# Understanding and Bridging the Gap Between Unsupervised Network Representation Learning and Security Analytics

Jiacen Xu
*University of California, Irvine*

Xiaokui Shu
*IBM Research*

Zhou Li
*University of California, Irvine*

*Abstract*—Cyber-attacks have become increasingly sophisticated, which also drives the development of security analytics that produce countermeasures by mining organizational logs, e.g., network and authentication logs. Graph security analytics (GSA) that can model the complex communication patterns between users/hosts/processes have been extensively developed and deployed. Among the techniques that power GSAs, Unsupervised Network Representation Learning (UNRL) is gaining traction, which learns a latent graph representation, i.e., node embedding, and customizes it for different downstream tasks. Prominent advantages have been demonstrated by UNRL-based GSAs, as UNRL trains a detection model in an unsupervised way and exempts the model developers from the duty of feature engineering.

In this paper, we revisit the designs of previous UNRL-based GSAs to understand how they perform in real-world settings. We found their performance is questionable on large-scale, noisy log datasets like LANL authentication dataset, and the main reason is that they follow the standard UNRL framework that trains a generic model in an attack-agnostic way. We argue that generic attack characteristics should be considered, and propose ARGUS, a UNRL-based GSA with new encoder and decoder designs. ARGUS is also designed to work on discrete temporal graphs (DTG) to exploit the graph temporal dynamics. Our evaluation of two large-scale datasets, LANL and OpTC, shows it can outperform the state-of-the-art approaches by a large margin.

*Index Terms*—Graph Neural Network, Security Analytics, Discrete Temporal Graph

## 1. Introduction

Modern cyber-attacks have reached an unprecedented level of complexity. To infiltrate a company, the attack usually takes multiple steps, including reconnaissance, exploitation, lateral movement and data exfiltration, as summarized in Advanced Persistent Threat (APT) kill-chain [1]. To counter such sophisticated attacks, a large number of enterprises have deployed security analytics [2], the systems that analyze logs (e.g., host and network logs) collected by the devices (e.g., web servers, hosts, and domain controllers) inside their networks and produce proactive security measures, like timely detection. A lot of recent security analytics, including the academic approaches [3]–[6] and commercial systems [7], [8], attempt to abstract communication graphs and perform graph-based methods, such as graph traversal, similarity testing, and embedding, to detect the completed or ongoing attacks. We term these systems as graph security analytics (GSA).

GSA has shown a lot of promise, as they are able to model the communications between devices internal to an enterprise and across enterprise boundaries. Though numerous sophisticated techniques are developed to hide the activities on a single device (e.g., exploiting 0-day vulnerabilities or erasing the host logs), the cross-device communication patterns are often hard to hide. One example is lateral movement, through which the attacker moves from a compromised machine to another high-profile machine with stolen credentials [9]. Such movement often introduces abnormal links or paths on a graph abstracted from the network and authentication logs, and it has been identified as a major use case for GSA [10], [11].

With the progress in deep learning, there is a trend of building GSA with deep learning methods, and one prominent direction is to apply Unsupervised Network Representation Learning (UNRL) [12]. At a high level, UNRL leverages a trained encoder to generate latent representation, e.g., node embedding, from the graph, and a decoder to compute node/edge scores for downstream tasks like node classification and link prediction. Graph autoencoder (GAE) is a prominent approach using Graph Convolutional Network (GCN) to realize UNRL [13]. Compared to the other methods, UNRL is able to learn complex graph patterns automatically and training the encoder does not require any labeled negative sample. Hence, a number of UNRL-based GSAs have been developed in recent years [10], [11], [14]–[16].

**Analysis of prior UNRL-based GSAs.** In this paper, we revisit the designs of these systems to understand their performance in a real-world setting. By comparing their results on LANL authentication dataset, which contains over 1 billion authentication events but only 749 malicious events, we found none of them achieve satisfactory detection accuracy. The best was achieved by Euler [14], but its average precision is less than 0.01. On the other hand, Euler is able to achieve over 0.9 average precision on non-security datasets like Enron [17], Colab [18] and Facebook [19],

suggesting there exists a big gap between non-security and security settings, and applying UNRL for GSA is non-trivial.

Three design issues were identified during our study. First, these GSAs directly map the problem of attack detection to link prediction, but malicious edges and negative edges, the classification targets of the two problems, could have very different distributions. Second, since standard encoders like GCN and GraphSage are used, these GSAs cannot incorporate edge features, like event attributes other than source and destination, into the framework. Third, the majority of GSAs abstract *static* graphs from the logs, which neglects the important temporal dynamics across different time snapshots.

**Our Approach.** In this paper, we present ARGUS[1], a GSA with new encoder and decoder designs that are aware of the generic attack characteristics. ARGUS abstracts the data into *discrete temporal graphs (DTG)* and uses Gated Recurrent Unit (GRU) to capture temporal dynamics across snapshots [20]. The encoder is inspired by Message Passing Neural Networks (MPNN) [21] to incorporate node features, edge weights and edge features all into the training process. For the decoder, instead of simply measuring the graph reconstruction loss [14], we consider the community patterns that are likely exhibited by the attacker [22] and the average precision as a major optimization goal [23].

We evaluate ARGUS on two large-scale datasets, LANL [24] and OpTC [25]. On LANL, ARGUS is able to achieve much better accuracy compared to the previous works, e.g., 0.3227 average precision (AP) compared to 0.0448 of Euler (state-of-the-art). On OpTC, ARGUS achieves 0.8074 AP (Euler has 0.6426 AP). The results suggest ARGUS could handle large-scale, realistic datasets, without triggering a large amount of false alerts that would overload the security analysts. Given that LANL and OpTC present different types of attacks (LANL focuses on malicious authentication attempts, while OpTC simulates a broader range of attack behaviors like communication with C2 servers), we believe the design of ARGUS is suitable to detect different types of attacks, in other words, not too attack-specific.

**Contributions.** We summarize the contributions as below.

- We revisit the designs of the existing UNRL-based GSAs and the gap between the UNRL framework and the challenges in security analytics.
- We develop ARGUS, a new UNRL-based GSA, to address the identified issues and achieve high detection accuracy.
- We evaluate ARGUS on two large-scale log datasets and demonstrate that its design goals can be achieved.
- We release the source code of this project to an open-source repository.[2]

---

1. ARGUS is short for <u>A</u>ttack-aware, <u>G</u>raph-based, <u>U</u>nsupervised <u>S</u>ecurity Analytics.
2. ARGUS: https://github.com/C0ldstudy/Argus

## 2. Background

In this section, we first explain the techniques related to UNRL and review the research that applies UNRL for security analytics. Then, we briefly review the representative encoders used to construct embeddings under UNRL. Next, we give a formal definition of temporal graphs that are used for data modeling in this paper. Finally, we briefly describe the two log datasets used in this study. The symbols used in this paper are defined in Table 1.

TABLE 1. THE MAIN SYMBOLS USED IN THE PAPER.

| Term | Symbol |
|---|---|
| Graph | $\mathcal{G}$ |
| Edges | $\mathcal{E}$ |
| Nodes | $\mathcal{V}$ |
| One edge | $e$ |
| One node | $v$ |
| Edge label | $y$ |
| Edge weights | $W$ |
| Edge features | $F$ |
| Node embeddings | $Z$ |
| Node features | $X$ |
| Snapshot time | $t$ |
| Adjacency matrix | $A$ |

### 2.1. Unsupervised Network Representation Learning (UNRL)

UNRL aims to generate latent representations for nodes, i.e., node embeddings, over graphs that come without labels [12]. The latent representation aims to capture the key structural features and be usable as input for various machine-learning models like neural networks. Different downstream tasks can be performed on the *same* latent representations, including node classification, link prediction, clustering, graph reconstruction, etc. Previous research shows outstanding performance can be achieved on large-scale relational data [26] and matrix completion / recommendation [27].

More formally, given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ represents nodes and $\mathcal{E}$ represents edges, UNRL aims to learn low-dimensional representations of each node $v \in \mathcal{V}$ so that connected nodes (e.g., similar neighbors) are closer in the embedding space. The distance between node embeddings can be used to generate node/edge scores for the downstream tasks. Take link prediction as an example, which predicts whether two nodes are likely to have a link (e.g., whether two users on Facebook actually know each other). An edge embedding can be generated by combining the node embeddings from the edge's two ends and converted into the probability of edge existence.

Recently, UNRL has been leveraged to develop graph-based security analytics (GSA), which aims to detect cyber-attacks by mining large-scale logs that can be represented as graphs (e.g., machines as $\mathcal{V}$ and network communications as $\mathcal{E}$). So far, most of the prior work modeled the attack detection as link prediction: assuming that in the

training period the links between entities (e.g., machines, users and processes) are normal, the links that are assigned with low likelihood scores in testing will be considered abnormal. Along this direction, prior work has attempted to detect lateral movement of APT [10], network attacks with heterogeneous graphs modeling [11], and host attacks with knowledge graph modeling [16]. Yet, in Section 3 we show that it is non-trivial to apply UNRL for GSA and design pitfalls exist in the prior work.

## 2.2. Graph Encoders

A number of approaches have been proposed to generate node embeddings from graph, e.g., Node2Vec [28], DeepWalk [29], NetMF [30], and HOPE [31]. Among them, graph autoencoder (GAE) is a prominent approach that leverages neural network models to encode graphs [13], [32]. The two most popular GAEs are Graph Convolutional Network (GCN) [33] and GraphSAGE [34]. ARGUS uses GCN, which is elaborated below.

GCN relies on graph convolution to predict the features of a node in the next layer by aggregating its neighbors' features. GCN chooses to use a multi-layer neural network for the graph convolution function, which is defined on the spectral domain. Each layer follows the propagation rule for neighbor aggregation:

$$H^{l+1} = \sigma(\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} H^l W^l) \tag{1}$$

Where $W^l$ is a trainable weight matrix of the layer $l$, $H^l$ is the hidden features of the layer $l$, $A$ is the adjacency matrix, $I$ is the identity matrix, $\tilde{A} = A + I$, $\tilde{D}_{uv} = \Sigma_v \tilde{A}_{uv}$, $W^l$ is the trainable weight matrix of $l$-th layer, and $\sigma$ is the activation function, such as ReLU. The node features $X$ are used to fill $H^0$. When GCN is used as an encoder, the output of the last layer becomes the node embedding.

## 2.3. Discrete Temporal Graph (DTG)

The GSAs listed in Section 2.1 train and test on *static* graphs. Recently, a few studies [14], [15] proposed to model the graph as a *Discrete Temporal Graph (DTG)*, where the graph evolves through time and updates at discrete periods. Within a period, each graph snapshot consists of all changes after the prior graph snapshot. Our framework ARGUS builds on DTG, which we illustrate in Figure 1.

Under the definition of static graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, we define a DTG snapshot observed in time slot $t$, with $\mathcal{G}_t = (\mathcal{V}_t, \mathcal{E}_t)$. Hence, a sequence of temporal graphs observed in $t = 1, 2, ..., \tau$ can be denoted as $\mathcal{G}_1, \mathcal{G}_2, ..., \mathcal{G}_\tau$. An edge $e = (u, v)$ exists in $\mathcal{E}_t$, when $u, v \in \mathcal{V}_t$ and some interactions happen between them during snapshot $t$.

Temporal link prediction is an important task on DTG, which predicts the apparition of new links at a given period of snapshot $t$, with the help of data observed in the previous periods. It has been used by GSAs to detect anomalous interactions in an organization. For instance, Netwalk [35] used clique embedding to learn the representation of
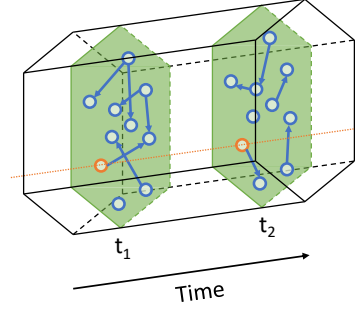


Figure 1. Example of a discrete temporal graph evolving through time.

TABLE 2. THE STATISTICS OF THE LANL DATASET AND THE OPTC DATASET AFTER FILTERING NETWORK FLOWS. NODES MEAN MACHINES (LANL) OR HOSTS (OPTC).

| Dataset | #Nodes | #Events | Duration |
|---|---|---|---|
| LANL-auth | 17,649 | 1,051,430,459 | 58 days |
| LANL-flows | 12,027 | 129,977,411 | 36 days |
| LANL-redteam | 305 | 749 | 28 days |
| OpTC | 814 | 92,073,717 | 8 days |
| OpTC-redteam | 28 | 21,784 | 3 days |

dynamic networks for graph anomaly detection. STEP combines GCN and LSTM (Long short-term memory) for event prediction [36]. StrGNN to detect unusual subgraph structures with GCN and GRU [37]. PIKACHU [15] and Euler [14] leveraged temporal random walk and GCN to detect malicious links in DTG. ARGUS, elaborated in Section 4, outperforms these models on temporal link prediction.

## 2.4. Host and Network Logs

Host-level logs (e.g., system audit logs and telemetry data generated by system monitors like auditd [38], sysmon [39], ETW [40], as well as EDR systems like Carbon-Black [41] and CrowdStrike [42]), and network logs (e.g., network flows and proxy logs) are main data sources for security analytics. Our framework ARGUS is tested on two open datasets, LANL and OpTC, to discover redteam events in them.

**LANL.** The LANL 2015 Comprehensive, Multi-Source Cyber-Security Events dataset [24] (or "LANL" for short) contains anonymized event data from four sources within Los Alamos National Laboratory's internal computer network, including authentication logs from individual computers and domain controller servers, process start and stop events, DNS logs at DNS servers, and network flow at routers. Simulated redteam attacks from the compromised machines were also performed during the data collection period, and their associated authentication events were labeled separately. We use the authentication logs ("LANL-auth"), network flows ("LANL-flows") and redteam events ("LANL-redteam") for evaluation. In Table 2 we summarize the dataset statistics. LANL posed prominent challenges in the high volume of events and very unbalanced distribution of benign and malicious events.

3592

**OpTC.** DARPA Transparent Computing (TC) program [43] aims to provide high-fidelity visibility into systems and aids modern attack campaign discovery, such as advanced persistent threat (APT) detection. The five-year program produces several batches of host-level telemetry data from its adversarial engagements, which contain various types of attacks across different OSes. We obtain the DARPA Operationally Transparent Cyber dataset (or "OpTC" for short) [25], which is generated at the end of the program and released in extended Cyber Analytics Repository (eCAR) format, for evaluation purposes. The dataset records all host-level activities between subjects like processes and objects like files and sockets on 814 hosts during one week. Like prior work [15], we use the "START" events related to the "FLOW" objects (i.e., network flows), and the statistics after filtering is shown in Table 2.

## 3. Motivation of ARGUS

While recent GSAs built on top of UNRL have shown prominent advantages over traditional approaches discussed in Section 7—in particular by training a model without malicious samples and learning the graph features automatically—several issues prevent them from being effective on large-scale, noisy log datasets. In this section, we first review the issues and then describe our solutions.

### 3.1. Issues with Prior Work

After careful reviewing and retraining the existing UNRL-based GSAs, we find the performance of many are largely impeded by low precision on large-scale datasets such as LANL [10], [11], [14], [15], [36][3] (we describe them in details in Appendix A.1). For instance,

- Euler [14], the state-of-the-art GSA based on the GCN+GRU architecture, yielded 0.0318 precision on LANL (376 true positives plus 11,464 false positives), though its false positive rate (FPR) is as low as 0.0045[4].
- PIKACHU [15] reported 0.0505 FPR, which is already much higher than 0.00578 of Euler, though its reported area under the ROC curve (AUC) is as high as 0.94.
- Log2Vec [11] only reported AUC at 0.91, which is even lower than 0.94 AUC from PIKACHU.
- GL-LV/GV [10] reported 0.009 FPR (635 true positives plus 107,960 false positives).

By analyzing the designs of these security analytics, we found three common issues:

**Issue-A: Gap between link prediction and attack detection.** Although it is common to model attack detection as link prediction for UNRL, the definitions of negative (non-exist) edges and malicious edges are indeed different,

which leads to a learning gap that a system trained to recognize negative edges does not detect malicious edges at a reasonable accuracy.

First, the distributions of benign edges, malicious edges, and non-exist edges (or their sub-spaces in the high dimensional detection space) could be dramatically different.

1) Benign edges follow expected program/system/network control- and data-flow.
2) Malicious edges follow allowed-but-unexpected control- or data-flow within a host or across hosts.
3) Negative edges generated for training purposes can belong to impossible control- or data-flow, not permitted by the binary or network topology.

Therefore, non-exist edges could be totally different from both benign and malicious edges and not always helpful in differentiating benign and malicious edges.

In fact, we found Euler on LANL is able to reach a precision over 0.9943 on the validation sets filled with negative edges, but its precision on the testing set filled with malicious edges is less than 0.01, suggesting malicious edges have a very different distribution from negative edges.

Second, prior GSAs employ standard loss functions defined for generic graph training tasks, e.g., cross-entropy (CE) loss used by Euler. These loss functions are situated to optimize the models for *accuracy* (i.e., the ratio of correctly predicted samples), but for datasets with a very large ratio of benign events like LANL, precision (i.e., the ratio of correctly predicted *malicious* samples) is more important, which would require different loss functions.

**Issue-B: Missing edge features.** Though a host/network event often contains many fields (e.g., packet size in network events), we found they are rarely used as edge features on the constructed graphs. As such, the information carried by each edge is limited. For instance, ShadeWatcher [16] assigns a type (e.g., read, write and clone) to an edge, but ignores other attributes such as action parameters; Euler [14] only computes the edge weights by the number of authentication events between two machines. We suspect that such a design choice is due to the limitations of standard graph encoders. For instance, GCN only accepts node features and edge weights [46] and GraphSAGE only accepts node features [47].

However, edge features could carry critical information in security graphs. Take LANL as an example: in addition to the source and destination hosts, each authentication event includes the user ID for login. Malicious login events could have user IDs that are rarely seen on the source or destination hosts. In addition to the authentication logs, LANL also collects other data that provides contextual information to logins, like network flows. Yet, challenges exist in what features to select and how to integrate them into a UNRL framework.

**Issue-C: Static graph modeling over long period.** Most GSAs, except Euler and PIKACHU, rely on learning/detecting on a gigantic static graph, which is usually constructed over a long period of time. For example, Shade-Watcher [16] merges events from an entire week to construct

---

3. STEP [36] reported over 0.98 accuracy on LANL, but its goal is to predict the existence of edges, rather than detecting redteam events.

4. Euler reported average precision, instead of precision, true positives and false positives, so we reran Euler [45] in dynamic link detection mode (explained in Section 4.1) with default parameters. We are able to obtain similar metrics like FPR as the paper (0.0045 vs 0.0057).
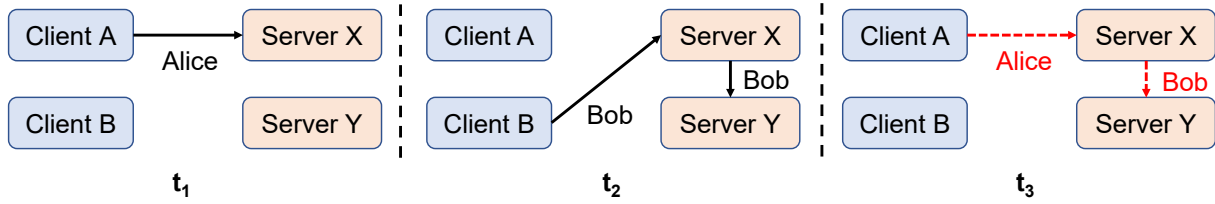
Figure 2. An example of anomalous logins, which is similar as [44]. Alice and Bob own two client machines A and B in an enterprise. Server X and Y can be logged in with Alice's and Bob's credentials remotely. The attacker conducts lateral movement at the period $t_3$ to log into Server Y.

a graph. The static graph provides rich information for learning, but also loses critical information for detection:

1) The long waiting time for all events to arrive to construct a static graph is not friendly for early alerting.
2) The temporal patterns ignored by the static graph could lead to evasion of advanced attacks. Here we use a lateral movement example as demonstration, which is also illustrated in Figure 2. Normal logins happen at $t_1$ and $t_2$ by Alice and Bob separately. In $t_3$, an attacker Eve uses Alice's credential to log into Server X, compromises Bob's credentials there (e.g., thorough pass-the-hash attacks [9]), and uses it to log into Server Y. If the detection model is trained on a graph merging events form $t_1$ to $t_2$, the malicious logins on $t_3$ are likely to be missed, because it exists in the training graph as a normal path. On the other hand, the logins in $t_3$ are likely to be considered as abnormal under DTG as each snapshot and their delta are modeled.

### 3.2. Solutions from ARGUS

To address the aforementioned issues, we believe following the generic workflow and framework of UNRL is insufficient, and the GSA should be carefully designed to incorporate security domain knowledge. Here we review the key components that are integrated by ARGUS, and elaborate on them in Section 4.

- To integrate the useful event fields as edge features, we designed a new encoder on top of Message Passing Neural Networks (MPNNs) [21].
- Instead of training GSA model to achieve high accuracy, we adopt new loss functions [23] to optimize the model for high precision.
- For the downstream decoder, we redesign it to capture the community patterns that are likely exhibited by the attacker (e.g., lateral movement and port scanning), by adjusting previous methods in graph anomaly detection [22]).
- Instead of static graph modeling, we model the events as DTG like [14], [15] to capture the temporal dynamics.

### 4. Design of ARGUS

In this section, we first describe the workflow of ARGUS. Then, we elaborate the design of our encoder. Finally, the decoder is described.

### 4.1. Operational Mode and Threat Model

ARGUS aims to detect traces related to ongoing or completed cyber-attacks, including unauthorized logins, malware infections, lateral movements, etc. Like other works built on UNRL [14]–[16], we assume ARGUS is trained using data from an *attack-free* period. During testing, ARGUS classifies an edge in a DTG snapshot as benign or malicious, using all node and edge information observed in this snapshot. This mode aligns with the *dynamic link detection* mode presented by Euler [14]. Noticeably, Euler has another mode named *dynamic link prediction*, which classifies an edge without any information of its current snapshot (it only relies on the prior snapshots). However, this mode incurs much higher error rate than detection, hence we leave this mode out. In Section 5.5, we conduct a preliminary study about how data poisoning impacts the performance of ARGUS.

ARGUS follows the standard threat model used by other security analytics, which assumes the correctness and integrity of the logs/data collected from hosts and network monitors. Note that advanced attackers could break this assumption, and systems that protect log integrity can be integrated as a defense [48], [49].

### 4.2. Workflow

As a UNRL system, ARGUS follows the generic GAE framework [13] to compute the node embedding from a graph and uses the decoder to compute edge probabilities from the node embedding. The original GAE uses GCN for the encoder and the inner-product of node embedding for the decoder. More formally,

$$\hat{A} = p(A|Z) = \sigma(ZZ^\top), \text{with } Z = \text{GCN}(X, A) \quad (2)$$

where $Z \in \mathbb{R}^{n \times r}$ is the node embedding generated by the encoder, $X$ represents the node features, $A$ is the adjacency matrix, $\sigma$ is the activation function, and $\hat{A}$ is the reconstructed adjacency matrix by the decoder, which can contain edge probabilities $p(A|Z)$. Training GAE aims to update the GCN parameters such that $\hat{A}$ and $A$ are similar.

Under DTG modeling, node embedding $Z_t$ should be generated for each graph snapshot $\mathcal{G}_t$, and the embeddings before $t$ could be leveraged to update $Z_t$. Hence, the GAE encoder needs to be redesigned to accommodate the prior embeddings. Like previous works [20], we stack the encoder upon a Recurrent Neural Network (RNN), e.g., Gated
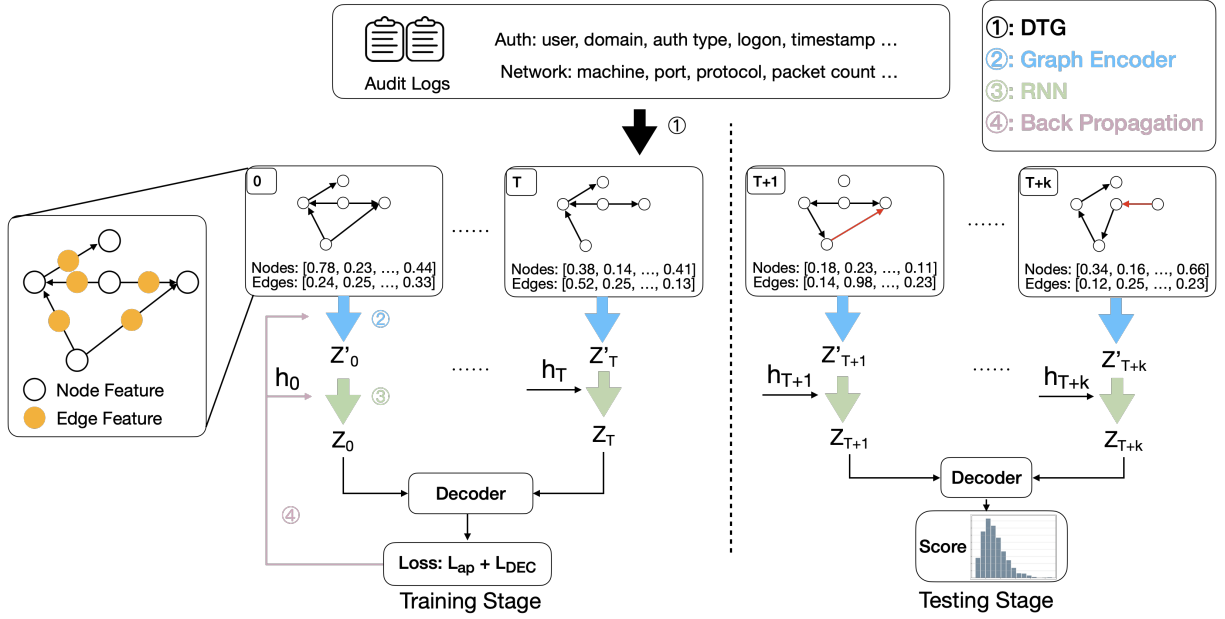
Figure 3. ARGUS workflow.

Recurrent Unit (GRU), to capture the topology and node changes in a sequence of DTG snapshots. More formally,

$$Z'_t = \text{ENC}(X_t, A_t, F_t) \tag{3}$$

where ENC is the trained encoder. $X_t$ and $A_t$ are the node features and adjacency matrix at $t$. When ENC uses GCN and only uses $X_t$ and $A_t$ as input, its layer-wise structure follows Equation 1. In Section 4.3, we show a new design of ENC that considers edge features $F_t$ as another input.

Assuming $T$ is the number of all observed snapshots, the RNN layer takes in $[Z'_0, Z'_1, ..., Z'_T]$ and updates them to $[Z_0, Z_1, ..., Z_T]$ by leveraging the observed temporal dynamics.

$$[Z_0, Z_1, ..., Z_T] = \text{RNN}([Z'_0, Z'_1, ..., Z'_T]) \tag{4}$$

The node embedding $Z_t$ can be extracted from the RNN output at index $t$. The temporal relation is automatically used by the RNN, which uses the embeddings from snapshots 0 to $T$ to update the same set of snapshots.

In a DTG snapshot $\mathcal{G}_t$, the decoder takes the $Z_t$ generated by the encoder to reconstruct the adjacency matrix $\hat{A}_t$. We note that the decoding function can be different from $\sigma(ZZ^\top)$, so we define the DTG decoder DEC as:

$$\hat{A}_t = p(A_t|Z_t) = \text{DEC}(Z_t) \tag{5}$$

In Section 4.4, we elaborate the design of our DEC.

During training, ARGUS loads event logs, splits them into $[\mathcal{G}_0, \mathcal{G}_1, ..., \mathcal{G}_T]$, and generates $X_t$, $A_t$ and $F_t$ for each snapshot $\mathcal{G}_t$ ($0 \leq t \leq T$). In the forward pass, the encoder produces node embeddings $[Z_0, Z_1, ..., Z_T]$ and the decoder reconstructs $[\hat{A}_0, \hat{A}_1, ..., \hat{A}_T]$ from them. The loss is computed between $[\hat{A}_0, \hat{A}_1, ..., \hat{A}_T]$ and the ground-truth $[A_0, A_1, ..., A_T]$. In the backpropagation pass, the loss is used to compute the gradients and update the RNN, encoder, and decoder (if it has trainable parameters). During testing, after logs of a snapshot duration $T+k$ ($k > 0$) are collected, its embedding $Z_{T+k}$ will be generated using $X_t$, $A_t$ and $F_t$, together with the prior embeddings $Z_0, Z_1, ..., Z_{T+k-1}$. The decoder generates $\hat{A}_{T+k}$ from $Z_{T+k}$. Instead of computing loss from this reconstructed adjacency matrix, we compare the probability score of each edge in $\hat{A}_{T+k}$ with a threshold $\tau$, and set off alarms for the ones under $\tau$. We summarize the whole workflow of ARGUS, including training and testing, in Algorithm 1 and Algorithm 2.

**Edge sampling.** In the standard UNRL setting, the training objective can be measured by a loss function:

$$\begin{aligned} \mathcal{L} &= -log(p(A_t|Z_t)) \\ &= -\frac{\Sigma_{e \in P_t} log(f(e))}{|P_t|} - \frac{\Sigma_{e' \in N_t} log(1 - f(e'))}{|N_t|} \end{aligned} \tag{6}$$

where $P_t$ and $N_t$ mean the positive (exist) and negative (non-exist) edges in $\mathcal{G}_t$ and $f$ computes a score from edge $e$ or $e'$. Computing on this loss function is intractable when the number of nodes is large. As a large graph is often sparse, negative sampling [50] is often performed to sample the negative edges, so $N_t$ will be reduced. The chances that a negative edge is sampled can be uniformly random or proportional to the node degrees. During evaluation, we set the number of sampled negative edges same as the positive ones.

## 4.3. Encoder

Different from previous works that only consider node features and edge weights (Issue-B in Section 3.1), we propose a new encoder that can integrate edge features, so the

3595

**Algorithm 1:** The pseudocode of training.

**Input:** the original graph $\mathcal{G}$, the snapshot windows $T$
**Output:** the encoder ENC, the RNN model RNN, the decoder DEC
$\mathcal{G}_0, \mathcal{G}_1, \ldots, \mathcal{G}_T = \text{Separate}(\mathcal{G}, \mathcal{T})$;
**while** ! *early_stop_requirement* **do**
    $loss = 0$;
    **for** $i \leftarrow 0$ **to** $T$ **do**
        $X_i^{train}, A_i^{train}, F_i^{train} = \text{data\_split}(\mathcal{G}_i)$;
        $Z_i' = \text{ENC}(X_i^{train}, A_i^{train}, F_i^{train})$;
        $Z_i = \text{RNN}(Z_i')$;
        $\hat{A}_i = \text{DEC}(Z_i)$;
        $loss = \mathcal{L}_{ap} + \beta \cdot \mathcal{L}_{DEC}$;
    **end**
    loss.back_propagate;
**end**
return ENC, RNN, DEC;

---

**Algorithm 2:** The pseudocode of testing.

**Input:** the original graph $\mathcal{G}'$, the snapshot windows $T'$, the encoder ENC, the RNN model RNN, the decoder DEC
**Output:** the edge classification results $Ys$
$\mathcal{G}_0, \mathcal{G}_1, \ldots, \mathcal{G}_T' = \text{Separate}(\mathcal{G}', \mathcal{T}')$;
$Ys = []$;
**for** $i \leftarrow 0$ **to** $T'$ **do**
    $X_i^{test}, A_i^{test}, F_i^{test} = \text{data\_split}(\mathcal{G}_i)$;
    $Z_i' = \text{ENC}(X_i^{test}, A_i^{test}, F_i^{test})$;
    $Z_i = \text{RNN}(Z_i')$;
    $\hat{A}_i = \text{DEC}(Z_i)$;
    $Y = \text{SCORE}(\hat{A}_i)$;
    $Ys.append(Y)$;
**end**
return $Ys$;

---

TABLE 3. THE ENCODER STRUCTURE. X_DIM EQUALS TO THE NUMBER OF NODES AND F_DIM EQUALS TO THE NUMBER OF FEATURES.

| Layer | Name | Size |
|---|---|---|
| 1 | $EW(\cdot)$ | (x_dim, 32) |
| 2 | $EW(\cdot)$ | (32, 32) |
| 3 | drop_out + activation function | |
| 4 | $EW(\cdot)$ | (32, 32) |
| 5 | drop_out + activation function | |
| 6 | $EF(\cdot)$, MLP | (32, 16), (f_dim, 8) |
| 7 | activation function | |

where $X_u$ denotes the node features of $u$, $F_{u,v}$ denotes the edge feature between the edge $(u, v)$, $N(u)$ denotes the neighborhood of $u$, $M_\theta(\cdot)$ aggregates the feature values passed by the neighborhood nodes, which can be simple a Multilayer perceptron (MLP), and $\sigma$ denotes the activation function.

$$\text{EW}(X, W) = \sigma(\hat{D}^{-\frac{1}{2}} W \hat{D}^{-\frac{1}{2}} X \Theta) \quad (8)$$

where $\hat{D}$ is the diagonal degree matrix to normalize the edge weights' matrix $W$, $\Theta$ represents the trainable parameters, and $\sigma$ denotes the activation function, which is set to Tanh.

In our experiment, we use three $EW(\cdot)$ layers and one $EF(\cdot)$ layer. The structure is shown in Table 3.

**Node features.** Node features $X_t$ can be simply initialized as the IDs (e.g., machine IDs assigned by the domain controller) if there is no other node information. The types of the nodes can also be considered, e.g., machine type like server or workstation, if they are logged, and converted into one-hot encoding and combined with the IDs.

**Edge weights.** For a pair of nodes, multiple events could happen in a snapshot $\mathcal{G}_t$ and they will be merged to form one edge. The frequency of events could be useful to detect abnormal behaviors (e.g., a sudden high-volume of authentication requests can be resulted from brute-force logins), which could fill edge weights. Given all edges of $\mathcal{E}_t$, its edge weights $W_t$ can be computed and normalized as:

$$W_t = \sigma\left(\frac{C_t}{\text{std}(C_t)}\right) \quad (9)$$

where $\sigma$ denotes the sigmoid function, $C_t$ is the frequency of all edges in $\mathcal{E}_t$, and std is the standard deviation.

**Edge features.** When merging events to an edge, the event fields could be aggregated to generate edge features. Given that an event can have numerous fields, feature selection should be done before presenting to the encoder. Our empirical evaluation suggests using the item frequency from the categorical fields (e.g., the number of logins initiated by normal users, administrators, and operating systems) and raw values from the numerical fields (e.g., the traffic volume of each network flow) can be used. The edge features $F_t$ for $\mathcal{E}_t$ can be computed and normalized similarly as Equation 9:

$$F_t = [\sigma\left(\frac{S_{t,0}}{\text{std}(S_{t,0})}\right), \sigma\left(\frac{S_{t,1}}{\text{std}(S_{t,1})}\right), ..., \sigma\left(\frac{S_{t,n}}{\text{std}(S_{t,n})}\right)] \quad (10)$$

where $n$ is the number of used features and $S_{t,i}$ ($0 \le i \le n$) has the values for feature $i$ in $\mathcal{E}_t$.

generated node embedding would have richer information about the host/network interactions. Below we describe the structure of the encoder, and then give some guidance on how to generate node features, edge weights, and edge features from the log data.

**Encoder structure.** Our design is inspired by MPNN [21], which uses a message-passing layer to merge edge features from the neighborhood nodes into node embedding. In particular, our encoder is composed of two types of layers which can take edge weights and edge features as input separately, and we name them $EF(\cdot)$ and $EW(\cdot)$. $EF(\cdot)$ uses a trainable function to aggregate the features from the neighboring nodes and $EW(\cdot)$ is designed similarly as GCN, which can process node features and edge weights. More formally, given a node $u \in \mathcal{V}$,

$$\text{EF}(X_u, F_{u,v}) = \sigma\left(X_u + \sum_{v \in N(u)} X_v \cdot M_\theta(F_{u,v})\right) \quad (7)$$

## 4.4. Decoder and Loss

The decoder defined in Equation 5 aims to reconstruct $\hat{A}_t$ from $Z_t$, but the attack-agnostic decoder chosen by the prior work could lead to inaccurate detection results (Issue-B in Section 3.1). We consider the attack characteristics that are generic to customize the decoder. One prominent characteristic is that to attack a target, the attacker would need to exploit the entities that is reachable to it (e.g., using a compromised client as the stepping stone to hack into the server), hence the neighborhood nodes could also be suspicious.

Based on such insights and previous works in graph anomaly detection [22], we design a random sampling aggregation operation from each node's neighborhoods to update the node embeddings. We acknowledge that this aggregation method might not be optimal as it does not exploit the differences of neighborhood nodes, and we plan to improve this design in the future. Equation 11 defines a new decoding operation based on Equation 5.

$$\hat{A}_t = \text{DEC}(Z_t^{agg}) = \text{Softmax}(\text{MLP}(Z_t^{agg})) \qquad (11)$$

where $Z_t^{agg} = S * A_t * Z_t$. $S$ is an identity matrix that samples $s$ neighborhoods from the adjacency matrix $A_t$ per row. $s$ is a constant representing the sample number. Similar to the basic decoder, we use cross-entropy (CE) to compute the loss of the decoder ($\mathcal{L}_{DEC}$) on the sampled edges.

$$\mathcal{L}_{DEC} = -log(p(A_t|\hat{A}_t)) \qquad (12)$$

**Average Precision (AP) loss.** Under CE, $\mathcal{L}_{DEC}$ still optimizes the GSA model under the accuracy metric. As described in Issue-A in Section 3.1, precision is a more suitable metric for imbalanced security datasets, and the learning objective can be designed to maximize *Area under Precision and Recall curve (AUPRC)*. Directly maximizing AUPRC incurs high overhead, and recently, Qi et al. proposed to use a surrogate Average Precision (AP) loss to approximate AUPRC [23]. We adapt the AP loss into ARGUS, and follow the implementation of LibAUC [51].

Specifically, the surrogate loss for AP can be written as the equation below:

$$\mathcal{L}_{ap} = \frac{1}{n_{P_t}} \sum_{e_i \in P_t} \frac{-\sum_{s=1}^n I(y_s = 1)l(e_s; e_i)}{\sum_{s=1}^n l(e_s; e_i)} \qquad (13)$$

where $P_t$ are positive edges in $\mathcal{G}_t$, $n_{P_t}$ is the number of positive edges, $e_i$ is one positive edge, $n$ is the number of all edges in $\mathcal{G}_t$, $e_s$ is any edge in $\mathcal{G}_t$, $y_s$ is the edge label ($y_s = 1$ means positive edge), $I$ is the identity function that outputs 1/0 when the argument is true/false. $l(e_s; e_i) = (m - (f(e_i) - f(e_s)))^2$ is the smooth squared loss that approximates $I(f(e_s) \geq f(e_i))$, with $f$ being the prediction model (i.e., GSA) and $m$ being the margin parameter. Minimizing $\mathcal{L}_{ap}$ is tractable by rewriting Equation 13 into a finite-sum of compositional functions and performing SGD-style or Adam-style stochastic optimization [23].

Finally, we combine $\mathcal{L}_{ap}$ and $\mathcal{L}_{DEC}$ and assign different weights to them, according to the dataset characteristics

(e.g., the highly imbalanced dataset would require a large weight on $\mathcal{L}_{ap}$). The final loss function becomes:

$$loss = \mathcal{L}_{ap} + \beta \cdot \mathcal{L}_{DEC} \qquad (14)$$

where $\beta$ is the parameter configured by the operator.

## 5. Evaluation

In this section, we present a comprehensive evaluation of ARGUS. We first describe the experiment settings, including the evaluation metrics, baseline models, and the running environment. Then, we describe our experiment on LANL and OpTC dataset separately, with comparison to the baseline models and the analysis of each component, etc. Finally, we evaluate the efficiency of ARGUS, which is described in Appendix A.3 due to page limit.

## 5.1. Settings

**Evaluation metrics.** Different from the standard link prediction that consider the positive edges as true positives (TP) and negative edges as true negatives (TN), given that our goal is to detect attacks (or malicious events), we define the edges containing at least one malicious event as TP, and the edges containing all normal events as TN. The malicious event is defined as *any* redteam event described in the ground-truth datasets or documents of LANL [24] and OpTC [25], and this is the same approach adopted by the baseline models like Euler and PIKACHU. False positives (FP) and false negatives (FN) are defined as the edges misclassified as malicious and normal respectively. With TP, TN, FP and FN defined, we compute the following metrics:

$$\text{Precision} = \frac{TP}{TP + FP}$$
$$\text{Recall} = \frac{TP}{TP + FN} \qquad (15)$$
$$\text{FPR} = \frac{FP}{TP + FP}$$

We also compute AUC for comparison with other works, but this number should be read with a grain of salt. Due to that the malicious and normal events are highly imbalanced, high AUC might yield misleading estimation of the effectiveness of a GSA ("Base Rate Fallacy" of [52]). For example, Euler has over 0.99 AUC, but the ratio between FP and TP is over 180x (see Section 3.1). Like Euler, we also compute average precision (AP) [53], which is defined as:

$$\text{AP} = \sum_n (R_n - R_{n-1}) \times P_n \qquad (16)$$

where $R_n$ and $P_n$ are the precision and recall at the $n$-th threshold. AP provides a better assessment than AUC on the imbalanced dataset. In addition to effectiveness, we also measure the efficiency of ARGUS, focusing on the time of training and testing.

TABLE 4. THE STATISTICS OF THE LANL DATASET FOR THE FIRST 14 DAYS.

| Dataset | Nodes | Events | Time |
|---------|-------|--------|------|
| LANL-auth | 15,610 | 239,558,591 | 14 days |
| LANL-flows | 11,504 | 114,584,051 | 14 days |
| LANL-redteam | 288 | 627 | 14 days |

**ARGUS Hyper-parameters.** The encoder hyperparameters have been described in Table 3. The RNN model is a two-layer GRU. The learning rate is 0.01. The node embedding size is 16. For the weight in the loss function, $\beta$ is 0.01 for LANL and 0.5 for OpTC based on their data distributions. For AP Loss, the margin parameter $m$ is 0.8. Qi et al. uses a moving average estimator for the stochastic optimization [23], and we set its moving average $\gamma$ to 0.01 for LANL and 0.1 for OpTC.

**Baseline models.** We consider 4 GSA models as baseline to compare with ARGUS, including Netwalk [35], PIKACHU [15], VGRNN [13], and Euler [14], which are all designed for anomaly detection on DTGs and providing source code. We also consider two non-GSA models, including Local Outlier Factor (LOF) [54] and Isolation Forest (IF) [55], which were also compared by other GSAs [10]. More details of these models and the parameters are shown in Appendix A.1.

**Computing Environment.** We run the experiments on a workstation which has an AMD Ryzen Threadripper 3970X 32-Core Processor and 256 GB CPU memory. ARGUS runs on CPU by default and in Section A.3, we assess the performance under GPU implementations, and our GPU is NVIDIA GeForce RTX 3090 with 24GB memory. We use PyTorch 1.10 and Python 3.9.12 as environment when building ARGUS. The OS is Ubuntu 20.04.2 LTS.

## 5.2. Evaluation on LANL

**Dataset and pre-processing.** Our first experiment evaluates the effectiveness of ARGUS on the LANL datasets, which has been briefly described in Section 2.4. We chose LANL because it was also evaluated by Euler, our primary baseline model. We perform attack detection on LANL-auth to identify the malicious authentications simulated by the redteam (recorded by LANL-redteam). We follow the implementation of Euler [45] to filter out the events without the keyword "NTML" (short for Windows New Technology LAN Manager [56]), which are unrelated to user authentication. To construct DTG, we split the LANL-auth data by 1-hour (3,600s) snapshots, and all events sharing the same source and destination nodes are merged into an edge. Later we will also evaluate the impact of different snapshot sizes. Similar as to Euler, we use the snapshots of the first 42 hours for model training, as after then the first redteam event is observed. Each snapshot has an average of 7,957 edges and we leave out 5% edges for validation. The remaining snapshots are used for testing (all edges are tested).

Within each snapshot, we use node index to construct node features $X_t$, normalized event frequency to construct

edge weights $W_t$ (described in Section 4.3). Since our encoder is able to ingest edge features, we select **3 features from LANL-auth events**, by the frequency of user-initiated, computer-initiated and anonymous logins, which can be learned from the first letter of the `source user@domain` field (U, C or A). By closely examining the other datasets provided by LANL, we found LANL-flows, which records the network flows coming through the central routers, are potentially useful in providing network traffic statistics. Hence, for each edge generated from LANL-auth, we search for the related events in **LANL-flows**, aggregate them by the pair of source and destination, and extract **7 additional features**: the number of flows, the mean of flow duration, the standard deviation of flow duration, the mean of packet count, the standard deviation of packet count, the mean of byte count, and the standard deviation of byte count. When examining LANL-flows, we found its daily volume drops significantly (by around 80%) after Day 14, and we found the LANL data provider admitted that there are issues with data collection [57]. Hence we test the snapshots within the first 14 days. Table 4 lists the statistics for this subset. Compared to the full datasets shown in Table 2, 83.7% (627 out of 749) redteam events are kept, and 88.44% (15,610 out of 17,649) nodes are included. The other baseline approaches are tested with the same training and testing snapshots.

TABLE 5. EVALUATION ON LANL DATASET. THE SNAPSHOT SIZE IS 3,600 SECONDS.

| Model | AP | AUC | Precision | Recall | FPR |
|-------|-----|-----|-----------|--------|-----|
| IOF | 0.0002 | 0.5048 | 0.0000 | 0.0023 | 0.0093 |
| IF | 0.0003 | 0.6969 | 0.0000 | 0.0137 | 0.1901 |
| Netwalk | 0.0029 | 0.7195 | 0.0000 | 0.0251 | 0.1908 |
| PIKACHU | 0.0038 | 0.9517 | 0.0040 | 0.9590 | 0.0413 |
| VGRNN | 0.0125 | 0.9660 | 0.0107 | 0.2232 | 0.0036 |
| Euler | 0.0448 | 0.9810 | 0.0318 | 0.8565 | 0.0045 |
| ARGUS | **0.3227** | 0.9983 | 0.2171 | 0.8269 | 0.0005 |

**Overall effectiveness.** In Table 5, we show the effectiveness of ARGUS and compare it to other baseline methods. In terms of AP, ARGUS is able to reach as high as 0.3227, which is **7.2x** from Euler and **25.8x** from VGRNN. We also note that both Euler and ARGUS achieve very high AUC (0.9810 and 0.9983), but a vast difference exists in AP, indicating AUC is imprecise in measuring the detection effectiveness. For the non-GSA models like IOF and IF, their AP are even lower (0.0002 and 0.0003) than every GSA model.

AP is averaged across precision and recall under different thresholds, as shown in Equation 16. In Table 5, we also report classification result under a threshold that is learned from the validation set. For ARGUS, reasonable precision (0.2171) and high recall (0.8269) can be achieved. In this case, 1,309 FP is triggered, which leads to very low FPR at 0.0005. In the meantime, 363 TP are detected. Though Euler and PIKACHU are able to detect more TP (376 for Euler and 421 for PIKACHU), a much higher number of FP is triggered (11,464 for Euler and 104,311 for PIKACHU), which are 8.76x and 79.68x compared to ARGUS. The precision-recall curves of all methods are shown in Figure 4.

Taking a close look at the FN, we found that many of them can be potentially detected at the later investigation stage. For instance, out of the 76 FN of ARGUS, 30 of them share the same source and destination with the TP. As mentioned by [14], LANL might also have a data labeling issue that the redteam events were not all logged (e.g., malicious activities following compromise events are not tracked). We also found some TN are potentially TP: we search LANL-auth for the edges with the same user ID, source and destination as the malicious edges but contained by different snapshots, and found **986** such edges. In Appendix A.2, we provide more details about the detection results separated by snapshots.
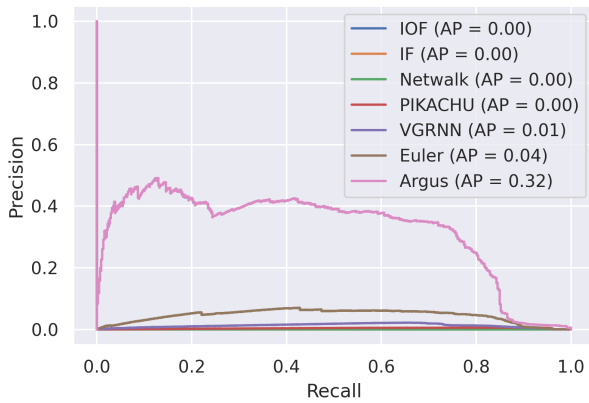
|  |  | AP | AUC | FPR |
|---|---|---|---|---|
| Activation | Tanh | 0.3227 | 0.9983 | 0.0005 |
|  | Softmax | 0.2780 | 0.9979 | 0.0008 |
|  | ReLU | 0.2426 | 0.9973 | 0.0006 |
|  | Sigmoid | 0.1338 | 0.9950 | 0.0005 |
| Snapshot (s) | 3,600 | 0.3227 | 0.9983 | 0.0005 |
|  | 1,800 | 0.1128 | 0.9930 | 0.0004 |
|  | 900 | 0.0550 | 0.9963 | 0.0015 |
|  | 360 | 0.0680 | 0.9968 | 0.0008 |



Figure 4. Precision-recall curves on LANL.
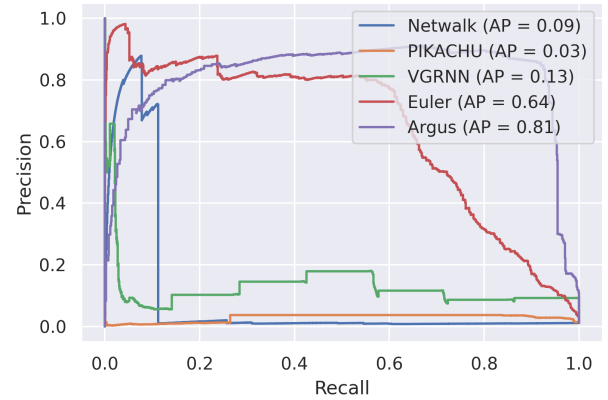


Figure 5. Precision-recall curves on OpTC.

**Different configurations.** Here we investigate the impact of different configurations of ARGUS on the overall effectiveness. We first evaluated the different choices of activation functions that are used by the encoder. We choose this component because the final edge scores heavily depend on the activation functions, and a good function should disperse the scores to a wider range, so it is easier to separate the benign and malicious edges by threshold. In particular, we tested the four most common activation functions: Tanh, Softmax, ReLU, and Sigmoid. In Table 6, all four activation function have high AUC (around 0.99) and low FPR. In addition to Tanh, Softmax and ReLU also has around 0.28 and 0.24 AP which are over 0.1 higher than Sigmoid.

We also assessed the impact of snapshot size, as the larger the snapshot, the more edges will be included, which can provide more contextual information of the attack and benign activities, but also increase the risk of wrong predictions, as more events will be merged and resulting in lower edge/event ratio. We found that 3,600 achieves the best result, and decreasing the snapshot size leads to a much lower AP (from 0.3227 to 0.0680 of 360).

## 5.3. Evaluation on OpTC

**Dataset and data preprocessing.** OpTC is another log dataset that has been used to evaluate DTG-based

GSAs [15], and we evaluate ARGUS on it and compare with the other GSAs. As described in Section 2.4, we use the network flows, so the hosts that are distinguished by IP addresses are used as nodes and the flows between pairs of hosts are merged into edges. We construct node features $X_t$ and edge weights $W_t$ in the same way as LANL. Though a network event contains fields like pid (process ID), ppid (parent process ID), source port, destination port and image path, we found it is difficult to extract useful statistical features from them. For example, pid and ppid are categorical fields but their ranges are very large. Hence, we did not use edge features in this experiment.

For the snapshot size, we set it to 360 seconds, which is much smaller than the default of LANL, because the number of nodes in OpTC (814) is much smaller than LANL (15,610) and the event frequency between a pair of hosts is much higher. Setting a large snapshot size will merge too many events. In the end, 1,680 snapshots are derived for the whole week. We use the first 5 days' snapshots for training and the remaining 3 days' snapshots for testing.

**Overall effectiveness.** Table 7 lists the overall effectiveness[5]. We found ARGUS is able to improve AP of Euler by 0.17 (increased from 0.6426 to 0.8074) and AUC of Euler by 0.013 (0.9840 to 0.9970). ARGUS has a higher FP

5. Our evaluation results of PIKACHU are worse than their paper (e.g., 0.99 AUC). One reason might be that data pre-processing is different (the code repo does not include this part [58]).

TABLE 7. Evaluation on OpTC. The snapshot is 360 seconds.

| Model | AP | AUC | Precision | Recall | FPR |
|---|---|---|---|---|---|
| Netwalk | 0.0901 | 0.7435 | 0.0259 | 0.3957 | 0.1690 |
| PIKACHU | 0.0285 | 0.9760 | 0.0370 | 0.8383 | 0.2479 |
| VGRNN | 0.1267 | 0.9473 | 0.0806 | 0.0493 | 0.0064 |
| Euler | 0.6426 | 0.9840 | 0.5273 | 0.6897 | 0.0070 |
| ARGUS | **0.8074** | 0.9970 | 0.2960 | 0.9647 | 0.0261 |

(2,795 vs 753) and higher TP (1,175 vs 840) over Euler, and we can balance the FP and TP of ARGUS by adjusting the classification threshold. Overall, the result shows ARGUS is able to achieve good performance consistently across datasets of different characteristics. The other GSAs except Euler are performing much worse, with the highest AP at 0.1267 by VGRNN. The precision-recall curves are shown in Figure 5.

**Different configurations.** In Table 8, we show the impact of the activation function, and ReLU achieves 0.015 better AP than Tanh but the differences are much smaller. We also show the impact of different snapshot sizes. ARGUS achieves the best performance at 360s snapshot size. Interestingly, less FP is triggered at a larger size, though the number of edges is much lower, suggesting some malicious events are hidden in merged edges. In Appendix A.2, we provide a case study to justify the need for DTG modeling.

TABLE 8. ARGUS with different activation functions (Snapshot size is 360) and different snapshot sizes on OpTC (activation function is Tanh).

| | | AP | AUC | FPR |
|---|---|---|---|---|
| | Tanh | 0.8074 | 0.9970 | 0.0261 |
| Activation | Softmax | 0.7834 | 0.9881 | 0.1022 |
| | ReLU | 0.8221 | 0.9969 | 0.0256 |
| | Sigmoid | 0.8064 | 0.9869 | 0.0563 |
| | 3,600 | 0.0238 | 0.9470 | 0.000 |
| Snapshot (s) | 1,800 | 0.5836 | 0.9875 | 0.0000 |
| | 900 | 0.5398 | 0.9885 | 0.0014 |
| | 360 | 0.8074 | 0.9970 | 0.0261 |

## 5.4. Ablation Study

To understand the contribution of each proposed component in ARGUS, we conduct an ablation study by switching one component with a standard method at a time and measuring the differences in effectiveness. The standard methods are cross-entropy (CE) loss (the same as Euler), GCN (2 layers, the same as Euler), LSTM, and inner product (IP, the same as Euler), which replace the AP loss, MPNN-based encoder, GRU-based RNN layer, and MLP-based decoder (in Equation 11) used by ARGUS. Table 9 and Table 10 show the results on the LANL and OpTC datasets.

For LANL, we observe the proposed encoder makes the biggest contribution, as the AP drops from 0.3227 to 0.0832 after replacing it with the standard GCN encoder that only processes node features and edge weights, suggesting edge features are vitally important. Prominent performance drops are also observed when using LSTM for RNN and CE for

loss. The impact of the MLP-based decoder is insignificant when switching to IP (less than 0.001 AP difference).

For OpTC, though edge features are not used, replacing the encoder with GCN still causes a large drop in AP (from 0.8074 to 0.3856), and the major reason could be that the baseline GCN structure is simple (only two layers). On the other hand, using LSTM for the RNN layer and CE for the loss could actually increase AP, though the improvements are both below 0.1. The similar observations have also been made by Euler: e.g., the default GCN+GRU setting performs best under dynamic link detection but worse than SAGE+LSTM under dynamic link prediction (Table VI of [14]).

TABLE 9. The ablation study of ARGUS on LANL.

| Encoder | Decoder | RNN | Loss | AP | AUC | FPR |
|---|---|---|---|---|---|---|
| ✓ | ✓ | ✓ | ✓ | 0.3227 | 0.9983 | 0.0005 |
| ✓ | ✓ | ✓ | CE | 0.2676 | 0.9966 | 0.0011 |
| GCN | ✓ | ✓ | ✓ | 0.0832 | 0.9929 | 0.0033 |
| ✓ | ✓ | LSTM | ✓ | 0.1691 | 0.9910 | 0.1653 |
| ✓ | IP | ✓ | ✓ | 0.3207 | 0.9985 | 0.2344 |

TABLE 10. The ablation study of ARGUS on OpTC.

| Encoder | Decoder | RNN | Loss | AP | AUC | FPR |
|---|---|---|---|---|---|---|
| ✓ | ✓ | ✓ | ✓ | 0.8074 | 0.9970 | 0.0261 |
| ✓ | ✓ | ✓ | CE | 0.8121 | 0.9966 | 0.0267 |
| GCN | ✓ | ✓ | ✓ | 0.3856 | 0.9873 | 0.0217 |
| ✓ | ✓ | LSTM | ✓ | 0.8718 | 0.9970 | 0.0063 |
| ✓ | IP | ✓ | ✓ | 0.7537 | 0.977 | 0.0020 |

## 5.5. Data Poisoning Attack

In Section 4.1, we assume the attacker cannot tamper with the training dataset. Here we revise this attacker's capability and study how "data poisoning" attack [59], through which the attacker manipulates the edges and/or nodes in the *training* set, impacts the effectiveness of ARGUS. We examine the attack on the LANL dataset and simulate two poisoning strategies. Table 11 lists the results.

First, we simulate a powerful attacker who is active in *both* training and testing stage (called "cross-stage attack"), and the malicious activities (i.e., authentications) conducted in the training stage are not detected (labeled as benign). The attacker does not need to know the exact training window (i.e., which data samples in the event stream are drawn for training). Specifically, we extend the training period to cover the first $K$ malicious edges in testing, and we set $K$ to 5, 10, and 20. It turns out AP drops significantly (e.g., from 0.3227 to 0.1281 when $K = 5$), and the main reason is that the redteam keeps running the same authentication attempts. For example, 15 malicious edges in the training set also appear in the remaining testing set, when $K = 20$.

Parallel to our work, Xu et al. examined data poisoning attack against GSAs and evaluated it against the LANL dataset (with Euler as the GSA) [60]. They proposed to inject "covering accesses" during training that is against the learning objective of a GSA model, assuming the attacker

has white-box access to the model [60]. For each malicious event, we name its source and destination nodes as malicious nodes. $N$ covering edges are injected for each malicious node to other nodes in the last training snapshot, and we select the covering edges randomly instead of using the optimization-based method like [60], as we did not have access to its source code (we call our version "edge injection attack"). We examined different $N$ values (2, 5, 10, 100), and found AP drops significantly (from 0.3227 to 0.1464) when $N = 100$. However, given that the average edge number for that snapshot is only 0.62, injecting many authentication events can expose the attacker under a simple anomaly detector that counts the event frequency. Interestingly, Xu et al. argued that "poisoning attack is still more difficult to launch" because only AUC is measured (0.997 AUC dropped to 0.988 when changing $N$ from 0 to 100) [60], but this conclusion is questionable when considering AP.

To notice, the adaptive attacker can attack GSAs in methods other than data poisoning. One such method is "evasion attack", which manipulates the graph structure at the *testing* stage. Wang et al. showed that graph-based classification can be fooled under optimization-based attacks [61], and recently Goyal et al. tested the methods against GSAs of simpler designs (e.g., Unicorn that computes Jaccard Similarity between subgraphs [62]). It is unclear how attackers can manipulate the malicious behaviors to evade UNRL-based GSAs and we plan to investigate this issue.

TABLE 11. THE RESULTS OF POISONING ATTACK ON LANL. "CSA" AND "EIA" MEAN "CROSS-STAGE ATTACK" AND "EDGE INJECTION ATTACK".

| Attack | | AP | AUC | FPR |
|---|---|---|---|---|
| CSA, $K=$ | 0 | 0.3227 | 0.9983 | 0.0005 |
| | 5 | 0.1281 | 0.9947 | 0.0012 |
| | 10 | 0.042 | 0.9877 | 0.0025 |
| | 20 | 0.0248 | 0.978 | 0.0013 |
| EIA, $N=$ | 2 | 0.2749 | 0.9983 | 0.0008 |
| | 5 | 0.2785 | 0.9983 | 0.0009 |
| | 10 | 0.2780 | 0.9984 | 0.0010 |
| | 100 | 0.1464 | 0.9738 | 0.0027 |

## 6. Discussion

**Lessons learned.** Overall, our study shows though UNRL is a promising technique to ease the development and deployment of GSA (e.g., by training without labeled malicious events and learning structural features without manual feature engineering), the current implementations still incur a high error rate, especially on realistic datasets collected by large organizations. We argue that the UNRL-powered GSA cannot be completely agnostic to the characteristics and distribution of the attack events. We also found some metrics can be misleading when the evaluation is carried out on the security datasets which are noisy and imbalanced (e.g., AUC is only presented by Log2Vec [11] for the evaluation on LANL, though it is over 0.9), and we suggest the metrics should be selected more carefully.

**Concept drift.** After the UNRL solutions, including AR-GUS, are deployed, the distribution of the normal behaviors in the testing data can be shifted away from the training data, which could increase FPR over time. This issue is called concept drift, and to combat this issue for security applications, recently, a few approaches like CADE [63], TRANSCENDENT [64] and OWAD [65] were proposed to detect concept drift, explain the drift, and retrain the model when necessary. In fact, OWAD has been tested on the same LANL dataset, and it shows the AUC can be increased by up to 0.07 (with GL-LV/GV as the anomaly detection model [10]). We expect similar results can be observed on ARGUS.

**Alignment of log files.** Due to the inaccuracy of computers' clocks and delays between when an event happens and gets recorded, the timestamps of logs are not always accurate. When merging multiple datasets like LANL-auth and LANL-flows, such "time skew" issue can cause the logs to be misaligned after the merge. This issue was observed by the creator of the LANL dataset and partially addressed by using the centralized logging server to write the timestamps [66]. Since ARGUS merges events in a snapshot of 360s to 3600s, the impact of time skew is further limited.

**Limitations and future work.** Here we list the limitations of ARGUS currently and outline the plan for future improvement. 1) Though ARGUS is able to outperform Euler, the SOTA method handling DTG, by a large margin when evaluating on LANL (e.g., 0.32 over 0.04 in AP), its performance should still be improved for a production environment. For instance, its precision is only 0.22 when recall reaches 0.8. One direction we will explore is generating different embeddings when a node is presented as a source node and context (or destination) node, as they have different meanings in the security setting (e.g., the attacker compromises a source node to attack a destination node that is not yet compromised). Yet, deep-learning-based encoders like GCN do not inherently support this setting [12], and a new encoder design is needed. On the other hand, as explained in Section 5.2, there are data labeling issues in LANL, which makes it nearly impossible to achieve high precision and high recall. 2) We use a subset of LANL for evaluation because the collection of network flows encountered issues after 14 days. Such collection issues are likely to happen in many organizations, and the trained model should be able to handle data corruption or missing when it is deployed. One solution could be synthesizing the replacement data and the approaches based on neural data augmentation can be employed [67]. 3) ARGUS is able to capture the abnormal behaviors on DTG, and is potentially applicable to the precise detection of lateral movement. However, we acknowledge that there exist more advanced attack techniques like ShadowMove [68] that can hide the malicious activities completely into legitimate communications, which will not introduce any new links. In this case, ARGUS might be evaded and other information about the hosts needs to be leveraged. 4) ARGUS's operation requires parameter tuning and we found different parameters are needed when the

datasets are different (e.g., LANL and OpTC). This might be inevitable given that even their embodied attack types are different. Our ablation study also shows not all components are effective for LANL and OpTC (e.g., replacing the default GRU with LSTM increases AP by about 0.08, shown in Table 10). How to infer the best parameters and components automatically from a sample of data can be an interesting direction to explore. 5) Since the encoder ARGUS relies on GCN, it can only conduct transductive learning [12] (explained in Section 2.2). Inductive learning might be necessary when the nodes in an organization frequently change, which requires a new encoder design. 6) Euler has better efficiency comparing to ARGUS, due to ARGUS uses a more complex encoder. Though we argue that improving detection accuracy is more important when the accuracy is low, we believe ARGUS can be optimized to be more efficient, e.g., by leveraging parallel workers like Euler.

## 7. Related Work

**Graph Learning for Security.** In Section 2.1, we have reviewed the studies that apply UNRL for security. Here we review other researches that apply graph learning for security analytics, which can be characterized into three directions:

1) *Graph traversal and tracking*: to identify the root cause of a security incident (e.g., data breach) on a graph that models network-level or host-level activities [69], provenance tracking [70], [71] has been proposed and shown promising results in tracing Advanced Persistent Threat (APT) [1]. Initially, provenance-aware systems [69] are designed to take queries (e.g., SQL) from analysts, compress/summarize matched graphs (e.g., through event abstraction [3], [4], execution partition [72], [73] and graph summation [74]), and return the results to analysts for further queries or attack confirmation. Recently, some works have investigated *automated* detection of the threats on the provenance graph, and the methods include tag propagation to label nodes [75], [76], sequence learning to identify the attack paths [5], attack attribution [6], and attack subgraph extraction [77]–[80].

2) *Graph isomorphism and similarity testing*: it is critical to find instances of malicious patterns in graphs of the monitored data. Both traditional graph isomorphism algorithms [79], [81] and graph learning approaches [82], [83] have been proposed, as well as methods to help derive patterns of interest in searching [84].

3) *Graph mining*: turning graph elements and structures into vectors enables neural networks to conduct classification and anomaly detection tasks such as malware detection [85]–[87], sybil detection [88]–[90], malicious website detection [91], [92]. Early works examined approaches based on message passing, like loopy belief propagation (LBP) and random walk (RW) for collective classification [93], which aims to classify the unlabeled nodes as positive or negative *simultaneously* with the help of the labeled nodes on the *same*

graph. Most of the previous works assign edge weights empirically, and Wang et al. proposed to learn the edge weights through an optimization-based approach [94]. UNRL was recently leveraged to build security analytics models in an unsupervised manner. And our study aims to largely improve its detection accuracy for practical use in production environments.

**Temporal Graph.** The security research that works on temporal graphs has been reviewed in Section 2.3. In applications other than security, temporal link prediction has been applied to the traffic prediction [95], fraud detection [96], etc., which typically combine GCN and RNN [20], [97], [98]. Singer proposed to use joint optimization to adapt static node embedding to different time points [99]. Beladev et al. learned graph representation per time step in the temporal graphs sequence [100]. Xu et al. proposed the temporal graph attention (TGAT) layer to efficiently aggregate temporal-topological neighborhood features [101]. Kumar et al. proposed JODIE which employs two RNNs to update the embedding and model the future interactions between nodes (e.g., users and items) [102]. Zheng et al. proposed TSNet to jointly learn temporal and structural feature, but the focus is on node classification [103]. Wen et al. proposed TREND which is built upon a Hawkes process-based GNN to achieve inductive learning on temporal graphs. [104] The log data handled by security analytics present unique challenges, e.g., large-scale and unbalanced distribution between benign and malicious events, and we design ARGUS to tackle them.

## 8. Conclusion

In this paper, we proposed ARGUS, a graph security analytics (GSA) that embodies unsupervised network representation learning (UNRL) to detect attacks from large-scale log datasets at high accuracy. Carefully designed to incorporate security domain knowledge in its components like encoder, ARGUS steps up and goes beyond generic UNRL to better model security graphs for attack detection and other security applications. Evaluated on two large-scale datasets, LANL and OpTC, ARGUS shows a clear advantage over other state-of-the-art GSAs like Euler by a large margin in average precision. With the development of ARGUS, we hope to advance practical GSA deployment in enterprises with manageable false positives to combat cyber-attacks.

## Acknowledgement

# References

[1] Lockheed Martin, "Cyber kill chain," https://www.lockheedmartin.com/en-us/capabilities/cyber/cyber-kill-chain.html, 2019.

[2] Forcepoint, "What is security analytics?" https://www.forcepoint.com/cyber-edu/security-analytics, 2021.

[3] W. U. Hassan, A. Bates, and D. Marino, "Tactical provenance analysis for endpoint detection and response systems," in *Proceedings of the IEEE Symposium on Security and Privacy*, 2020.

[4] L. Yu, S. Ma, Z. Zhang, G. Tao, X. Zhang, D. Xu, V. E. Urias, H. W. Lin, G. Ciocarlie, V. Yegneswaran *et al.*, "Alchemist: Fusing application and audit logs for precise attack provenance without instrumentation," in *NDSS*, 2021.

[5] A. Alsaheel, Y. Nan, S. Ma, L. Yu, G. Walkup, Z. B. Celik, X. Zhang, and D. Xu, "{ATLAS}: A sequence-based learning approach for attack investigation," in *30th {USENIX} Security Symposium ({USENIX} Security 21)*, 2021.

[6] Y. Liu, M. Zhang, D. Li, K. Jee, Z. Li, Z. Wu, J. Rhee, and P. Mittal, "Towards a timely causality analysis for enterprise security," in *Proceedings of the 25th Network and Distributed System Security Symposium (NDSS)*, 2018.

[7] IBM, "Theory behind kestrel?" https://kestrel.readthedocs.io/en/latest/theory.html, 2021.

[8] NEC, "Automated security intelligence (asi) with auto detection of unknown cyber-attacks," https://www.nec.com/en/global/techrep/journal/g16/n01/160110.html, 2016.

[9] MITRE, "Lateral movement, tactic ta0008," https://attack.mitre.org/tactics/TA0008/, 2021.

[10] B. Bowman, C. Laprade, Y. Ji, and H. H. Huang, "Detecting lateral movement in enterprise computer networks with unsupervised graph AI," in *23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2020)*, 2020, pp. 257–268.

[11] F. Liu, Y. Wen, D. Zhang, X. Jiang, X. Xing, and D. Meng, "Log2vec: A heterogeneous graph embedding based approach for detecting cyber threats within enterprise," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 1777–1794.

[12] M. Khosla, V. Setty, and A. Anand, "A comparative study for unsupervised network representation learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 33, no. 5, pp. 1807–1818, 2019.

[13] T. N. Kipf and M. Welling, "Variational graph auto-encoders," *arXiv preprint arXiv:1611.07308*, 2016.

[14] I. J. King and H. H. Huang, "Euler: Detecting network lateral movement via scalable temporal link prediction," in *Proceedings of the 29th Network and Distributed System Security Symposium (NDSS). The Internet Society, San Diego, California, USA*, 2022.

[15] R. Paudel and H. H. Huang, "Pikachu: Temporal walk based dynamic graph embedding for network anomaly detection," in *NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2022, pp. 1–7.

[16] J. Zengy, X. Wang, J. Liu, Y. Chen, Z. Liang, T.-S. Chua, and Z. L. Chua, "Shadewatcher: Recommendation-guided cyber threat analysis using system audit records," in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022, pp. 489–506.

[17] C. E. Priebe, J. M. Conroy, D. J. Marchette, and Y. Park, "Scan statistics on enron graphs," *Computational & Mathematical Organization Theory*, vol. 11, pp. 229–247, 2005.

[18] M. Rahman and M. A. Hasan, "Link prediction in dynamic networks using graphlet," in *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2016, Riva del Garda, Italy, September 19-23, 2016, Proceedings, Part I 16*. Springer, 2016, pp. 394–409.

[19] B. Viswanath, A. Mislove, M. Cha, and K. P. Gummadi, "On the evolution of user interaction in facebook," in *Proceedings of the 2nd ACM workshop on Online social networks*, 2009, pp. 37–42.

[20] E. Hajiramezanali, A. Hasanzadeh, K. Narayanan, N. Duffield, M. Zhou, and X. Qian, "Variational graph recurrent neural networks," *Advances in Neural Information Processing Systems*, vol. 32, pp. 10 701–10 711, 2019.

[21] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *International conference on machine learning*. PMLR, 2017, pp. 1263–1272.

[22] L. Ouyang, Y. Zhang, and Y. Wang, "Unified graph embedding-based anomalous edge detection," in *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2020, pp. 1–8.

[23] Q. Qi, Y. Luo, Z. Xu, S. Ji, and T. Yang, "Stochastic optimization of areas under precision-recall curves with provable convergence," *Advances in Neural Information Processing Systems*, vol. 34, pp. 1752–1765, 2021.

[24] A. D. Kent, "Comprehensive, Multi-Source Cyber-Security Events," Los Alamos National Laboratory, 2015.

[25] Five Directions, "Operationally transparent cyber (optc) data release," https://github.com/FiveDirections/OpTC-data, 2020.

[26] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. v. d. Berg, I. Titov, and M. Welling, "Modeling relational data with graph convolutional networks," in *European semantic web conference*. Springer, 2018, pp. 593–607.

[27] R. v. d. Berg, T. N. Kipf, and M. Welling, "Graph convolutional matrix completion," *arXiv preprint arXiv:1706.02263*, 2017.

[28] 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*, 2016, pp. 855–864.

[29] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2014, pp. 701–710.

[30] J. Qiu, Y. Dong, H. Ma, J. Li, K. Wang, and J. Tang, "Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec," in *Proceedings of the eleventh ACM international conference on web search and data mining*, 2018, pp. 459–467.

[31] M. Ou, P. Cui, J. Pei, Z. Zhang, and W. Zhu, "Asymmetric transitivity preserving graph embedding," in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, 2016, pp. 1105–1114.

[32] Z. Zhang, P. Cui, and W. Zhu, "Deep learning on graphs: A survey," *IEEE Transactions on Knowledge and Data Engineering*, 2020.

[33] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[34] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 2017, pp. 1025–1035.

[35] W. Yu, W. Cheng, C. C. Aggarwal, K. Zhang, H. Chen, and W. Wang, "Netwalk: A flexible deep embedding approach for anomaly detection in dynamic networks," in *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, 2018, pp. 2672–2681.

[36] Q. Cheng, Y. Shen, D. Kong, and C. Wu, "Step: Spatial-temporal network security event prediction," *arXiv preprint arXiv:2105.14932*, 2021.

[37] L. Cai, Z. Chen, C. Luo, J. Gui, J. Ni, D. Li, and H. Chen, "Structural temporal graph neural networks for anomaly detection in dynamic graphs," in *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, 2021, pp. 3747–3756.

[38] Redhat, "Chapter 7. system auditing," https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/security_guide/chap-system_auditing, 2022.

[39] Microsoft, "Sysmon - windows sysinternals," https://docs.microsoft.com/en-us/sysinternals/downloads/sysmon, 2022.

[40] ——, "Event tracing for windows (etw)," https://docs.microsoft.com/en-us/windows-hardware/drivers/devtest/event-tracing-for-windows--etw-, 2022.

[41] VMware, "Vmware carbon black endpoint," https://www.vmware.com/products/carbon-black-cloud-endpoint.html, 2022.

[42] CrowdStrike, "Crowdstrike: Stop breaches. drive business." https://www.crowdstrike.com/, 2022.

[43] DARPA, "Transparent computing (archived)," https://www.darpa.mil/program/transparent-computing, 2014.

[44] G. Ho, M. Dhiman, D. Akhawe, V. Paxson, S. Savage, G. M. Voelker, and D. Wagner, "Hopper: Modeling and detecting lateral movement," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021.

[45] iHeartGraph, "Euler source code," https://github.com/iHeartGraph/Euler, 2022.

[46] PyTorch Geometric, "TORCH_GEOMETRIC.NN GCNConv," https://pytorch-geometric.readthedocs.io/en/latest/modules/nn.html#torch_geometric.nn.conv.GCNConv, 2022.

[47] ——, "TORCH_GEOMETRIC.NN SAGEConv," https://pytorch-geometric.readthedocs.io/en/latest/modules/nn.html#torch_geometric.nn.conv.SAGEConv, 2022.

[48] A. Bates, D. J. Tian, K. R. Butler, and T. Moyer, "Trustworthy {Whole-System} provenance for the linux kernel," in *24th USENIX Security Symposium (USENIX Security 15)*, 2015, pp. 319–334.

[49] R. Paccagnella, P. Datta, W. U. Hassan, A. Bates, C. Fletcher, A. Miller, and D. Tian, "Custos: Practical tamper-evident auditing of operating systems using trusted execution," in *Network and distributed system security symposium*, 2020.

[50] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," *Advances in neural information processing systems*, vol. 26, 2013.

[51] Z. Yuan, Z.-H. Qiu, G. Li, D. Zhu, Z. Guo, Q. Hu, B. Wang, Q. Qi, Y. Zhong, and T. Yang, "Libauc: A deep learning library for x-risk optimization," 2022.

[52] E. Quiring, F. Pendlebury, A. Warnecke, F. Pierazzi, C. Wressnegger, L. Cavallaro, and K. Rieck, "Dos and don'ts of machine learning in computer security," in *31st USENIX Security Symposium (USENIX Security 22), USENIX Association, Boston, MA*, 2022.

[53] sklearn, "sklearn.metrics.average_precision_score," https://scikit-learn.org/stable/modules/generated/sklearn.metrics.average_precision_score.html, 2022.

[54] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "Lof: identifying density-based local outliers," in *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, 2000, pp. 93–104.

[55] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *2008 eighth ieee international conference on data mining*. IEEE, 2008, pp. 413–422.

[56] Microsoft, "NTLM user authentication," https://docs.microsoft.com/en-us/troubleshoot/windows-server/windows-security/ntlm-user-authentication, 2021.

[57] A. D. Kent, "Cybersecurity Data Sources for Dynamic Network Research," in *Dynamic Networks in Cybersecurity*. Imperial College Press, Jun. 2015.

[58] rpaudel42, "Pikachu source code," https://github.com/rpaudel42/Pikachu, 2022.

[59] X. Liu, S. Si, X. Zhu, Y. Li, and C.-J. Hsieh, "A unified framework for data poisoning attack to graph-based semi-supervised learning," *arXiv preprint arXiv:1910.14147*, 2019.

[60] X. Xu, Q. Hao, Z. Yang, B. Li, D. Liebovitz, G. Wang, and C. Gunter, "How to cover up anomalous accesses to electronic health records," in *32nd {USENIX} Security Symposium ({USENIX} Security 23)*, 2023.

[61] B. Wang and N. Z. Gong, "Attacking graph-based classification via manipulating the graph structure," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 2023–2040.

[62] X. Han, T. Pasquier, A. Bates, J. Mickens, and M. Seltzer, "Unicorn: Runtime provenance-based detector for advanced persistent threats," in *NDSS*, 2020.

[63] L. Yang, W. Guo, Q. Hao, A. Ciptadi, A. Ahmadzadeh, X. Xing, and G. Wang, "Cade: Detecting and explaining concept drift samples for security applications." in *USENIX security symposium*, 2021, pp. 2327–2344.

[64] F. Barbero, F. Pendlebury, F. Pierazzi, and L. Cavallaro, "Transcending transcend: Revisiting malware classification in the presence of concept drift," in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022, pp. 805–823.

[65] D. Han, Z. Wang, W. Chen, K. Wang, R. Yu, S. Wang, H. Zhang, Z. Wang, M. Jin, J. Yang, X. Shi, and X. Yin, "Anomaly detection in the open world: Normality shift detection, explanation, and adaptation," in *NDSS*, 2023.

[66] N. M. Adams and N. A. Heard, *Dynamic networks and cybersecurity*. World Scientific, 2016, vol. 1.

[67] S. T. Jan, Q. Hao, T. Hu, J. Pu, S. Oswal, G. Wang, and B. Viswanath, "Throwing darts in the dark? detecting bots with limited data using neural data augmentation," in *2020 IEEE symposium on security and privacy (SP)*. IEEE, 2020, pp. 1190–1206.

[68] A. Niakanlahiji, J. Wei, M. R. Alam, Q. Wang, and B.-T. Chu, "Shadowmove: A stealthy lateral movement strategy," in *29th USENIX Security Symposium (USENIX Security 20)*, 2020, pp. 559–576.

[69] A. Bates and W. U. Hassan, "Can data provenance put an end to the data breach?" *IEEE Security & Privacy*, vol. 17, no. 4, pp. 88–93, 2019.

[70] S. T. King and P. M. Chen, "Backtracking intrusions," *ACM Transactions on Computer Systems (TOCS)*, vol. 23, no. 1, pp. 51–76, 2005.

[71] X. Jiang, A. Walters, D. Xu, E. H. Spafford, F. Buchholz, and Y.-M. Wang, "Provenance-aware tracing ofworm break-in and contaminations: A process coloring approach," in *Distributed Computing Systems, 2006. ICDCS 2006. 26th IEEE International Conference on*. IEEE, 2006, pp. 38–38.

[72] W. U. Hassan, M. A. Noureddine, P. Datta, and A. Bates, "Omegalog: High-fidelity attack investigation via transparent multi-layer log analysis," in *Proc. NDSS*, 2020.

[73] Y. Kwon, W. Wang, J. Jung, K. H. Lee, and R. Perdisci, "C2sr: Cybercrime scene reconstruction for post-mortem forensic analysis," in *Network and Distributed Systems Security (NDSS) Symposium*, 2021.

[74] Z. Xu, P. Fang, C. Liu, X. Xiao, Y. Wen, and D. Meng, "Depcomm: Graph summarization on system audit logs for attack investigation," in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022, pp. 540–557.

[75] S. M. Milajerdi, R. Gjomemo, B. Eshete, R. Sekar, and V. Venkatakrishnan, "Holmes: real-time apt detection through correlation of suspicious information flows," in *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019, pp. 1137–1152.

[76] M. N. Hossain, S. Sheikhi, and R. Sekar, "Combating dependence explosion in forensic analysis using alternative tag propagation semantics," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 1139–1155.

[77] Y. Kwon, F. Wang, W. Wang, K. H. Lee, W.-C. Lee, S. Ma, X. Zhang, D. Xu, S. Jha, G. F. Ciocarlie *et al.*, "Mci: Modeling-based causality inference in audit logging for attack investigation." in *NDSS*, 2018.

[78] S. M. Milajerdi, B. Eshete, R. Gjomemo, and V. 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*, 2019, pp. 1795–1812.

[79] X. Shu, F. Araujo, D. L. Schales, M. P. Stoecklin, J. Jang, H. Huang, and J. R. Rao, "Threat intelligence computing," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 1883–1898.

[80] P. Fang, P. Gao, C. Liu, E. Ayday, K. Jee, T. Wang, Y. F. Ye, Z. Liu, and X. Xiao, "Back-propagating system dependency impact for attack investigation," in *31st USENIX Security Symposium (USENIX Security), Boston, MA*, 2022, pp. 10–12.

[81] L. Cordella, P. Foggia, C. Sansone, and M. Vento, "A (sub)graph isomorphism algorithm for matching large graphs," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 10, pp. 1367–1372, 2004.

[82] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" *arXiv preprint arXiv:1810.00826*, 2018.

[83] S. Wang, Z. Chen, X. Yu, D. Li, J. Ni, L.-A. Tang, J. Gui, Z. Li, H. Chen, and P. S. Yu, "Heterogeneous graph matching networks for unknown malware detection," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. International Joint Conferences on Artificial Intelligence Organization, 7 2019, pp. 3762–3770. [Online]. Available: https://doi.org/10.24963/ijcai.2019/522

[84] B. Zong, X. Xiao, Z. Li, Z. Wu, Z. Qian, X. Yan, A. K. Singh, and G. Jiang, "Behavior query discovery in system-generated temporal graphs," *Proc. VLDB Endow.*, vol. 9, no. 4, p. 240–251, dec 2015.

[85] G. Stringhini, Y. Shen, Y. Han, and X. Zhang, "Marmite: spreading malicious file reputation through download graphs," in *Proceedings of the 33rd Annual Computer Security Applications Conference*, 2017, pp. 91–102.

[86] M. A. Rajab, L. Ballard, N. Lutz, P. Mavrommatis, and N. Provos, "Camp: Content-agnostic malware protection," in *NDSS*, 2013.

[87] A. Tamersoy, K. Roundy, and D. H. Chau, "Guilt by association: large scale malware detection by mining file-relation graphs," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2014, pp. 1524–1533.

[88] J. Jia, B. Wang, and N. Z. Gong, "Random walk based fake account detection in online social networks," in *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2017, pp. 273–284.

[89] H. Zheng, M. Xue, H. Lu, S. Hao, H. Zhu, X. Liang, and K. Ross, "Smoke screener or straight shooter: Detecting elite sybil attacks in user-review social networks," in *NDSS*, 2018.

[90] D. Yuan, Y. Miao, N. Z. Gong, Z. Yang, Q. Li, D. Song, Q. Wang, and X. Liang, "Detecting fake accounts in online social networks at the time of registrations," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 1423–1438.

[91] Z. Gyongyi, H. Garcia-Molina, and J. Pedersen, "Combating web spam with trustrank," in *Proceedings of the 30th international conference on very large data bases (VLDB)*, 2004.

[92] Z. Li, S. Alrwais, Y. Xie, F. Yu, and X. Wang, "Finding the linchpins of the dark web: a study on topologically dedicated hosts on malicious web infrastructures," in *2013 IEEE Symposium on Security and Privacy*. IEEE, 2013, pp. 112–126.

[93] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad, "Collective classification in network data," *AI magazine*, vol. 29, no. 3, pp. 93–93, 2008.

[94] B. Wang, J. Jia, and N. Z. Gong, "Graph-based security and privacy analytics via collective classification with joint weight learning and propagation," in *NDSS*, 2019.

[95] L. Zhao, Y. Song, C. Zhang, Y. Liu, P. Wang, T. Lin, M. Deng, and H. Li, "T-gcn: A temporal graph convolutional network for traffic prediction," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 9, pp. 3848–3858, 2019.

[96] D. Cheng, X. Wang, Y. Zhang, and L. Zhang, "Graph neural network for fraud detection via spatial-temporal attention," *IEEE Transactions on Knowledge and Data Engineering*, 2020.

[97] A. Pareja, G. Domeniconi, J. Chen, T. Ma, T. Suzumura, H. Kanezashi, T. Kaler, T. Schardl, and C. Leiserson, "Evolvegcn: Evolving graph convolutional networks for dynamic graphs," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 04, 2020, pp. 5363–5370.

[98] Y. Seo, M. Defferrard, P. Vandergheynst, and X. Bresson, "Structured sequence modeling with graph convolutional recurrent networks," in *International Conference on Neural Information Processing*. Springer, 2018, pp. 362–373.

[99] U. Singer, I. Guy, and K. Radinsky, "Node embedding over temporal graphs," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI)*, S. Kraus, Ed., 2019.

[100] M. Beladev, L. Rokach, G. Katz, I. Guy, and K. Radinsky, "td-graphembed: Temporal dynamic graph-level embedding," in *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, 2020, pp. 55–64.

[101] D. Xu, C. Ruan, E. Körpeoglu, S. Kumar, and K. Achan, "Inductive representation learning on temporal graphs," in *8th International Conference on Learning Representations (ICLR)*, 2020.

[102] S. Kumar, X. Zhang, and J. Leskovec, "Predicting dynamic embedding trajectory in temporal interaction networks," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019, pp. 1269–1278.

[103] C. Zheng, B. Zong, W. Cheng, D. Song, J. Ni, W. Yu, H. Chen, and W. Wang, "Node classification in temporal graphs through stochastic sparsification and temporal structural convolution," in *Proceedings of European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML-PKDD)*, September 2020.

[104] Z. Wen and Y. Fang, "Trend: Temporal event and node dynamics for graph representation learning," in *Proceedings of the ACM Web Conference 2022*, 2022, pp. 1159–1169.

[105] NEC Labs, "Netwalk github," https://github.com/chengw07/NetWalk, 2018.

[106] PyTorch, "PyTorch Distributed Overview," https://pytorch.org/tutorials/beginner/dist_overview.html, 2022.

# Appendix A.

## A.1. Details of the Other Security Analytics

In Section 3.1, we listed the effectiveness of 4 UNRL-based GSAs on LANL. Here we provide details about their designs and the parameters we set for the comparative study.

- **Euler [14]** uses GCN as encoder to generate node embedding on individual snapshot and GRU to accommodate temporal dynamics. Its decoder is the inner product decoder as shown in Equation 2. The loss function is a standard CE loss over the ground-truth

TABLE 12. THE COMPARISON BETWEEN ARGUS AND OTHER GSAS. AUC, AP AND FPR ARE MEASURED ON LANL.

| System | DTG | Edge Feature | Encoder | Loss | RNN | AUC[1] | AP[1] | FPR[1] |
|---|---|---|---|---|---|---|---|---|
| GL-LV/GV [10] | × | × | Continuous-Bag-of-Words | Noise Contrastive Estimation | × | - | - | 0.0090 |
| Log2Vec [11] | × | × | Word2vec | CE | × | 0.91 | - | - |
| Netwalk [35] | × | × | Network walk Autoencoder | Clique Embedding Loss | × | 0.7195 | 0.0029 | 0.1908 |
| PIKACHU [15] | ✓ | × | GRU-based Autoencoder | CE | GRU | 0.9517 | 0.0038 | 0.0413 |
| VGRNN [20] | ✓ | × | GCN | Variational Lower Bound | GRNN | 0.9660 | 0.0125 | 0.0036 |
| Euler [14] | ✓ | × | GCN | CE | GRU | 0.9810 | 0.0448 | 0.0045 |
| **ARGUS** | ✓ | ✓ | MPNN | AP + Community | GRU | 0.9983 | 0.3227 | 0.0005 |

[1] The results for GL-LV/GV and Log2Vec are obtained from the papers, while the remaining results were generated after we run their code.

and reconstructed adjacency matrices. Euler is able to distribute the DTG snapshots into multiple workers for parallel processing, under its scalability goal. We use the code from [45] and use its default parameters (learning rate = 0.01, threshold weight = 0.6).

- **PIKACHU [15]** splits the graphs into sequences with random walk and uses Skip-gram to train the node embedding. A GRU-based autoencoder is used to learn long-term information. We use the default parameters listed in the paper [15] (embedding dimension=64, walk length=500, context size for training=5, learning rate=0.001, neighbor number=10).
- **Log2Vec** [11] proposes 10 rules to build heterogeneous graphs from different types of logs and uses random walk and word2vec to generate node embeddings and detect abnormal edges.
- **GL-LV/GV** [10] aims to detect lateral movement from authentication logs. It uses random walk and a Contiuous-Bag-Of-Words model to generate node embeddings and identifies low-probability authentication events via a learned logistic regression link predictor.

In Section 5, we include VGRNN and Netwalk as baseline models in addition to Euler and PIKACHU, and we provide details about them below. EvolveGCN [97] is another model that performs temporal link prediction but it exceeds the limit of our CPU memory when running on LANL, so it was not selected as a baseline. In Table 12 we summarize the aforementioned GSAs.

- **VGRNN [20]** uses GCN as the encoder for a DTG snapshot and stacks it up on a GC-LSTM to capture the temporal relations. It was also compared by Euler [14] and we use the same model parameters (learning rate=0.01, hidden embedding size=32).
- **Netwalk [35]** performs network walk (a variation of random walk) to generate node embedding, and uses a reservoir sampling strategy to update the embedding. A newly arrived nodes or edges that do not belong to any cluster will be considered as anomalies. We use the code from [105], and use its default parameters (learning rate=0.01, epoch=30, number of clusters=3).

Except GSAs, we also include two other non-GSA methods as baseline models.

- **Local Outlier Factor (LOF) [54]** is a non-graph anomaly detection approach that extracts relative local densities from a sample and compare the sample with its neighbors for anomaly detection. To notice, this

method was compared with GL-LV/GV [10] and 1-hot encoding was applied on the authentication logs to generate features. In this paper, we use the same edge features extracted by ARGUS instead. The parameters set by [10] are used (neighbors=2).

- **Isolation Forest (IF) [55]** is another non-graph model compared with GL-LV/GV [10]. It uses simple decision trees to identify the abnormal samples. We use the default parameter (random state=0).

## A.2. More Evaluation Results

**Results by snapshots on LANL.** If ARGUS is deployed as a near real-time defense that issue alerts to the security analysts after each snapshot, it should avoid triggering too many FPs in one snapshot. In other words, "smoothness" should be achieved on the detection results. Hence, we log the TP, FP, and FN triggered by ARGUS in each snapshot (in total 311 snapshots) and show them in Figure 6. We found none of the snapshots would trigger more than 39 FP. In most cases (280 out of 295 snapshots), less than 10 FP is triggered. In some snapshots, the attack activities are intensive (e.g., 38 malicious edges observed in snapshot 171), and ARGUS is able to capture most of them (33 TP captured).

In Figure 7, we draw the number of edges observed in each snapshot, and we found there exists a vast divergence among snapshots (the number ranges from 5,836 to 16,352). High FP and FN also have strong relations with the edge numbers. On the other hand, clear temporal patterns are observed across snapshots: weekends and outside-of-working hours see lower edge numbers. By modeling such patterns during training (e.g., adding snapshot weights to RNN based on their time windows), ARGUS might achieve better results.

Then, we compute the detection statistics by days and found the average FP per day is 85, suggesting the extra workload for analysts is moderate.

**Case study of OpTC.** Here we demonstrate how ARGUS is able to pinpoint the attack traces at finer grain based on DTG modeling. We select the snapshots that are related to the simulated "Plain PowerShell Empire" campaign by the redteam, as documented in [25]. The attackers first connected to a host to download the attack tool "PowerShell Empire stager", and then launched privilege escalation to collect user credentials by Mimikatz. After that they conducted port scanning, and set their foothold on another host with WMI.
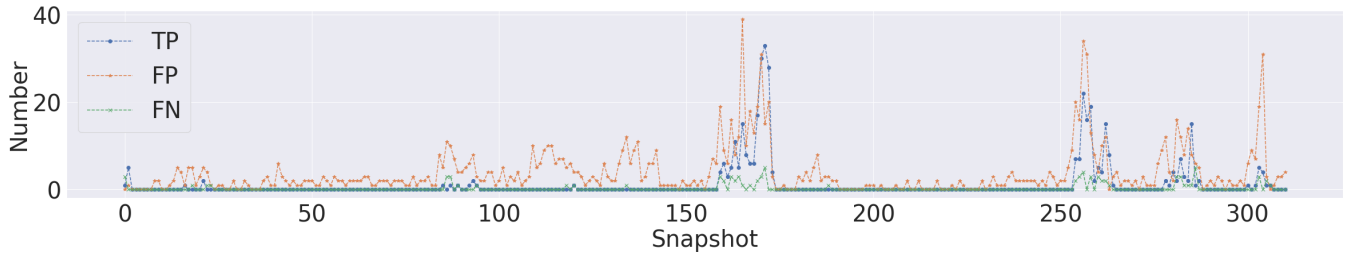
Figure 6. TP, FP, FN of each LANL snapshot (1 hour). Snapshot is numbered by their timing order. TN is not drawn because it is much larger than TP, FP and FN, and it is close to the number of edges.
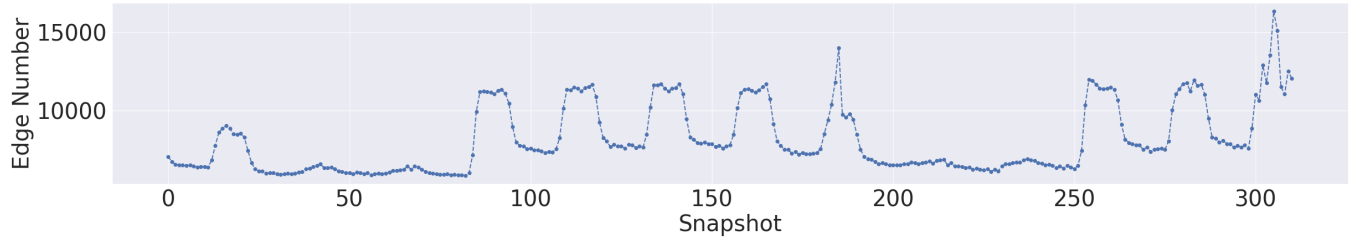


Figure 7. Number of edges of each LANL snapshot (1 hour). Snapshot is numbered by their timing order.
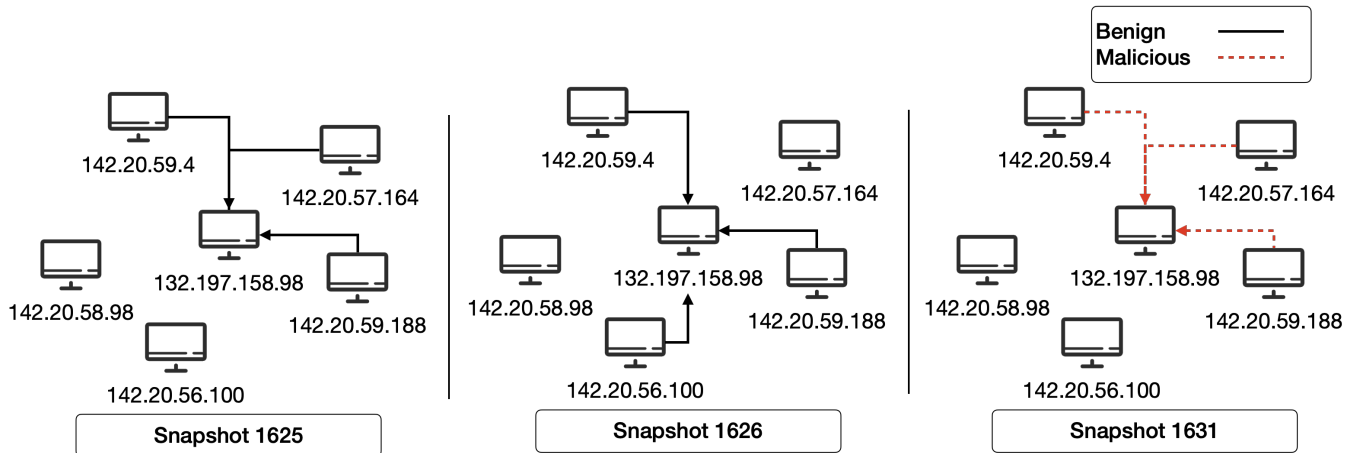


Figure 8. Case study of "Plain PowerShell Empire" redteam attack in OpTC. Lines in red dash represent the redteam events.

Finally, they are able to reach the domain controller and harvest users' password hashes with lsadump.

Figure 8 draws three representative snapshots. `132.197.158.98` is the C2 (Command and Control) server that keeps the attack tool. It was connected by clients like `142.20.59.4` and `142.20.59.188` in snapshot 1625 and 1626. However, these connections happened before the first redteam event (attack tool downloading), which suggests these connections could be benign (e.g., the C2 server was not fully owned by the attacker and the clients viewed the normal content). ARGUS is able to correctly detect the attack events in snapshot 1631 that matches timestamps of the redteam events. If we choose static graph modeling rather than DTG, it will incur high efforts for the analysts to locate the attack events.

## A.3. Efficiency

Here we compare the training and testing efficiency of ARGUS to Euler. Euler is designed to process snapshots in parallel, which leveraged the distributed framework of PyTorch [106]. We are more interested in how Euler and ARGUS performs at the algorithmic level, so we we set the number of workers and threads of Euler both to 1. We focus on the overhead of the machine-learning components and leave out the latency caused by data loading. We tested on the LANL dataset with 3,600 seconds snapshot size, and Euler's training time is 164.86 seconds while the testing time is 77.09 seconds. For ARGUS, if we use CPU for both as a fair comparison (Euler's source code uses CPU), because the overhead of processing edge features, ARGUS runs slower than Euler with the training and testing time at 636.55 and 81.74 seconds. To notice, 42 hours and 12

3607

TABLE 13. TRAINING AND TESTING TIME OF ARGUS UNDER DIFFERENT SNAPSHOT SIZES FOR LANL. FOR GPU-IMPLEMENTATION OF ARGUS, WE USE THE CROSS-ENTROPY (CE) LOSS.

| Snapshot (s) | Training (s) | Testing (s) |
|---|---|---|
| 3,600 (Euler) | 164.86 | 77.09 |
| 3,600 (GPU) | **5.81** | 97.04 |
| 3,600 (CPU) | 636.55 | 81.14 |
| 1,800 (CPU) | 800.48 | 90.50 |
| 900 (CPU) | 933.71 | 238.83 |
| 360 (CPU) | 2,141.5 | 445.35 |

days LANL data are used for training and testing, and we believe the overhead is acceptance under such data volume. If we run ARGUS on GPU, only 5.8 seconds are needed for training, and the testing time is slightly increased because all the data need to be loaded to the GPU from CPU memory. To notice, due to the memory limitation of our GPU (24 GB), supporting the full loss of Equation 14 runs out of memory. Hence, we change the loss to cross-entropy (CE) loss, which reaches 0.2676 AP and is still higher than Euler.

To assess the impact of snapshot size, we vary the size from 3,600 to 1,800, 900, 360 seconds. The result is listed in Table 13, which shows the snapshot size has a big impact on the training time and testing time, as more than 3x slowdown was observed when reducing the snapshot size from 3,600 seconds to 360 seconds.

# Appendix B.
# Meta-review

## B.1. Summary of Paper

This paper aims to address performance issues encountered by unsupervised network representation learning techniques, which stem mainly from the attack-agnostic learning objective and other factors that are specific to graphical data. The paper proposes ARGUS, which incorporates domain knowledge about security analysis into learning. Empirical results show that the approach offers superior performance to both graphical and non-graphical neural network baselines.

## B.2. Scientific Contributions

- Creates a New Tool to Enable Future Science
- Provides a Valuable Step Forward in an Established Field

## B.3. Reasons for Acceptance

1) A systematic analysis of prior approaches that sheds light on why they perform poorly on real-world data
2) Introduces a new unsupervised graphical security analysis method that builds on these findings
3) Illustrates how security domain knowledge can be incorporated into learning
4) Experimental results provide compelling evidence that supports the effectiveness of ARGUS's design