CAPTOR: Cyber Attack Protection via Temporal Online Graph Representation Learning

Bishal Lakha, Janet Layne, Edoardo Serra, Francesco Gullo, and Sushil Jajodia, *Fellow,IEEE*

Abstract—Online intrusion detection systems (IDSs), i.e., tools for detecting live cyberattacks in the form of unauthorized access gained to a (networking) system, play a role of paramount importance in the cybersecurity landscape. Among the plethora of existing IDSs, the ones based on temporal graph anomaly detection (TGAD) process a temporal graph representing entities of interest of the underlying system and time-varying relationships among them, and identify anomalous temporal edges in such a graph as potential intrusions. TGAD-based IDSs are superior to various other existing types of IDS for their peculiarities of high generality and powerfulness in data representation and types of cyberattack identifiable. However, existing TGAD-based IDSs are still far from being suitable for real-world settings, due to their severe limitations in efficiently and effectively handling the underlying temporal graphs, which are typically really big. In this paper, we devise CAPTOR ("Cyber Attack Protection via Temporal Online graph Representation learning"), a novel TGAD-based IDS which addresses the limitations of the state of the art. CAPTOR consists of a careful selection and clever combination of graph representation learning (GRL), TGAD, and temporal aggregation of GRL representations (embeddings). These design choices make CAPTOR achieve the best tradeoff between accuracy and scalability in TGAD-based intrusion detection, as testified by extensive experiments on real cybersecurity datasets. As such, with this work we take a significant step forward towards rendering the important TGAD-based IDS technology actually applicable in real-world cybersecurity scenarios.

Index Terms—cybersecurity, online intrusion detection systems, scalable data science, big graph processing, large-scale graph representation learning, temporal graph anomaly detection

1 Introduction

NOWADAYS, in the era of cyber defense, identifying *cyberattacks*[1] to computer systems has become of vital importance for enterprises, organizations, and governments [56]. If undetected, these attacks may result in significant damages, such as business interruption or privacy/data breaches, which ultimately cause severe financial losses and reputational harm [57].

Intrusion Detection Systems (IDSs). Identifying cyberattacks to enterprise networks is an interdisciplinary task which has attracted significant attention in the cybersecurity, data-management, data-science, and machine-learning scientific communities [7], [10], [12], [15], [16], [17], [18], [26], [35], [49], [69]. This has led to the proliferation of *Intrusion Detection Systems* (IDSs), i.e., automated methodologies to monitor/analyze *events* that occur within the target computer system (e.g., network traffic, authentications within machines of a network, running behavior of processes) with the goal of recognizing/predicting malicious activity [35], [69]. The IDS landscape is really vast. In Section 2, we elaborate on this slightly further, but for a detailed argumentation we defer to comprehensive surveys [2], [40], [69].

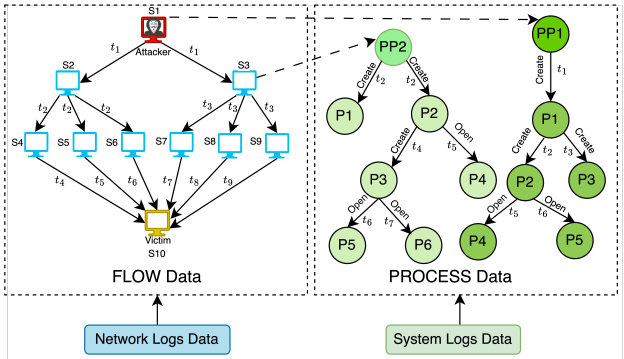


Fig. 1: Example of temporal graph processed by TGAD-based IDSs, extracted from the real DARPA OpTC dataset [3] (see Section 4). Edges are labeled with their timestamps of occurrence ("t_x") and, in the right panel, with the corresponding semantic action too. The overall temporal graph results from the integration of a FLOW graph (left) and a provenance graph (right). The FLOW graph is derived from network logs. It represents hosts as nodes, and interaction from a source host to a destination host as a (directed) edge. The provenance graph corresponds to process interaction data extracted from system logs. It depicts actions (edges) between two processes (nodes), i.e., from a parent process to a child process. The two graphs are integrated into one by adding edges from a host to the processes running on that host.

IDSs based on Temporal Graph Anomaly Detection (TGAD). Among the different categories of IDS, the ones based on *temporal graph anomaly detection* (TGAD) [5], [13], [33], [49], [65] represent events as a *temporal graph* and identify cyberattacks by performing TGAD [42] on it. Specifically, TGAD-based IDSs cast cyberattack detection into the problem of identifying *anomalous temporal edges* in a temporal graph that represents the events of interest in terms of entities (nodes) and time-varying relationships

- *B. Lakha, J. Layne and E. Serra are with Boise State University, Boise, ID, USA.*
  (e-mail: bishallakha@u.boisestate.edu; janetlayne@boisestate.edu; edoardoserra@boisestate.edu)
- *F. Gullo is with University of L'Aquila, L'Aquila, Italy.*
  (e-mail:francesco.gullo@univaq.it)
- *S. Jajodia is with George Mason University, Fairfax, VA, USA.*
  (e-mail:jajodia@gmu.edu)

1. Throughout this paper, we focus on a particular type of cyberattack consisting of intrusion activities on enterprise networking systems, specifically those considered in [3], [4], [31], [55]. For the sake of simplicity, with a little abuse of terminology, we refer to these collectively as "cyberattacks".

among them (temporal edges) [33], [49] (see Figure 1). A temporal edge is recognized as anomalous – and, as such, likely corresponding to (part of) a cyberattack – based on how unexpected/abnormal its presence is with respect to the usual/expected temporal graph structure. Events that are popularly exploited by TGAD-based IDSs come from *provenance data* [24], and their graph-like representation as *provenance graphs*, which usually encompass system entities as nodes, e.g., processes, files, sockets, threads, and interactions between those entities as edges, e.g., system calls, authentications, user events [52], [61], [64] (see Figure 1).

TGAD-based IDSs are generally preferable to other existing categories of IDS. This superiority has been widely assessed [5], [6], [26], [33], [34], [41], [49], [51], [65]. In Section 2, we elaborate further on other types of IDS and benefits of TGAD-based IDSs over them.

TGAD-based IDS, Like any IDS in general, can operate in two main different modes. *Online* (or *monitoring*, or *live*) mode, where events of the target computer system are analyzed on the fly, while they occur. Or *offline* mode, where an IDS is used for a-posteriori *forensic analysis*, e.g., to perform an audit to identify critical events among the ones that have already occurred. In this work, *our main focus is on TGAD-based IDSs operating in an online mode*.

Requirements for TGAD-based IDSs. First and foremost, TGAD-based IDSs should identify cyberattacks as *accurately* as possible (**Requirement R1**). IDSs are usually employed as *decision support systems*, whose main role is to raise alerts that will be then (typically manually) inspected by domain experts. As such, it is important for TGAD-based IDSs to guarantee accuracy in terms of low rates of both false negatives (to miss actual cyberattacks as little as possible) and false positives (so as to limit expert intervention).

But accuracy is not enough: in the cybersecurity landscape, it is extremely important to have cyberattacks identified *in a timely manner* (**Requirement R2**). In fact, only this way one can promptly adopt countermeasures before attacks proliferate and cause irreparable harm [43]. Detecting cyberattacks in a timely manner means that TGAD-based IDSs – while operating in *online mode* – need to be highly *scalable* in the size of the temporal graphs they typically process. These graphs may be really big, as real-world cybersecurity scenarios yield events at a very high rate. For instance, the network that generated the well-established DARPA OpTC dataset [3] (see Section 4) produces more than 10 000 events (i.e., temporal edges) per minute. Additionally, in order to better exploit their generality and potential, it is desirable to have TGAD-based IDSs able to process temporal graphs that are the result of the integration of multiple data sources. Obviously, this further increases the size of the resulting temporal graphs, which necessitates even stricter scalability requirements for TGAD-based IDSs.

Motivation. The state-of-the-art TGAD-based IDSs, like Pikachu [49] and Euler [33], fail in satisfying either **R1** or **R2**. In fact, Pikachu was originally conceived to work in offline mode only, thus it clearly fails in **R2** (efficiency). Euler is trained on a link-prediction task which is seemingly not very effective for achieving satisfactory accuracy in an online setting (cf. Section 4). As such, Euler fails in **R1** (accuracy). *This incapability of the state of the art to simultaneously fulfill* **R1** *and* **R2** *constitutes the main motivation for this work.*

Contributions. In this paper, we address the limitations of the state of the art in a critical real-world problem in the cybersecurity applied domain, i.e., the problem of *online* detection of cyberattacks via TGAD (temporal graph anomaly detection). Specifically, we identify an important gap in the TGAD-based IDS literature consisting in the fact that existing TGAD-based IDSs are still not really suitable for real online cybersecurity settings. This unsuitability mainly derives from the incapability of existing methods to fulfill the above **R1–R2** requirements simultaneosly.

Here, we take a significant step forward towards filling this gap, by devising CAPTOR ("Cyber Attack Protection via Temporal Online graph Representation learning"), a novel TGAD-based IDS which is both *very effective* and *highly scalable* when used in an online setting. As such, CAPTOR comes with a major conceptual novelty and contribution, which consists in its ability to meet both **R1–R2** requirements. To achieve this, CAPTOR adopts the following technical choices, which differentiate it from the existing methods and make it technically novel and impactful too.

First, unlike existing TGAD-based IDSs [33], [49] which perform *proximity-based graph representation learning* (GRL), where embeddings reflect closeness of the nodes in the graph [23], CAPTOR adopts *structural* GRL, where embeddings express similarity about nodes' neighborhood (based on graph isomorphism) [53]. This makes CAPTOR independent of specific predictive tasks, and thus capable of self-adaptation to the vast and heterogeneous landscape of cyberattack patterns, including the ones not based on node proximity. However, although fast (and accurate) structural temporal GRL (TGRL) methods exist [36], they fail in processing temporal graphs *incrementally*, thus are still too inefficient for our hard scalability requirements. CAPTOR overcomes this by first generating time-unaware embeddings on every snapshot of the input temporal graph. This is accomplished via a structural GRL method [28], [37] which possesses all the desiderata we need: it is efficient, effective, and inductive. Then, time-unaware embeddings are aggregated over time so as to produce time-aware embeddings. This is done via a novel temporal-aggregation methodology that is fast, incrementally computable, and not relying on machine learning. The latter makes it more lightweight and faster than other methodologies adopted by existing TGAD-based IDSs [33], [49], which are based on recurrent neural networks [19]. All these technical choices enable CAPTOR to meet **R1–R2**, as desired: it achieves high accuracy–better than both Pikachu and Euler, and high efficiency–comparable to the fastest existing method (Euler).

Summary and roadmap. The contributions of this work are:

- We investigate the applicability in a real-world online setting of a category of IDS that plays a role of crucial importance in the cybersecurity applied domain, i.e., IDSs based on temporal graph anomaly detection (TGAD). (Section 2).
- We devise CAPTOR, a TGAD-based IDS which addresses the state-of-the-art limitations (Section 3).
- We provide extensive experiments (Section 4). Results show that CAPTOR achieves the best accuracy, and is comparable efficiency to the fastest competitor.

2 Background and Related Work

Building blocks of TGAD-based IDSs. Both the existing TGAD-based IDSs [33], [49] and the proposed CAPTOR, include (T)GRL and TGAD as main components at the core of their methodology.

Graph representation learning (GRL) is the problem of assigning elements of a graph (e.g., nodes, edges, subgraphs) numerical vectors – termed embeddings – such that the similarity between those elements in the graph corresponds to the similarity between their embeddings [23], [30], [53]. Based on the notion of similarity, GRL approaches can be categorized into *proximity-based* and *structural*. Proximity-based GRL approaches [23] assign similar embeddings for nodes close in the graph. Conversely, structural GRL [53] adopts similarity about nodes' neighborhood topology, regardless of whether nodes are close or not.

Temporal GRL (TGRL) [30] brings GRL to temporal graphs, requiring embeddings to additionally consider temporal dynamics.

Temporal graph anomaly detection (TGAD) is the general problem of identifying anomalies in a temporal graph. The (many) existing TGAD methods differ from each other based on the level at which anomaly is detected (e.g., node, edge, or subgraph level in a single graph, or entire-graph level in a set of multiple graphs), the definition of anomaly, and the specific algorithmic techniques [42].

General scheme adopted by TGAD-based IDSs is as follows. The input temporal graph is processed via TGRL. xThen, an *anomaly scoring* (AS) module computes anomaly scores of every temporal edge based on the embeddings of its endpoints. Anomaly scores are ultimately used for recognizing temporal edges as (likely) corresponding to cyberattacks, typically via thresholding. The TGRL and AS modules are trained (offline) on an *historical temporal graph* of past *benign-only* events. Then, the actual online part employs the learned TGRL and AS models to *incrementally* process a temporal graph of live events.

State-of-the-art TGAD-based IDSs include Pikachu [49] and Euler [33]. Both those methods follow the above general scheme of a TGAD-based IDS and differentiate from each other on the specific TGRL and AS implementations. These design choices make neither Pikachu nor Euler able to simultaneously meet both the **R1–R2** requirements discussed in the Introduction. In fact, Pikachu was originally conceived to work in offline mode only. The main reason that prevents it from being profitably used in online mode is that it adopts a *transductive* TGRL module, which needs to reprocess the historical graph entirely for producing embeddings on the live events. This way, Pikachu clearly fails in **R2** (scalability). Instead, Euler trains its TGRL module on a link-prediction task which is seemingly not very effective for achieving satisfactory accuracy in an online setting. This is mainly due to the the negative-random-sampling step required for link prediction, which (*i*) overlooks the fundamental difference between actual non-existing edges and edges that exist but are malicious, and (*ii*) comes with performance instability as the randomly chosen non-existing edges might not always reflect the actual malicious patterns (see Section 4). As such, Euler fails in **R1** (accuracy).

Other types of IDS. There exist a plethora of different IDSs [2], [40], [69], which can be categorized according to several orthogonal dimensions. For instance, according to the ID technique utilized, they can be broadly classified into *signature-based*, *classification-based*, and *anomaly-based*.

Signature-based IDSs look at distinctive malicious patterns – and *exactly* those patterns – that have been recognized as particular to specific cyberattacks [38], [54]. As such, signature-based IDSs are *unable to go beyond known cyberattack patterns*.

Classification-based IDSs adopt machine learning to learn general models of malicious patterns from cyberattack examples [1], [2], [8], [13], [29], [32], [39], [40], [58], [59]: this improves upon signature-based IDSs which instead rely on fixed malicious pattern. However, classification-based IDSs need examples of known cyberattacks: they hence *struggle to identify novel attacks for which no prior example exists*.

Anomaly-based IDSs identify cyberattacks as actions different from the usual, mostly benign behavior, without relying on any (a-priori defined/learned) distinctive malicious patterns [2]. As such, anomaly-based IDSs are *better able to recognize attacks with previously unseen characteristics*.

Another orthogonal classification is about the representation format of the data utilized by an IDS. According to this dimension, the counterpart of graph-based IDSs is given by *IDSs operating on flat data*, i.e., numerical [1], [8], [29], [32], [39], [58], [59], or textual [21], [22], [46], [60], [63] system log data. Flat-data-based IDSs have several limitations with respect to the graph-based counterpart. These limitations are better described in the last paragraph of this section.

There exist also *non-temporal graph-based IDSs* [6], which are *limited by their inability to properly operate in online mode*.

IDSs can be classified according to other orthogonal dimensions Here, we (briefly) overviewed IDS categorizations according to the aspects most relevant for the target TGAD-based IDSs. For more details, we defer to existing comprehensive surveys [2], [35], [69].

Benefits of TGAD-based IDSs. TGAD-based IDSs are generally preferable to IDSs operating on non-graph-data, i.e., flat (numerical or textual) data, for the following main reasons. (**B1**) Graphs are a much more expressive data representation model than flat representations. This corresponds to better chance for an IDS to detect complex cyberattacks patterns. In fact, signature-based or classification-based IDSs rely on predefined or learned patterns of malicious behavior. These patterns are typically either frequency-based, where anomalies are identified as activities that significantly deviate from normal frequency, or event-based, where features like packet count, protocols, and other characteristics are analyzed for commonalities. However, many attacks are impossible to capture using this traditional method. For instance, attacks using stolen credentials may exhibit completely normal features and frequencies during the "low-and-slow" lateral movement stage, often bypassing traditional detection methods. However, these attacks create atypical connections between entities, which can only be identified through relational data, such as graphs. (**B2**) Graphs provide a general and abstract way of representing data. In fact, a change in the underlying events (which corresponds to a change in the graph itself representing those events) is typically transparent to a graph-based IDS,

because the IDS simply operates on *any* graph-like data representation. This also means that (**B3**) different sources of event can easily be integrated into a single abstract graph representation, and possibly be exploited altogether, likely resulting in a more effective cyberattack detection (see Figure 1). (**B4**) Detecting cyberattacks through the identification of anomalous (temporal) edges corresponds to high generality, as a variety of cyberattacks can be abstractly modeled as anomalous edges of a (temporal) graph. (**B5**) In dynamically evolving systems like enterprise networks, temporal graphs enhance the ability to capture time-evolving relationships, enabling TGAD-based IDS to detect cyberattacks as *timely* as possible. (**B6**) TGAD-based IDSs typically employ TGRL, which, among others, has the main benefit of avoiding time/effort-consuming feature engineering. Finally, TGAD-based IDSs fall into the category of anomaly-based IDS. Thus, as discussed above, with respect to non-anomaly-based IDSs such as signature- or classification-based IDSs, TGAD-based IDSs have the main benefit that (**B7**) they are much better suited for identifying new, previously unknown cyberattacks.

symbol	*description*
$\mathcal{G} = \{G_1, \ldots, G_k\}$	temporal graph
$G_i = (V_i, E_i, w_i)$	graph snapshot representing events occurring during time interval $\Delta_i = [s_i, e_i]$
nbr_u^i	neighborhood of node u in G_i
α_e	anomaly score of edge $e \in E_i$
$\mathcal{G}_{hist}$	historical temporal graph of benign-only events used for training
τ	anomaly threshold
c	number of clusters of representations in the proposed CAPTOR
d	depth of neighborhood exploration in the proposed CAPTOR
δ	parameter to simulate a time window of interest
γ	parameter to modulate the contribution between timestamps
β	
$\mathbf{P}^i$	cluster probability matrix of snapshot i
$\mathbf{F}_u^j$	frequency the various clusters of representation appear in u's embeddings up to time j
$\mathbf{A}^{u,j}$	cluster transition matrix for node u up to time j
$\mathbf{E}^j$	time-aware embedding matrix at time j
$\mathbf{x}_e$	numerical representation of temporal edge e

TABLE 1: Main notations used in this paper.

3 Proposed Methodology

3.1 Problem statement

The raw input to TGAD-based intrusion detection is a sequence of *timestamped events* encompassing a set V of entities of interest. These events are represented as *temporal edges*, i.e., (u, v, t) triples denoting that $u, v \in V$ have interacted at timestamp t. Temporal edges in $\mathcal{E}$ are collected in a *temporal graph* $\mathcal{G} = \{G_1, \ldots, G_k\}$ composed of k (temporal) snapshots. Every snapshot G_i contains the temporal edges for all the timestamps of a time interval $\Delta_i = [s_i, e_i]$ (identified by start/end timestamps s_i and e_i). This includes Δ_i corresponding to a single timestamp as a special case (in that case, $s_i = e_i$). We assume $s_i > e_{i-1}$, for all $i = 2, \ldots, k$, i.e., we assume the various Δ_i's be non-overlapping, not necessarily contiguous, and sorted by increasing order of their start timestamp. Specifically, every $G_i = (V_i, E_i, w_i)$ is a *static edge-weighted* graph, with node set $V_i = \{u \mid \exists (u, v, t) \in \mathcal{E}, t \in \Delta_i\} \subseteq V$, edge set $E_i = \{(u, v) \mid \exists (u, v, t) \in \mathcal{E}, t \in \Delta_i\}$, and edge weighting function $w_i \colon E_i \to \mathbb{N}^+$ which assigns to every edge $(u, v) \in E_i$ a positive integer $w_i(u, v) = |\{(u, v, t) \in \mathcal{E} \mid t \in \Delta_i\}|$ corresponding to the number of times (u, v) has occurred during Δ_i. Should Δ_i correspond to a single timestamp, clearly $w_i(u, v) = 1$, for all $(u, v) \in E_i$. We denote by $nbr_u^i = \{v \in V_i \mid (u, v) \in E_i\}$ the neighborhood of node u in G_i. For the sake of generality, we assume every G_i be a *directed graph*.

The problem tackled in this work is as follows:

***Problem 1* (Online TGAD ID).** Given a temporal graph $\mathcal{G} = \{G_1, \ldots, G_k\}$, where snapshot $G_i = (V_i, E_i, w_i)$ represents events occurred during a certain time interval $\Delta_i = [s_i, e_i]$ for all $i = 1, \ldots, k$, compute a real-valued score $\alpha_e \in \mathbb{R}$ for every edge $e \in E_i$, $i = 1, \ldots, k$, which expresses how anomalous the presence of e in snapshot G_i is.

The way in which anomaly scores are computed is left unspecified in Problem 1 as it depends on the specific TGAD-based IDS. The ultimate decision about whether temporal edges are indicative of cyberattacks is made via thresholding: given a threshold $\tau \in \mathbb{R}$, edge e is recognized as a likely cyberattack if $\alpha_e \geq \tau$. τ can be either learned from past history, or established via domain expertise.

Algorithms for Problem 1 typically encompass an offline *training* phase where machine-learning models are learned from a historical temporal graph $\mathcal{G}_{hist}$ of past benign-only events. Once trained, those models are then used in the online phase to perform *inference* on the live events (i.e., to actually solve Problem 1).

The main notations used in the paper are summarized in Table 1.

3.2 Algorithmic desiderata

TGAD-based IDSs employ the following general scheme:

- (S1) Perform (static) GRL on every snapshot, so as to yield a *time-unaware* embedding of every node in every snapshot.
- (S2) For every node and every snapshot, perform temporal aggregation (TAGG) of the embeddings of that node produced in that snapshot and the preceding snapshots. This yields a *time-aware* embedding of every node in every snapshot.
- (S3) Use time-aware embeddings for anomaly scoring (AS) of the temporal edges of every snapshot.

The main desiderata for an accurate and scalable instantiation of the above scheme are as follows:

- (D1) [Inductive GRL] GRL methods in (S1) need to be *inductive*. This means that a GRL model must be able to be trained on past examples, i.e., the $\mathcal{G}_{hist}$ graph, so as to use such a model for online inference without accessing $\mathcal{G}_{hist}$ anymore.
- (D2) [Task-independent GRL training] GRL training should not rely on specific predictive tasks. This is because the landscape of malicious patterns is wide

and multifaceted. Thus, constraining embeddings to be produced on the basis of a unique, a-priori fixed predictive task is far too restrictive to effectively capture the vast cyberattack heterogeneity.

(D3) [Incremental TAGG] The TAGG in (S2) needs to be *incrementally computable*. TAGG produces embeddings in every snapshot by aggregating over the current snapshot and its preceding ones. Incremental TAGG means that aggregation up to snapshot i can be performed by updating the embeddings aggregated up to snapshot $i-1$, without reprocessing the snapshots preceding i. If TAGG is not incremental, aggregation would take time quadratic in the number of snapshots, which is a severe scalability bottleneck.

(D4) [Machine-learning independent TAGG] For further efficiency, the TAGG in (S2) should also be as lightweight as possible. Thus, it would be desirable to have TAGG not relying on any (typically computationally heavy) trainable machine-learning components (such as, e.g., RNNs).

The state-of-the-art TGAD-based IDSs meet only part of the above desiderata. Specifically, Euler [33] fails in (D2), as it trains its GRL module via a fixed link-prediction self-supervised task. Pikachu [49] fails in (D1), as it uses transductive GRL based on random-walk sampling and skipgram model learning. Also, both Euler and Pikachu fail in (D4), as they employ an RNN for TAGG.

3.3 An immediate (but impracticable) solution

At first glance, the above (D1)–(D4) desiderata could be fulfilled by employing an existing *structural* TGRL method, e.g., the recent T-SIRGN [36]. In fact, unlike the proximity-based counterpart, structural (T)GRL does not need any fixed predictive task for training, as it captures nodes' similarities based on the (graph-isomorphism-based) similarity of their neighborhoods. As such, structural (T)GRL methods are inherently self-adaptive in recognizing heterogeneous distinctive patterns of node representation: this fulfils (D2). Also, the aforementioned T-SIRGN is inductive and adopts a non-machine-learning temporal aggregation: this way, it would satisfy (D1) and (D4) as well. Conversely, though, structural TGRL methods – including T-SIRGN – fail in (D3), as they perform embedding and temporal aggregation in bundle. In particular, T-SIRGN produces the ultimate time-aware embeddings by adopting a methodology that iteratively clusters node representations (embeddings), computes temporal transitions among such clusters, and properly aggregates such cluster temporal transitions over time. The ultimate time-aware embedding of any node u produced by T-SIRGN includes a (flattenized version of) a cluster transition matrix CT_u, which stores, for every pair j, l of clusters of representations, the expected number of times j is observed to come before l in time in u's neighborhood. The computation of CT_u involves both multiplications of auxiliary vectors and sum over the various timestamps of the time period at hand. As such, CT_u is not amenable to incremental computation, which makes the overall temporal aggregation of T-SIRGN not incrementally computable either. This is confirmed by experimental evidence too (cf. Section 4, Figure 6).

In conclusion, structural TGRL methods are not amenable to incremental temporal aggregation, as desired: to compute temporal embeddings for snapshot i, they need to process all the snapshots up to i. As said above, this gives overall time complexity quadratic in the number of snapshots, which is a showstopper for our strict scalability requirement. Thus, using T-SIRGN or any other structural TGRL method is not a viable option.

3.4 Proposed CAPTOR algorithm

The proposed CAPTOR method still adheres to the aforementioned general scheme of a TGAD-based IDS, but it makes different technical choices, which are described in detail next.

Design principles and methodology overview. Although structural TGRL methods are not a proper choice (as explained above), still the idea of adopting structural GRL is promising, as it allows for fulfilling (D2). Also, this would allow for recognizing cyberattack patterns that do not depend on node proximity.

Motivated by this, our CAPTOR method still resorts to structural GRL. However, instead of using a method that performs embedding computation and temporal aggregation as a whole, we split such two phases and use different methodologies for either phase. We compute non-temporal embeddings on each snapshot with a structural GRL method which is fast, accurate and inductive, namely SIR-GN [28], [37]. Then, we devise a novel temporal aggregation which is incrementally computable, not relying on machine learning, and is well-defined for the specific embeddings produced by SIR-GN. With these two ingredients, our CAPTOR satisfies all (D1)–(D4) desiderata, and it remains sound and principled.

Overall CAPTOR's methodology is summarized in Figure 2 and Algorithm 1. CAPTOR is composed of three main components, described below: (i) time-unaware embedding generation, (ii) time-aware embedding generation, and (iii) anomaly scoring.

We remark that CAPTOR is a TGAD-based IDS. As such, it is designed to train on benign events, such that it can recognize as malicious any event that is not consistent with (i.e., "anomalous" enough with respect to) the learned benign patterns. This makes CAPTOR capable of identifying attack types that have never yet encountered, which is one of its main strengths.

Time-unaware embedding generation adopts SIR-GN [28], a method for structural (non-temporal) GRL that has been recognized as highly effective and efficient. The logic underlying SIR-GN emulates the *Weisfeiler-Lehman* algorithm [27], [62], a popular method designed (among others) to test for graph isomorphism. SIR-GN encodes the structural representation of a node by using basic topological properties (i.e., the degree) as a starting point, and iteratively updating those representations via *clustering* and *neighbor aggregation*. At each iteration, the current node representations are first grouped into c clusters, where $c > 0$ is a (user-defined) integer parameter (corresponding to the embedding dimension too). Then, the probability for a node that its current

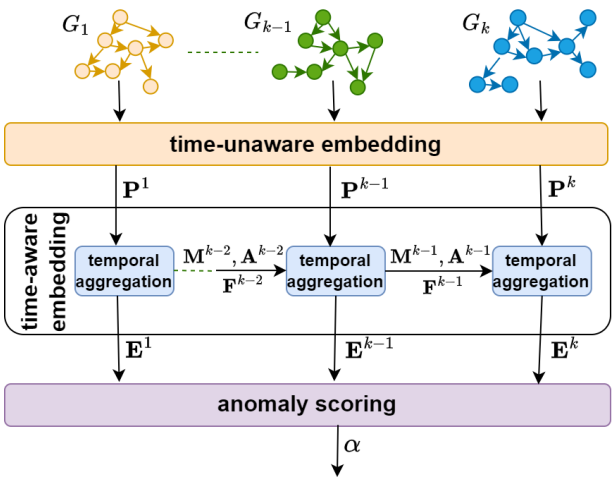


Fig. 2: Block diagram of the proposed CAPTOR algorithm.

representation belongs to any cluster of representations is computed and placed in a probability vector. At the next iteration, the new structural representation of any node u will be the aggregation (sum) of probability vectors of u's neighbors yielded in the previous iteration. This process is repeated for d iterations, where $d > 0$ is a further (user-defined) parameter, denoting the depth of neighborhood exploration. This way, the ultimate node embeddings are representative of d-hop nodes' neighborhoods. SIR-GN handles directed graphs and edge features (which, in our context, are given by edge frequencies $w_i(\cdot,\cdot)$).

SIR-GN can easily be made *inductive* by decomposing it into training and inference phases [37]. SIR-GN's training phase consists in computing c cluster centroids on a training graph, i.e., the historical $\mathcal{G}_{hist}$ graph in our context. Such centroids constitute an embedding machine-learning model EMB. At inference time, given c, d, the trained embedding model is applied to every snapshot G_i of the input temporal graph $\mathcal{G}$ of live events, i.e.,

$$\text{EMB}(G_i, d, c), \tag{1}$$

whose output is a time-unaware embedding for every node $u \in V$ on snapshot i, and is represented as a *cluster probability* matrix

$$\mathbf{P}^i \in [0,1]^{|V|\times c}. \tag{2}$$

Every row $\mathbf{P}^i_u$ is a probability distribution of the representation of node u at time i to belong to the c clusters of representations.

Time-aware embedding generation performed at time j yields a *time-aware embedding matrix*

$$\mathbf{E}^j \in \mathbb{R}^{|V|\times(c^2+c)}, \tag{3}$$

where every row $\mathbf{E}^j_u \in \mathbb{R}^{c^2+c}$ is the time-aware embedding of node u capturing the temporal dynamics of time-unaware u's embeddings up to time j. $\mathbf{E}^j_u$ is a concatenation of vectors $\mathbf{T}^j_u \in \mathbb{R}^{c^2}$ and $\mathbf{F}^j_u \in \mathbb{R}^c$:

$$\mathbf{E}^j_u = \text{concatenate}(\mathbf{T}^j_u, \mathbf{F}^j_u). \tag{4}$$

$\mathbf{F}^j_u$ expresses the *frequency* (i.e., expected number of times) the various clusters of representation appear in u's embeddings up to time j, i.e., $\sum_{i=1}^{j} \mathbf{P}^i_u$. Every term of this sum is multiplied by a *time elapse* factor $e^{\frac{-\ell(i,j)}{\delta}} \in [0,1]$ which smooths the contribution of distant timestamps. $\ell(i,j)$ quantifies the amount of time between timestamps i and j. Specifically, for any two timestamps $x, y \in [1,k]$, corresponding to time intervals $\Delta_x = [s_x, e_x]$, $\Delta_y = [s_y, e_y]$, $\ell(x,y)$ is the difference between the middle timestamps of Δ_y and Δ_x, i.e., $\ell(x,y) = (s_y + e_y - s_x - e_y)/2$. $\delta \geq 0$ is a (user-defined) parameter that modulates the impact of the time elapse factor, so that a lower (resp. higher) δ makes $e^{\frac{-\ell(i,j)}{\delta}}$ closer to 0 (resp. 1). Playing with δ allows for simulating a *time window of interest*, so that timestamps i which are too far from j, and, as such, outside the desired time window, are implicitly and automatically discarded in the temporal aggregation. To sum up, $\mathbf{F}^j_u$, for all $u \in V$, is defined as:

$$\mathbf{F}^j_u = \textstyle\sum_{i=1}^{j} e^{\frac{-\ell(i,j)}{\delta}} \mathbf{P}^i_u. \tag{5}$$

$\mathbf{T}^j_u$ expresses the *temporal transitions* among the various clusters of representation. It is defined based on a *cluster transition matrix* $\mathbf{A}^{u,j} \in \mathbb{R}^{c\times c}$. Every (a,b) entry of $\mathbf{A}^{u,j}$ corresponds to the frequency (i.e., expected number of times) cluster a is observed to come before cluster b in time in u's time-unaware embeddings up to time j, further multiplied by two time elapse terms. The first time elapse term is conceptually and mathematically equal to the one used for the definition of $\mathbf{F}^j_u$ (Equation (5)): it smooths the contribution of timestamps i too distant from the current timestamp j. The second time elapse term is mathematically similar to the first one and is regulated by a (user-defined) parameter $\gamma \geq 0$ which plays the same role as δ. However, a major difference is that this second time elapse term modulates the contribution of distant timestamps i to *any other* timestamp $i' \in [i,j]$, and not to timestamp j only. Ultimately, matrix $\mathbf{A}^{u,j}$ and vector $\mathbf{T}^j_u$ are defined as:

$$\begin{aligned} \mathbf{A}^{u,j} &= \textstyle\sum_{i=1}^{j} \sum_{i'=i+1}^{j} e^{\frac{-\ell(i,j)}{\delta}} e^{\frac{-\ell(i,i')}{\gamma}} (\mathbf{P}^i_u)^\top \mathbf{P}^{i'}_u, \\ \mathbf{T}^j_u &= \text{flatten}(\mathbf{A}^{u,j}). \end{aligned} \tag{6}$$

Incremental temporal aggregation. Importantly, $\mathbf{F}^{j+1}_u$ and $\mathbf{A}^{u,j+1}$, for all $u \in V$, can be *computed incrementally* from $\mathbf{F}^j_u$ and $\mathbf{A}^{u,j}$ as:

$$\mathbf{F}^{j+1}_u = e^{\frac{-\ell(j,j+1)}{\delta}} \mathbf{F}^j_u + \mathbf{P}^{j+1}_u, \tag{7}$$

$$\mathbf{A}^{u,j+1} = e^{\frac{-\ell(j,j+1)}{\delta/\gamma}} \mathbf{A}^{u,j} + (\mathbf{M}^{j+1}_u)^\top \mathbf{P}^{j+1}_u, \tag{8}$$

where $\mathbf{M}^{j+1} \in \mathbb{R}^{|V|\times c}$ is an auxiliary matrix, which is still incrementally computable as:

$$\mathbf{M}^{j+1} = e^{\frac{-\ell(j,j+1)}{\delta} - \frac{\ell(j,j+1)}{\gamma}} \mathbf{M}^j + \mathbf{P}^{j+1}. \tag{9}$$

Note that, for ease of presentation, $\mathbf{P}^i_u$, $\mathbf{E}^i_u$, $\mathbf{T}^i_u$, $\mathbf{F}^i_u$, $\mathbf{M}^i_u$, and $\mathbf{A}^{u,i}$ are assumed to be computed for nodes $u \in V \setminus V_i$ too. However, an actual implementation of CAPTOR can avoid generating/processing vectors/matrices for nodes $\notin V_i$, for obvious efficiency reasons.

Rationale of the proposed temporal aggregation. CAPTOR's temporal aggregation yields time-aware embeddings

TABLE 2: Dataset characteristics. $|V|$: number of overall nodes. $|\mathcal{E}|_{hist}$, k_{hist}: number of temporal edges and snapshots in the $\mathcal{G}_{hist}$ historical graph (used for training). $|\mathcal{E}|$, k: number of temporal edges and snapshots in the temporal graph $\mathcal{G}$ of live events (used for inference). *attacks*: number of $\mathcal{G}$'s temporal edges corresponding to cyberattacks. For each dataset, snapshots aggregate events occurring during a 1-minute time period.

dataset	$\lvert V\rvert$	$\lvert\mathcal{E}_{hist}\rvert$	k_{hist}	$\lvert\mathcal{E}\rvert$	k	*attacks*
OpTC	683	38 392 890	3 455	8 642 610	2 308	24 231
OpTC-ext	49 304	93 130 638	3 455	19 918 000	2 308	24 231
LANL	16 082	812 936	2 460	26 866 807	81 060	788
CICIDS	19 129	529 918	487	2 300 825	1 967	589 102

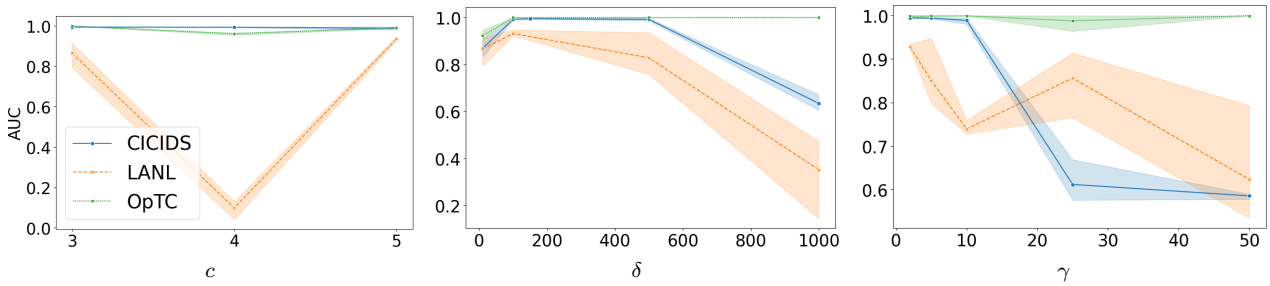


Fig. 3: CAPTOR's parameter tuning.

Algorithm 1 CAPTOR

Input: Machine-learning models EMB and VAE (trained on a historical temporal graph); temporal graph $\mathcal{G} = \{G_i = (V_i, E_i, w_i)\}_{i=1}^{k}$, with $V = \bigcup_{i=1}^{k} V_i$; integers $c, d > 0$; real numbers $\delta, \gamma, \beta \geq 0$.

Output: Real number α_e for every $e \in E_i$, $\forall i = 1, \ldots, k$.

1: Initialize matrices $\mathbf{P} = 0^{|V|\times c}$, $\mathbf{F} = 0^{|V|\times c}$, $\mathbf{M} = 0^{|V|\times c}$, $\forall u \in V$: $\mathbf{A}^u = 0^{c\times c}$
2: **for all** $i = 1, \ldots, k$ **do**
▷ time-unaware embedding generation
3: $\mathbf{P} \leftarrow \text{EMB}(G_i, d, c)$ ▷ Equations (1)–(2)
▷ time-aware embedding generation
4: **for all** $u \in V_i$ **do**
5: $\mathbf{F}_u \leftarrow e^{\frac{-\ell(i-1,i)}{\delta}} \mathbf{F}_u + \mathbf{P}_u$ ▷ Equation (7)
6: $\mathbf{M}_u \leftarrow e^{\frac{-\ell(i-1,i)}{\delta} - \frac{\ell(i-1,i)}{\gamma}} \mathbf{M}_u + \mathbf{P}_u$ ▷ Equation (9)
7: $\mathbf{A}^u \leftarrow e^{\frac{-\ell(j,j+1)}{\delta/\gamma}} \mathbf{A}^u + (\mathbf{M}_u)^\top \mathbf{P}_u$ ▷ Equation (8)
8: $\mathbf{E}_u \leftarrow \text{concatenate}(\text{flatten}(\mathbf{A}^u), \mathbf{F}_u)$ ▷ Equations (4) and (6)
9: **end for**
▷ anomaly scoring
10: **for all** $e = (u, v) \in E_i$ **do**
11: $\mathbf{x}_e \leftarrow \text{concatenate}(\mathbf{E}_u, \mathbf{E}_v)$, $\hat{\mathbf{x}}_e \leftarrow \text{VAE}(\mathbf{x}_e)$ ▷ Equations (10) and (11)
12: $\alpha_e = \text{MSE}(\mathbf{x}_e, \hat{\mathbf{x}}_e) + \beta\ \text{D}_{\text{KL}}(q_\phi(\mathbf{z}|\mathbf{x}_e) \| p(\mathbf{z}))$ ▷ Equation (12)
13: **end for**
14: **end for**

as a combination of temporal transitions among the various types (clusters) of structural representation and their frequencies. This general design has been shown to be principled and effective [36], thus we borrow it.However, CAPTOR adopts a new way to compute and combine temporal transitions and frequencies. A major novelty is that CAPTOR's temporal transitions and frequencies do not aggregate over neighbors' embeddings of a node. In fact, this is not really needed, as it has already been performed upstream, in time-unaware embeddings. This simple but clever trick enables incremental computation. Conversely, temporal aggregation adopted in the literature, e.g., in T-SIRGN [36], performs neighborhood aggregation before every step of temporal aggregation, and it is, as such, not easily amenable to incremental computation.

Anomaly scoring. We utilize a *Variational Autoencoder* (VAE) to yield anomaly scores α_e for every temporal edge e of the input temporal graph $\mathcal{G}$. The VAE is trained on a historical temporal graph $\mathcal{G}_{hist}$ of past benign-only events, and produces a machine-learning model VAE. The VAE takes as input a numerical representation $\mathbf{x}_e \in \mathbb{R}^{2(c^2+c)}$ of a temporal edge $e = (u, v, i)$ defined as the concatenation

$$\mathbf{x}_e = \text{concatenate}(\mathbf{E}_u^i, \mathbf{E}_v^i) \tag{10}$$

of the (time-aware) embeddings of e's endpoints at time i. Given $\mathbf{x}_e$, VAE's output is the

$$\hat{\mathbf{x}}_e = \text{VAE}(\mathbf{x}_e) \tag{11}$$

reconstruction of $\mathbf{x}_e$ based on the VAE.

The ultimate anomaly score α_e of an edge e of $\mathcal{G}$ is computed as the mean squared error $\text{MSE}(\mathbf{x}_e, \hat{\mathbf{x}}_e)$ between e's representation $\mathbf{x}_e$ and its VAE-based reconstruction $\hat{\mathbf{x}}_e$, plus a regularization term corresponding to the Kullback-Leibler divergence D_{KL} between VAE encoder's distribution $q_\phi(\mathbf{z}|\mathbf{x}_e)$ and a prior distribution $p(\mathbf{z})$:

$$\alpha_e = \text{MSE}(\mathbf{x}_e, \hat{\mathbf{x}}_e) + \beta\ \text{D}_{\text{KL}}(q_\phi(\mathbf{z}|\mathbf{x}_e) \| p(\mathbf{z})), \tag{12}$$

where β is a coefficient that balances the reconstruction error and regularization. The rationale is that the higher the VAE-based reconstruction error, the more unexpected the edge is for the VAE. Unexpected edges e are those that diverge from a benign behavior: hence, it is sound to have a high anomaly score α_e for them.

Algorithm analysis. The time complexity of Algorithm 1 is as follows. Matrix initialization (Line 1) takes $O(c|V|)$ time.

For each timestamp i, time-unaware embeddings (Line 3) take $O(d\ c\ (|V_i|+|E_i|))$ time [37], time-aware embeddings (Lines 4–9) take $O(c^2\ |V_i|)$ time, and anomaly scoring (Lines 10–13) takes $O(c^2|E_i|)$ time (reasonably assuming the few linear transformation of the VAE model to be negligible compared to the rest). Thus, the overall runtime of Algorithm 1 for processing a single snapshot $G_i = (V_i, E_i, w_i)$ is $O((d\ c + c^2)\ (|V_i|+|E_i|))$, which is linear in the size of G_i. The overall runtime for processing the entire input temporal graph $\mathcal{G}$ is $O((d\ c + c^2)\sum_{i=1}^{k}(|V_i|+|E_i|)$, which is, again linear in the size of the input. This attests the high efficiency of the proposed CAPTOR.

The space complexity of Algorithm 1 is small too: $O(c^2\ |V|)$, given by the (asymptotic) size of the biggest auxiliary matrix (i.e., $\mathbf{E}$).

4 Experimental Evaluation

Datasets. We experiment with well-established real datasets, which have been traditionally employed in cybersecurity, and whose characteristics are summarized in Table 2 and described below.

(DARPA-)OpTC [3], [4] is a dataset collected as part of a technology transition pilot study funded by Boston Fusion Corp.'s cyber APT scenarios for Enterprise Systems (CASES) project. We use the START events from the FLOW object, extracted from network logs. This gives a communication graph whose nodes are IP addresses of network hosts, and temporal edges are communications from a source to a destination host occurring at a certain time. $\mathcal{G}_{hist}$ includes the first 3 days of activity, containing benign events only.

We also integrated FLOW-START data with PROCESS data, resulting in a an extended OpTC-ext dataset. PROCESS data represent a provenance graph of timestamped actions (temporal edges) between two processes (nodes), namely from a parent process to a child process. PROCESS data and FLOW-START data are integrated by adding edges from a host to the processes running on that host.

LANL [31] refers to the Comprehensive Multi-source Cyber Security Events dataset developed by the Los Alamos National Lab (LANL). This is a further authentication dataset where events are authentications among network hosts. The first 40 hours contain benign-only events which are used to build $\mathcal{G}_{hist}$. The full dataset spans $> 1B$ events. We selected a subset of it by adopting the approach in [49]. We first removed all computer-account activities, along with events generated from LOCAL, SYSTEM, ANONYMOUS, and ADMINISTRATOR accounts, so as to keep human-initiated activities only. Then, we included in the ultimate dataset all malicious users along with 5% randomly chosen benign users.

The CICIDS dataset [55] includes network traffic data which are represented as a graph in a way similar to the OpTC dataset (interactions between IP addresses). The first day of activity – containing benign events only – constitutes $\mathcal{G}_{hist}$.

Methods. We compare our CAPTOR (Algorithm 1) to the state-of-the-art TGAD-based IDSs: Pikachu [49] and Euler [33]. We consider two configurations of Euler: with and without a Long Short-Term Memory (LSTM). The latter is denoted Euler-noLSTM. As per Euler's design [33], we use the link-prediction predictive task to train it for the online setting, and link detection for the offline setting. Remarkably, Pikachu has been conceived to work in an offline setting only. It cannot process in reasonable time our large-sized online datasets. Thus, we involve it in offline experiments only.

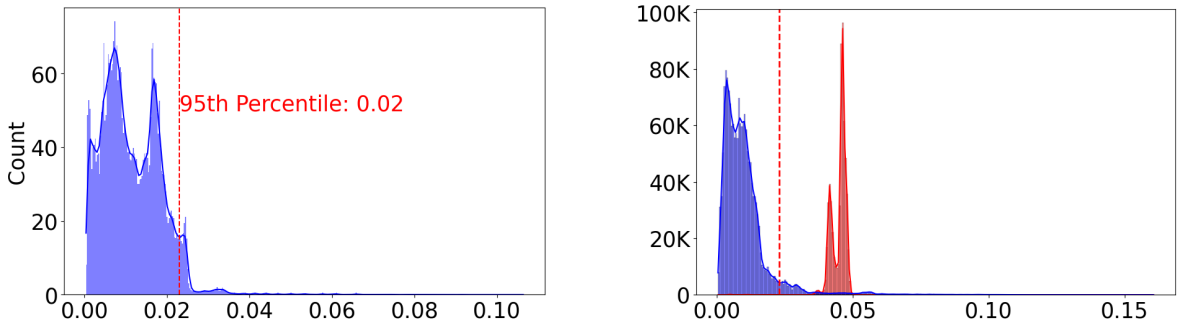
Assessment. Accuracy of the various tested methods is measured in terms of number of false positives (FP) and false negatives (FN), true positive (TPR) and false positive (FPR) rates, and area under the receiver operating characteristic curve (AUC). Efficiency is measured in terms of runtime of the training and inference phases.

Anomaly threshold – offline setting. To compute the τ anomaly threshold for experiments for the offline setting, we employ a strategy that is a well-established one in the IDS literature [33], [49]: τ is set to the value that maximizes $\min\{TPR, 1-FPR\}$ on test data (i.e., on the graph $\mathcal{G}$ itself of live events). This is a supervised strategy, which works for offline forensic analysis only, as it needs to look at future events. As such, we employ this strategy for the offline-setting experiments only.

Anomaly threshold – online setting. For the online-setting experiments, the threshold should be set by looking at the $\mathcal{G}_{hist}$ training data only. Since $\mathcal{G}_{hist}$ contains benign events only, it is popular in the IDS literature to set the τ anomaly threshold to a value that ranges from (roughly) the 90th and the 100th percentile of the distribution of the anomaly scores observed on $\mathcal{G}_{hist}$ [20], [44], [45], [47], [66]. Thus, we stick to this common practice, and we set τ equal to the 95th percentile of the α_e scores of the $\mathcal{G}_{hist}$ training data for all the experiments of our CAPTOR in an online setting.

As for the Euler competitor (the only one amenable to an online setting), we use a τ threshold that is computed according to a strategy suggested in Euler's original work [49]. That strategy holds a subsets of snapshots out of the $\mathcal{G}_{hist}$ training data and utilizes them as a validation set to learn a proper threshold value. Specifically, the threshold is set to the value that allows for optimizing a convex combination of TPR and $-FPR$ on such validation snapshots. As the validation snapshots are part of the training data, they contain benign events only. Thus, to compute TPR and FPR, a negative sampling strategy is adopted to simulate malicious events. This strategy simply randomly samples temporal edges that do not exist in the validation snapshots and interprets them as malicious edges to be used to compute TPR and FPR. This negative-sampling-based threshold setting strategy comes with two major downsides. First, it overlooks the fundamental difference between actual non-existing edges and edges that exist but are malicious. Second, its inherent nondeterministic nature (as it relies on random negative edge sampling) makes the strategy result in high instability in model's ultimate effectiveness in detecting malicious events, as the randomly chosen non-existing edges might not always reflect the actual malicious patterns. This instability phenomenon is a known issue, well-observed in the literature [65] and confirmed in our experiments too (see below).

Parameters. CAPTOR's parameters were set on each dataset by measuring the performance (in terms of AUC) on a



(a) CAPTOR – distribution of anomaly scores α_e on training data. (b) CAPTOR – distribution of anomaly scores α_e on test data.

Fig. 4: Distribution of the α_e anomaly scores on CICIDS dataset: for every anomaly score (x axis), the number of temporal edges exhibiting it (y axis), in (a) training data and (b) test data. Blue represents benign events and red represents malicious events. In (a), the dashed vertical line corresponds to the 95th percentile, i.e, to the value used for setting CAPTOR's τ anomaly threshold. Applying the same threshold in the test data in (b) shows how effectively it separates benign events from malicious ones.

TABLE 3: Accuracy results for the online setting. Mean and standard deviation over multiple execution runs are reported.

dataset	*method*	TPR	FPR	AUC
OpTC	Euler	.689 ± .327	.102 ± .039	.922 ± .058
	Euler-noLSTM	.553 ± .205	.145 ± .037	.872 ± .048
	CAPTOR	.960 ± .005	.002 ± .001	.998 ± .001
LANL	Euler	.560 ± .226	.049 ± .015	.918 ± .029
	Euler-noLSTM	.647 ± .198	.017 ± .004	.946 ± .025
	CAPTOR	.918 ± .016	.081 ± .016	.953 ± .015
CICIDS	Euler	.704 ± .414	.048 ± .039	.977 ± .02
	Euler-noLSTM	.928 ± .219	.176 ± .028	.864 ± .016
	CAPTOR	.997± .001	.084 ± .017	.980 ± .002

validation set of live events (distinct from the one used for testing). The results of this parameter-tuning stage are illustrated in Figure 3. Based on that, we selected the following values: $c = 3$ for OpTC and CICIDS datasets; $c = 5$ for LANL; $d = 5$ for all datasets; $\delta = 250$, $\gamma = 5$ for OpTC and LANL; $\delta = 150$, $\gamma = 5$ for CICIDS. As for the VAE, we set $\beta = 1$ for OpTC and LANL $\beta = 10^{-6}$ for CICIDS. We varied the size of the latent space from a $[2, 8]$ range. According to the results on the validation set, we picked a latent-space size of 4 for OpTC and LANL, and of 6 for CICIDS. We utilized the Adam optimization algorithm, adjusting the learning rate within a range of 0.1 to 0.01 and applying a weight decay parameter set to $1e$-6.

For both Pikachu and Euler, parameters were tested on datasets OpTC and LANL, properly tuned and reported in their original paper to be used for further experiments. Thus, on OpTC and LANL datasets, we used those suggested parameters for both Pikachu and Euler. The CICIDS dataset, however, was not used in Pikachu's or Euler's original work, thus no suggestion about parameters was provided for such a dataset. For this reason, on CICIDS, we performed a grid search nearby the default values of Pikachu's and Euler's parameters, and chose the best performing values. For Pikachu, we varied the GRU first layer size from 32 to 64 (default 32) and second layer from size 64 to 256 (default 128). We also tested embedding dimension $h = \{50, 100, 150, 200, 300\}$ and support set (i.e., number of neighbors sampled for each node during the learning process) $s = \{2, 5, 10, 15, 20, 25\}$. The optimal values were found to be size 32 for Layer 1 and 128 for layer 2, $h = 10$ and $s = 5$. For Euler, we varied the hidden dimension from 16 to 32 (default 16) and embedding dimension from 8 to 64 (default 32) and learning rate of 0.01 to 0.001 (default 0.01). The optimal parameters were found to be 32-dimensional hidden and 32-dimension embedding size with a learning rate of 0.01. All reported results for Pikachu and Euler on CICIDS refer to those best-performing parameter values.

Implementation. The proposed CAPTOR method was implemented in `Python`. The SIR-GN and VAE building blocks were implemented using `scikit-learn` [9] and `PyTorch` [48], respectively. We used a random seed of 42 throughout the experiments. As for the competitors, we used their official implementations [33], [49]

Testing environment. All the experiments were run on an Ubuntu 22.04 server with 16-core 3.80GHz Intel(R) Xeon(R) Gold 5222 CPU, 1TB RAM, and NVIDIA RTX 6000 Ada Generation GPU.

Accuracy results – online setting (Table 3, Figure 4). The proposed CAPTOR achieves consistently high (average) TPR and AUC, and low FPR across all datasets. It outperforms its primary online competitor Euler and its Euler-noLSTM variant in all the metrics and datasets, but FPR in LANL and CICIDS. Apart from the inferior average performance, a perhaps even more critical downside of Euler and Euler-noLSTM is their high instability in their performance, which is testified by the very high standard deviations they exhibit over the various experimental runs. The reason of this instability lies in the way how Euler and Euler-noLSTM set the anomaly threshold (discussed in the above "Anomaly threshold – online setting" paragraph). Conversely, CAPTOR exhibits very small variation over the various runs (mainly due to the nondeterministic nature of its VAE component).

Accuracy results – offline setting (Table 4). The proposed CAPTOR exhibits high performance in the offline setting too: it outperforms all the competitors on all the datasets and assessment criteria, with the only exception of FPR and AUC on LANL, where, however, the difference with the best-performing method (Euler) is tiny. Our CAPTOR achieves an improvement over Pikachu (the competitor specifically designed for an offline setting) up to 11-fold

TABLE 4: Accuracy results for the offline setting.

method	OpTC			LANL			CICIDS		
	TPR	FPR	AUC	TPR	FPR	AUC	TPR	FPR	AUC
Euler	.956	.044	.994	.901	.099	.967	.983	.016	.997
Euler-noLSTM	.896	.110	.961	.940	**.061**	**.985**	.969	.028	.989
Pikachu	.995	.006	.997	.728	.272	.735	.821	.128	.953
CAPTOR	**.998**	**.002**	**.998**	**.942**	.063	.968	**.989**	**.011**	**.997**

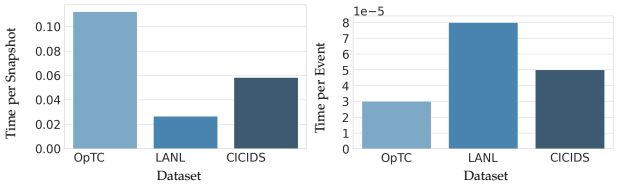


Fig. 5: Average inference time (seconds) of the proposed CAPTOR method.

in terms of false positives ($19.6k$ vs. $223.3k$) and 14-fold in terms of false negatives ($6.8k$ vs. $99.9k$), on the CICIDS dataset. The improvement remains consistent on the other two datasets too: 3-fold on OpTC and 4-fold on LANL.

Efficiency results (Table 5, Figure 5). This set of experiments is mainly conceived to compare the proposed CAPTOR to Euler, in both training (on the historical $\mathcal{G}_{hist}$ graph) and inference (while processing the graph $\mathcal{G}$ of live events). The other competitor, Pikachu, was not designed to work in online mode. In fact, to process all the live events in $\mathcal{G}$, it took 16 hours on CICIDS, and > 50 hours on OpTC and LANL (despite using 8 parallel processes).

Here we aggregate events in snapshots spanning 1-hour intervals, and use no parallelization for Euler. Results in Table 5 reveal that CAPTOR's training is faster than Euler's on LANL and CICIDS: this is likely due to a more lightweight methodology employed by CAPTOR, which relies much less than Euler on machine-learning components. On OpTC, Euler's training is instead faster than CAPTOR's, though of the same order of magnitude. This is likely due to a significantly lower number of nodes (683) in OpTC than the other datasets, which is particularly beneficial for a graph neural network [11] (one of main Euler's components). Euler's inference runtimes are generally lower than CAPTOR's. However, the difference between the two methods is small, and, in general, CAPTOR's runtimes are very low. This can be better appreciated in Figure 5, which shows that average CAPTOR's runtime per snapshot and per event is in the order of 10^{-1} and 10^{-4} seconds, respectively: this attests the high scalability of CAPTOR when used in a real-world online setting, where snapshots/events need to be processed incrementally and extremely fast.

Data integration results (Table 6). As discussed in Sections 1–2, a major benefit of TGAD-based IDSs is that the integration of multiple data sources is transparent to them, thus it can be easily adopted. In fact, using more data sources can be beneficial for improving the effectiveness of machine learning tasks in general [14], [50], [68], including intrusion detection [25], [67]. The main reason is that using more data sources brings complementary information about system activities, thus providing context to network events. This allows any IDS, including our CAPTOR, for better distinguishing between benign and malicious events. Simply put, more data improves the ability to correctly identify attacks, thus improving performance. Here we test this capability, by comparing CAPTOR's accuracy values on the basic OpTC dataset to the ones obtained on the OpTC-ext dataset, which corresponds to the basic OpTC augmented with further data (see Table 2 and the description at the beginning of this section). Results in Table 6 show that integrating more data is indeed beneficial: FP get a substantial reduction, by a a factor of roughly 3 (from $17\,455$ to $5\,606$), and FN even get to zero.

Ablation study (Tables 7–9). Table 7 depicts the performance of our CAPTOR without incorporating the time-aware embedding generation component. This is best understood as removing the update loop in Lines 4–8 in Algorithm 1. For all datasets, the use of the time-aware embedding generation component leads to nearly double AUC, nearly double TPR, and a decrease of FPR by at least a factor of 5. These results confirm that the novel incremental temporal aggregation introduced in CAPTOR is indeed crucial for its performance.

Table 8 and Table 9 show how the performance of CAPTOR varies when different anomaly scoring algorithms (other than the VAE) are used. Table 8 show that Isolation Forest (iForest) (trained on all the dataset with number of estimators varying from 50 to 150) does not perform as well as the VAE. In Table 9, we focused on two other algorithms, namely One-class SVM and Local Outlier Factor (LOF). We report results for these two further anomaly scoring algorithms for CICIDS dataset only as they were too slow to run on the bigger OpTC and LANL datasets. According to the results in the table, these algorithms are consistently outperformed by the VAE. Together, Tables 8–9 make it clear the contribution of our choice of VAE as an anomaly scoring algorithm for the performance of CAPTOR.

Additional experimental insights (Figure 6, Table 10). We also provide experimental evidence that existing structural TGRL methods are not amenable to incremental temporal aggregation. To this end, we compare inference runtimes to calculate one graph snapshot for our CAPTOR versus T-SIRGN [36] in the OpTC-ext dataset. The results of this experiment are illustrated in Figure 6. We used two different levels of granularity to aggregate events in graph snapshots, namely 30-second and 60-second intervals, and measured runtimes as the window of considered timestamps is increased. The window is simply the distance in previous timestamps that contributes to the calculation of time i. As discussed in Section 3.3, in order to calculate the time-aware

TABLE 5: Runtimes (seconds), online setting, train and inference phases.

method	OpTC		LANL		CICIDS	
	train	*inference*	*train*	*inference*	*train*	*inference*
Euler	44.08	4.47	359.14	465.2	87.04	13.47
Euler-noLSTM	42.05	4.46	272.03	448.3	49.5	13.4
CAPTOR	70.2	9.925	145.62	517.7	18.2	19.81

TABLE 6: CAPTOR's accuracy results while integrating multiple data sources.

dataset	FP	FN	TPR	FPR	AUC
OpTC	17 455	45	.998	.002	.998
OpTC-ext	**5 606**	**0**	**.999**	**0**	**.999**

TABLE 7: CAPTOR's accuracy results with and without time-aware embedding generation component.

dataset	without time-aware emb.			with time-aware emb.		
	TPR	FPR	AUC	TPR	FPR	AUC
OpTC	.496	.503	.497	**.998**	**.002**	**.998**
LANL	.301	.350	.643	**.942**	**.063**	**.968**
CICIDS	.499	.503	.499	**.989**	**.011**	**.997**

TABLE 8: CAPTOR's accuracy results by equipping it with iForest or VAE as a component for anomaly scoring.

dataset	iForest			VAE		
	TPR	FPR	AUC	TPR	FPR	AUC
OpTC	.948	.081	.934	**.998**	**.002**	**.998**
LANL	.689	.318	.708	**.942**	**.063**	**.968**
CICIDS	.686	.314	.741	**.989**	**.011**	**.997**

TABLE 9: CAPTOR's accuracy results in the CICIDS dataset by varying the anomaly scoring algorithm utilized in it.

anomaly scorer	TPR	FPR	AUC
iForest	.686	.314	.741
One-class SVM	.515	.479	.603
LOF	.447	.552	.368
VAE	**.989**	**.011**	**.997**

embeddings for time i, T-SIRGN must process all timestamps in the desired window up to time i. Our CAPTOR, instead, is able to perform incremental computation for producing the time-aware embeddings for time i, without needing to reprocess snapshots before i. This different way of computing time-aware embeddings by the two methods is faithfully reflected in the runtimes reported in Figure 6. In fact, it is clear from the figure that T-SIRGN in all but the smallest window and the largest granularity is not able to process a graph snapshot in the time increment equal to the size of the snapshot. This rules out its use in any online setting. In contrast, CAPTOR's method of computing embeddings updates incrementally, and, as such, takes the same amount of time to update regardless of the number of previous timestamps contributing information to the embedding.

Finally, we complement our evaluation with an experiment focusing on intrusion detection formulated as a binary edge classification task instead of anomaly detection, with classes being malicious or benign events. The main rationale of this experiment is to test the ability of CAPTOR to detect malicious patterns too, not only benign ones. More specifically, we considered the entire temporal graph underlying the CICIDS dataset, and built a numerical representation

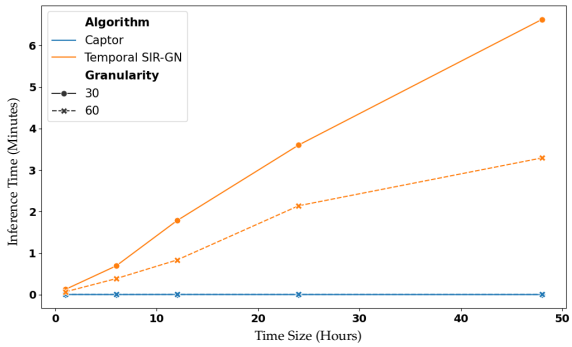


Fig. 6: Inference time for generating time-aware embeddings using the proposed CAPTOR and T-SIRGN [36] in OpTC-ext dataset by aggregating events in snapshots spanning 30-second and 60-second intervals.

TABLE 10: CAPTOR's accuracy results on CICIDS dataset when intrusion detection is formulated as a binary edge classification task instead of anomaly detection, with classes being malicious or benign events.

Precision	*Recall*	$F1$	AUC	TPR	FPR
0.99	0.99	0.99	0.98	0.96	0.006

of it by representing every temporal edge with the ultimate embedding produced by CAPTOR. Each (numerical representation of a) temporal edge is assigned with a label denoting whether it is malicious or benign. We performed a 80/20 random split of the resulting dataset, using the 80% part as a training set to build a (logistic regression) classifier, and the remaining 20% as a test set to measure the performance of such a trained classifier. The results of this experiments are provided in Table 10. These results show very high accuracy of CAPTOR, e.g., 0.99 F1, 0.96 TPR, 0.006 FPR, which fully supports its hypothesized ability to identify malicious patterns too.

5 Conclusion

In this paper, we present CAPTOR ("Cyber Attack Protection via Temporal Online graph Representation learning"), a novel temporal-graph-anomaly-detection (TGAD)-based intrusion detection system (IDS) which addresses the accuracy and scalability shortages of the state-of-the-art TGAD-based IDSs. CAPTOR performance is attested by extensive experiments on real cybersecurity datasets. As such, CAPTOR constitutes an important step forward towards making the valuable TGAD-based IDS technology applicable in real-world scenarios of the cybersecurity applied domain.

Acknowledgments

This research was funded by a National Centers of Academic Excellence in Cybersecurity grant H98230-22-1-0300,

which is part of the National Security Agency and ONR grant N00014-23-1-2132.

REFERENCES

[1] I. Al-Turaiki and N. Altwaijry, "A convolutional neural network for improved anomaly-based network intrusion detection," *Big Data*, vol. 9, no. 3, pp. 233–252, 2021.

[2] A. Aldweesh, A. Derhab, and A. Z. Emam, "Deep learning approaches for anomaly-based intrusion detection systems: A survey, taxonomy, and open issues," *Knowledge and Information Systems (KAIS)*, vol. 189, 2020.

[3] M. M. Anjum, S. Iqbal, and B. Hamelin, "Analyzing the usefulness of the DARPA OpTC dataset in cyber threat detection research," in *Proc. of Symp. on Access Control Models and Technologies (SACMAT)*, 2021, pp. 27–32.

[4] R. Arantes, C. Weir, H. Hannon, and M. Kulseng, "Operationally transparent cyber (OpTC)," https://doi.org/10.21227/edq8-nk52, 2022.

[5] E. Bautista, L. Brisson, C. Bothorel, and G. Smits, "MAD: Multi-Scale Anomaly Detection in Link Streams," in *Proc. of Int. Conf. on Web Search and Data Mining (WSDM)*, 2024, pp. 38–46.

[6] T. Bilot, N. El Madhoun, K. Al Agha, and A. Zouaoui, "Graph neural networks for intrusion detection: A survey," *IEEE Access*, vol. 11, pp. 49 114–49 139, 2023.

[7] H. Binyamini, R. Bitton, M. Inokuchi, T. Yagyu, Y. Elovici, and A. Shabtai, "A framework for modeling cyber attack techniques from security vulnerability descriptions," in *Proc. of ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD)*, 2021, pp. 2574–2583.

[8] A. Bivens, C. Palagiri, R. Smith, B. Szymanski, M. Embrechts *et al.*, "Network-based intrusion detection using neural networks," *Intelligent Engineering Systems through Artificial Neural Networks*, vol. 12, no. 1, pp. 579–584, 2002.

[9] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux, "API design for machine learning software: experiences from the scikit-learn project," in *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, 2013, pp. 108–122.

[10] 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 *Proc. of Int. Conf. on Information and Knowledge Management (CIKM)*, 2021, p. 3747–3756.

[11] W.-L. Chiang, X. Liu, S. Si, Y. Li, S. Bengio, and C.-J. Hsieh, "Cluster-GCN: An efficient algorithm for training deep and large graph convolutional networks," in *Proc. of ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD)*, 2019, pp. 257–266.

[12] K. H. Chow, U. Deshpande, S. Seshadri, and L. Liu, "SRA: smart recovery advisor for cyber attacks," in *Proc. of ACM Int. Conf. on Management of Data (SIGMOD)*, 2021, pp. 2691–2695.

[13] G. Duan, H. Lv, H. Wang, G. Feng, and X. Li, "Practical Cyber Attack Detection With Continuous Temporal Graph in Dynamic Network System," *IEEE Transactions on Information Forensics and Security (TIFS)*.

[14] Y. Ektefaie, G. Dasoulas, A. Noori, M. Farhat, and M. Zitnik, "Multimodal learning with graphs," *Nature Machine Intelligence*, vol. 5, no. 4, pp. 340–350, 2023.

[15] C. Feng and P. Tian, "Time series anomaly detection for cyber-physical systems via neural system identification and bayesian filtering," in *Proc. of ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD)*, 2021, pp. 2858–2867.

[16] P. Gao, F. Shao, X. Liu, X. Xiao, H. Liu, Z. Qin, F. Xu, P. Mittal, S. R. Kulkarni, and D. Song, "A system for efficiently hunting for cyber threats in computer systems using threat intelligence," in *Proc. of IEEE Int. Conf. on Data Engineering (ICDE)*, 2021, pp. 2705–2708.

[17] P. Gao, F. Shao, X. Liu, X. Xiao, Z. Qin, F. Xu, P. Mittal, S. R. Kulkarni, and D. Song, "Enabling efficient cyber threat hunting with cyber threat intelligence," in *Proc. of IEEE Int. Conf. on Data Engineering (ICDE)*, 2021, pp. 193–204.

[18] P. Gao, X. Xiao, Z. Li, K. Jee, F. Xu, S. R. Kulkarni, and P. Mittal, "A query system for efficiently investigating complex attack behaviors for enterprise security," *Proc. of the VLDB Endowment (PVLDB)*, vol. 12, no. 12, pp. 1802–1805, 2019.

[19] B. Ghojogh and A. Ghodsi, "Recurrent neural networks and long short-term memory networks: Tutorial and survey," *CoRR*, vol. abs/2304.11461, 2023.

[20] S. Givnan, C. Chalmers, P. Fergus, S. Ortega-Martorell, and T. Whalley, "Anomaly detection using autoencoder reconstruction upon industrial motors," *Sensors*, vol. 22, no. 9, p. 3166, 2022.

[21] A. Golczynski and J. A. Emanuello, "End-to-end anomaly detection for identifying malicious cyber behavior through NLP-based log embeddings," *CoRR*, vol. abs/2108.12276, 2021.

[22] H. Guo, S. Yuan, and X. Wu, "LogBERT: Log anomaly detection via BERT," in *Proc. of Int. Joint Conf. on Neural Networks (IJCNN)*, 2021, pp. 1–8.

[23] W. L. Hamilton, *Graph Representation Learning*, ser. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2020.

[24] X. Han, T. Pasquier, and M. Seltzer, "Provenance-based intrusion detection: opportunities and challenges," in *Proc. of USENIX Workshop on the Theory and Practice of Provenance (TaPP)*, 2018.

[25] H. He, X. Sun, H. He, G. Zhao, L. He, and J. Ren, "A novel multimodal-sequential approach based on multi-view features for network intrusion detection," *IEEE Access*, vol. 7, pp. 183 207–183 221, 2019.

[26] X. Hu, W. Gao, G. Cheng, R. Li, Y. Zhou, and H. Wu, "Toward early and accurate network intrusion detection using graph embedding," *IEEE Transactions on Information Forensics and Security (TIFS)*, vol. 18, pp. 5817–5831, 2023.

[27] N. Huang and S. Villar, "A short tutorial on the Weisfeiler-Lehman test and its variants," in *Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP)*, 2021, pp. 8533–8537.

[28] M. Joaristi and E. Serra, "SIR-GN: A fast structural iterative representation learning approach for graph nodes," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 15, no. 6, pp. 100:1–100:39, 2021.

[29] G. Karatas and O. K. Sahingoz, "Neural network based intrusion detection systems with different training functions," in *Proc. of Int. Symp. on Digital Forensic and Security (ISDFS)*, 2018, pp. 1–6.

[30] S. M. Kazemi, R. Goel, K. Jain, I. Kobyzev, A. Sethi, P. Forsyth, and P. Poupart, "Representation learning for dynamic graphs: A survey," *Journal of Machine Learning Research (JMLR)*, vol. 21, pp. 70:1–70:73, 2020.

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

[32] J. Kim, Y. Shin, E. Choi *et al.*, "An intrusion detection model based on a convolutional neural network," *Journal of Multimedia Information System (JMIS)*, vol. 6, no. 4, pp. 165–172, 2019.

[33] I. J. King and H. H. Huang, "Euler: Detecting network lateral movement via scalable temporal link prediction," *ACM Transactions on Privacy and Security (TOPS)*, vol. 26, no. 3, pp. 35:1–35:36, 2023.

[34] B. Lakha, S. L. Mount, E. Serra, and A. Cuzzocrea, "Anomaly detection in cybersecurity events through graph neural network and transformer based model: A case study with BETH dataset," in *Proc. of IEEE Int. Conf. on Big Data*, 2022, pp. 5756–5764.

[35] D. H. Lakshminarayana, J. Philips, and N. Tabrizi, "A survey of intrusion detection techniques," in *Proc. of IEEE Int. Conf. On Machine Learning and Applications (ICMLA)*, 2019, pp. 1122–1129.

[36] J. Layne, J. Carpenter, E. Serra, and F. Gullo, "Temporal SIR-GN: efficient and effective structural representation learning for temporal graphs," *Proc. of the VLDB Endowment (PVLDB)*, vol. 16, no. 9, pp. 2075–2089, 2023.

[37] J. Layne and E. Serra, "Inferential SIR-GN: scalable graph representation learning," *CoRR*, vol. abs/2111.04826, 2021.

[38] H. Liao, C. R. Lin, Y. Lin, and K. Tung, "Intrusion detection system: A comprehensive review," *Journal of Network and Computer Applications*, vol. 36, no. 1, pp. 16–24, 2013.

[39] O. Linda, T. Vollmer, and M. Manic, "Neural network based intrusion detection system for critical infrastructures," in *Proc. of Int. Joint Conf. on Neural Networks (IJCNN)*, 2009, pp. 1827–1834.

[40] H. Liu and B. Lang, "Machine learning and deep learning methods for intrusion detection systems: A survey," *Applied Sciences*, vol. 9, no. 20, 2019.

[41] W. W. Lo, S. Layeghy, M. Sarhan, M. Gallagher, and M. Portmann, "E-GraphSAGE: A graph neural network based intrusion detection system for IoT," in *Proc. of IEEE/IFIP Network Operations and Management Symposium (NOMS)*, 2022, pp. 1–9.

[42] X. Ma, J. Wu, S. Xue, J. Yang, C. Zhou, Q. Z. Sheng, H. Xiong, and L. Akoglu, "A comprehensive survey on graph anomaly detection with deep learning," *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, vol. 35, no. 12, pp. 12 012–12 038, 2023.

[43] A. Macrae, "Identifying threats in real time," *Network Security*, vol. 2013, no. 11, pp. 5–8, 2013.
[44] B. Mali, "Comparison of the statistical and autoencoder approach for anomaly detection in big data," in *2024 5th International Conference on Big Data Analytics and Practices (IBDAP)*, 2024, pp. 22–25.
[45] N. Merrill and A. Eskandarian, "Modified autoencoder training and scoring for robust unsupervised anomaly detection in deep learning," *IEEE Access*, vol. 8, pp. 101 824–101 833, 2020.
[46] S. Nedelkoski, J. Bogatinovski, A. Acker, J. Cardoso, and O. Kao, "Self-attentive classification-based anomaly detection in unstructured logs," in *Proc. of IEEE Int. Conf. on Data Mining (ICDM)*, 2020, pp. 1196–1201.
[47] Y. N. Nguimbous, R. Ksantini, and A. Bouhoula, "Anomaly-based intrusion detection using auto-encoder," in *2019 International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, 2019, pp. 1–5.
[48] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Z. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An imperative style, high-performance deep learning library," in *Proc. of Conf. on Advances in Neural Information Processing Systems (NeurIPS)*, 2019, pp. 8024–8035.
[49] R. Paudel and H. H. Huang, "Pikachu: Temporal walk based dynamic graph embedding for network anomaly detection," in *Proc. of IEEE/IFIP Network Operations and Management Symposium (NOMS)*, 2022, pp. 1–7.
[50] C. Peng, J. He, and F. Xia, "Learning on multimodal graphs: A survey," 2024. [Online]. Available: https://arxiv.org/abs/2402.05322
[51] D. Pujol-Perich, J. Suárez-Varela, A. Cabellos-Aparicio, and P. Barlet-Ros, "Unveiling the potential of graph neural networks for robust intrusion detection," *SIGMETRICS Performance Evaluation Review*, vol. 49, no. 4, pp. 111–117, 2022.
[52] M. U. Rehman, H. Ahmadi, and W. U. Hassan, "FLASH: A comprehensive approach to intrusion detection via provenance graph representation learning," in *Proc. of IEEE Symp. on Security and Privacy (SP)*, 2024, pp. 139–139.
[53] R. A. Rossi, D. Jin, S. Kim, N. K. Ahmed, D. Koutra, and J. B. Lee, "On proximity and structural role-based embeddings in networks: Misconceptions, techniques, and applications," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 14, no. 5, pp. 63:1–63:37, 2020.
[54] K. A. Scarfone and P. M. Mell, "SP 800-94. Guide to intrusion detection and prevention systems (IDPS)," Tech. Rep., 2007.
[55] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," in *Proc. of Int. Conf. on Information Systems Security and Privacy (ICISSP)*, 2018, pp. 108–116.
[56] R. Sharp, *Introduction to Cybersecurity - A Multidisciplinary Challenge*, ser. Undergraduate Topics in Computer Science. Springer, 2024.
[57] K. Shortridge and J. Dykstra, "Opportunity cost and missed chances in optimizing cybersecurity," *Communications of the ACM (CACM)*, vol. 66, no. 7, pp. 96–104, 2023.
[58] S. Sriram, A. Shashank, R. Vinayakumar, and K. Soman, "DCNN-IDS: deep convolutional neural network based intrusion detection system," in *Proc. of Int. onf. on Computational Intelligence, Cyber Security and Computational Models (ICC3)*, 2020, pp. 85–92.
[59] N. N. Tran, R. A. Sarker, and J. Hu, "An approach for host-based intrusion detection system design using convolutional neural network," in *Proc. of Int. Conf. on Mobile Networks and Management (MONAMI)*, 2017, pp. 116–126.
[60] F. Ullah, S. Ullah, G. Srivastava, and J. C.-W. Lin, "Ids-int: Intrusion detection system using transformer-based transfer learning for imbalanced network traffic," *Digital Communications and Networks*, vol. 10, no. 1, pp. 190–204, 2024.
[61] S. Wang, Z. Wang, T. Zhou, H. Sun, X. Yin, D. Han, H. Zhang, X. Shi, and J. Yang, "THREATRACE: detecting and tracing host-based threats in node level through provenance graph learning," *IEEE Transactions on Information Forensics and Security (TIFS)*, vol. 17, pp. 3972–3987, 2022.
[62] B. Weisfeiler and A. A. Lehman, "The reduction of a graph to canonical form and the algebra which appears therein," *Nauchno-Technicheskaya Informatsia*, vol. 2, no. 9, pp. 12–16, 1968.
[63] Z. Wu, H. Zhang, P. Wang, and Z. Sun, "RTIDS: A robust transformer-based approach for intrusion detection system," *IEEE Access*, vol. 10, pp. 64 375–64 387, 2022.
[64] Y. Xie, D. Feng, Y. Hu, Y. Li, S. Sample, and D. Long, "Pagoda: A hybrid approach to enable efficient real-time provenance based intrusion detection in big data environments," *IEEE Transactions on Dependable and Secure Computing*, vol. 17, no. 6, pp. 1283–1296, 2018.
[65] J. Xu, X. Shu, and Z. Li, "Understanding and Bridging the Gap Between Unsupervised Network Representation Learning and Security Analytics," in *Proc. of IEEE Symp. on Security and Privacy, (SP)*, 2024, pp. 3590–3608.
[66] W. Xu, J. Jang-Jaccard, A. Singh, Y. Wei, and F. Sabrina, "Improving performance of autoencoder-based network anomaly detection on nsl-kdd dataset," *IEEE Access*, vol. 9, pp. 140 136–140 146, 2021.
[67] R. Yasaei, Y. Moghaddas, and M. A. Al Faruque, "Iot-graf: Iot graph learning-based anomaly and intrusion detection through multi-modal data fusion," in *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2024, pp. 1–6.
[68] S. Zheng, Z. Zhu, Z. Liu, Z. Guo, Y. Liu, Y. Yang, and Y. Zhao, "Multi-modal graph learning for disease prediction," *IEEE Transactions on Medical Imaging*, vol. 41, no. 9, pp. 2207–2216, 2022.
[69] M. Zipperle, F. Gottwalt, E. Chang, and T. S. Dillon, "Provenance-based intrusion detection systems: A survey," *ACM Computing Surveys (CSUR)*, vol. 55, no. 7, pp. 135:1–135:36, 2023.

Bishal Lakha is a PhD in Computing (Computer Science) student at Boise State University, Boise, Idaho, USA. He received his MS in Computer Science from Boise State and B.Eng degree in Electronics and Communication Engineering from IOE Pulchowk Campus, Tribhuvan University, Kathmandu, Nepal. His current research interests include Graph ML, xAI and Anomaly Detection.

Janet Layne is a PhD student in Computing at Boise State University. She has a MS in Biology, also from Boise State. Her current research focus is in graph machine learning, with her most recent work published in VLDB. Previous research included genetic modifications to crop organisms, and the utility of nanotechnology to cancer applications.

Edoardo Serra received his Ph.D. degree in Computer Science Engineering from the University of Calabria, Italy, in 2012. After his PhD, he was a Research Associate at the University of Maryland (till August 2015). From August 2015 to July 2021, he was an Assistant Professor in the computer science department at Boise State University. From August 2021 he is an Associate Professor in the computer science department at Boise State University. Moreover, since June 2020, he has joint position as a senior scientist at Pacific Northwest National Laboratory. His research interests are in the field of data science with applications in cyber and national security.

Francesco Gullo is an associate professor of computer science at the University of L'Aquila (Italy), in the Department of Information Engineering, Computer Science, and Mathematics (DISIM). He received his PhD, in "Computer and Systems Engineering", from the University of Calabria (Italy), in 2010. During his PhD, he was an intern at the George Mason University (US), and a teaching/research assistant at the University of Catanzaro (Italy). After his graduation, he was a postdoc at the University of Calabria (Italy), a postdoc and a research scientist at the Yahoo Labs (Spain), a research scientist at the Fundacio Barcelona Media (Spain), and a senior associate researcher at the UniCredit banking group (Italy). His research falls into the broad areas of artificial intelligence and data science, with emphasis on algorithmic aspects. His recent interests include graph machine learning, graph data management, natural language processing, and trustworthy AI. His research has been published in premier venues such as SIGMOD, VLDB, KDD, WWW, ICDM, CIKM, EDBT, WSDM, ECML-PKDD, SDM, TODS, TKDE, TKDD, MACH, DAMI, JCSS, TNSE, PR. He has also been serving the scientific community: he was/is/will be Associate Editor of EPJ Data Science JDSA journals, Finance Chair of CIKM'24, Workshop Chair of KDD'24 and ICDM'16, Industry Track Program co-Chair of ASONAM'24, Program co-Chair of MIDAS workshop @ECML-PKDD['16-'24], MultiClust symposium @SDM'14, MultiClust workshop @KDD'13, 3Clust workshop @PAKDD'12), as well as (senior) program-committee member of major conferences, including SIGMOD, ICDE, KDD, WWW, IJCAI, AAAI, CIKM, SIGIR, ICDM, WSDM, SDM, ECML-PKDD, ECAI, ICWSM.

Suhil Jajodia is currently a University Professor, a BDM International Professor, and the Center for Secure Information Systems Director at George Mason University. Prior to joining Mason, he held permanent positions at NSF, NRL, and the University of Missouri-Columbia. He has sustained a highly active research agenda spanning database and cyber security for over 30 years. He has authored or coauthored 7 books, edited 52 books and conference proceedings, and published more than 500 technical articles in refereed journals and conference proceedings. He holds 23 U.S. patents and has received a number of prestigious awards in recognition of his research accomplishments. According to Google Scholar, he has over 54,000 citations, and his H-index is 116.