detection is achieved by applying our trained LR link predictor to new authentication events. First, authentication events are parsed into a set of edges between authenticating entities. Next, we perform an embedding lookup for the node embeddings generated during the training stage. The anomaly detection function A can be expressed as:

$$A(h_a, h_b) = \begin{cases} 1, & if\ f(h_a \circ h_b) < \delta \\ 0, & otherwise \end{cases} \tag{9}$$

where h_a and h_b are the embeddings for nodes a and b, and the function $f(\cdot)$ is the logistic regression link predictor trained on the true and false edges generated from our training graph. The parameter δ is the threshold for generating an alert. In this paper, we use a threshold of $\delta = 0.1$, or 10%, which we will show shortly yields good performance.

4 Evaluation

In this section we will evaluate our technique for detecting malicious authentication in enterprise networks. First we will discuss the datasets we used for evaluation, followed by a detailed description of the various methods we evaluated, and an analysis of our results. In an effort to further reduce false positives, we make some observations about the data and our results, and update our algorithm accordingly.

4.1 Datasets

We apply our malicious authentication detection to two datasets generated from contrasting computer networks. Table 1 provides details on each dataset. We discuss both datasets in detail below.

Table 1: Dataset Details

	PicoDomain	**LANL**
Duration in Days	3	58
Days with Attacks	2	18
Total Records	4686	1.05 B
Total Attack Records	129	749
User and Machine Accounts	86	99968
Computers	6	17666

PicoDomain is a dataset we generated in-house for cyber security research. It is designed to be a highly scaled-down environment which contains only the most critical elements commonly found in enterprise-level domains. Specifically, the PicoDomain consists of a small Windows-based environment with five workstations, a domain controller, a gateway firewall and router, and a small-scale internet that houses several websites as well as the adversarial infrastructure. A Zeek network sensor was installed inside the environment and placed such that it had visibility of traffic entering and leaving the network from the simulated Internet (north/south), as well as traffic between local systems in the simulated enterprise network (east/west). A total of three days of network traffic was captured. During this three day period, there was benign activity performed in a typical 9-5 workday pattern, such as browsing the web, checking e-mail, etc. Additionally, on days 2 and 3, we ran an APT-style attack campaign which included all stages of the kill chain. The attack campaign started with a malicious file downloaded from an e-mail attachment. This gave the attacker the initial foothold in the network. The attacker then proceeded to perform various malicious actions typically associated with APT-level campaigns. This included exploiting system vulnerabilities for privilege escalation, registry modifications to maintain persistence, credential harvesting via the tool Mimikatz, domain enumeration, and lateral movement to new systems via the legitimate Windows Management Instrumentation (WMI) service. At the end of the campaign, the attacker was able to compromise a domain admin account, resulting in full network ownership by the attacker.

Comprehensive Cyber Security Events is a dataset released by Los Alamos National Labs (LANL) and consists of 58 consecutive days of anonymized network and host data [11]. There are over 1 billion events containing authentication activity for over 12,000 users and 17,000 computers in the network. An APT-style attack was performed during the data capture, and relevant authentication log entries were labeled as being malicious or benign. No further details were provided in the dataset as to what types of attacks were performed during the exercise. This is a limiting factor of this dataset, and, in fact, led to the generation of the previously mentioned PicoDomain dataset.

4.2 Methods Evaluated

We evaluate two variants of our proposed graph learning methods, as well as four different baseline techniques, which include two non-graph-based machine learning algorithms, as well as two traditional rule-based heuristics. We will discuss each below.

Graph Learning with Local View (GL-LV). This is our graph learning technique configured in such a way as to have a more localized view in our graph. This means our embeddings and link predictor will be optimized for nodes within a close proximity. To achieve this, we generate 20 random walks of length 10 for every node, and generate a 128-dimension embedding for each node based on a context window size of 2. This means each node will only consider a neighborhood of 2-hop neighbors in the embedding process. Our anomaly detection threshold is set at $\delta = 0.1$.

Graph Learning with Global View (GL-GV). This is our second graph learning variant which is very similar to the first, however this time configured to have a more global view of the graph. This means our embeddings and link predictor will be optimized for nodes that are further apart in our graph. To that end we used the same configuration as previously, however now setting the window size to 5. This means nodes will consider at most 5-hop neighbors during the embedding and link prediction process, which will give the algorithm a much broader view of the graph.

Local Outlier Factor (LOF) [2]. For a non-graph-based machine learning comparison, we implement the LOF anomaly detection algorithm. The LOF is a density-based anomaly detection approach, where relative local densities are compared between each sample, and those which are very different from their neighbors are considered anomalous. In order to generate features for this algorithm, we 1-hot encode the authentication events into an authentication vector containing a dimension for all authenticating entities. For each event, the dimensions corresponding to the various authenticating entities for that particular record will be set to 1, and all other dimensions will be 0. We then apply the LOF algorithm to these vectors to identify anomalies.

Isolation Forest (IF) [15]. This is a second non-graph-based machine learning comparison technique. The Isolation Forest algorithm identifies samples that can be easily isolated from the dataset by simple decision trees as being anomalous. This is applied to the same authentication vectors as in the previous LOF method.

Unknown Authentication (UA). This is a more traditional rule-based heuristic which simply identifies all first-time authentication events as anomalous. During the training period, a list of known authentications is generated for each authenticating entity in the network. During the testing phase, any authentication event which was not seen during the training phase is considered as anomalous. After an anomalous result is generated the first time, the authentication event is added to the set of known authentications for the particular entity. This way we do not generate repeated anomalies for the same event.

Failed Login (FL). This is a second traditional rule-based heuristic which considers all failed login events as anomalous. As this technique does not requiring any training data, we only evaluate it on the test portion of the datasets.

4.3 Detection Analysis

Next we apply the six different algorithms discussed previously and evaluate their ability to detect malicious authentication in our two datasets. For all techniques, we report the number of true positives (TP), false positives (FP), as well as the true positive rate (TPR), and false positive rate (FPR).

PicoDomain. First we apply all techniques to the simulated PicoDomain dataset. We split the dataset into training and testing, with the training data consisting of authentication activity before the APT attack began, and the testing data containing all other activity. As this is a small dataset focused on malicious activity, the majority of the time period contains malicious events. As a result, there was only roughly 20% clean training data available. Thus our final train/test split on this data was about 20%/80%. For all 6 detection techniques, we only generate accuracy metrics on the testing dataset.

Table 2 shows the results for all six techniques. Not unsurprisingly, the **UA** detector performed very well, with 100% TPR, and only 1.5% FPR. This means all of the lateral movement associated with the APT campaign involved systems which did not have authentication activity during the training period, a characteristic that is likely only to hold in relatively small and simulated environments. We can also see that the failed login (**FL**) rule generated very few results, and only managed to detect a single event associated with the malicious activity. This is due to the fact that the APT campaign did not involve any brute-force password guessing attempts. The single failed login is likely due to user error during the attack.

Both ML techniques (**LOF** and **IF**) struggled to detect malicious events, with TPRs well below 50%, and FPRs as high

Table 2: Anomaly Detection Results on PicoDomain Dataset

Algorithm	TP	FP	TPR (%)	FPR (%)
UA	**129**	11	**100**	1.5
FL	1	15	0.8	2.0
LOF	41	19	32	2.5
IF	34	62	26	8.3
GL-LV	102	**0**	80	**0.0**
GL-GV	102	**0**	80	**0.0**

as 8.3%. This indicates that the pure authentication activity between entities, without the additional information present in the graph topology, is not sufficient for detecting lateral movement.

Our graph learning techniques, **GL-LV** and **GL-GV**, performed much better than the comparison ML techniques, achieving 80% TPR. This shows the strength of the graph topology for the detection of lateral movement. Additionally, the graph-learning approaches were able to reduce the FPR to 0% compared with the 1.5% of the **UA** detector. A low false positive rate is critical for anomaly detection techniques, as will be made clear by the next experiment on the LANL dataset. Interestingly, we see that the global view and local view had no effect on the performance. This again is likely due to the extremely small scale of this dataset. The average shortest path between any two nodes in the PicoDomain graph is slightly over 2 hops. This means the additional visibility that the **GL_GV** detector provides will not contribute significantly more information on the graph structure.

LANL. Here we apply the same 6 detectors to the LANL Comprehensive Cyber Security Events dataset. In a similar manner, we split the data into training and testing sets. The training set consists of 40 days on which no malicious activity is reported, and the testing set of 18 days with malicious activity. This is equivalent to roughly 70% training data, and 30% testing data. Due to the large scale of this dataset, it was necessary that we perform an additional down sampling for the two ML techniques **LOF** and **IF**, which was accomplished by removing timestamps from the training and testing dataset, and removing duplicate events. The TPR and FPR for these two techniques have been adjusted to account for this.

Table 3: Anomaly Detection Results on LANL Dataset

Algorithm	TP	FP	TPR (%)	FPR (%)
UA	542	530082	72	4.4
FL	31	116600	4	1.0
LOF	87	169460	12	9.6
IF	65	299737	9	16.9
GL-LV	503	146285	67	1.2
GL-GV	**635**	**107960**	**85**	**0.9**

Table 3 shows the results for the six anomaly detectors. The impact of scale is readily evident in these results, with a significant number of false positives for all detectors, despite reasonably small false-positive rates.

We can see that the **UA** detector performs again reasonably well, with a significant 72% of the malicious authentication events detected. However, with this real-world dataset, we can see how noisy this detector is, with a FPR of 4.4% resulting in over 500,000 false positives. The **FL** detector again fails to perform, indicating that for APT style campaigns, simple failed login attempts are not suitable detectors. Similarly, both ML approaches generated many false positives, and few true positives, again showing that simple authentication events without the added information in the authentication graph are insufficient for malicious authentication detection.

The two graph learning techniques were able to provide the best TPR at the least FPR. The **GL-LV** detector, although returning less true positives than the simple UA detector, was still able to detect 67% of the malicious activity, at only 1.2% FPR compared to 4.4% by the **UA** detector. The best performing predictor on this dataset is the **GL_GV** detector, which was able to detect the most malicious authentication events with a TPR of 85%, while maintaining the lowest FPR of 0.9%. For this dataset, the increased context window of the **GL-GV** over the **GL-LV** contributed significantly to the added performance. The average shortest path between any two nodes in the LANL graph is roughly 4 hops. This explains why, in this case, the broader view of the **GL_GV** detector was able to capture more information from the graph structure in the node embeddings, resulting in a better performing link predictor.

It is important to note here that all of the previous experiments were performed on commodity server hardware. Specifically, we utilized a server with two Intel Xeon CPU E5-2683 CPUs, and 512 GB of ram. This provided enough memory and compute power to run any of the detectors discussed on the full 58-day LANL dataset in under 6 hours. We believe that the techniques used here would be supported by the infrastructure already available to our network defenders.

4.4 Reducing False Positives

As we can see from the previous experiment, and specifically Table 3, the effect of false positives on the datasets of the scale found in the real-world can be very detrimental. Even for the best performing detector, the **GL_GV** detector, a false positive rate of 0.9% resulted in over 100,000 individual false positive results in the test data. As these results will ultimately be used by cyber analysts to investigate the threats, it is important that we do our best to keep the false positives to a minimum. In this section, we present some of our observations of the data and results, and design several filters to further reduce the false positive rate by nearly 40%, while reducing true positives by less than 1%.

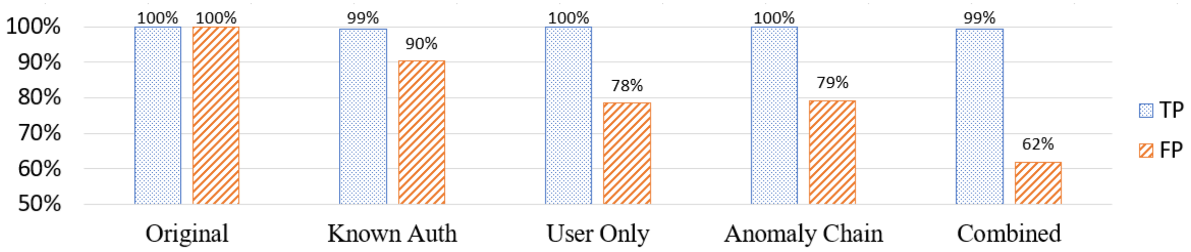

Figure 5: Impact of various approaches in reducing the number of false positives returned on the LANL dataset.

Observation 1: The malicious authentication events are predominantly first authentication events.

This observation was made based on the fact that the simple unknown authentication (UA) detector performed very well at identifying the malicious events. However, its false positive rate was far too high to use on its own. Based on this observation, we use the inverse of this detector as a false positive filter. More precisely, all anomalies generated by the graph learning approach are passed through a filter based on the known authentication events. We discard any of the anomalous authentication events that were previously seen during the training period. This filter corresponds to the "Known Auth" filter in Figure 5. We can see that we achieved about a 10% reduction in false positives, while reducing true positives by less than 1%.

Observation 2: The malicious authentication events are predominantly based on user interactions.

Our authentication graph includes interactions between users and computers, but also interactions between purely computers. Some of the interactions are possibly associated with the red team exercise, however, the labeling scheme utilized by LANL only labeled authentication events involving user accounts as being malicious. Without further details on exactly what the red team activity entailed, it is impossible to label other interactions as malicious or benign that could have been associated with the red team exercise. Based on this, we modify our anomaly detection algorithm, and again add a new filter where the results that are generated and do not involve at least one user account are discarded. This filter corresponds to the "User Only" filter in Figure 5. We can see this had a significant impact on the results, reducing false positives by over 20% from the original, while not reducing the true positives at all.

Observation 3: The malicious authentication events are predominantly related to specific user accounts and systems.

This observation makes sense from a practical standpoint. When an adversary gains access to a network, it is unlikely that they have multiple initial footholds. Typically a single foothold would be established, and then access throughout the network would expand from there. This means that all of the malicious edges in our authentication graph should be close together, or even form a connected component in the graph. Based on this observation, we build a third filter, where all of the anomalous results are chained together based on their shared nodes and edges. Any anomalous results which do not form a chain with at least one other anomalous event is discarded. This filter corresponds to the "Anomaly Chain" filter in Figure 5. This resulted again in about a 20% reduction in false positives from the original, and no reduction in true positives.

To summarize, the last bars labeled as "Combined" in Figure 5 represent the results when combining all of the previous filters together. We can see this resulted in the best performance, and was able to reduce the number of FPs on the LANL dataset by nearly 40%, while losing < 1% of the true positives.

5 Related Work

This section studies the related works in terms of anomaly detection and node embedding methods.

Anomaly detection for APT identification has been extensively studied. However, the majority of works are based on expensive host-based log analysis, with the goal of anomalous process activity, indicative of malware or exploitation [23] [6] [24] [16]. Some go so far as mining information from user-driven commands for anomaly detection [14]. While host logs may be available in some environments, it would be a significant burden for most large enterprises to capture and store verbose host-based logs such as system call traces.

At the network level, there are techniques for detecting web-based attacks [13], as well as botnet activity [1] utilizing anomaly detection algorithms. A highly related technique [4] combines host information with network information to detect lateral movement. However, they require process-level information from hosts, making this technique a poor fit at the enterprise scale. As lateral movement detection is such a hard problem, some approaches instead focus on detecting the degree to which environments are vulnerable to lateral movement attacks [10].

There are also approaches that look for deviations from known, specification-driven, rules of how an environment should behave, such as Holmes [20] and Poirot [19]. While these work reasonably well and are able to reduce false positives by explicitly defining what behavior is deemed malicious, they are still based on knowledge derived from a human, and thus risk circumvention by new and novel attack paths. In addition, these techniques require constant maintenance and upkeep to develop new specifications for the constantly evolving attack surface.

Node embedding methods aiming at learning representative embeddings for each node in a graph have been successfully applied to various downstream machine learning tasks, such as node classification [22], link prediction [7], and node recommendation [30]. Existing methods usually take two steps to generate node embeddings. First, they sample meaningful paths to represent structural information in the graph. Second, they apply various data mining techniques from domains such as natural language processing (NLP), utilizing technologies such as word2vec [18] for learning meaningful vector embeddings.

The major difference between existing methods lie in the first step, i.e., how to mine better paths to capture the most important graph information. In this context, the early work DeepWalk [22] applies random walks to build paths for each node. In order to give more importance to close-by neighbors, Line [27] instead applies a breadth-first search strategy, building two types of paths: one-hop neighbors and two-hop neighbors. Further, the authors of node2vec [7] observe that the node embeddings should be decided by two kinds of similarities, homophily and structural equivalence. The homophily strategy would embed the nodes closely that are highly interconnected and in similar cluster or community, while the structural equivalence embeds the nodes closely that share similar structural roles in the graph. Based on these strategies, node2vec implements a biased random walk embedding process which is able to model both similarity measures.

There are additionally many other graph neural network architectures recently proposed, such as the convolution-based GCN [12], attention-based GAT [28], and many variants based on both [9]. However, they are mostly designed for semi-supervised or supervised tasks, and are not as suitable for unsupervised learning as the random-walk based approaches mentioned previously.

6 Limitations & Future Work

Although our results are promising, there are several limiting factors of our approach. The first limitation is the problem of explainability, which is not specific to our technique, but rather a limitation of machine learning techniques in general. When our graph learning algorithms label an event as an anomaly, it is relatively challenging to determine why it has done so. There is current and active research on explaining machine learning and AI algorithms [3], and many even specific to explaining the results of graph learning algorithms in particular [29]. We may be able to use some of these techniques in the future which would allow us to identify what nodes were most important when generating both the embedding, and ultimately the link prediction scores.

Our detection algorithm is based on the assumption that we will have historic data for each entity we plan to perform link prediction on in the future. If we have never seen an entity authenticate before, then we will not have an embedding generated for that entity, and thus we will be unable to perform the link prediction. There are many ways to handle this problem, such as assigning new entities a generic "new node" embedding, or assigning the new node embedding to the average embedding of its neighbors (provided that they have embeddings themselves), however we have not explored the impact of these various approaches. We believe that, at least in the case of enterprise network authentication, it is a fair assumption to believe that for the vast majority of user accounts in the network, there should be some history of their behavior provided a sufficiently long historic window.

In this work we focused specifically on log data pertaining to authentication events. However, there is a myriad of additional data that we could add to our graph and ultimately to our graph learning algorithms. In the future we plan to add finer grained detail of actions performed by users, such as DNS requests and file-share accesses. This will allow us to also expand our detection algorithm to identify other stages of the kill chain beyond lateral movement, such as command and control traffic, which would likely cause anomalous DNS requests.

7 Conclusion

In this work we discussed the challenging problem of detecting lateral movement of APT-level adversaries within enterprise computer networks. We explained why existing signature-based intrusion detection techniques are insufficient, and existing behavioral analytics are too fine grained. We introduced our technique of abstracting a computer network to a graph of authenticating entities, and performing unsupervised graph learning to generate node behavior embeddings. We discussed how we use these embeddings to perform link prediction, and ultimately anomaly detection for malicious authentication events. We applied our techniques to both simulated and real-world datasets and were able to detect anomalous authentication links with both increased true positive rates, and decreased false positive rates, over rule-based heuristics and non-graph ML anomaly detectors. We analyzed the results of our algorithm, and developed several simple filters to further reduce the false positive rate of our technique.

Acknowledgment

This work was supported in part by DARPA under agreement number N66001-18-C-4033 and National Science Foundation CAREER award 1350766 and grants 1618706 and 1717774, as well as support from the ARCS Foundation. The views, opinions, and/or findings expressed in this material are those of the authors and should not be interpreted as representing the official views or policies of the Department of Defense, National Science Foundation, ARCS, or the U.S. Government.

References

[1] Leyla Bilge, Davide Balzarotti, William Robertson, Engin Kirda, and Christopher Kruegel. Disclosure: detecting botnet command and control servers through large-scale netflow analysis. In *Proceedings of the 28th Annual Computer Security Applications Conference*, pages 129–138. ACM, 2012.

[2] Markus M Breunig, Hans-Peter Kriegel, Raymond T Ng, and Jörg Sander. Lof: identifying density-based local outliers. In *ACM sigmod record*, volume 29, pages 93–104. ACM, 2000.

[3] Mengnan Du, Ninghao Liu, Qingquan Song, and Xia Hu. Towards explanation of dnn-based prediction with guided feature inversion. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1358–1367, 2018.

[4] 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)*, pages 21–30, Sep. 2016.

[5] FireEye. M-trends 2019. https://content.fireeye.com/m-trends/rpt-m-trends-2019, 2019.

[6] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff. A sense of self for unix processes. In *Proceedings 1996 IEEE Symposium on Security and Privacy*, pages 120–128, May 1996.

[7] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864. ACM, 2016.

[8] Michael Gutmann and Aapo Hyvärinen. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 297–304, 2010.

[9] William L Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584*, 2017.

[10] John R Johnson and Emilie A Hogan. A graph analytic metric for mitigating advanced persistent threat. In *2013 IEEE International Conference on Intelligence and Security Informatics*, pages 129–133. IEEE, 2013.

[11] Alexander D. Kent. Comprehensive, Multi-Source Cyber-Security Events. Los Alamos National Laboratory, 2015.

[12] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[13] Christopher Kruegel and Giovanni Vigna. Anomaly detection of web-based attacks. In *Proceedings of the 10th ACM conference on Computer and communications security*, pages 251–261. ACM, 2003.

[14] Wenke Lee, Salvatore J Stolfo, and Kui W Mok. A data mining framework for building intrusion detection models. In *Proceedings of the 1999 IEEE Symposium on Security and Privacy (Cat. No. 99CB36344)*, pages 120–132. IEEE, 1999.

[15] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. In *2008 Eighth IEEE International Conference on Data Mining*, pages 413–422. IEEE, 2008.

[16] Emaad Manzoor, Sadegh M Milajerdi, and Leman Akoglu. Fast memory-efficient anomaly detection in streaming heterogeneous graphs. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1035–1044. ACM, 2016.

[17] Lockheed Martin. The cyber kill chain. https://www.lockheedmartin.com/en-us/capabilities/cyber/cyber-kill-chain.html, 2020. Accessed: 2020-01-16.

[18] 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.

[19] Sadegh M. Milajerdi, Birhanu Eshete, Rigel Gjomemo, and V.N. Venkatakrishnan. Poirot: Aligning attack behavior with kernel audit records for cyber threat hunting. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, page 1795–1812, New York, NY, USA, 2019. Association for Computing Machinery.

[20] 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.

[21] MITRE. Mitre att@ck. https://attack.mitre.org/, 2020. Accessed: 2020-01-16.

[22] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710. ACM, 2014.

[23] R Sekar, Mugdha Bendre, Dinakar Dhurjati, and Pradeep Bollineni. A fast automaton-based method for detecting anomalous program behaviors. In *Proceedings 2001 IEEE Symposium on Security and Privacy. S&P 2001*, pages 144–155. IEEE, 2000.

[24] Xiaokui Shu, Danfeng Yao, and Naren Ramakrishnan. Unearthing stealthy program attacks buried in extremely long execution paths. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 401–413. ACM, 2015.

[25] Snort. Snort. https://www.snort.org/, 2020. Accessed:2020-01-16.

[26] Splunk. Splunk. https://www.splunk.com, 2020. Accessed:2020-01-16.

[27] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *Proceedings of the 24th international conference on world wide web*, pages 1067–1077. International World Wide Web Conferences Steering Committee, 2015.

[28] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[29] Rex Ying, Dylan Bourgeois, Jiaxuan You, Marinka Zitnik, and Jure Leskovec. Gnn explainer: A tool for post-hoc explanation of graph neural networks. *arXiv preprint arXiv:1903.03894*, 2019.

[30] Xiao Yu, Xiang Ren, Yizhou Sun, Quanquan Gu, Bradley Sturt, Urvashi Khandelwal, Brandon Norick, and Jiawei Han. Personalized entity recommendation: A heterogeneous information network approach. In *Proceedings of the 7th ACM international conference on Web search and data mining*, pages 283–292, 2014.

[31] Zeek. The zeek network security monitor. https://zeek.org, 2020. Accessed: 2020-01-16.