IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 24 April 2023, accepted 9 May 2023, date of publication 12 May 2023, date of current version 24 May 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3275789

SURVEY

Graph Neural Networks for Intrusion Detection: A Survey

TRISTAN BILOT[1,2,3], NOUR EL MADHOUN[3,4], KHALDOUN AL AGHA[2], AND ANIS ZOUAOUI[1]
[1]Iriguard, 92800 Puteaux, France
[2]Laboratoire Interdisciplinaire des Sciences du Numérique, CNRS, Université Paris-Saclay, 91190 Gif-sur-Yvette, France
[3]LISITE Laboratory, ISEP, 92130 Issy-les-Moulineaux, France
[4]LIP6, CNRS, Sorbonne Université, 75005 Paris, France

Corresponding author: Tristan Bilot (tristan.bilot@universite-paris-saclay.fr)

This work was supported by the ISEP (Institut Supérieur d'Electronique de Paris).

ABSTRACT Cyberattacks represent an ever-growing threat that has become a real priority for most organizations. Attackers use sophisticated attack scenarios to deceive defense systems in order to access private data or cause harm. Machine Learning (ML) and Deep Learning (DL) have demonstrate impressive results for detecting cyberattacks due to their ability to learn generalizable patterns from flat data. However, flat data fail to capture the structural behavior of attacks, which is essential for effective detection. Contrarily, graph structures provide a more robust and abstract view of a system that is difficult for attackers to evade. Recently, Graph Neural Networks (GNNs) have become successful in learning useful representations from the semantic provided by graph-structured data. Intrusions have been detected for years using graphs such as network flow graphs or provenance graphs, and learning representations from these structures can help models understand the structural patterns of attacks, in addition to traditional features. In this survey, we focus on the applications of graph representation learning to the detection of network-based and host-based intrusions, with special attention to GNN methods. For both network and host levels, we present the graph data structures that can be leveraged and we comprehensively review the state-of-the-art papers along with the used datasets. Our analysis reveals that GNNs are particularly efficient in cybersecurity, since they can learn effective representations without requiring any external domain knowledge. We also evaluate the robustness of these techniques based on adversarial attacks. Finally, we discuss the strengths and weaknesses of GNN-based intrusion detection and identify future research directions.

INDEX TERMS Cyberattacks, cybersecurity, deep learning (DL), graph neural networks (GNNs), intrusion detection (IDS), machine learning (ML).

I. INTRODUCTION

Cyberattacks are ubiquitous in our daily lives and their detection has become a priority for all infrastructures relying on information systems. These attacks have evolved both in quantity and complexity, resulting in many organizations using legacy cyberdefense systems that may not prevent from such attacks. Enterprise networks are even more complicated to fully secure due to the multiplicity of devices connected to the internet that create important attack vectors. Identifying attacks before or during their execution thus becomes a necessity to preserve the privacy of data and maintain the proper functioning of computer systems. Intrusions in such systems can have severe consequences, ranging from financial losses to the theft of sensitive information or even critical infrastructure disruptions. Throughout the years, the field of cybersecurity has followed this technical progress by implementing new innovative cyberdefense tools. For a long time, rule-based detection systems were integrated in antiviruses and Intrusion Detection Systems (IDS) to detect network- and system-level intrusions from previously known patterns. The limitation of these methods lies in the fact that they poorly generalize to new patterns and only a few modifications to the attack sequence suffice to bypass the rule or signature detection. To counter these issues, Machine Learning (ML) has seen great success in cybersecurity tasks

The associate editor coordinating the review of this manuscript and approving it for publication was Prakasam Periasamy.

like intrusion detection [1], [2], [3] and malware detection [4], [5]. The strength of these techniques lies in their ability to yield satisfactory results across various scenarios, using a small quantity of data along with proper feature engineering. However, this feature engineering part is very error-prone, since it requires domain experts for the selection of useful features that can be used in downstream detection systems.

On the contrary, Deep Learning (DL) makes the learning process more efficient as the features are learned from the model itself, leading to meaningful features to detect complex cyberattacks such as intrusions. Some researchers prefer considering the intrusion detection task as anomaly detection, by leveraging unsupervised approaches based on Deep Belief Network (DBN) [6] or multiple variants of Auto Encoder (AE) [7], whereas others leverage supervised approaches usually based on Convolutional Neural Network (CNN) [8], Recurrent Neural Network (RNN) [9] or traditional Multi Layer Perceptron (MLP) [3], [10], [11].

Although these models achieve remarkable results in learning new patterns, they are trained from inherently flat data structures such as vectors or grids. These simple structures fail to capture complex structural patterns essential to the detection of Advanced Persistent Threats (APTs) and 0-day attacks. Indeed, such threats are often characterized by new attack patterns involving weak signals, which elapse for an arbitrary period of time. These signals are challenging to capture using traditional Deep Learning methods on raw and flat data, as such representations are not robust enough to transcend the obfuscated behaviors of attackers. Graphs are universal representations that offer a high-level and abstract view of a system which is much harder to evade by attackers. Indeed, graphs are expressive structures that provide even more semantic in contrast to flat data, that loose valuable relational information, ubiquitous in cyberattack scenarios. Indeed, intrusions are essentially characterized by a sequence of suspicious and benign interactions between entities such as hosts in a network or processes in a host system. When represented as a graph, these interactions can be learned by Graph Representation Learning (GRL) models such as Graph Neural Networks (GNNs), that have achieve promising results in a variety of cybersecurity tasks such as vulnerability detection [12], [13], threat intelligence [14], [15] or malware detection [16], [17]. GRL and GNN methods have also been successfully applied to intrusion detection, and to the best of our knowledge, there exists no paper reviewing this topic yet.

This survey aims to give a comprehensive review of existing techniques employed in intrusion detection with graph-structured data and GNNs. We believe that the application of GRL and GNNs to intrusion detection is a promising research direction and hope that this survey will offer a concise and normalized view of recent techniques for future research. Our contribution is threefold:

- We first introduce the different types of graphs used in graph-based intrusion detection, along with the random walk and GNN models commonly employed. We also provide a general architecture that summarizes most approaches based on GNN for intrusion detection.
- To facilitate the easy comparison of existing works in the field, we have classified the state-of-the-art papers and provided a comprehensive review of current methods and datasets for GNN-based intrusion detection. For both network and system data, we also present the graph structures that can be used, along with their strength and weaknesses.
- Finally, we study the robustness of GNN classifiers based on adversarial attacks and further discuss the strengths and limitations of using these techniques. We also provide recommendations for future research to improve the field.

The content of this survey is organized as follows. Section II introduces background on graphs, GRL and GNN models. Sections III and IV respectively review the state-of-the-art literature on GNNs applied to network-based and host-based intrusion detection. Section V focuses on adversarial attacks that may be led against these GNN-based methods. Finally, Sections VI and VII are devoted to future research ideas and conclusion.

II. BACKGROUND

A. REPRESENTATION LEARNING ON GRAPHS

Graph Representation Learning involves extracting informative features, known as embeddings, from a graph. Depending on the specific method used for representation learning, different information can be captured in the embeddings, such as node attributes or structural properties of the graph. By leveraging these learned embeddings, downstream tasks can benefit from the rich semantic contained in the graph structure, even when faced with incomplete or noisy data. A general approach is to compute node embeddings, namely find a differentiable mapping function that projects nodes to a fixed-size embedding vector in a manner that similar nodes are embedded close together in the embedding space. Other approaches are dedicated to compute edge-wise and graph-wise embeddings, depending on the use case. These techniques can be applied to different types of graphs, described in this section and summarized in Figure 1.

Graph: A graph is simply represented as $G = (V, E)$ where V is a set of nodes (or entities) and E is a set of edges (or relations) between those nodes.

Heterogeneous Graph: A heterogeneous graph contains multiple types of nodes and edges. Formally, it can be defined as $G = (V, E, \phi_v, \phi_e)$, where ϕ_v and ϕ_e are functions that respectively map each node and each edge to a type.

Attributed Graph: The attributed graph attaches attributes to the elements of the graph (i.e. nodes and/or edges). A node-attributed graph is described as $G = (V, E, X_v)$, where X_v is a mapping function that maps each node $v \in V$ to a list of attributes. Similarly, an edge-attributed graph is described as $G = (V, E, X_e)$, where X_e maps each edge $e \in E$ to a list of attributes.

Spatio-Temporal Graph: A spatio-temporal graph is a graph with static structure and time-varying features. Formally, it can be described as $G_t = (V, E, X_{v,t}, X_{e,t})$, where $X_{v,t}$ and $X_{e,t}$ are functions that respectively map each node $v \in V$ and each edge $e \in E$ to a feature vector that represents the attributes of each element at timestamp t.

Dynamic Graph: The dynamic graph models the time-varying structure along with the time-varying features in the graph. Indeed, dynamic graphs associate every edge and/or node to a timestamp representing the time of the event (e.g. add, remove or update a node/edge). A majority of papers use Discrete-Time Dynamic Graphs (DTDGs), where the evolving graph is modeled by a sequence of S graph snapshots $G_1, \ldots, G_S$ where G_t is a snapshot of the graph captured at timestamp t. Formally, it can be described as $G_t = (V_t, E_t, X_{v,t}, X_{e,t})$, where V_t and E_t represent the graph structure at each timestamp t, whereas $X_{v,t}$ and $X_{e,t}$ represent the time-varying features. Other approaches leverage Continuous-Time Dynamic Graphs (CTDGs), which associate an individual timestamp to each event observed in the graph, resulting in a new snapshot created for each event. CTDGs provide a more fine-grained representation of events compared to DTDGs but are still little used in literature.

When learning from graphs, two approaches are commonly employed:

- **Inductive learning**: in this approach, the goal is to learn from multiple graphs and then generalize to new graphs. Similarly as in traditional ML, the model is trained on a set of graphs and then predict labels on new graphs, made of nodes and edges not seen during training. In this case, the train, validation and test sets are composed of different graphs.
- **Transductive learning**: in this setting, the predictions are taking place on a single large graph and cannot generalize to new graphs. Consequently, the model can predict labels solely from nodes and edges that have been seen during training. In contrast to the inductive setting, the train, validation and test sets cannot be divided into different graphs as a single graph is available here. Each set is thus composed of nodes and edges, and during training, the features and labels from the training nodes will be gathered. However, GNNs essentially aggregate the features from neighbors, leading to a potential aggregation of nodes from the validation and test sets during the training. In this case, it should be considered that the nodes in the training set could benefit from the features of the nodes in the other sets.

In the special case of intrusion detection, an inductive detection model may be able to generalize on new enterprise networks or new host machines, whereas a transductive detection model may be trained on a specific network with a fix set of hosts, or on a specific host machine. The former model has the ability to perform inference in new situations given its pre-trained weights, whereas the latter needs to be entirely retrained if the graph changes.

B. RANDOM WALK-BASED LEARNING

Random walks have been successfully applied to a large range of domains such as recommender systems and computer vision [18]. Despite the current success of recent GNNs, random walks are still used in recent graph-based intrusion detection works [19], [20], [21], [22]. Indeed, these algorithms demonstrate strong capabilities at capturing graph information and node co-occurrence relations while using self-supervised embedding techniques, namely the graph structure and features are used as the label for the predictive task. In this section, we focus on two major random walk-based embedding techniques used in intrusion detection: DeepWalk [23] and node2vec [24]. We then present how variants of these models can be applied to more complex graphs, and discuss the challenges induced by these methods.

1) RANDOM WALK ON GRAPHS

DeepWalk: Given a starting node, a neighbor node is randomly selected to continue the random walk, following a uniform distribution probability. This step is repeated K times to create a random walk of length K. To capture full graph information, each node present in the graph is considered as a starting node for a random walk. In total, $N \times R$ random walks are generated, where N is the number of nodes and R is a hyperparameter indicating the number of random walks per node. To create embeddings, DeepWalk builds a co-occurrence list L where all nodes that co-occur in the random walk are present. For any tuple present in L, one node is defined as the center node v_i and the other as the context node v_j. The node embeddings are then created using the Skip-gram model [25], by trying to reconstruct v_j based on v_i, namely by maximizing the probability of observing v_j in the context of node v_i. Using this process, close nodes tend to co-occur in many random walks and their low-dimensional vector representation tends to be similar in latent space.

node2vec: In DeepWalk, the construction of random walks is limited by the random selection of co-occurring nodes. node2vec proposes an improvement by introducing a second-order biased random walk which captures local and global structures using Breadth First Search (BFS) and Depth First Search (DFS). Two hyperparameters p and q can be tuned in order to respectively control the likelihood of immediately revisiting a node and the traversing behavior.

2) RANDOM WALK ON ATTRIBUTED GRAPHS

Previous methods only capture the structural information, without taking into account the features that may be stored in the nodes/edges of the graph. Such features can provide further information such as text or values, making the embedding process aware of these attributes. Paper [26] proposed Text-Associated DeepWalk (TADW), an improvement to DeepWalk that considers the text information stored in nodes. Another model inspired by DeepWalk is Max-Margin DeepWalk (MMDW) [27], which computes embeddings in a semi-supervised way by leveraging node labeling information. Planetoid [28] is another approach that consists of a transductive and an inductive variant where the graph structure and node features are leveraged together to learn the embeddings.

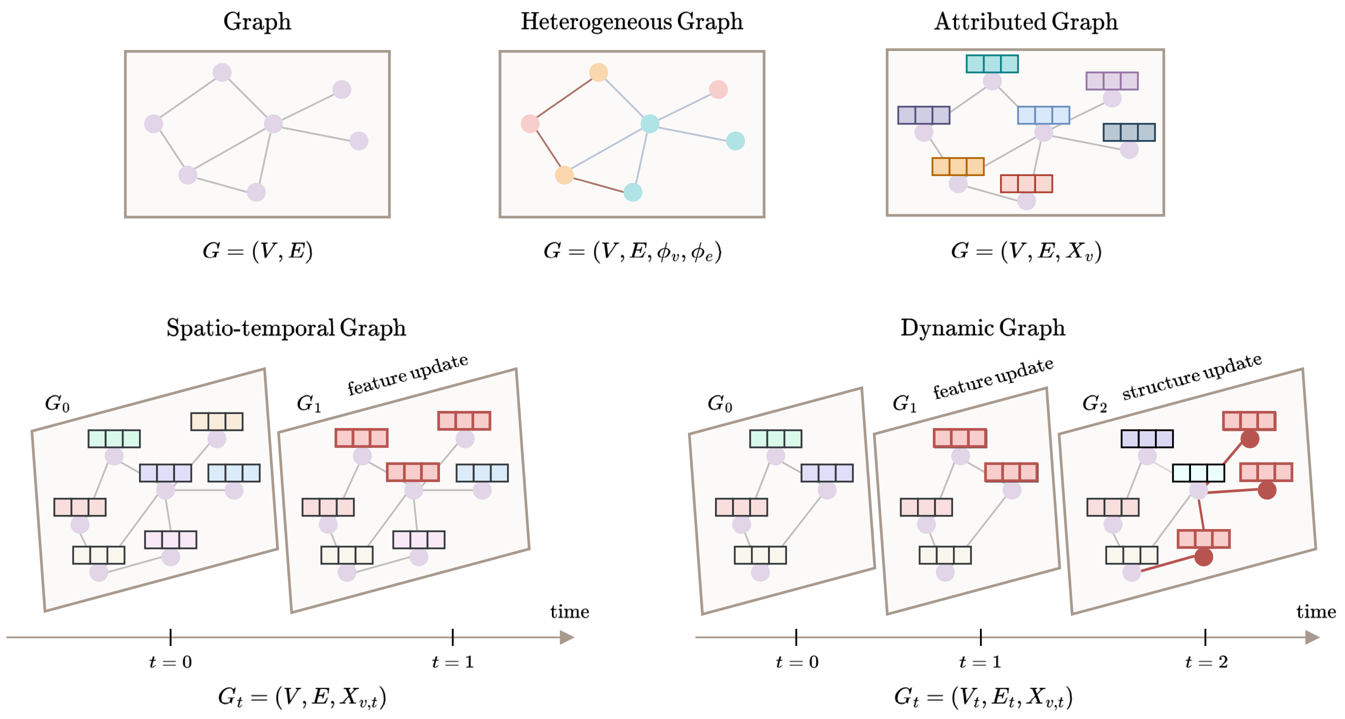


FIGURE 1. Representation of graph structures used in graph-based intrusion detection. A graph consists in a structure made of nodes interconnected by edges, whereas heterogeneous graphs refer to graphs with multiple types of nodes/edges, and attributed graphs assign a vector of features to nodes/edges. In this illustration, we present a view of node-attributed graphs. Spatio-temporal and dynamic graphs leverage an additional temporal dimension, where the former updates its features with respect to time, and the latter can update both its features and structure.

3) RANDOM WALK ON DYNAMIC GRAPHS

In many real-world scenarios including cybersecurity, graphs may be inherently dynamic. Generally, such graphs are modeled by adding a temporal dimension that contains a stream of discrete graph snapshots taken at different time intervals. A temporal random walk can then cross through dynamic graphs by walking into this temporal dimension in an increasing order of time. Formally, it can be described as a sequence of nodes $(v_{t_1}, v_{t_2}, \ldots, v_{t_n})$ with $t_1 \leq t_2 \leq \ldots \leq t_n$ where t represents a timestamp. Following the same idea, authors in [29] leverage temporal random walks to capture temporal relations with a high granularity. Paper [30] proposes an online learning approach to continually learn embeddings using temporal random walks and in [31], an inductive method named Causal Anonymous Walk (CAW) is introduced to extract causal anonymous walks from temporal random walks.

4) CHALLENGES WITH RANDOM WALK-BASED METHODS

Node representation learning with random walks has been a powerful method to capture the structure of graphs and represent it in embedding space. However, most of these techniques are usually challenging to apply in an inductive setting and they do not share parameters between nodes [32]. Furthermore, these methods are highly dependent on the hyperparameters and tend to prioritize proximity information above structural information [33].

In the next section, we discuss how Graph Neural Networks can bypass these limitations and be applied on large graphs and in an inductive setting.

C. GRAPH NEURAL NETWORK-BASED LEARNING

Learning on graphs is a challenging task but has been proven to be successful in a number of complex tasks. The popularity of deep learning has allowed the emergence of new methods involving spectral and spatial convolutions applied to graph structures, making it possible to benefit from both the expressive structure of graphs and the power of representation learning. Spectral GNNs such as ChebNet [34] leverage Laplacian matrix eigen decomposition in Fourier space to analyze the underlying structure of the graph. Contrarily, spatial GNNs work directly on the adjacency matrix and capture local neighborhood of nodes in graph domain, which avoids the time-consuming switch in spectral domain. Furthermore, spectral methods tend to be less used as they are inherently transductive [35], meaning that they cannot generalize to unseen nodes or edges. We summarize in this section some of the fundamental spatial GNNs that are commonly used and that should be understood before diving into their applications to intrusion detection.

1) ORIGINAL GNN MODEL

The Graph Neural Network (GNN) model [36], [37] is the origin of the first application of deep neural networks to graph-structured data. Although deep learning on graphs has only been recently democratized, the first GNN [36] dates back to 2005 and is originally inspired from Recurrent Neural Networks (RNNs). The authors explain that RNNs can directly process graph data but they are limited to graph-focused tasks on directed acyclic graphs. In response to this problem, the

GNN model can be applied to directed, undirected, labeled and cyclic graphs using the concept of neighbor information propagation. More precisely, every node shares its information with its local neighborhood until convergence to a certain equilibrium. This model can be formalized as:

$$\mathbf{h}_u = f(\mathbf{x}_u, \mathbf{x}_v, \mathbf{h}_v),\ \ u \in V, v \in N(u), \tag{1}$$

where h_u corresponds to the embedding of node u, $\mathbf{x}_i$ is the feature vector of a given node i, $N(u)$ represents the neighboring nodes of u, and f is a parametric function that extracts the dependencies between a node and its neighborhood.

2) MESSAGE-PASSING NEURAL NETWORK

More recently, authors in [38] proposed a general framework to encapsulate common state-of-the-art GNN models. As indicated by the name, Message-Passing Neural Networks (MPNNs) exchange messages between nodes and their neighbors. The framework is essentially made of two steps: message-passing and readout.

First, message-passing can be described as a function M which computes a message for each node by gathering its previous embedding along with its neighbor embeddings:

$$\mathbf{m}_u^{t+1} = \sum_{v \in \mathcal{N}(u)} M^t\left(\mathbf{h}_u^t, \mathbf{h}_v^t, \mathbf{e}_{(u,v)}\right),\ \ \mathbf{e}_{(u,v)} \in E, \tag{2}$$

where $\mathbf{m}_u^{t+1}$ represents the messages collected at layer $t+1$, $\mathcal{N}(u)$ is the list of neighbors of u, $\mathbf{h}_u^t$, $\mathbf{h}_v^t$ are the embeddings of nodes u and v at layer t and $\mathbf{e}_{(u,v)}$ is the edge feature vector of edge (u, v). The message function M is generally a differentiable function such as a neural network taking as input the concatenation of node and edge embeddings [39].

The computed message is then leveraged by an update function U to update the new representation of a node:

$$\mathbf{h}_u^{t+1} = U^t\left(\mathbf{h}_u^t, \mathbf{m}_u^{t+1}\right), \tag{3}$$

where function U could be a neural network such as a Gated Recurrent Unit (GRU) [40].

Finally, the readout step consists in transforming all node embeddings into a fixed-size vector $\hat{y}$ which captures the global information in the graph:

$$\hat{y} = R\left(\left\{\mathbf{h}_u^T \mid u \in V\right\}\right), \tag{4}$$

where R is a permutation-invariant readout function like a sum. The framework can become even more general by replacing the sum operation in Eq. 2 by an arbitrary aggregation function.

3) CONVOLUTIONS ON GRAPHS WITH GCN

The Graph Convolutional Network (GCN) model [41] is a generalization of convolutions on graphs, where node embeddings are learned from node features along with the graph structure. One GCN layer, presented in Eq. 5, corresponds to the aggregation of the 1-hop neighborhood of every node in the graph, passed into a neural network such that

$$H^{t+1} = \sigma\left(\mathcal{A}H^tW^t\right), \tag{5}$$

where σ is the ReLU activation, H^t and W^t are respectively the node embeddings and learnable parameters matrices at layer t, with $H^0 = X$, the initial node feature matrix. Normalized adjacency matrix $\mathcal{A}$ is created after applying a re-normalization trick to the undirected adjacency matrix A in order to avoid numerical instabilities and exploding gradients due to the eigenvalues range:

$$\mathcal{A} = \tilde{D}^{-\frac{1}{2}}(A+I)\tilde{D}^{-\frac{1}{2}}, \tag{6}$$

where $A+I$ is the adjacency matrix with self-loops and $\tilde{D}$ is the degree matrix of $A+I$. A visual representation of GCN and following models is provided in Figure 2.

4) SCALING TO LARGE GRAPHS WITH GRAPHSAGE

In traditional ML, the samples selected from the dataset are usually considered statistically independent. This assumption makes it possible to train models on large datasets by dividing the dataset into chunks that can be trained independently using mini-batches. Contrarily, nodes in a graph are inherently connected and are thus not considered independent and uniformly distributed. This makes the GNNs harder to be scaled on very large graphs with billions of nodes or edges. Indeed, models like GCN and GAT require the storage of the whole adjacency matrix with corresponding features into memory, which makes these models unusable on very large graphs.

GraphSAGE [42] was the first model to address the scalability issue by sampling a fixed number of nodes s from the neighborhood of a given node u, rather than using the full neighborhood. This sampling operation is denoted by:

$$\mathcal{N}_s(u) = \text{SAMPLE}(\mathcal{N}(u), s), \tag{7}$$

where $\mathcal{N}_s(u)$ corresponds to the uniformly sampled neighborhood of node u and s is the number of samples to select from the full neighborhood $\mathcal{N}(u)$. By doing so, GraphSAGE reduces the computational and memory requirements of the model while preserving its ability to capture the structural information of the graph.

To compute node embeddings using the sampled neighbors, GraphSAGE employs two operations. The first operation is the aggregation of the neighbors' embeddings, denoted by

$$\mathbf{h}_{\mathcal{N}_s(u)}^t = \text{AGGREGATE}_t\left(\left\{\mathbf{h}_v^{t-1}, \forall v \in \mathcal{N}_s(u)\right\}\right), \tag{8}$$

where $\mathbf{h}_{\mathcal{N}_s(u)}^t$ represents the aggregated neighborhood and the aggregation function AGGREGATE_t can be a weighted sum, a max-pooling, or a mean-pooling, among others.

The second operation consists in updating the embedding of u using its previous embedding and the aggregation of the neighbors' embeddings:

$$\mathbf{h}_u^t = \sigma\left(\mathbf{W}^t \cdot \left[\mathbf{h}_u^{t-1}, \mathbf{h}_{\mathcal{N}_s(u)}^t\right]\right), \tag{9}$$

where $\mathbf{h}_u^t$ is the node embedding of node u at layer t, $\mathbf{W}^t$ is a weight matrix, and [., .] is the concatenation operation. The fixed neighborhood sampling strategy used in GraphSAGE has several advantages such as mini-batch training and inductive training, which make this model adapted to large graphs with changing structures.

5) ATTENTION APPLIED TO GRAPHS WITH GAT

Traditional GNNs such as the GCN, typically weigh the embeddings of a node's neighbors during aggregation by their node degree, which is computed using the degree matrix D. While this approach helps to stabilize the aggregation process during training, it assumes that each node equally attends to each of its neighbors, which may not be optimal in some cases. To address this limitation, the Graph Attention Network (GAT) [43] was introduced. GAT uses the attention mechanism, originally proposed in Natural Language Processing (NLP) [44], to learn an importance weight for each neighbor of a given node. By attending to the most relevant neighbors, GAT is able to capture more fine-grained relationships between nodes, leading to better communication and higher quality aggregation of information. Formally, the attention weight α_{uv} between two nodes u and v can be computed as follows:

$$\alpha_{uv} = \frac{\exp(\sigma\left(\mathbf{a}^T\left[\mathbf{W}h_u, \mathbf{W}h_v\right]\right))}{\sum_{k\in\mathcal{N}(u)}\exp(\sigma\left(\mathbf{a}^T\left[\mathbf{W}h_u, \mathbf{W}h_k\right]\right))}, \tag{10}$$

where a and W respectively represent a weight attention vector and a shared weight matrix. h_u and h_v denote the hidden representations of nodes u and v, respectively, whereas $\mathcal{N}(u)$ is the full neighborhood of u and σ corresponds to the LeakyReLU activation function. Softmax is used here to map the weights to a distribution of $|\mathcal{N}(u)|$ possible outcomes.

Ultimately, the final node representation h'_u is computed by combining neighbor embeddings with the corresponding attention weights:

$$h'_u = \sigma\left(\sum_{v\in\mathcal{N}(u)} \alpha_{uv}\mathbf{W}h_v\right). \tag{11}$$

For training stability reasons, multi-head attention may also be used to calculate K independent attention weights in parallel.

6) ATTENTION IN HETEROGENEOUS GRAPHS WITH HAN

Real-world data are diverse and complex, with a wide range of entities and connections that traditional attributed graphs may not adequately represent. To address this, researchers are increasingly turning to heterogeneous graph learning for attack detection [45], [46], [47], [48]. With heterogeneous graphs containing multiple types of objects, specific methods are needed to handle them. Meta-paths, which are sequences of node and edge types that capture specific semantics in the graph, are often used in heterogeneous graphs due to their heterogeneous composition and strong semantic extraction capabilities. The Heterogeneous Graph Attention Network (HAN) [49] uses a similar attention mechanism to the original GAT to aggregate information from meta-paths. More specifically, two attention mechanisms are learned: a node-level

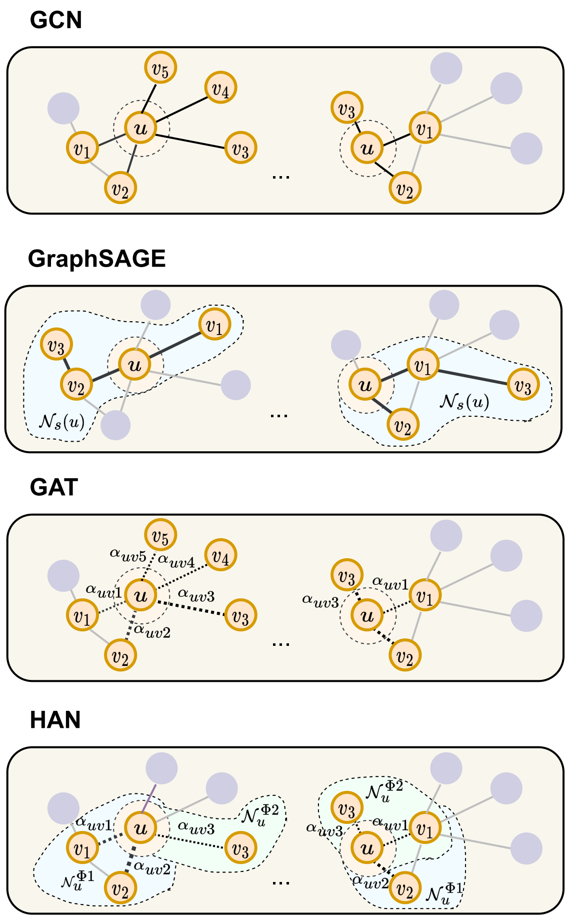


FIGURE 2. Illustration of common GNNs used in literature. GCN aggregates its 1-hop neighborhood; GraphSAGE samples a fixed number of neighbors and aggregates them; GAT leverages the attention mechanism to weight neighbors depending on their importance; HAN applies attention to heterogeneous graphs by learning attention weights between nodes from a same meta-path.

attention, that consists in learning the importance between a node and its meta-path-based neighbors, and a semantic-level attention, that learns the importance weight to give to each meta-path. In a meta-path Φ, the importance of a node j for a node i can be calculated with the node-level attention α_{ij}^{Φ} described as:

$$\alpha_{ij}^{\Phi} = \frac{\exp\left(\sigma(\mathbf{a}_{\Phi}^T.[h'_i, h'_j])\right)}{\sum_{k\in\mathcal{N}_i^{\Phi}}\exp\left(\sigma(\mathbf{a}_{\Phi}^T.[h'_i, h'_k])\right)}, \tag{12}$$

where h'_i is the embedding of node i after projection in such a way that $h'_i = \mathbf{M}_{\Phi i}.h_i$ with $\mathbf{M}_{\Phi i}$ a type-specific transformation matrix for node type Φi and h_i the original feature vector of node i. The vector $\mathbf{a}_{\Phi}$ represents the node-level attention for meta-path Φ, and is learned by the network, whereas $\mathcal{N}_i^{\Phi}$ denotes the meta-path-based neighbors of node i (included), and σ is an activation function. Meta-path-based neighbors of a node i on a given meta-path Φ, written as $\mathcal{N}_i^{\Phi}$, denote all nodes that connect i through a meta-path of the same type Φ. The meta-path-based embedding of a node i can then be aggregated using the attention coefficients:

$$z_i^{\Phi} = \sigma\left(\sum_{j\in\mathcal{N}_i^{\Phi}} \alpha_{ij}^{\Phi}.h_j'\right). \tag{13}$$

The coefficients may also be calculated using multi-head attention, for training stability reasons.

In a second phase, semantic-level attention is learned to give appropriate importance between different meta-paths. This is measured as the similarity of the transformed embedding with a semantic-level attention vector $\mathbf{q}$. A softmax function and an average over all embeddings are then applied to get a normalized importance coefficient of each meta-path i, denoted as $\beta_{\Phi i}$:

$$\beta_{\Phi i} = \text{softmax}\left(\frac{1}{|\mathcal{V}|}\sum_{i\in\mathcal{V}} \mathbf{q}^T.\tanh\left(\mathbf{W}.z_i^{\Phi} + \mathbf{b}\right)\right), \tag{14}$$

where $\mathbf{q}$ is the semantic-level attention vector, $\mathcal{V}$ is the set of nodes, $\mathbf{W}$ and $\mathbf{b}$ respectively denote the weight and bias parameters. Final embeddings are obtained by combining the computed attention coefficients, such that

$$\mathbf{Z} = \sum_{i=1}^{P} \beta_{\Phi i}.Z_{\Phi i}, \tag{15}$$

where $Z_{\Phi i}$ represents the semantic-specific node embeddings from meta-path Φi, and P is the total number of meta-paths.

D. INTRUSION DETECTION WITH GNNs

In this section, we propose a 3-step architecture to summarize GNN-based intrusion detection. Indeed, most of the studies reviewed in this survey follow a similar scheme that can be divided into three steps, which are Preprocessing, GNN Embedding, and Detection & Training. This architecture is illustrated in Figure 3.

1) **Preprocessing**: In the first step, raw data such as network captures or system logs are converted into graph structures. Important engineering may be required in this step to handle the possibly large input data, and techniques such as graph sampling may be applied to divide the graph into batches or snapshots. Optionally, node and edge features may be preprocessed to extract even more semantic, and meta-paths can be crafted to handle heterogeneous graphs. The graph structures used for the representation of network and host systems are presented in Sections III-A and IV-A.
2) **GNN Embedding**: The second step involves projecting the nodes and edges of an input graph into an embedding space to capture desired similarities among them, such as structural and contextual similarity, or feature similarity. The GNNs used in intrusion detection draw inspiration from models discussed in Section II-C, with some adjustments to better capture intrusion-based patterns. Typically, the goal is to create embeddings that closely embed malicious flows, attacker hosts, or suspicious authentication requests, differentiating them from benign ones in embedding space. We describe in Sections III-B and IV-B, the different GNN encoders used for the creation of meaningful embeddings.
3) **Detection & Training**: The last step of the intrusion detection process is to leverage the embedding space to search for potential anomalies or attacks. Depending on the use case, a classifier can predict the label of a node, an edge, or the whole graph. Node-level tasks such as malicious host detection often consider a classifier that takes as input node embedding, whereas edge-level tasks like malicious flow detection typically leverage both endpoint nodes along with the edge embedding, to make a prediction. Graph classification tasks such as malicious provenance graph detection are performed by reducing all the node or edge embeddings into a final graph embedding vector using a readout operation. A loss function then compares the prediction to a ground truth value, and backpropagation updates the weights of the model. In supervised setting, the ground truth is a label, whereas unsupervised settings are usually based on self-supervised learning, where the goal is to reconstruct parts of the adjacency matrix or feature matrix to learn embeddings [50], [51]. These techniques will also be reviewed in Sections III-B and IV-B.

In the following sections of this paper, we showcase the various approaches proposed in the literature for extracting graph structures, along with the different types of GNN encoders employed for intrusion detection. Additionally, we discuss the (training) strategies used to effectively train such encoders.

III. NETWORK INTRUSION DETECTION

Network intrusion detection aims to analyze network data and build tools to detect abnormal behaviors. In this section, we first discuss the representation of network data as graphs and then review how researchers leverage these data structures to learn useful embeddings using GNNs and classify them with downstream classifiers. All the datasets mentioned in the literature review are described at the end of the section.

A. REPRESENTING NETWORK DATA AS GRAPHS

A network is by definition a system made of interconnected entities. This is why networks have intuitively been represented as graphs since the beginning of the Internet era. Nodes usually represent hosts (e.g. computers, printers, IoT and physical devices) and edges model the interactions between those hosts (e.g. requests, authentications, communications). Literature commonly relies on three types of network data to model network traffic as a graph:

Packet Graph: A packet is a layer-3 normalized unit of data transmitting on the network, notably composed of a payload that contains the actual data to be transmitted, and a header storing the necessary information to deliver the packet, such as the source and destination IP addresses and ports. The packet payload may contain sensitive information such as authentication credentials or visited website content.

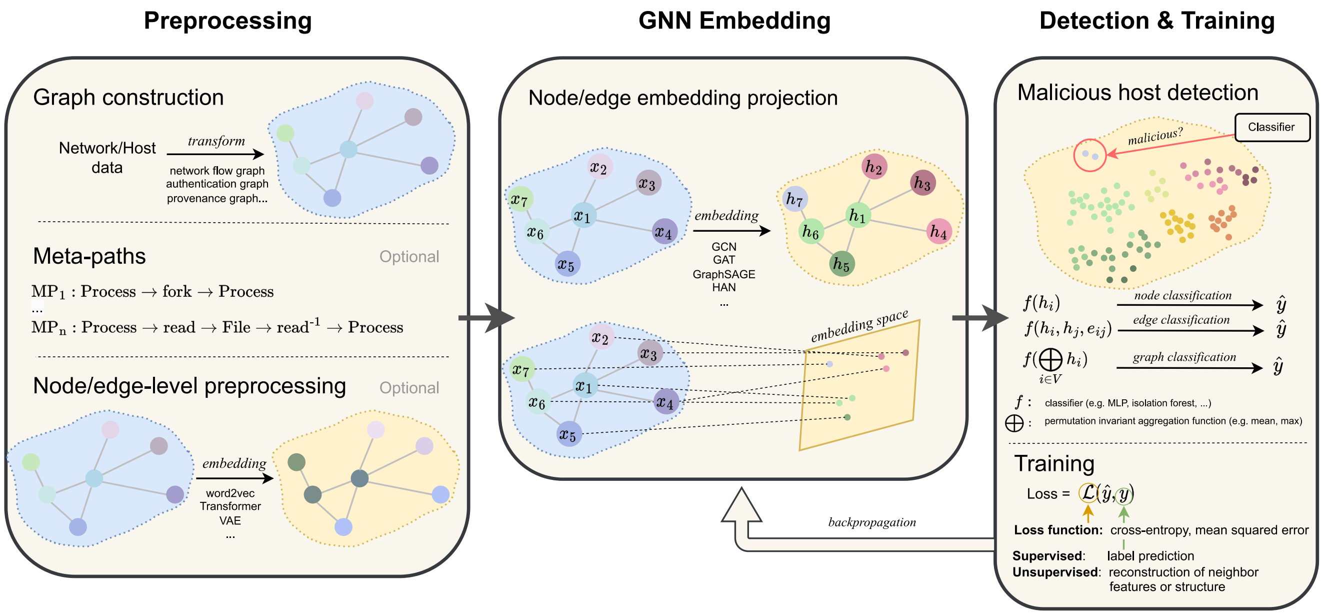

Preprocessing
GNN Embedding
Detection & Training
Graph construction
Network/Host data
transform
network flow graph
authentication graph
provenance graph...
Meta-paths
Optional
MP$_1$: Process → fork → Process
MP$_n$: Process → read → File → read^{-1} → Process
Node/edge-level preprocessing
Optional
embedding
word2vec
Transformer
VAE
Node/edge embedding projection
embedding
GCN
GAT
GraphSAGE
HAN
embedding space
backpropagation
Malicious host detection
Classifier
malicious?
$f(h_i)$ node classification $\hat{y}$
$f(h_i, h_j, e_{ij})$ edge classification $\hat{y}$
$f(\bigoplus_{i \in V} h_i)$ graph classification $\hat{y}$
f : classifier (e.g. MLP, isolation forest, ...)
$\bigoplus$: permutation invariant aggregation function (e.g. mean, max)
Training
Loss = $\mathcal{L}(\hat{y}, y)$
Loss function: cross-entropy, mean squared error
Supervised: label prediction
Unsupervised: reconstruction of neighbor features or structure

FIGURE 3. General architecture for intrusion detection with GNN-based methods. In a first **Preprocessing** step, the graph structure is built from raw flat data such as network flows or system provenance data. Optional preprocessing operations may include the construction of meta-paths to handle heterogeneous host-based graphs or the pre-computation of node and edge embeddings. Then, the **GNN Embedding** part aims to encode information from feature space (represented as blue area) to embedding space (yellow area). In this step, various GNN architectures may be employed or created to extract different kinds of embeddings. The last **Detection & Training** step leverages the computed embeddings for task-specific detection methods. The model is finally trained by computing the loss based on the previous predictions.

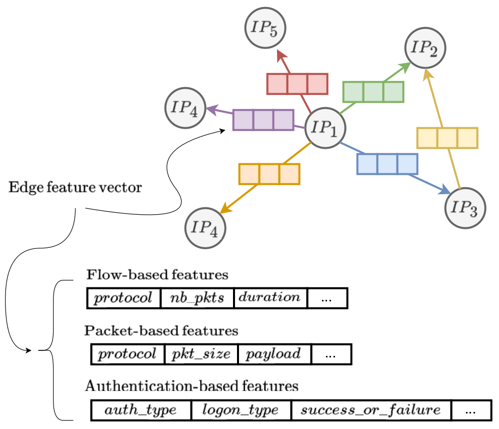

IP_5
IP_2
IP_4
IP_1
IP_3
IP_4
Edge feature vector
Flow-based features
protocol | nb_pkts | duration | ...
Packet-based features
protocol | pkt_size | payload | ...
Authentication-based features
auth_type | logon_type | success_or_failure | ...

FIGURE 4. Example of an edge-attributed graph for the representation of network communications. Nodes are identified by IP addresses and sometimes by the port on which the communication takes place. Edges can possibly store an attribute vector, usually created based on flow, packet or authentication features. Some works prefer storing the attributes in the nodes, whereas others leverage only the graph topology, without any domain-specific features.

However, the payload cannot be analyzed in HTTPS communications as it is encrypted using TLS protocol. Google transparency report estimates in 2022 that 95% of Google's traffic is encrypted under HTTPS [52], making the payload analysis unreliable for intrusion detection at first glance. Contrarily, some companies intercept encrypted packets and rely on SSL inspection [53] to decrypt, analyze and re-encrypt them for security concerns. Packets can be transformed intuitively as graphs, by associating nodes to IP addresses and by creating an edge for each packet transmitted between a pair of nodes. However, creating separate edges for every packets usually results in scalability issues, which make this graph structure rarely used in literature.

Flow Graph: Flows are abstractions made of multiple packets captured in a period of time. One flow is usually identified as a 5-tuple (Src IP addr., Dst IP addr., Src port, Dst, port, Protocol) and summarizes the packets transmitted between the two endpoints using statistics such as the number of packets sent by the source or destination, the mean of the packet size or the duration of the flow. These statistics are usually stored as an edge feature vector but other variants exist such as [54] that creates an intermediate attributed node storing the flow statistics between the source and destination nodes, or [55] that builds a line graph [56] where all nodes are flows, and edges are communications between one endpoint of the flow and an endpoint from another flow. Generally, a TCP flow is ended upon connection teardown (by a FIN packet), whereas an UDP flow is segmented by a time window. The flow can be unidirectional or bi-directional. The former creates two separate flows for outgoing and incoming packets, whereas the latter creates a unique flow combining outgoing and incoming packets. The flow direction is also an important information to consider before analyzing flow data, especially in intrusion detection, where the attacker and target must be identified. Furthermore, flows are often represented as graphs, which offer a more compact representation of communication between hosts by condensing potentially thousands of individual packets into a single edge.

Authentication Graph: Rather than analyzing all network communications, certain detection methods focus specifically on authentication requests. These methods are commonly used to detect attacks that rely on malicious authentications, such as lateral movements in enterprise networks [21], [57]. In such scenarios, nodes may represent IP addresses or identified users in a network, whereas edges are associated to authentication requests.

By capturing different levels of abstraction in the network, these types of data represent various features that can be leveraged in downstream models to detect specific attacks [3]. Many network monitoring tools are available to collect network data, such as Zeek [58], Argus [59], Wireshark [60], Joy [61], Splunk [62], Snort [63] or Suricata [64].

B. GRAPH-BASED NETWORK INTRUSION DETECTION

Detecting intrusions in network graphs typically involves learning representations of nodes and edges, which can be used in downstream clustering or classification models. Depending on the downstream task, various classification methods may be employed. Currently, most state-of-the-art approaches model malicious entity detection as a node classification task and malicious flow detection as an edge classification problem, while network graph classification is less common. In this section, we review existing state-of-the-art works, grouped by graph type, and also summarized in Table 1.

1) NETWORK INTRUSION DETECTION WITH FLOW GRAPHS

Against botnet attacks, some researchers attempt to leverage the graph representation of network flows to detect complex botnet communication patterns between hosts. Zhou et al. [65] propose a supervised approach based on a GCN and network traces. Here, only the topological structure of the graph is considered. Indeed, they omitted edge features and initialized node features as a vector full of ones. The graphs used for training are significant and contain in average more than 140k nodes and 700k edges. A GCN model first computes the node embeddings across all the graph and malicious botnet nodes are then classified using a neural network at node-level. The performance of the model has been evaluated using a synthetic dataset presented in Section III-B4.

Similarly, Zhang et al. [66] suggest using a GCN for botnet detection. In order to capture the long-term dependencies present in large botnet architectures, the authors used 12 GCN layers. However, the use of very deep GCN models is subject to over-smoothing [80] and dedicated methods are usually required to deal with these deep networks. The model was evaluated on the same dataset proposed by Zhou et al. [65].

To prevent the previous over-smoothing problem, XG-BoT [67] leverages grouped reversible residual connections [81] along with a GIN model to capture hidden topological patterns of botnets while maintaining more stability during the training of very deep models. XG-BoT was also evaluated on the dataset by Zhou et al. [65] and outperforms the two previous works [65], [66] for the detection of botnet nodes. Forensics analysis is also considered by leveraging GNNExplainer [82] to highlight a relevant subgraph along with corresponding node features, ultimately resulting in an explainable graph that security teams can analyze.

On a different dataset built from background traffic and synthetic botnet topologies, the authors in paper [68] suggest to leverage Inferential SIR_GN [83], a model able to generalize on unseen and very large graphs and that privileges node structural similarity. Indeed, the authors suggest that GNNs may not be fitted for botnet detection as they consider the node proximity similarity an important indicator in the learning process. They further explain that proximity-based similarity generates node representations that are close in embedding space when the nodes are near or highly connected in the graph space. In the case of botnet and target nodes, which are often close in graph space, proximity similarity can create a relationship between these two nodes. For these reasons Inferential SIR_GN takes into account node structural similarity to create more structural-based embeddings, that are then fed into a traditional neural network for node-level classification.

Although previous botnet detection works rely on homogeneous graphs, Zhao et al. [69] represent network flows as a multi-attributed heterogeneous graph, where a node is one flow entity from the 6-tuple (IP_src, IP_dst, port, protocol, request, response) and edges are actions between flow entities such as *access* or *connect*. The heterogeneous graph is made of multiple node features, such as a timestamp and a user-agent for an IP address node. Knowledge is extracted from this graph using 10 hand-designed meta-paths, which capture useful symmetries hidden in botnet topologies (see Section II-C6 for the definition of meta-path). Weighted similarity graphs are built by calculating the similarity between pairs of nodes and GCN layers are applied on these graphs to characterize discriminative features of bots. Malicious nodes are then detected using a standard feed-forward network at node-level, trained with cross-entropy loss in an end-to-end manner. The proposed model outperforms other graph-based baselines such as a HAN and a standalone GCN, on private network data and on the CTU-13 dataset.

A more general approach to detect intrusions with an heterogeneous graph is proposed in paper [54]. The graph is built based on network flows and for each flow, three nodes are created: the source host node, destination host node and the flow node, where two undirected edges exist between source host and flow nodes and between flow and destination host nodes. By creating a separate flow node, the authors explain that it fits the way GNN models operate and they will naturally better learn the embeddings from flows. To handle heterogeneity between host and flow nodes, a non-standard MPNN has been implemented where messages are aggregated using two distinct functions σ_{sf}, σ_{fd} respectively associated to host-to-flow and flow-to-host edges. New node embeddings are computed by learning an update function δ_{type} and by concatenating the previous node embedding with the message as explained in Eq. 16 and 17.

$$a_i^t = \frac{1}{|\mathcal{N}(i)|} \sum_{j \in \mathcal{N}(i)} \sigma_{\text{type}} \left(\left[h_i^t, h_j^t \right] \right), \tag{16}$$

TABLE 1. State-of-the-art papers for network-based intrusion detection with GRL.

Data	Graph type	Classification	Learning	Models	Year	Paper
Flows	Graph	Node	Supervised	GCN	2020	Zhou et al. [65]
				GCN	2022	Zhang et al. [66]
				GIN, Grouped reversible connections, GNNExplainer	2022	XG-BoT [67]
				Inferential SIR_GN	2022	isirgn1 [68]
	Heterogeneous	Node	Semi-supervised	GCN, Meta-path	2020	Bot-AHGCN [69]
			Supervised	GNN, GRU	2021	Pujol-Perich [54]
	Attributed	Node	Supervised	Graph Network	2020	Protogerou et al. [70]
		Edge		E-GraphSAGE, GraphSAGE	2021	E-GraphSAGE [71]
		Node		Attention, Adaptive Gate Fusion	2021	LGANet [72]
		Node		GAT, E-GraphSAGE, Residual connections	2021	E-ResGAT [55]
		Edge		E-GraphSAGE	2022	E-minBatch GraphSAGE [73]
		Edge	Self-supervised	E-GraphSAGE, Isolation forest	2022	Anomal-E [50]
		Graph	Supervised	GIN	2022	GraphDDoS [74]
Packets	Spatio-temporal	Edge	Supervised	ST-GCN	2021	Cao et al. [75]
	Attributed	Node	Supervised, Meta-learning	Auto Metric GNN	2022	Govindaraju et al. [76]
Authentication logs	Heterogeneous	Edge	Unsupervised	Random Walk, CBOW, LR	2020	Bowman [21]
		Edge	Supervised	Meta-path, CNN, Attention	2021	MLTracer [77]
		Path	Unsupervised	Meta-path, metapath2vec++, Autoencoder	2022	LMTracker [78]
	Temporal	Edge	Unsupervised	Various GNN encoders, RNN	2022	EULER [57]
Authentication logs+Flows	Dynamic	Edge	Unsupervised	Random Walk, GRU, Autoencoder, Skip-gram	2022	Pikachu [19]
	Heterogeneous	Edge	Semi-supervised	Meta-path, GNN with attention	2022	HetGLM [79]

The table categorizes all the network-based papers studied in this survey. **Data** corresponds to the input data type; **Graph Type** refers to the type of graph constructed from the input data. We characterize a graph as attributed if either hand-crafted features, embeddings or raw features such as raw text are intentionally assigned to a node or an edge; **Classification** indicates the downstream classification task; **Learning** represents the learning method; **Models** summarizes the types of models used by each paper; **Year** refers to the publication year, whereas the **Paper** column provides references to each work.

$$h_i^{t+1} = \delta_{\text{type}} \left(\left[h_i^t, a_i^t \right] \right), \tag{17}$$

where a_i^t and h_i^t are respectively the message and the embedding of node i at iteration t, [., .] is the concatenation operation, $\mathcal{N}(i)$ is the neighborhood of node i, σ_{type} is one of the two message functions σ_{sf} and σ_{fd}, and δ_{type} is one of two update functions associated with the type of node. Note that σ_{type} and δ_{type} respectively represent a fully connected network and a GRU. A final MLP is used at node-level to classify among multiple attack classes using softmax along with the categorical cross-entropy loss for training. The custom MPNN has been tested against the CIC-IDS2017 dataset, after dropping 90% of benign examples for unbalanced classes reasons, and achieves a similar accuracy compared to benchmarked ML methods such as MLP, random forest or AdaBoost. However, the robustness of their model has been demonstrated by modifying flow features like packet size and inter-arrival times via multiple adversarial attacks [84]. The results show that their accuracy remains good despite the attacks, whereas other ML algorithms see their performance diminished. This demonstrates that GNNs offer more robustness and generalization capabilities than traditional ML techniques.

For the detection of IoT-based network attacks, Protogerou et al. [70] place multiple agents into a network to collect flow statistics that are then attributed to nodes and edges in a graph built from IP communications. For instance, a node may store the average number of packets sent from this node and an edge can contain the average number of packets sent between a pair of nodes. To propagate information in the graph and compute embeddings, a Graph Network (GN) [85] is used. This is a generalization block built from various GNNs, including MPNNs and Non-Local Neural Networks (NLNNs). Two Multi-Layer Perceptrons (MLPs) are then used for attack detection, one at node-level and another at edge-level, that are both leveraged to create a final prediction. The authors built a synthetic dataset containing benign and attack examples based on the CTU-13 dataset and Mirai attack's distribution. The proposed solution has been evaluated against this custom dataset and outperforms the compared ML techniques such as SVM and random forest, while consuming less energy.

E-GraphSAGE [71] is another method for network intrusion detection in IoT environment. When building the graph, nodes are identified by an IP address and a port, whereas edges represent the flow communication between two nodes. The source IP addresses are randomly mapped to local addresses on a given range to avoid unintentional label for attack traffic. The actual flow features are stored in edges, unlike the original GraphSAGE model that only works with node features. For these reasons, they tuned the GraphSAGE model to work with edge-level features. The aggregation and update functions are similar to the ones used in GraphSAGE (See Eq. 8 and Eq. 9). However, the newly created neighborhood aggregator embeds sampled neighbor edges instead of neighbor nodes, as explained in Eq. 18 and 19.

$$\mathbf{h}^{t}_{\mathcal{N}_s(u)} = \mathrm{AGG}_t\left(\left\{\mathbf{e}^{t-1}_{uv}, \forall v \in \mathcal{N}_s(u), uv \ \in E\right\}\right), \quad (18)$$

$$\mathbf{h}^{t}_{u} = \sigma\left(\mathbf{W}^t \cdot \left[\mathbf{h}^{t-1}_{u}, \mathbf{h}^{t}_{\mathcal{N}_s(u)}\right]\right), \quad (19)$$

where $\mathbf{e}^{t-1}_{uv}$ is the embedding of edge (u, v) at the previous layer and E is the set of edges. No node features are used here (i.e. a node is just a all-one vector). The model is composed of two E-GraphSAGE layers to build the embeddings, meaning that edges are aggregated up to a 2-hop neighborhood. After message-passing, edge embeddings are used for supervised edge classification. The performance has been evaluated on the BoT-IoT, ToN-IoT datasets along with NF-BoT-IoT and NF-ToN-IoT. These are the Netflow normalized versions of the datasets, described in Section III-B4.

For the detection of peer-to-peer botnets, LGANet [72] proposes to consider each node in the network graph as a centroid and builds a local graph from its 1-hop neighborhood, on which the correlation between nodes is modeled using attention. Indeed, P2P bots usually communicate with peers to provide malicious commands and attention is leveraged to capture such relations between hosts. The performance of this custom model has been demonstrated on a dataset created by merging multiple public datasets such as CTU-13, PeerRush [86] and MAWI [87].

Based on the work of E-GraphSAGE [71], Chang et al. [55] propose an improved version of the model by leveraging residual connections to train deeper models and to deal with the high class imbalance of most current intrusion detection datasets. The graph is originally bipartite and is built from flow data, where a node is identified by a tuple (IP address, port), and an edge between two nodes is attributed with flow statistics. The bipartite graph is then transformed into a line graph [56] by converting edges into nodes, resulting in a node classification task. As in E-GraphSAGE, random address mapping is applied to source IP addresses to avoid unintentional label for malicious traffics. In a first step, a fixed-size set of neighbors is sampled from the nodes' neighborhoods. Neighboring edges are then aggregated using a mean operation and as described in Eq. 20, node embeddings are created by concatenating the node embeddings from the previous layer with the aggregated edges:

$$h^t_v = \sigma(W^t.[h^{t-1}_v, h^t_{\mathcal{N}(v)}]), \quad (20)$$

where h^t_v is the node embedding of node v at iteration t, $h^t_{\mathcal{N}(v)}$ represents the aggregated neighboring edge embeddings of node v, W^t is a trainable weight matrix and σ is the ReLU activation function. Node features are initialized with all-one vectors with the same dimension as edge features storing flow statistics. Edge features are then updated by concatenating node embeddings from both endpoints along with the initial edge features as a residual connection:

$$z_{uv} = [h^K_u, h^K_v, e_{uv}], \quad (21)$$

where z_{uv} is the final edge embedding between nodes u and v, h^K_i is the final embedding of a given node i after K iterations, e_{uv} is the initial feature vector of edge (u, v). Here, $(u, v) \in \mathcal{B}$ where $\mathcal{B}$ represents a batch of edges. In a second part, the E-ResGAT model is presented. It consists of a GAT model with additional residual connection and operates on the same line graph as the first method. The authors evaluate the performance of both models on UNSW-NB15, CIC-DarkNet, CSE-CIC-IDS and ToN-IoT datasets. However, only the original GAT is used as a baseline for the benchmark.

Another improvement to E-GraphSAGE is provided by Lan et al. [73], by introducing a pre-sampling step before training the model, resulting in smaller graphs and better scalability. The performance of this model has been compared to the original E-GraphSAGE along with other baselines on the UNSW-NB15 dataset where the accuracy and F1-score are slightly improved on both binary classification and multi-class classification tasks. However, the performance of this model has not been demonstrated on other datasets such as the ones originally used in the E-GraphSAGE paper.

Authors of E-GraphSAGE also contributed to an important improvement to the model, by applying it to a self-supervised approach. This new method named Anomal-E [50] leverages the same graph structure (i.e. flows and endpoints respectively represent edges and nodes) while not requiring any label. Deep Graph Infomax (DGI) [33] is leveraged here to build embeddings for positive and negative graph samples and to compare them to a summarized version of the graph to maximize local-global mutual information. In DGI, negative samples are generated using a corruption function (e.g. shuffling elements in feature matrix or adjacency matrix). Embeddings are then built using an encoder, like GraphSAGE or GCN, and the global summary is computed by a readout function like the average operation. A discriminator then computes scores for both positive and negative embeddings, in order to be used by the binary cross-entropy loss to maximize mutual information. DGI was initially made to train an encoder to learn node embeddings but the authors deal here with an edge classification problem, so they modified the model to learn edge embeddings instead. Here, the negative examples are created by shuffling the edge features, while keeping the same adjacency matrix. E-GraphSAGE is chosen as the encoder to compute positive edge embeddings, global graph summary and negative edge embeddings. A discriminator function presented in Eq. 22 and 23 computes two scores by comparing positive and negative edge embeddings with the graph summary.

$$\mathcal{D}(z_{uv}^k, \vec{s}) = \sigma(z_{uv}^{K^T}\mathbf{W}\vec{s}), \tag{22}$$

$$\mathcal{D}(\tilde{z}_{uv}^k, \vec{s}) = \sigma(\tilde{z}_{uv}^{K^T}\mathbf{W}\vec{s}), \tag{23}$$

where z_{uv}^k and $\tilde{z}_{uv}^k$ respectively represent the true embedding and negative embedding of edge (u, v) at depth k. W is a trainable parameter and $\vec{s}$ is the global graph summary defined as the average of node embeddings passed in a sigmoid activation:

$$\vec{s} = \sigma(\frac{1}{n}\sum_{i=1}^{n} z_i^K), \tag{24}$$

where z_i^K represents the final ith node embedding and σ is the sigmoid function. These scores are finally used in the binary cross-entropy loss to train the model:

$$L = -\frac{1}{2n}\sum_{i=1}^{n}\left(\mathbb{E}_G \log\mathcal{D}\left(z_{uv}^K, \vec{s}\right) + \mathbb{E}_{\tilde{G}}\log\left(1 - \mathcal{D}\left(\tilde{z}_{uv}^K, \vec{s}\right)\right)\right), \tag{25}$$

where $\mathcal{D}\left(z_{uv}^K, \vec{s}\right)$ is a trainable bilinear binary classifier that takes as input an edge embedding along with a graph summary. Anomal-E can then classify malicous flows based on the embeddings using unsupervised classifiers. On both NetFlow datasets presented in [88], Anomal-E achieves close performance compared to the original supervised baseline [88], using an extra tree classifier. However, the current approach does not require any label, which makes this solution more applicable to real-world scenarios.

Against DDoS attacks, a graph classification approach is proposed by Li et al. [74]. In this work, the presented GraphDDoS model aims to detect both low-rate and high-rate DDoS attacks by considering the relationship of packets from a single flow and the relationship between flows. To achieve this, network packets from same source and destination IP addresses are divided into groups that are used to obtain an endpoint graph as final representation. Message-passing is performed on these graphs using GIN and final graph embeddings are computed after passing the embeddings into a readout function. On the CIC-IDS2017 and CIC-DoS2017 datasets, GraphDDoS notably outperforms the node-based method proposed in [54].

2) NETWORK INTRUSION DETECTION WITH PACKET GRAPHS

Although flow-based detection is largely employed in intrusion detection systems, packets offer a fine-grained view of the data that are actually transferred in the network. The authors in [75] propose to represent traffic of packets using a spatio-temporal graph, that is able to model features that vary with respect to time. The proposed method aims to detect DDoS attacks in software-defined networking (SDN) networks based on a Spatio-Temporal Graph Convolutional Network (ST-GCN) [89], a GNN model that is originally used for traffic forecasting. They experiment their model on a real-world enterprise network made of 20 programmable switches with 1 SDN controller. DDoS attacks are simulated with hping3 [90] and are embedded in CAIDA traffic traces similarly to in [65]. The model has been evaluated on the synthetic dataset against CNN, GCN and SVM baselines, and the ST-GCN achieves an important accuracy increase and false positive rate reduction.

Against intrusions in IoT environment, paper [76] leverages the Auto-Metric Graph Neural Network (MAGNN) [91] along with meta-heuristic optimization methods based on HTTP requests data. The MAGNN was initially introduced to be used for the diagnosis of Alzheimer's disease, where it outperforms most graph-based methods like GCN or GAT on biomedical datasets [91] by using a meta-learning paradigm [92]. Here, the authors apply this model to IoT node classification using 2 datasets: CSIC-2010 and ISCXIDS-2012. On both datasets, the proposed model outperforms the compared baselines.

3) NETWORK INTRUSION DETECTION WITH AUTHENTICATION GRAPHS

In enterprise networks, attackers often tend to authenticate into other host machines via lateral movement, in an attempt to access new permissions. In their research paper, Bowman et al. [21] leverage autentication data along with an unsupervised framework to detect such lateral movements. A heterogeneous graph is constructed from Kerberos authentication logs. In this graph, a node can either be an IP address, a user or a service, and an edge represents an authentication event. Furthermore, no additional features are used here, just the graph topology is leveraged. In a first offline phase, random walks randomly sample paths in the graph and Continuous Bag Of Words (CBOW) [25] along with negative sampling [93] compute the node embeddings. In a second online phase, inference on new edges is done based on embedding lookups. As explained in Section II-B1 with the Skip-gram method, here CBOW predicts a center node given its context instead of predicting context from center node. Negative sampling is used to efficiently train embeddings in a self-supervised way. In the context of CBOW, this technique consists in sampling k nodes that are not the target node (i.e. center node) and minimizing their co-occurrence probability with the context node. Eq. 26 describes the corresponding loss function.

$$L = -\left[\log\sigma(h_{con}^T h_{cen}) + \sum_{k\sim\mathcal{P}(w)}\left[\log\sigma(-h_{con}^T h_k)\right]\right], \tag{26}$$

where h_{cen} and h_{con} are respectively the embeddings of center and context nodes, $\mathcal{P}(w)$ is the noise distribution to randomly sample nodes that are different from the center node, and σ is the sigmoid function. The first term maximizes the probability of co-occurrence between center and context nodes, whereas the second term minimizes the probability of co-occurrence between context nodes and negative center node samples. Following this process, users who regularly authenticate to similar entities are close in embedding space. Finally, edge prediction (i.e. authentication prediction) is done between every pair of nodes a and b by using a logistic regression on embeddings $a \circ b$, where $\circ$ is the Hadamard product. The authentication subset of LANL dataset and PicoDomain custom dataset are used for evaluation and the proposed method outperforms other unsupervised ML techniques, on both true positive and false positive rates.

To overcome malicious logins and credential-based lateral movements, MLTracer [77] models login activity from logs as a heterogeneous information network (HIN) and extract contextual semantic with meta-paths. For each type of meta-path, they apply a CNN along with co-attention to learn vector representations, and a fully-connected layer predicts malicious edges given as input the concatenation of source and destination node embeddings.

The lateral movement detection problem is also tackled by LMTracker [78], where the authors also model the interactions between entities using a heterogeneous graph. The metapath2vec++ model [94] computes all node embeddings using meta-paths and random walks. An autoencoder then tries to reconstruct the embeddings, in an attempt to find outliers in embedding space when the reconstruction of malicious examples fail. The final task is to predict a potential lateral movement path, namely a malicious path along multiple edges in the graph. For training, only benign samples from LANL and CERT datasets are leveraged.

Another solution to lateral movement detection is proposed by HetGLM [79]. Rich semantics are captured on the heterogeneous graph with meta-paths to profile each network entity and distinguish authentication activities that deviate from benign activities. The problem thus becomes a weakly-supervised anomaly detection task, where only few benign examples are required as input. Random walks along with attention compute the node embeddings and a dual-decoder identifies abnormal edges given the concatenation of endpoints nodes.

EULER [57] consists in a model-agnostic framework especially designed for the detection of lateral movements with GNNs in temporal graphs. This method captures the temporal relations present in those graphs to perform anomaly link prediction. Indeed, this temporal structure appears to be a meaningful feature that seems to improve the prediction of APTs. Network data such as authentication requests or flows can be model as a temporal graph $G_t = (V, E_t)$, where each t-th graph represents a snapshot with fixed node structure and dynamic edges. The authors propose a distributed architecture based on the leader/worker paradigm, to concurrently perform the time-consuming message-passing operations. Indeed, a leader dispatches graph snapshots to multiple workers, that are responsible for loading the graphs into memory and for computing the embeddings using a given GNN encoder. These snapshot embeddings are then sent back to the leader to capture the temporal dimension using an RNN block and results are dispatched to the workers to perform decoding and anomaly detection. Final back-propagation is performed on the leader node, and training is done in an end-to-end manner with an unsupervised loss. Experiments are conducted on authentication logs from LANL 2015 dataset, with multiple encoders such as GAT, GCN or GraphSAGE, along with various RNNs like GRU and LSTM.

Paudel et al. [19] also leverage the temporal dimension of graphs for network-based anomaly detection. They propose PIKACHU, an unsupervised dynamic graph embedding technique that considers both the dynamic edges and nodes. The network is modeled as a stream of graphs, where a temporal dimension represents the superposition of discrete graph snapshots, with possibly different structures. No features are used, except the IP addresses and timestamps required to build the dynamic graph. Temporal random walks are used in combination with Skip-gram model to generate embeddings that encode the spatial and short-term temporal dependencies in the dynamic graph. A GRU-based auto-encoder also captures long-term temporal dependencies. An anomalous edge can then be detected when it has a low probability to exist between two nodes u and v at a given timestamp T. This probability is expressed in Eq. 27 and 28.

$$p_v = \mathcal{P}(v|u, \mathcal{N}(u))_T, \tag{27}$$
$$p_u = \mathcal{P}(u|v, \mathcal{N}(v))_T, \tag{28}$$

where p_v and p_u respectively represent the probability that there exists an edge (v, u) and (u, v). $\mathcal{N}(i)$ is the neighborhood of a given node i. This probability is estimated by the softmax function and cross-entropy is used as loss to train the model. Finally, an anomaly score is computed based on p_v and p_u:

$$e_{score} = \frac{(1 - p_v) + (1 - p_u)}{2}. \tag{29}$$

In their experimental setup, the authors use the LANL and DARPA OpTC datasets, where 1 snapshot corresponds to 1 hour of data. Three dynamic graph-based anomaly detection methods have been compared to PIKACHU and it notably achieves a lower false positive rate and higher AUC.

4) NETWORK INTRUSION DATASETS

This section aims to review the different network intrusion datasets that were employed in previous studies. A comprehensive listing of these datasets is also provided in Table 2.

CAIDA [95]: Between 2008 and 2019, the Center for Applied Internet Data Analysis (CAIDA) recorded passive network traces from high-speed monitors on a commercial backbone link. Hundreds of Gigabytes of requests were recorded over these years and can be accessed via pcap files. These data are usually used as background traffic for creating synthetic cyberattack detection datasets [65], [75].

Zhou et al. [65]: In their botnet detection method based on GCN, Zhou et al. released a graph-based dataset made of both synthetic and real botnet topologies within a real background network traffic. The CAIDA 2018 passive network traces were used as background traffic and random subsets of nodes were chosen to embed botnet topologies using real botnets or overlaid P2P topologies. A total of 4 synthetic topologies (DE BRUIJN, KADEMLIA, CHORD, LEET-CHORD) and 2 real

TABLE 2. Datasets used by each network-based paper.

Paper	Datasets	Performance
Zhou et al. [65]	Zhou et al.	99.03%, 99.51% acc
Zhang et al. [66]	Zhou et al.	98%-98.91% F1
XG-BoT [67]	Zhou et al.	99.52%, 99.47% F1
isirgn1 [68]	CAIDA+Synthetic samples	97.85%-99.78% F1
Bot-AHGCN [69]	CTU-13	98.27%, 98.22% micro-F1
Pujol-Perich [54]	CIC-IDS2017	99% weighted-F1
Protogerou et al. [70]	CTU-13+Synthetic samples	99% AUC
E-GraphSAGE [71]	BoT-IoT, ToN-IoT, NF-BoT-IoT, NF-ToN-IoT	100%, 99%, 97%, 100% F1
LGANet [72]	CTU-13+PeerRush+MAWI	95% F1
E-ResGAT [55]	UNSW-NB15, CIC-Darknet2020, ToN-IoT, CSE-CIC-IDS2018	99.5%, 92.32%, 99.88%, 96.5% weighted-F1
E-minBatch GraphSAGE [73]	UNSW-NB15	99.88% F1
Anomal-E [50]	NF-UNSW-NB15-v2, NF-CSE-CIC-IDS2018-v2	92.35%, 95.39% macro-F1
GraphDDoS [74]	CIC-IDS2017, CIC-DoS-2017	99.59%, 94.56% F1
Cao et al. [75]	CAIDA+Synthetic samples	~90% acc
Govindaraju et al. [76]	CSIC 2010, ISCXIDS2012	90.75%-97.96%, 93.88%-96.97% F1
Bowman [21]	PicoDomain, LANL 2015	80%, 85% TPR
MLTracer [77]	LANL 2015, Private	99.98%, 99.97% AUC
LMTracker [78]	LANL 2015, CERT	95%, 94% AUC
EULER [57]	LANL 2015	99.16%
Pikachu [19]	LANL 2015, DARPA OpTC	94%, 99% AUC
HetGLM [79]	LANL 2015, CERT, PicoDomain	89.28%, 91.28%, 92.68% F1

Categorization of all datasets used in the surveyed papers. **Paper** and **Datasets** refer to the paper reference and the datasets used in the corresponding paper, respectively. When Datasets are separated by a "+" symbol, it indicates a global dataset assembled from each mentioned dataset. However, if a comma is used, it means that the authors conducted experiments on separate datasets. If multiple comma-separated datasets are present in Datasets, and only one metric is assigned in Performance, then that metric refers to the performance of the first dataset. Otherwise, each dataset is assigned to a specific performance metric. If there are more metrics in Performance than there are datasets, multiple variants of the models may be proposed. For further information, we suggest referring to the original paper; **Performance** refers to the metrics, defined as follows: "acc" represents accuracy, calculated as (TP+TN)/(TP+TN+FP+FN), where TP, TN, FP, and FN correspond to true positive, true negative, false positive, and false negative, respectively. "F1" represents the harmonic mean of precision and recall, calculated as 2×(Precision×Recall)/(Precision+Recall) where Precision=TP/(TP+FP) and Recall=TP/(TP+FN). "AUC" represents the Area Under the Receiver Operating Characteristic Curve.

botnet topologies (C2 and P2P) are generated and divided into 6 labeled datasets. Each dataset contains 960 undirected graphs with self-loops and is composed in average of 140,000 nodes and 700,000 edges. No additional flow features are provided in this dataset and existing works rather leverage representation learning to learn directly embeddings from graph topology, as done in [66] and [67] with C2 and P2P datasets.

CTU-13 [96]: CTU-13 was delivered by CTU University, Czech Republic and is made of network traffic captures containing benign traffic and Botnet attacks. Two types of data are available for this dataset: packet-based pcap captures (more than 850M packets) and flows captured with Argus in Netflow format (around 20M bi-directional flows). The dataset consists of 13 attack scenarios exploiting various protocols, divided into 13 files. A graph can possibly be built using the source and destination IP addresses of either the packet or flow data. However, flow data are in practice much more used in graph approaches due to the large size of packet-based captures.

CIC-IDS2017 [97]: CIC-IDS2017 is a network dataset proposed by the Canadian Institute of Cybersecurity (CIC), composed of benign and attack network flows. For each flow, 80 features have been extracted into pcap and CSV files using CICFlowMeter [98] for 5 days in a simulated environment. Seven types of web attacks are present in the dataset: Brute Force, HeartBleed, Botnet, DoS, DDoS, Web Attack, Infiltration. In each attack type, one or multiple attacks are performed, for a total of 13 distinct labeled attacks. Each of them is isolated based on a time range, similarly to most other attack datasets. Experiments were conducted on the 80 original features, concluding that 8 selected features are sufficient to achieve valuable results [97].

CSE-CIC-IDS2018 [99]: CSE-CIC-IDS2018 is a network dataset born from a joint project between the Communications Security Establishment (CSE) and the Canadian Institute for Cybersecurity (CIC). The data cover examples from the same seven attack types as CICIDS2017, and CICFlowMeter was also used to extract 80 flow features that are available in pcap format.

UNSW-NB15 [100]: This dataset was released by the Cyber Range Lab of UNSW Canberra. Network flows were captured in a private environment, resulting in 2,218,761 benign flows (87.35%) and 321,283 attack flows (12.65%)

divided into 10 classes: Benign, Analysis, Backdoor, DoS, Exploits, Fuzzers, Generic, Reconnaissance, Shellcode, Worms. A comprehensive range of data formats are available, including packet-based (pcap files) and flow-based (CSV, Zeek, Argus) network logs. This dataset is known to suffer from class imbalance and class overlap [101], meaning that these two issues should be addressed before usage. As an example, [102] used bagging, undersampling and synthetic minority oversampling to tackle the class imbalance problem, resulting in a better classification accuracy.

BoT-IoT [103]: Also developed by the Cyber Range Lab of UNSW Canberra, this dataset was created by simulating a network environment and running multiple botnet attacks on it. Both packet- and flow-based data are captured and made public in pcap, Argus and CSV file formats. The pcap files store around 70GB of packet-based data and CSV files store 16.7GB of flow-based data. The overall dataset contains 477 benign flows (0.01%) and more than 3.6M attack flows (99.99%). Four categories of attacks are present in the dataset, including DDoS, DoS, OS and service scan, as well as keylogging and data exfiltration, for a total of 10 labeled attacks.

ToN-IoT [104]: ToN-IoT is another dataset created by the Cyber Range Lab of UNSW Canberra. It is composed of 3 categories of datasets: Telemetry of IoT and IIoT sensors, Windows and Ubuntu OS metrics and Network traffic. These data were captured from a simulated and realistic network. For NIDS applications, the provided network dataset leveraged Zeek to capture 796,380 benign flows (3.56%) and 21,542,641 attack flows (96.44%), along with 44 flow features extracted from Argus and saved in CSV files. Nine attacks are represented: DoS, DDoS, backdoor, injection, MITM, password, ransomware, scanning and XSS.

NF-datasets [105]: NF-BoT-IoT, NF-ToN-IoT, NF-CSE-CIC-IDS2018 and NF-UNSW-NB15 have been introduced as part of a standardization process that aims to use NetFlow as the default standard for NIDS datasets (NF- prefix stands for NetFlow). Indeed, all four initial datasets have few features in common, making the benchmark of ML methods impossible on each of them. The authors therefore propose a new version of these datasets, using only 8 NetFlow features (12 by considering source and destination IP addresses and ports) instead of the original dissociated features. All the standardized datasets are then merged in order to create a global dataset named NF-UQ-NIDS. The nProbe tool [106] was used to convert the original pcap files from datasets into CSV files storing the NetFlow v9 features. Hereafter are described the statistics of the standardized datasets:

- NF-UNSW-NB15: 1,550,712 benign flows (95.54%) and 72,406 attack flows (4.46%) divided into 10 classes: Benign, Fuzzers, Analysis, Backdoor, DoS, Exploits, Generic, Reconnaissance, Shellcode, Worms.
- NF-BoT-IoT: 13,859 benign flows (2.31%) and 586,241 attack flows (97.69%) divided into 5 classes: Benign, Reconnaissance, DDoS, DoS, Theft.
- NF-ToN-IoT: 270,279 benign flows (19.6%) and 1,108,995 attack flows (80.4%) divided into 10 classes: Benign, Backdoor, DoS, DDoS, Injection, MITM, Password, Ransomware, Scanning, XSS.
- NF-CSE-CIC-IDS2018: 7,373,198 benign flows (87.86%) and 1,019,203 attack flows (12.14%) divided into 7 classes: Benign, Brute Force, Bot, DoS, DDoS, Infiltration, Web Attacks.
- NF-UQ-NIDS: 9,208,048 benign flows (76.77%) and 2,786,845 attack flows (23.23%) divided into all previous classes.

The same authors also provided a second version of these datasets (with -v2 suffix) [88] that integrates a total of 43 flow features instead of the original 12.

CIC-Darknet2020 [107]: CIC-Darknet2020 is a network dataset used for VPN and Tor applications classification, developed by the Canadian Institute for Cybersecurity. It is composed of 77 flow features extracted with CICFlowMeter from 134,348 attack scenarios (84,67%) and 24,311 benign activities (15,33%), divided into 9 categories: Benign, Audio-Stream, Browsing, Chat, Email, P2P, Transfer, Videao-Stream, VOIP.

LANL-2015 [108]: The Los Alamos National Lab (LANL) dataset is made up of 58 consecutive days of data gathered from the Los Alamos National Laboratory's internal computer network. The dataset is composed of 5 types of data: red-team activities, network flows, process start and stop events, window-based authentication events, and DNS lookups. A typical APT campaign produced 1,648,275,307 events across 17,684 Windows machines, including 305 compromised computers and 749 harmful events. With this dataset, graphs can be built thanks to the use of either network flows or authentication data events.

DARPA OpTC [109]: DARPA OpTC contains more than 17 billion network-based and host-based events following APT-like scenarios, generated in an enterprise network by DARPA, where 3 attack scenarios are represented: powershell empire staging, data exfiltration and malicious software upgrade. All samples follow the eCar format, which is an extension of MITRE's CAR data model [110], where events are identified in temporal space using a 3-tuple (object, action, fields), inspired from object-oriented programming.

CSIC 2010 [111]: The CSIC 2010 dataset is composed of 36,000 normal and 25,000 attack HTTP/1.1 requests generated from an eCommerce website developed in the Information Security Institute of CSIC. Multiple types of attacks are present in the dataset, such as SQL injection, buffer overflow, information gathering and XSS. A graph can be built based on the HTTP requests, where two nodes are the host and destination URLs and where an edge represents a request.

ISCXIDS2012 [112]: Initiated by the Information Security Centre of Excellence (ISCX) at the University of New Brunswick, this dataset is composed of 7 days of network activity monitoring from a simulated realistic environment where normal bahaviors and attack scenarios happen on specific days. Multiple kinds of attacks are present: network infiltration from inside, HTTP DoS, DDoS with IRC Botnet, SSH BruteForce. Roughly 125,000 attack packets and 2.1M benign packets are recorded. For each day, a pcap file can be downloaded. Multiple standard features are available, including packet payload and total number of bytes sent or received.

PicoDomain [113]: PicoDomain is a light-weight dataset containing network flows captured with Zeek in a small-size network environment with multiple departments and Windows Active Directory. The attack campaign was registered during 3 days and follows the Mandiant Attack Lifecycle (MAL), which comprises multiple cyclic phases appearing during adversarial campaigns. This dataset aims to be compact (∼ 220MB of Zeek logs) and useful for small analysis research projects, where small computation time and memory consumption are required.

IV. HOST INTRUSION DETECTION

Host intrusion detection aims to monitor and analyze the internals of a system, looking for suspicious behaviors. In this section, we show that there are multiple ways to model host data as graphs and we review the state-of-the-art works that leverage these graph representations with GNNs for host intrusion detection. We also present host-based datasets at the end of the section.

A. HOST GRAPH REPRESENTATIONS

Host machines are the source of many system data that can be modeled as a graph. These input data could be for instance captured from the Operating System (OS) logs and application events [114], [115] but may also be captured from the network traffic generated by the host [116], [117]. Monitoring host data is usually straightforward as a sensor needs to be placed directly on the host machine, without requiring a strategic position like in the monitoring of network traffic. However, managing the installation and maintenance of host data monitoring tools can be challenging in large companies as a remote or a physical access to the machines may be required, inducing privacy and scaling issues. However, the low-level and fine-grained information provided by host-based analysis is fundamental to the detection of most attacks that cannot be handled by network intrusion detection systems.

Effective graph representations are thus fundamental to build models able to detect complex attacks that take place at the host level. In this section, we present two data representations commonly used in host-based detection, which are also illustrated in Figure 5.

Provenance Graph: Provenance graphs are abstract representations of the origin and evolution of data for a given system. They are usually made of system entities as nodes (e.g. processes, files, sockets, threads) and interactions between those entities as edges (e.g. system calls, authentications, user events), making their graph structure compatible with various graph learning techniques. Such graph representations capture important relationships between system events, which facilitates the discovery of malicious events that are temporally distant [118]. These advantages have made the provenance graphs more and more used in recent threat detection research works [119], [120]. In such graphs, the direction of edges is especially important because it gives an important insight into a system event. For instance, `cmd.exe` forking `word.exe` is less suspicious than `word.exe` forking `cmd.exe` [121]. In order to improve compatibility between data provenance tools, multiple data models such as Open Provenance Model [122] and W3C-PROV [123] have been released. In these two examples, provenance data is represented as a directed acyclic graph (DAG). CamFlow [124] is a tool that creates such provenance graphs that can be used in downstream ML tasks.

System Call Graph: System calls are low-level functions used for the communication between programs and the operating system's kernel. Sequences of system calls can be monitored on a host device, and transformed into graphs to represent the sequential relations between calls.

A wide variety of host-based data monitoring tools is publicly available and leveraged in research (e.g. Windows and Linux event logs, CamFlow [124], Sysmon [125], auditd [126]). On top of CamFlow, Flurry [127] is a framework that captures both kernel-level and user-level data to build provenance graphs for downstream graph representation learning experiments.

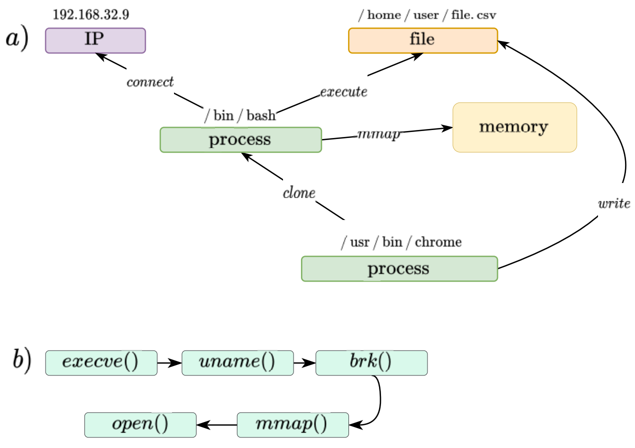


FIGURE 5. Examples of provenance graph (a) and system call graph (b).

B. GRAPH-BASED HOST INTRUSION DETECTION

In this section, we review the state-of-the-art papers regarding host intrusion detection based on GRL and GNNs. We notably show that a majority of works model host systems using heterogeneous provenance graphs, where the goal is to predict abnormal nodes or graphs. We also demonstrate that some works prefer leveraging other data sources such as system calls, authentication requests and network flows for other classification tasks. A summary of the host-based papers studied in this section is provided in Table 3.

1) PROVENANCE GRAPH-BASED HOST INTRUSION DETECTION

Provenance graphs are useful data structures for learning representations. In existing literature, graph representation learning is succesfully applied to learn embeddings from these graphs, in an attempt to detect malicious system

TABLE 3. State-of-the-art papers for host-based intrusion detection with GRL.

Data	Graph type	Classification	Learning	Models	Year	Paper
Provenance Graph	Heterogeneous	Node	Unsupervised	Random Walk, word2vec	2019	Log2vec [22]
	Attributed	Graph matching	Supervised	GCN, Attention, NTN	2021	DeepHunter [15]
	Heterogeneous	Node	Supervised	HAN, R-GCN	2021	Lv et al. [47]
		Node	Semi-supervised	GraphSAGE	2022	THREATRACE [48]
		Node	Supervised	HAN, R-GCN	2022	APT-KGL [128]
		Graph	Supervised	GIN, Attention	2022	PROV-GEM [129]
		Graph	Unsupervised	GNN, Deep SVDD	2022	OC-DHetGNN [121]
	Attributed	Node	Unsupervised	NWR-GAE, Transformer, VAE, Isolation forest	2022	Lakha et al. [130]
Syscall	Graph	Graph	Supervised	Random Walk, word2vec	2021	Hu et al. [20]
Provenance + Flows	Heterogeneous	Edge, Graph	Self-supervised, Unsupervised	HAN, GAT, Autoencoder	2021	Li et al. [51]

entities or advanced threats. As a first example, Log2vec [22] extracts provenance-like data from system logs to detect threats, with an unsupervised graph embedding approach for anomaly detection. A heterogeneous graph is built using ten hand-crafted rules from log entries that consist in multiple system-level attributes. One log entry is a tuple composed of an object (e.g. file, website, removable storage), an operation type (e.g. logon, file operation, browser usage), a time and a host machine. Each element can store features that will be gathered when processing embeddings in the graph. Three types of relations are taken into consideration while creating the graph: causal and sequential relationships within a day, logical relationships among days, and logical relationships among objects. Only benign examples are required to train the model as the goal here is to detect anomalies (i.e. suspicious behaviors). A custom approach based on random walks and word2vec is used to extract the context of each node and to construct embeddings. Using this technique, close nodes (i.e. log entries with close relationship) will be embedded close in the embedding space. Finally, they apply a clustering method with threshold to separate benign and anomalous nodes in latent space. Log2vec has been compared to 11 baselines on the CERT dataset and 2 methods on the LANL dataset. In both cases, Log2vec widely outperforms the baseline methods.

DeepHunter [15] tries to hunt known threats by applying graph pattern matching between a target provenance graph and a query graph, that is built from Indicators Of Compromise (IOCs) extracted from public Cyber Threat Intelligence (CTI). Both graphs possess the same structure, attributes and relations. A subject is namely associated to a process and an object represents another system object such as a file, a socket or a Windows registry. An attribute embedding network first computes embeddings for each node attribute using word2vec. A node embedding is then aggregated with all its attribute embeddings using attention coefficients learned for each attribute. The nodes of the small and noise-free query graph are then embedded using a GCN. However, provenance graphs are more complex and noisy by nature and the authors aggregate the information from distant nodes using Layer-wise Dense-connected Aggregator [49] along with attention. Graph embeddings are calculated for both graphs with Global Context-Aware Attention, as proposed in SimGNN [131]. Finally, the matching between the two graph embeddings is done by leveraging Neural Tensor Network (NTN) [132].

Another heterogeneous approach based on provenance data is proposed by Lv et al. [47] where hand-designed meta-paths are used by a Heterogeneous graph Attention Network (HAN) to extract useful semantic information, as explained in section II-C6. Here, a meta-path represents a sequence of system events like two processes reading a sensitive file or two processes accessing the Internet. HAN is used as the encoder to build embeddings from the heterogeneous graph and classification is done at node-level to detect malicious processes. Online attack detection is also possible based on the embeddings, by inferring on sampled areas of the heterogeneous graph.

THREATRACE [48] is a real-time intrusion detection framework also based on provenance graphs. Each node is a system entity (e.g. file or process) and an edge is a system call associated with a timestamp identifying the event time. The model is inspired by GraphSAGE to aggregate the nodes and to build the embeddings. However, only benign examples are required here. The idea is to learn normal behaviors to detect anomalous ones in embedding space, without requiring anomalous examples. A total of 7 state-of-the-art algorithms are used as baselines, where 3 are anomaly-based host threats detectors such as Unicorn [118], ProvDetector [133] and StreamSpot [134], and 4 are anomalous log detectors like Log2vec [22] or DeepLog [135]. On the StreamSpot, Unicorn and DARPA TC datasets, THREATRACE outperforms all methods, especially in terms of false positive and false negative rates.

For the detection of advanced APT attacks, APT-KGL [128] samples a subgraph around a new incoming node to gather the embedding information already computed by a GNN in previous iterations. This technique makes it possible to infer new system entities in a reasonable time as only a subgraph is considered. A new incoming node is aggregated with its local subgraph using the propagation rule of the relational graph convolutional network (R-GCN) [136]. The new node embedding is thus aggregated in this way:

$$h_i^{(l+1)} = \sigma\left(\sum_{r\in R}\sum_{j\in\mathcal{N}_i^r}\frac{1}{|\mathcal{N}_i^r|}\mathbf{W}_r^{(l)}h_j^{(l)} + \mathbf{W}_0^{(l)}h_i^{(l)}\right), \tag{30}$$

where $h_i^{(l)}$ is the embedding of node i at layer l, $\mathcal{N}_i^r$ denotes the neighbors of node i under relation $r \in R$, with R the set of relation types. $\mathbf{W}_r$ and $\mathbf{W}_0$ are trainable weights. This model is trained using cross-entropy loss after applying softmax to all final embeddings. Due to the complexity of finding real-life APT scenarios, the authors introduce a module responsible for creating synthetic attack graph samples by extracting threat knowledge from public CTIs and Tactics Techniques and Procedures (TTPs). The graphs are generated from threat reports written in natural language by domain experts and are then embedded into real benign provenance graphs to enhance the authenticity of data. Multiple evaluation strategies are compared on private data and on the public DARPA TC dataset. On the private dataset, the model is outperformed by CONAN [137], a state-of-the-art APT detection method based on hand-designed rules. However, APT-KGL outclasses other baselines by a large gap on DARPA TC despite a relatively low precision (i.e. high number of false positives).

PROV-GEM [129] first pre-processes host-level data collected with CamFlow in order to build a unified embedding framework for detecting anomalous behaviors in provenance data. Nodes, edges, and their provenance types are standardized using W3C-PROV labels [138], resulting in a heterogeneous graph with 3 node types (agent, entity, activity) and multiple edge types (e.g. wasDerivedFrom, used, wasGeneratedBy). A Locality Sensitive Hashing technique (TLSH [139]) converts the high-dimensional variable-length vectors generated by CamFlow into a fixed-length space of lower dimension that preserves the features' information. After this normalization step, message-passing is applied by using a similar update function as GIN [140]. To deal with the heterogeneity of the graph, the authors propose to aggregate neighbors based on the 3-tuple representing the source node type, edge type, destination node type. Such a structure is called a relation type and is defined as $r = (t_v, t_e^{(v,u)}, t_u)$ where $r \in R$, the set of all relation types.

$$\phi_r^k(\mathcal{N}(v,r)) = \text{MLP}^k\left(h_v^{(k-1)} + \sum_{u\in\mathcal{N}(v,r)} h_u^{(k-1)}\right), \tag{31}$$

where $\mathcal{N}(v,r)$ is the neighborhood of node v on relation r and h_v^{k-1} is the embedding of node v at previous layer. To take into account all relation types, a new node embedding representation is obtained by concatenating the representations computed for all relations types:

$$\tilde{h}_v^k = \text{CONCAT}\left(\phi_r^k\left(\mathcal{N}(v,r)\right) \mid \forall r \in R\right). \tag{32}$$

Semantic attention is applied on top of the embedding process in order to learn attention coefficients for each relation type:

$$a_v^k = \text{softmax}\left(\mathbf{w}^k.\tanh\left(\mathbf{W}^k.\tilde{h}_v^k\right)\right), \tag{33}$$

where a_v^k represents the relation attention coefficients, $\mathbf{w}^k$ and $\mathbf{W}^k$ are respectively a trainable relation attention matrix and a traditional weight matrix. The final node embedding is computed using a simple dot product as follows:

$$h_v^k = a_v^k.\tilde{h}_v^k. \tag{34}$$

The given model is evaluated on the graph classification task by only aggregating the 1-hop neighborhood and by using as readout function a sum along all node embeddings. The loss is computed with binary cross entropy and experiments were done on StreamSpot and Unicorn datasets, where both original methods are outperformed. However, no other GNN-based methods were compared here.

Huang et al. [121] push the host-based intrusion detection task further by introducing an unsupervised method that considers multiple modalities at node- and edge-level. An heterogeneous directed graph is built from system interactions, where nodes are processes or files and edges are events such as a process accessing a file or a process forking other processes. Every process is uniquely identified using the PID and the command that invoked it, and the file is identified by its path. A different aggregation function is implemented for each node type, namely the parent process, the child process and the accessed file. Node embeddings are then computed using a custom aggregation that preserves the direction of edges:

$$H_P^{l+1} = \mathbf{W}^l\left[H_i^{l+1}, H_P^l, H_o^{l+1}, H_F^{l+1}\right], \tag{35}$$

where H_P^{l+1} is the embedding of the process at next layer, H_i^{l+1} is the embedding of the parent process, H_P^l is the embedding of the current process node, H_o^{l+1} is the embedding of the child process node, H_F^{l+1} is the embedding of the accessed file. The directionality is preserved by concatenating the embeddings in a precise order, starting from the parent process to the accessed file. Finally, a deep SVDD model [141] is leveraged to compute a global score based on the graph embedding (obtained via mean-pooling), and a local score based on node embeddings. Considering these two scores in the loss allows the model to detect attacks from both local and global areas in the graph. The proposed model outperforms other baselines such as GCN and GraphSAGE. However, the experiments are done only on a private enterprise dataset.

2) SYSTEM CALL GRAPH-BASED HOST INTRUSION DETECTION

System calls (syscall) offer a fine-grained representation of the running host system, which can be structured as a graph. In reference [20], a random walk-based approach is proposed to detect host intrusions from system call traces. The graph is built from a sequence of system calls captured from local computers, where a node represents a syscall and the sequential relation between two syscalls is an edge. Only the topology of the graph is leveraged here, no features are used. Node embeddings are computed with random walk and word2vec. The intrusion detection is here a graph classification task, and the authors use a hierarchical pooling method in order to learn hierarchical representations. Indeed, traditional pooling operations are inherently flat and do not learn hierarchical representations of graphs. This limitation is especially problematic for the graph classification task, where all embeddings are generally reduced into a fixed-size graph embedding for downstream classification. In hierarchical pooling, the dimension reduction of the graph is done layer after layer and the pooling process may be differentiable and jointly learned with the neural network. For instance, DiffPool [142] is a differentiable hierarchical pooling technique where the graph is coarsened by clustering nodes after each layer. A mapping function is learned to map each node to a cluster at the next layer, until the dimension is sufficiently reduced for downstream graph classification. A similar technique is used in this work, where the graph is pooled successively until getting a final node embedding capturing the whole graph information. Multiple alternatives of random walk and embedding methods are benchmarked on the ADFA-LD dataset, with a MLP or kNN as back-end classifier.

3) HYBRID HOST AND NETWORK INTRUSION DETECTION

Although rarely evoked in current graph-based intrusion detection works due to the lack of real-world datasets, some researchers are interested in leveraging hybrid approaches, where both host- and network-level data are taken into consideration when building the graph. To our knowledge, Li et al. [51] are the first authors to propose a graph deep learning method that leverages both kinds of data to detect complex attacks such as APTs. They propose a framework that comprehensively captures the behaviors of the full APT lifecycle by building an Intrahost Provenance Graph (IPG) to capture host-based features, and an Interhost Interactive Graph (IIG), that models all network-based communications among hosts. The IPG is created using a similar heterogeneous graph as in previous works (i.e. interactions between process, file or socket), whereas the IIG represents the hosts as nodes and the edges are either network flows, authentication requests or DNS lookups, with heterogeneous feature vectors. Both graphs are trained using different methods as the graph structure and the semantic is not the same in both cases. First, the embeddings of the IPG are trained using a similar method as HAN (see Section II-C6) to deal with the heterogeneity of the graph with meta-paths that are aggregated together with attention. The resulting graph embeddings are then fed into a deep autoencoder for unsupervised anomaly detection. The goal is to compress the input graph into a latent space and learn to reconstruct it using the low-dimension embeddings. By training the model to decode normal graphs, the malicious graphs will be decoded with different embeddings that may be detected in downstream anomaly detection tasks. Specifically, the model is trained using Mean Squared Error (MSE) loss, where the l_2-norm is leveraged to measure the reconstruction error between the reconstructed embeddings and the actual graph embeddings h_g. The anomaly is then measured by a score that can be compared to a threshold value or used as a ranking of the most suspicious hosts:

$$s(h_g) = \|\mathrm{Dec}(\mathrm{Enc}(h_g)) - h_g\|^2. \tag{36}$$

Concerning the IIG, the graph is made of uniform nodes but heterogeneous edges, thus a specific method is implemented to deal with multiple edge features. The authors propose an alternative to GAT, that considers the edge features by learning an attention coefficient for specific feature channels. For a given P-dimensional edge feature tensor E_{vu} between two nodes v and u, the p-th feature channel is denoted as E^p_{vu}. The attention score e^p_{vu} for the edge (v, u) on the feature channel p is measured as in GAT but also multiplied by the p-th feature channel. The normalized attention score is computed in Eq. 38 using softmax along all neighbors of starting node v:

$$e^p_{vu} = \mathbf{a}^T([\mathbf{W}h'_v, \mathbf{W}h'_u])E^p_{vu}, \tag{37}$$

$$\alpha^p_{vu} = \frac{\exp(\sigma(e^p_{vu}))}{\sum_{i\in\mathcal{N}(v)}\exp(\sigma(e^p_{vi}))}, \tag{38}$$

where $\mathbf{a}$ is the weight attention vector, h'_i is the embedding of a given node i and $\mathbf{W}$ is a weight matrix. $\mathcal{N}(v)$ and σ are respectively the neighborhood of node v and the LeakyReLU activation function. A final node embedding h_v is obtained by aggregating neighboring node embeddings, edge features and corresponding attention score, for all feature channels, in this way:

$$h_v = \|_{p=1}^{P}\sigma\left(\sum_{u\in\mathcal{N}(v)}\alpha^p_{vu}\mathbf{W}(h'_u, E^p_{vu})\right), \tag{39}$$

where $\|_{p=1}^{P}$ is the concatenation operation along the P edge feature dimensions. The IIG model is trained in a self-supervised fashion using negative sampling by maximizing the anomaly score of edges between the malicious hosts discovered in the IPG and by minimizing the anomaly score of benign edges. Thus, the loss takes into consideration the predictions of both IPG and IIG models. These two methods have respectively been evaluated separately on the StreamSpot and LANL datasets. However, the complete architecture using both models has not yet been experimented due to the lack of datasets.

4) HOST-BASED DATASETS

The extensive research that was undertaken in HIDS over the last decade led to the emergence of large datasets that researchers can leverage to build various representations for downstream tasks. In this section, we describe the datasets that were leveraged in the studied papers, also summarized in Table 4.

DARPA TC [143]: DARPA TC dataset was created as part of the DARPA Transparent Computing program, that aimed to develop technologies to record and preserve the provenance of systems for security and system research. More precisely, multiple experiments involving APT attacks have been conducted by the DARPA and the recorded provenance system logs are, since then, publicly available. In most approaches studied here, the data are transformed into provenance graphs and the interactions between entities is learned with graph representation learning.

StreamSpot [144]: This dataset hosted on GitHub [145] is composed of 6 * 100 system call flow graphs derived from 5 normal behavior scenarios and 1 attack scenario. The normal behavior graphs are represented as benign syscall flows of activities such as checking emails in Gmail, browsing CNN.com, downloading files and watching videos on YouTube. The attack scenario is made of malicious syscall flows from a drive-by download attack triggered from a malicious URL browsing that exploits a victim host and gains root access. The dataset is by default graph-oriented, meaning that it has been designed with the aim to be used as graph structures and in a real-time environment. Multiple node types exist such as socket, file or memory, whereas edges represent actual system calls like read or fork.

Unicorn [118]: The Unicorn dataset consist of SC-1 and SC-2 datasets, that are created in a private lab environment following the cyber kill chain model. The system provenance was captured with CamFlow during benign and attack activities and over the course of three days. Therefore, provenance graphs can be intuitively built from these datasets.

BETH [146]: The BETH dataset provides more than 8 million data points collected from 23 honeypots in a network. Attacks have been conducted and both network and kernel-level host data were collected, labeled into benign, unusual and evil classes. A graph can be built using the interactions between events and processes available in logs [130].

ADFA-LD [147]: A host intrusion detection dataset based on Linux system calls that was developed by the Australian Defence Force Academy. All data were gathered from hosts with normal background behaviors such as web browsing or LaTeX document editing. A total of 5,925 examples are present in the dataset, already separated into train/validation/attack sets, including 7 classes: Benign, Hydra-FTP, Hydra-SSH, Adduser, Java-meterpreter, Meterpreter, Web shell.

CERT [148]: The CERT Insider Threat Test Dataset is a synthetic insider threat dataset provided by Carnegie Mellon University [149]. In the last v6.2 release version, five scenarios are proposed such as a user logging into another account and searching for files, a user working after hours, etc. Five types of entities exist in the dataset: LDAP, device, email, http and logon with four to five features such as id, date, user, pc, url. More than 135M operations with 101.4GB of data are recorded from the activity of 4,000 users with 46 distinct roles.

V. ROBUSTNESS AGAINST ADVERSARIAL ATTACKS

The use of ML algorithms in intrusion detection is presenting new possibilities for defense, but also new risks. Studies have demonstrated that these learning models can be susceptible to adversarial attacks, which manipulate input data in order to make it harder for the model to detect threats. To better understand the challenges posed by adversarial attacks on graph based intrusion detection, in this section, we will review the existing literature on this topic.

A. BACKGROUND ON ADVERSARIAL ATTACKS AGAINST INTRUSION DETECTION

Adversarial attacks refer to the act of introducing small changes to data with the intention of deceiving a detection model, leading it to produce inaccurate results that benefit the attacker. These changes are designed to be undetectable by humans. In computer vision, this objective is relatively easy to achieve by modifying a small number of pixels [150]. However, for intrusion detection, the efficacy of adversarial attacks is highly dependent on the specific domain, and what works well in one domain may not work in another [151], [152].

Adversarial attacks can target both the training and testing phases of the ML model. During training, an attacker may rely on poisoning attacks to modify the training data to influence the detection performance of the model. Formally, given a training dataset $D = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ and a model f, the attacker seeks to modify D to obtain a new dataset $D' = \{(x'_1, y_1), \ldots, (x'_n, y_n)\}$, where $x'_i \approx x_i$ for all i, such that the performance of f on D' is degraded or manipulated in a specific way. This can be achieved, for instance, by injecting new samples or by flipping the labels of some examples [153].

In contrast, during inference (i.e., when the model is already trained), an attacker may leverage evasion attacks to modify the input data and deceive the model. Precisely, given an input sample x and a model f, the attacker seeks to find a perturbed sample x' such that $f(x') \neq f(x)$ and $x' \approx x$. The goal of evasion attacks is to cause the model to misclassify malicious samples as legitimate. Evasion attacks can be particularly challenging to detect and defend against, as they can take many forms and can be customized to the specific model and scenario. Authors in [154] demonstrated that by altering the payload size of packets and decreasing the packet rate, attackers can create adversarial examples against an NIDS implemented on a Software-Defined Network (SDN) for a DDoS attack. Another approach [152] involves applying legitimate transformations, such as dividing a packet into multiple packets or modifying the delay between packets, to mislead the NIDS while preserving the underlying network protocols.

B. GRAPH-BASED ADVERSARIAL ATTACKS

Adversarial attacks on graph-based models such as GNNs involve manipulating the graph representation of data to mislead the model's predictions. Developing robust defense mechanisms has thus become a necessity for the use of these models in real-life scenarios. Furthermore, as the threat

TABLE 4. Datasets used by each host-based paper.

Paper	Datasets	Performance
Log2vec [22]	CERT, LANL 2015	93%, 91% AUC
DeepHunter [15]	DARPA TC	95.1-100% AUC
Lv et al. [47]	DARPA TC, Private	83.14%, 95.3% macro-F1
THREATRACE [48]	StreamSpot, Unicorn, DARPA TC	99%, 93-95%, 69-95% F1
APT-KGL [128]	DARPA TC, Private	87.04%, 97.8% macro-F1
PROV-GEM [129]	StreamSpot, Unicorn	97%, 89% F1
OC-DHetGNN [121]	Private	96.3% AUC
Lakha et al. [130]	BETH	93.2%, 95.1% AUC
Hu et al. [20]	ADFA-LD	95.55% AUC
Li et al. [51]	StreamSpot, LANL 2015	98%, 83% F1

landscape is constantly evolving, ongoing research is necessary to keep up with the development of more advanced and sophisticated attacks.

In paper [155], a novel approach is introduced to generate Hierarchical Adversarial Attacks (HAA) in order to specifically target GNN-based IDSs in Internet of Things (IoT) networks. The proposed method uses a saliency map technique [156] to identify critical features that can be modified with minimal perturbations. Additionally, a hierarchical node selection algorithm based on random walk with restart is employed to select the most vulnerable nodes with high attack priority. This adversarial attack works in black-box scenario, meaning that only the output prediction is known by the attacker. The proposed HAA method is evaluated using the UNSW-SOSR2019 dataset [157], and results show that it can reduce classification precision by more than 30%. The findings suggest that this approach is an effective strategy for implementing level-aware black-box adversarial attacks against GNN-based IDSs in IoT environment.

The authors of paper [54] presented in Section III-B1, verify the robustness of their GNN model against flow-based adversarial attacks. They notably modify flow-level features to test the robustness of each system, such as incrementing the packet size of attack-related flows, or increasing the inter-packet arrival times, to serve traffic at lower rates. After experiments, the authors claim that their proposed GNN-based NIDS is more robust than traditional ML-based NIDS for several reasons. Indeed, traditional flow-based ML-based NIDS can be highly vulnerable to variations in individual flow features, as these methods solely rely on flow-level features to detect attacks. However, GNN-based NIDSs are able to capture the structural flow patterns of attacks, which remain unchanged even after altering flow features. This allows the model to remain robust in the face of such adversarial attacks. As a result, the proposed GNN model achieves similar accuracy to state-of-the-art ML-based NIDS, but benefits of a more robust prediction against various attacks.

In THREATRACE [48] (see Section IV-B1), the robustness of the GraphSAGE-based model is evaluated against adversarial attacks on host-based intrusion detection. These attacks aim to evade the detection of malicious nodes by finding a small perturbation on the node's features, that is constructed with edges between the node and its neighbors. During the attacks, the feature should thus evade detection while keeping the original malicious function intact. Attacks are developed using optimization-based evasion, that are designed to find the optimal perturbation while minimizing the cost. The authors evaluate the robustness of the model using two settings: one considers that the attacker knows the training data, and the other assumes the attacker knows the model. In both cases, THREATRACE maintains a stable false negative rate against these optimization-based evasion attacks, which further demonstrates the robustness capabilities of GNN-based detection systems.

VI. DISCUSSION AND FUTURE DIRECTIONS

Representing host and network data as graphs results in interesting proprieties that seem promising for cybersecurity applications. Graph representation learning techniques such as GNNs are strong candidates to leverage these graphs and learn robust representations that can detect complex attacks and transcend obfuscation techniques. Indeed, the literature reviewed in Section V-B demonstrates the robustness of GNN-based IDS faced with adversarial attacks, thanks to the robust structure of graphs, challenging to be bypassed by attackers. Furthermore, in Sections III and IV, we showed that in most cases, these GNN models outperform traditional ML algorithms and usually surpass deeper models based on DL, which manifests the powerful faculties of GNNs for intrusion detection.

Despite relatively poor existing literature compared to other domains, applications of GNNs to cybersecurity, and especially intrusion detections, holds great promise. Consequently, we provide future directions to improve research in this area and to further democratize the use of GNNs in attack detection applications.

- We think that the temporal dimension of attacks is a key element to consider for their detection. Indeed, all intrusions take place on a given system spread over time, and the temporal distance between events should also be learned by the model in order to further profile the behavior of attackers. Future research could focus on the incorporation of temporal information in the input data, in order to fully leverage powerful structures such as spatio-temporal graphs and dynamic graphs.

- Obtaining labeled samples from real-life attacks such as APTs remains a main bottleneck in the development of robust detection systems based on ML. For this reason, we think that future research should focus on self-supervised techniques, where the inherent structure of graphs can be leveraged for self-training, or weakly-supervised learning, where even noisy attack behaviors could be considered. Unsupervised methods based on anomaly detection could also be further explored, but we believe that they are not sufficient for attack detection, as an anomaly is not necessarily an attack, leading to a large number of false positives.
- As justified by current literature, the representation of network- and host-based systems as graphs is especially concentrated around two graph structures, which are respectively network flows (leveraged by ∼78% of network papers studied in this survey) and provenance graphs (∼58% of host papers). Consequently, other data such as network sessions and system calls could be further explored, in an attempt to obtain more efficient graph representations for the detection of specific attacks.
- Although most papers focus on the innovation of new models, we believe that important efforts should also be dedicated to the development of large datasets that represent real-life scenarios. Indeed, we showed that a number of works achieve near-perfect detection performance on either small noise-free datasets or on private enterprise data. However, intrusion detection is a hard problem and datasets employed for their detection should be larger and contain various examples in order to achieve reliable performance on a variety of unseen attacks. We therefore think that larger datasets are required to continue the constant progress in this field.
- Finally, it is important to note that there is a gap between the research implementations presented in papers and the actual applications in production environments. Scaling GNN classifiers to large graphs can present significant challenges, as the graph may no longer fit into memory, and computation time can become infeasible. To address these engineering issues, researchers have proposed graph sampling methods to divide the graph into manageable batches and distribute the training across multiple workers [57]. Additionally, model interpretability is a critical concern for many applications, including those in cybersecurity [67], [158]. Despite their importance, these topics remain less discussed in current literature and represent areas for future research.

VII. CONCLUSION

The recent surge of interest in graph machine learning, in particular GNNs, has led to a plethora of applications across various domains. Although the use of GNNs in intrusion detection is relatively new, existing research has demonstrated that representing systems as graph structures offers properties that can enhance the accuracy of detection models. In this paper, we provide knowledge on the extraction of graph structures along with the training of GNN models for downstream classification tasks on both network-based and host-based intrusion detection. We comprehensively review and categorize the state-of-the-art approaches and the datasets used, highlighting the potential of GNNs in generating efficient embeddings for robust detection of various types of intrusions. Finally, we discuss the challenges that may arise when using GNNs in intrusion detection and propose directions for future research.

REFERENCES

[1] P. Mishra, V. Varadharajan, U. Tupakula, and E. S. Pilli, "A detailed investigation and analysis of using machine learning techniques for intrusion detection," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 1, pp. 686–728, 1st Quart., 2019.

[2] M. Alkasassbeh and M. Almseidin, "Machine learning methods for network intrusion detection," 2018, *arXiv:1809.02610*.

[3] H. Liu and B. Lang, "Machine learning and deep learning methods for intrusion detection systems: A survey," *Appl. Sci.*, vol. 9, no. 20, p. 4396, Oct. 2019.

[4] A. Shalaginov, S. Banin, A. Dehghantanha, and K. Franke, "Machine learning aided static malware analysis: A survey and tutorial," in *Cyber Threat Intelligence*. Cham, Switzerland: Springer, 2018, pp. 7–45.

[5] D. Gibert, C. Mateu, and J. Planes, "The rise of machine learning for detection and classification of malware: Research developments, trends and challenges," *J. Netw. Comput. Appl.*, vol. 153, Mar. 2020, Art. no. 102526.

[6] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural Comput.*, vol. 18, no. 7, pp. 1527–1554, Jul. 2006.

[7] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, P.-A. Manzagol, and L. Bottou, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *J. Mach. Learn. Res.*, vol. 11, no. 12, pp. 3371–3408, 2010.

[8] Y. LeCun and Y. Bengio, "Convolutional networks for images, speech, and time series," *Handbook Brain Theory Neural Netw.* vol. 3361, no. 10, p. 1995, Apr. 2015.

[9] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[10] J. Zhang, L. Pan, Q.-L. Han, C. Chen, S. Wen, and Y. Xiang, "Deep learning based attack detection for cyber-physical system cybersecurity: A survey," *IEEE/CAA J. Autom. Sinica*, vol. 9, no. 3, pp. 377–391, Mar. 2022.

[11] Y. Wu, D. Wei, and J. Feng, "Network attacks detection methods based on deep learning techniques: A survey," *Secur. Commun. Netw.*, vol. 2020, pp. 1–17, Aug. 2020.

[12] V.-A. Nguyen, D. Q. Nguyen, V. Nguyen, T. Le, Q. H. Tran, and D. Phung, "ReGVD: Revisiting graph neural networks for vulnerability detection," in *Proc. IEEE/ACM 44th Int. Conf. Softw. Eng., Companion (ICSE-Companion)*, May 2022, pp. 178–182.

[13] C. Lin, Y. Xu, Y. Fang, and Z. Liu, "VulEye: A novel graph neural network vulnerability detection approach for PHP application," *Appl. Sci.*, vol. 13, no. 2, p. 825, Jan. 2023.

[14] Y. Zhang, C. Yang, K. Huang, and Y. Li, "Intrusion detection of industrial Internet-of-Things based on reconstructed graph neural networks," *IEEE Trans. Netw. Sci. Eng.*, early access, Jun. 21, 2022, doi: 10.1109/TNSE.2022.3184975.

[15] R. Wei, L. Cai, L. Zhao, A. Yu, and D. Meng, "DeepHunter: A graph neural network based approach for robust cyber threat hunting," in *Proc. Int. Conf. Secur. Privacy Commun. Syst.* Cham, Switzerland: Springer, Sep. 2021, pp. 3–24.

[16] M. R. Norouzian, P. Xu, C. Eckert, and A. Zarras, "Hybroid: Toward Android malware detection and categorization with program code and network traffic," in *Proc. Int. Conf. Inf. Secur.*, Cham, Switzerland: Springer, 2021, pp. 259–278.

[17] C. Liu, B. Li, J. Zhao, Z. Zhen, X. Liu, and Q. Zhang, "FewM-HGCL: Few-shot malware variants detection via heterogeneous graph contrastive learning," *IEEE Trans. Depend. Secure Comput.*, early access, Oct. 1, 2022, doi: 10.1109/TDSC.2022.3216902.

[18] F. Xia, J. Liu, H. Nie, Y. Fu, L. Wan, and X. Kong, "Random walks: A review of algorithms and applications," *IEEE Trans. Emerg. Topics Comput. Intell.*, vol. 4, no. 2, pp. 95–107, Apr. 2020.

[19] R. Paudel and H. H. Huang, "Pikachu: Temporal walk based dynamic graph embedding for network anomaly detection," in *Proc. NOMS IEEE/IFIP Netw. Oper. Manage. Symp.*, Apr. 2022, pp. 1–7.

[20] Z. Hu, L. Liu, H. Yu, and X. Yu, "Using graph representation in host-based intrusion detection," *Secur. Commun. Netw.*, vol. 2021, pp. 1–13, Dec. 2021.

[21] B. Bowman, C. Laprade, Y. Ji, and H. H. Huang, "Detecting lateral movement in enterprise computer networks with unsupervised graph AI," in *Proc. 23rd Int. Symp. Res. Attacks, Intrusions Defenses (RAID)*, 2020, pp. 257–268.

[22] 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 *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2019, pp. 1777–1794.

[23] B. Perozzi, R. Al-Rfou, and S. Skiena, "DeepWalk: Online learning of social representations," in *Proc. 20th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2014, pp. 701–710.

[24] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2016, pp. 855–864.

[25] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," 2013, *arXiv:1301.3781*.

[26] C. Yang, Z. Liu, D. Zhao, M. Sun, and E. Chang, "Network representation learning with rich text information," in *Proc. 24th Int. Joint Conf. Artif. Intell.*, 2015, pp. 1–7.

[27] C. Tu, "Max-margin deepwalk: Discriminative learning of network representation," in *Proc. IJCAI*, 2016, pp. 3889–3895.

[28] Z. Yang, W. Cohen, and R. Salakhudinov, "Revisiting semi-supervised learning with graph embeddings," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 40–48.

[29] G. H. Nguyen, J. B. Lee, R. A. Rossi, N. K. Ahmed, E. Koh, and S. Kim, "Dynamic network embeddings: From random walks to temporal random walks," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Dec. 2018, pp. 1085–1092.

[30] F. Béres, D. M. Kelen, R. Pálovics, and A. A. Benczúr, "Node embeddings in dynamic graphs," *Appl. Netw. Sci.*, vol. 4, no. 1, pp. 1–25, Dec. 2019.

[31] Y. Wang, Y.-Y. Chang, Y. Liu, J. Leskovec, and P. Li, "Inductive representation learning in temporal networks via causal anonymous walks," 2021, *arXiv:2101.05974*.

[32] W. L. Hamilton, R. Ying, and J. Leskovec, "Representation learning on graphs: Methods and applications," 2017, *arXiv:1709.05584*.

[33] P. Velickovic, W. Fedus, W. L. Hamilton, P. Liò, Y. Bengio, and R. D. Hjelm, "Deep graph infomax," in *Proc. ICLR*, vol. 2, no. 3, 2019, p. 4.

[34] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Proc. Adv. Neural Inf. Process. Syst.*, 29, 2016, pp. 3844–3852.

[35] (2021). *How to Get Started With Graph Machine Learning*. Accessed: Feb. 20, 2023. [Online]. Available: https://gordicaleksa.medium.com/

[36] M. Gori, G. Monfardini, and F. Scarselli, "A new model for learning in graph domains," in *Proc. IEEE Int. Joint Conf. Neural Netw.*, Jul. 2005, pp. 729–734.

[37] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Trans. Neural Netw.*, vol. 20, no. 1, pp. 61–80, Jan. 2009.

[38] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 1263–1272.

[39] P. Battaglia, "Interaction networks for learning about objects, relations and physics," in *Proc. Adv. Neural Inf. Process. Syst.*, 29, 2016, pp. 4509–4517.

[40] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," 2014, *arXiv:1412.3555*.

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

[42] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 1025–1035.

[43] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," 2017, *arXiv:1710.10903*.

[44] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," 2014, *arXiv:1409.0473*.

[45] J. Zhao, X. Liu, Q. Yan, B. Li, M. Shao, H. Peng, and L. Sun, "Automatically predicting cyber attack preference with attributed heterogeneous attention networks and transductive learning," *Comput. Secur.*, vol. 102, Mar. 2021, Art. no. 102152.

[46] J. Zhao, Q. Yan, X. Liu, B. Li, and G. Zuo, "Cyber threat intelligence modeling based on heterogeneous graph convolutional network," in *Proc. 23rd Int. Symp. Res. Attacks, Intrusions Defenses (RAID)*, 2020, pp. 241–256.

[47] M. Lv, C. Dong, T. Chen, T. Zhu, Q. Song, and Y. Fan, "A heterogeneous graph learning model for cyber-attack detection," 2021, *arXiv:2112.08986*.

[48] 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 Trans. Inf. Forensics Security*, vol. 17, pp. 3972–3987, 2022.

[49] X. Wang, H. Ji, C. Shi, B. Wang, Y. Ye, P. Cui, and P. S. Yu, "Heterogeneous graph attention network," in *Proc. World Wide Web Conf.*, May 2019, pp. 2022–2032.

[50] E. Caville, W. W. Lo, S. Layeghy, and M. Portmann, "Anomal-E: A self-supervised network intrusion detection system based on graph neural networks," *Knowl.-Based Syst.*, vol. 258, Dec. 2022, Art. no. 110030.

[51] Z. Li, X. Cheng, L. Sun, J. Zhang, and B. Chen, "A hierarchical approach for advanced persistent threat detection with attention-based graph neural networks," *Secur. Commun. Netw.*, vol. 2021, pp. 1–14, May 2021.

[52] (2023). *HTTPS Encryption on the Web Google Transparency Report*. Accessed: Dec. 23, 2022. [Online]. Available: https://transparencyreport.google.com/https/overview?hl=en

[53] T. Radivilova, L. Kirichenko, D. Ageyev, M. Tawalbeh, and V. Bulakh, "Decrypting SSL/TLS traffic for hidden threats detection," in *Proc. IEEE 9th Int. Conf. Dependable Syst., Services Technol. (DESSERT)*, May 2018, pp. 143–146.

[54] D. Pujol-Perich, J. Suarez-Varela, A. Cabellos-Aparicio, and P. Barlet-Ros, "Unveiling the potential of graph neural networks for robust intrusion detection," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 49, no. 4, pp. 111–117, Jun. 2022.

[55] L. Chang and P. Branco, "Graph-based solutions with residuals for intrusion detection: The modified E-GraphSAGE and E-ResGAT algorithms," 2021, *arXiv:2111.13597*.

[56] F. Harary and R. Z. Norman, "Some properties of line digraphs," *Rendiconti del Circolo Matematico di Palermo*, vol. 9, no. 2, pp. 161–168, May 1960.

[57] I. J. King and H. H. Huang, "EULER: Detecting network lateral movement via scalable temporal link prediction," *ACM Trans. Privacy Secur.*, Mar. 2023.

[58] *Zeek is a Powerful Network Analysis Framework That is Much Different From the Typical IDS You May Know*. Accessed: Nov. 12, 2023. [Online]. Available: https://github.com/zeek/zeek

[59] (2022). *Openargus Home*. Accessed: Nov. 12, 2022. [Online]. Available: https://openargus.org

[60] *Wireshark Download*. Accessed: Nov. 12, 2023. [Online]. Available: https://www.wireshark.org/download.html

[61] *Cisco/Joy: A Package for capturing, Analyzing Network Flow Data, Intraflow Data for Network Research Forensics, and Security Monitoring*. Accessed: Nov. 12, 2022. [Online]. Available: https://github.com/cisco/joy

[62] *Splunk | The Key to Enterprise Resilience*. Accessed: Nov. 12, 2022. [Online]. Available: https://www.splunk.com

[63] *Snort Network Intrusion Detection and Prevention System*. Accessed: Nov. 12, 2022. [Online]. Available: https://www.snort.org

[64] *Home Suricata*. Accessed: Nov. 12, 2022. [Online]. Available: https://suricata.io

[65] J. Zhou, Z. Xu, A. M. Rush, and M. Yu, "Automating botnet detection with graph neural networks," 2020, *arXiv:2003.06344*.

[66] B. Zhang, J. Li, C. Chen, K. Lee, and I. Lee, "A practical botnet traffic detection system using GNN," in *Proc. Int. Symp. Cyberspace Saf. Secur.* Cham, Switzerland: Springer, 2021, pp. 66–78.

[67] W. Weng Lo, G. K. Kulatilleke, M. Sarhan, S. Layeghy, and M. Portmann, "XG-BoT: An explainable deep graph neural network for botnet detection and forensics," 2022, *arXiv:2207.09088*.

[68] J. Carpenter, J. Layne, E. Serra, and A. Cuzzocrea, "Detecting botnet nodes via structural node representation learning," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Dec. 2021, pp. 5357–5364.

[69] J. Zhao, X. Liu, Q. Yan, B. Li, M. Shao, and H. Peng, "Multi-attributed heterogeneous graph convolutional network for bot detection," *Inf. Sci.*, vol. 537, pp. 380–393, Oct. 2020.

[70] A. Protogerou, S. Papadopoulos, A. Drosou, D. Tzovaras, and I. Refanidis, "A graph neural network method for distributed anomaly detection in IoT," *Evolving Syst.*, vol. 12, no. 1, pp. 19–36, Mar. 2021.

[71] 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. NOMS IEEE/IFIP Netw. Oper. Manage. Symp.*, Apr. 2022, pp. 1–9.

[72] Y. Yang and L. Wang, "LGANet: Local graph attention network for peer-to-peer botnet detection," in *Proc. 3rd Int. Conf. Adv. Comput. Technol., Inf. Sci. Commun. (CTISC)*, Apr. 2021, pp. 31–36.

[73] J. Lan, J. Z. Lu, G. G. Wan, Y. Y. Wang, C. Y. Huang, S. B. Zhang, Y. Y. Huang, and J. N. Ma, "E-minBatch GraphSAGE: An industrial Internet attack detection model," *Secur. Commun. Netw.*, vol. 2022, pp. 1–12, Jul. 2022.

[74] Y. Li, R. Li, Z. Zhou, J. Guo, W. Yang, M. Du, and Q. Liu, "GraphDDoS: Effective DDoS attack detection using graph neural networks," in *Proc. IEEE 25th Int. Conf. Comput. Supported Cooperat. Work Design (CSCWD)*, May 2022, pp. 1275–1280.

[75] Y. Cao, H. Jiang, Y. Deng, J. Wu, P. Zhou, and W. Luo, "Detecting and mitigating DDoS attacks in SDN using spatial–temporal graph convolutional network," *IEEE Trans. Depend. Secure Comput.*, vol. 19, no. 6, pp. 3855–3872, Nov. 2022.

[76] S. Govindaraju, W. V. R. Vinisha, F. H. Shajin, and D. A. Sivasakthi, "Intrusion detection framework using auto-metric graph neural network optimized with hybrid woodpecker mating and capuchin search optimization algorithm in IoT network," *Concurrency Comput., Pract. Exper.*, vol. 34, no. 24, Nov. 2022, Art. no. e7197.

[77] F. Liu, Y. Wen, Y. Wu, S. Liang, X. Jiang, and D. Meng, "MLTracer: Malicious logins detection system via graph neural network," in *Proc. IEEE 19th Int. Conf. Trust, Secur. Privacy Comput. Commun. (TrustCom)*, Dec. 2020, pp. 715–726.

[78] Y. Fang, C. Wang, Z. Fang, and C. Huang, "LMTracker: Lateral movement path detection based on heterogeneous graph embedding," *Neurocomputing*, vol. 474, pp. 37–47, Feb. 2022.

[79] X. Sun and J. Yang, "HetGLM: Lateral movement detection by discovering anomalous links with heterogeneous graph neural network," in *Proc. IEEE Int. Perform., Comput., Commun. Conf. (IPCCC)*, Nov. 2022, pp. 404–411.

[80] G. Li, M. Ḿuller, A. Thabet, and B. Ghanem, "DeepGCNs: Can GCNs go as deep as CNNs?" in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 9267–9276.

[81] G. Li, M. Müller, B. Ghanem, and V. Koltun, "Training graph neural networks with 1000 layers," in *Proc. Int. Conf. Mach. Learn.*, 2021, pp. 6437–6449.

[82] Z. Ying, D. Bourgeois, J. You, M. Zitnik, and J. Leskovec, "GNNExplainer: Generating explanations for graph neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 32, 2019, pp. 9244–9255.

[83] J. Layne and E. Serra, "Inferential SIR-GN: Scalable graph representation learning," 2021, *arXiv:2111.04826*.

[84] I. Corona, G. Giacinto, and F. Roli, "Adversarial attacks against intrusion detection systems: Taxonomy, solutions and open issues," *Inf. Sci.*, vol. 239, pp. 201–225, Aug. 2013.

[85] P. W. Battaglia, "Relational inductive biases, deep learning, and graph networks," 2018, *arXiv:1806.01261*.

[86] B. Rahbarinia, R. Perdisci, A. Lanzi, and K. Li, "PeerRush: Mining for unwanted p2p traffic," in *Proc. Int. Conf. Detection Intrusions Malware, Vulnerability Assessment*, Berlin, Germany: Springer, Jul. 2013, pp. 62–82, 2013.

[87] *MAWI Working Group Traffic Archive*. Accessed: Jul. 2, 2023. [Online]. Available: http://mawi.wide.ad.jp/mawi/

[88] M. Sarhan, S. Layeghy, and M. Portmann, "Towards a standard feature set for network intrusion detection system datasets," *Mobile Netw. Appl.*, vol. 27, no. 1, pp. 357–370, Feb. 2022.

[89] B. Yu, H. Yin, and Z. Zhu, "Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting," 2017, *arXiv:1709.04875*.

[90] *Hping3(8) Linux Man Page*. Accessed: Oct. 1, 2023. [Online]. Available: https://linux.die.net/man/8/hping3

[91] X. Song, M. Mao, and X. Qian, "Auto-metric graph neural network based on a meta-learning strategy for the diagnosis of Alzheimer's disease," *IEEE J. Biomed. Health Informat.*, vol. 25, no. 8, pp. 3141–3152, Aug. 2021.

[92] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 1126–1135.

[93] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Proc. Adv. Neural Inf. Process. Syst.*, 26, 2013, pp. 3111–3119.

[94] Y. Dong, N. V. Chawla, and A. Swami, "metapath2vec: Scalable representation learning for heterogeneous networks," in *Proc. 23rd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2017, pp. 135–144.

[95] (2009). *The CAIDA Anonymized Internet Traces Dataset*. Accessed: Jul. 2, 2023. [Online]. Available: https://www.caida.org/data/passive/passive_dataset.xml

[96] S. García, M. Grill, J. Stiborek, and A. Zunino, "An empirical comparison of botnet detection methods," *Comput. Secur.*, vol. 45, pp. 100–123, Sep. 2014.

[97] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," in *Proc. 4th Int. Conf. Inf. Syst. Secur. Privacy*, 2018, pp. 108–116.

[98] (2017). *Applications | Research | Canadian Institute for Cybersecurity | UNB*. Accessed: Jul. 2, 2023. [Online]. Available: https://www.unb.ca/cic/research/applications.html

[99] (2018). *IDS 2018 | Datasets | Research | Canadian Institute for Cybersecurity | UNB*. Accessed: Jul. 2, 2023. [Online]. Available: https://www.unb.ca/cic/datasets/ids-2018.html

[100] N. Moustafa and J. Slay, "UNSW-NB15: A comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)," in *Proc. Mil. Commun. Inf. Syst. Conf. (MilCIS)*, Nov. 2015, pp. 1–6.

[101] Z. Zoghi and G. Serpen, "UNSW-NB15 computer security dataset: Analysis through visualization," 2021, *arXiv:2101.05067*.

[102] C. Wheelus, E. Bou-Harb, and X. Zhu, "Tackling class imbalance in cyber security datasets," in *Proc. IEEE Int. Conf. Inf. Reuse Integr. (IRI)*, Jul. 2018, pp. 229–232.

[103] N. Koroniotis, N. Moustafa, E. Sitnikova, and B. Turnbull, "Towards the development of realistic botnet dataset in the Internet of Things for network forensic analytics: Bot-IoT dataset," *Future Gener. Comput. Syst.*, vol. 100, pp. 779–796, Nov. 2019.

[104] N. Moustafa, "A new distributed architecture for evaluating AI-based security systems at the edge: Network TON_IoT datasets," *Sustain. Cities Soc.*, vol. 72, Sep. 2021, Art. no. 102994.

[105] M. Sarhan, S. Layeghy, N. Moustafa, and M. Portmann, "Netflow datasets for machine learning-based network intrusion detection systems," in *Proc. Int. Conf. Big Data Technol. Appl., Int. Wireless Internet Conf.* Cham, Switzerland: Springer, 2021, pp. 117–135.

[106] *nProbe: An Extensible NetFlow v5/v9/IPFIX Probe for IPv4/v6*. Accessed: Feb. 20, 2023. [Online]. Available: https://www.ntop.org/products/netflow/nprobe/

[107] A. H. Lashkari, G. Kaur, and A. Rahali, "DIDarknet: A contemporary approach to detect and characterize the darknet traffic using deep image learning," in *Proc. 10th Int. Conf. Commun. Netw. Secur.*, Nov. 2020, pp. 1–13.

[108] A. D. Kent, "Cyber security data sources for dynamic network research," in *Dynamic Networks and Cyber-Security*. Singapore: World Scientific, 2016, pp. 37–65.

[109] R. Arantes, C. Weir, H. Hannon, and M. Kulseng, "Operationally transparent cyber (OPTC)," *IEEE Dataport*, to be published, doi: 10.21227/edq8-nk52.

[110] (2022). *Data Model | MITRE Cyber Analytics Repository*. Accessed: Jul. 2, 2023. [Online]. Available: https://car.mitre.org/data_model/

[111] C. T. Giménez, A. P. Villegas, and G. A. Marañón, "Http data set CSIC 2010," in *Information Security Institute of CSIC (Spanish Research National Council)*, vol. 64, 2010. [Online]. Available: https://www.tic.itefi.csic.es/dataset/

[112] A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani, "Toward developing a systematic approach to generate benchmark datasets for intrusion detection," *Comput. Secur.*, vol. 31, no. 3, pp. 357–374, May 2012.

[113] C. Laprade, B. Bowman, and H. H. Huang, "PicoDomain: A compact high-fidelity cybersecurity dataset," 2020, *arXiv:2008.09192*.

[114] D. J. Pohly, S. McLaughlin, P. McDaniel, and K. Butler, "Hi-Fi: Collecting high-fidelity whole-system provenance," in *Proc. 28th Annu. Comput. Secur. Appl. Conf.*, Dec. 2012, pp. 259–268.

[115] D. Tariq, M. Ali, and A. Gehani, "Towards automated collection of application-level data provenance," in *Proc. TaPP*, vol. 12, 2012, p. 16.

[116] A. Chen, Y. Wu, A. Haeberlen, W. Zhou, and B. T. Loo, "Differential provenance: Better network diagnostics with reference events," in *Proc. 14th ACM Workshop Hot Topics Netw.*, Nov. 2015, pp. 1–7.

[117] W. Zhou, Q. Fei, A. Narayan, A. Haeberlen, B. T. Loo, and M. Sherr, "Secure network provenance," in *Proc. 23rd ACM Symp. Operating Syst. Princ.*, Oct. 2011, pp. 295–310.

[118] X. Han, T. Pasquier, A. Bates, J. Mickens, and M. Seltzer, "UNICORN: Runtime provenance-based detector for advanced persistent threats," 2020, *arXiv:2001.01525*.

[119] M. Zipperle, F. Gottwalt, E. Chang, and T. Dillon, "Provenance-based intrusion detection systems: A survey," *ACM Comput. Surv.*, vol. 55, no. 7, pp. 1–36, Jul. 2023.

[120] Z. Li, Q. A. Chen, R. Yang, Y. Chen, and W. Ruan, "Threat detection and investigation with system-level provenance graphs: A survey," *Comput. Secur.*, vol. 106, Jul. 2021, Art. no. 102282.

[121] Z. Huang, Y. Gu, and Q. Zhao, "One-class directed heterogeneous graph neural network for intrusion detection," in *Proc. 6th Int. Conf. Innov. Artif. Intell. (ICIAI)*, Mar. 2022, pp. 178–184.

[122] (2010). *The Open Provenance Model*. Accessed: Dec. 23, 2022. [Online]. Available: https://openprovenance.org/opm

[123] (2013). *PROV-Overview*. Accessed: Dec. 23, 2022. [Online]. Available: https://www.w3.org/TR/prov-overview

[124] (2018). *CamFlow: Practical Whole-System Provenance for Linux*. Accessed: Dec. 23, 2022. [Online]. Available: https://camflow.org

[125] (2023). *Sysmon Sysinternals | Microsoft Learn*. Accessed: Dec. 23, 2022. [Online]. Available: https://learn.microsoft.com/en-gb/sysinternals/downloads/sysmon

[126] *Auditd(8): Audit Daemon Linux Man Page*. Accessed: Dec. 23, 2022. [Online]. Available: https://linux.die.net/man/8/auditd

[127] M. Kapoor, J. Melton, M. Ridenhour, T. Moyer, and S. Krishnan, "Flurry: A fast framework for provenance graph generation for representation learning," in *Proc. 31st ACM Int. Conf. Inf. Knowl. Manage.*, Oct. 2022, pp. 4887–4891.

[128] T. Chen, C. Dong, M. Lv, Q. Song, H. Liu, T. Zhu, K. Xu, L. Chen, S. Ji, and Y. Fan, "APT-KGL: An intelligent APT detection system based on threat knowledge and heterogeneous provenance graph learning," *IEEE Trans. Depend. Secure Comput.*, early access, Dec. 26, 2022, doi: 10.1109/TDSC.2022.3229472.

[129] M. Kapoor, J. Melton, M. Ridenhour, S. Krishnan, and T. Moyer, "PROV-GEM: Automated provenance analysis framework using graph embeddings," in *Proc. 20th IEEE Int. Conf. Mach. Learn. Appl. (ICMLA)*, Dec. 2021, pp. 1720–1727.

[130] 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. IEEE Int. Conf. Big Data (Big Data)*, Dec. 2022, pp. 5756–5764.

[131] Y. Bai, H. Ding, S. Bian, T. Chen, Y. Sun, and W. Wang, "SimGNN: A neural network approach to fast graph similarity computation," in *Proc. 12th ACM Int. Conf. Web Search Data Mining*, Jan. 2019, pp. 384–392.

[132] R. Socher, D. Chen, C. D. Manning, and A. Ng, "Reasoning with neural tensor networks for knowledge base completion," in *Proc. Adv. Neural Inf. Process. Syst.*, 2013, pp. 926–934.

[133] Q. Wang, W. U. Hassan, D. Li, K. Jee, X. Yu, K. Zou, J. Rhee, Z. Chen, W. Cheng, C. A. Gunter, and H. Chen, "You are what you do: Hunting stealthy malware via data provenance analysis," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2020, pp. 1–17.

[134] E. Manzoor, S. M. Milajerdi, and L. Akoglu, "Fast memory-efficient anomaly detection in streaming heterogeneous graphs," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2016, pp. 1035–1044.

[135] M. Du, F. Li, G. Zheng, and V. Srikumar, "DeepLog: Anomaly detection and diagnosis from system logs through deep learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2017, pp. 1285–1298.

[136] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. V. D. Berg, I. Titov, and M. Welling, "Modeling relational data with graph convolutional networks," in *Proc. Eur. Semantic Web Conf.* Cham, Switzerland: Springer, 2018, pp. 593–607.

[137] C. Xiong, T. Zhu, W. Dong, L. Ruan, R. Yang, Y. Cheng, Y. Chen, S. Cheng, and X. Chen, "A practical real-time APT detection system with high accuracy and efficiency," *IEEE Trans. Depend. Secure Comput.*, vol. 19, no. 1, pp. 551–565, Jan. 2022.

[138] P. Groth and L. Moreau, "PROV-overview. An overview of the PROV family of documents," *World Wide Web Consortium*, Apr. 2013.

[139] J. Oliver, C. Cheng, and Y. Chen, "TLSH—A locality sensitive hash," in *Proc. 4th Cybercrime Trustworthy Comput. Workshop*, Nov. 2013, pp. 7–13.

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

[141] L. Ruff, R. Vandermeulen, N. Goernitz, L. Deecke, S. A. Siddiqui, A. Binder, E. Müller, and M. Kloft, "Deep one-class classification," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 4393–4402.

[142] Z. Ying, J. You, C. Morris, X. Ren, W. Hamilton, and J. Leskovec, "Hierarchical graph representation learning with differentiable pooling," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 4805–4815.

[143] (2023). *Transparent Computing*. Accessed: Jun. 2, 2023. [Online]. Available: https://www.darpa.mil/program/transparent-computing

[144] X. Han, "Streamspot dataset, Harvard dataverse," Harvard Univ., Cambridge, MA, USA, Tech. Rep., 2018, doi: 10.7910/DVN/83KYJY.

[145] (2016). *Sbustreamspot/Sbustreamspot-Data: Datasets Used in the StreamSpot Experiments*. Accessed: Jun. 2, 2023. [Online]. Available: https://github.com/sbustreamspot/sbustreamspot-data

[146] K. Highnam, K. Arulkumaran, Z. Hanif, and N. R. Jennings, "BETH dataset: Real cybersecurity data for anomaly detection research," *TRAINING*, vol. 763, pp. 1–8, Jan. 2021.

[147] G. Creech and J. Hu, "A semantic approach to host-based intrusion detection systems using contiguousand discontiguous system call patterns," *IEEE Trans. Comput.*, vol. 63, no. 4, pp. 807–819, Apr. 2014.

[148] J. Glasser and B. Lindauer, "Bridging the gap: A pragmatic approach to generating insider threat data," in *Proc. IEEE Secur. Privacy Workshops*, May 2013, pp. 98–104.

[149] (2020). *Insider Threat Test Dataset*. Accessed: Jun. 2, 2023. [Online]. Available: https://kilthub.cmu.edu/articles/dataset/Insider_Threat_Test_Dataset/12841247/1

[150] J. Su, D. V. Vargas, and K. Sakurai, "One pixel attack for fooling deep neural networks," *IEEE Trans. Evol. Comput.*, vol. 23, no. 5, pp. 828–841, Oct. 2019.

[151] G. Apruzzese, M. Andreolini, L. Ferretti, M. Marchetti, and M. Colajanni, "Modeling realistic adversarial attacks against network intrusion detection systems," *Digit. Threats, Res. Pract.*, vol. 3, no. 3, pp. 1–19, Sep. 2022.

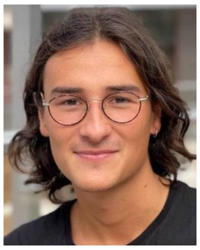
[152] M. J. Hashemi, G. Cusack, and E. Keller, "Towards evaluation of NIDSs in adversarial setting," in *Proc. 3rd ACM CoNEXT Workshop Big DAta, Mach. Learn. Artif. Intell. Data Commun. Netw.*, Dec. 2019, pp. 14–21.

[153] M. Kloft and P. Laskov, "Online anomaly detection under adversarial impact," in *Proc. 13th Int. Conf. Artif. Intell. Statist., JMLR Workshop Conf.*, 2010, pp. 405–412.

[154] J. Aiken and S. Scott-Hayward, "Investigating adversarial attacks against network intrusion detection systems in SDNs," in *Proc. IEEE Conf. Netw. Function Virtualization Softw. Defined Netw. (NFV-SDN)*, Nov. 2019, pp. 1–7.

[155] X. Zhou, W. Liang, W. Li, K. Yan, S. Shimizu, and K. I.-K. Wang, "Hierarchical adversarial attacks against graph-neural-network-based IoT network intrusion detection system," *IEEE Internet Things J.*, vol. 9, no. 12, pp. 9310–9319, Jun. 2022.

[156] M. Ahmadi, M. Hajabdollahi, N. Karimi, and S. Samavi, "Context-aware saliency map generation using semantic segmentation," in *Proc. Electr. Eng. (ICEE), Iranian Conf.*, May 2018, pp. 616–620.
[157] (2019). *Data Collected for ACM SOSR 2019/Attack and Benign Data*. Accessed: Feb. 20, 2023. [Online]. Available: https://iotanalytics.unsw.edu.au/attack-data.html
[158] H. He, Y. Ji, and H. H. Huang, "Illuminati: Towards explaining graph neural networks for cybersecurity analysis," in *Proc. IEEE 7th Eur. Symp. Secur. Privacy (EuroS&P)*, Jun. 2022, pp. 74–89.



TRISTAN BILOT received the M.Sc. degree in computer security and systems from the EPITA Engineering School, France, in 2022. He is currently pursuing the Ph.D. degree with the ROCS Team (LISN), Université Paris-Saclay. He is also a Research Scientist with Iriguard as part of his thesis. At EPITA, he was jointly with LSE on the applications of graph machine learning to the detection of phishing websites. Between 2019 and 2022, he gained professional experience by working as a Software Engineer and a Data Engineer apprentice within the retail corporation Carrefour. His current research interests include the applications of deep learning and graph representation learning to cybersecurity.

NOUR EL MADHOUN received the joint master's degree in networks/computer science from Sorbonne Université and Télécom ParisTech, in 2014, and the Ph.D. degree in cybersecurity/computer science from Sorbonne Université, in 2018. She is currently an Associate Professor in computer science, cybersecurity, and blockchain and the Head of the Engineering Cycle "Digital Security and Networks" at ISEP - Engineering School in Paris. She is also an Associate Researcher at Sorbonne Université/LIP6 and Université Paris Saclay/LISN. At Sorbonne Université, she became an ATER in 2017. In 2018, she gained industry experience through working as a Post-Doctoral Researcher on blockchain and smart-contract technologies at Orange Labs. From 2019 to 2020, she joined ISEP - Engineering School in Paris, as an Associate Professor in cybersecurity and blockchain in addition to overseeing the engineering cycle "Digital Security and Networks". From 2020 to 2022, she joined, EPITA - Engineering school in Paris, as an Associate Professor in cybersecurity and blockchain. From 2020 to 2022, she joined with EPITA and the Engineering School in Paris as an Associate Professor in cybersecurity and blockchain. Her current research interests include network security, machine learning, deep learning, cryptographic protocols and blockchain, and smart-contracts technologies.

KHALDOUN AL AGHA received the Graduate degree from CentraleSupelec and the Ph.D. degree in computer science. He is currently a Full Professor with Paris-Saclay University. He led several international projects on mobile networks and was an invited professor in Japan, Spain, and Chile. He co-founded EIT Digital, a pan-European community that aims to support digital innovation. From 2010 to 2013, he directed the action line on digital cities. He is also the Co-Founder of Green Communications, a company that proposes edge computing solutions to reduce the internet carbon footprint.

ANIS ZOUAOUI received the engineering degree from Ecole Nationale des Sciences de l'Informatique (ENSI), Tunisia, in 2003, and the M.B.A. degree from the University of Liverpool, in 2010. As an expert in bytecode instrumentation, profiling, and performance engineering. He has extensive experience working on source code audits for performance and security optimization. In 2010, he founded Adservio, a technology company that has since been recognized among Deloitte fast 50 in France (2015 and 2016), financial times' top 1000 fast-growing companies in Europe, since 2017. He received the Seal of Excellence from the European Commission, in 2019, and was awarded the IT Night innovation Trophy for Co-Construction for a Machine Learning Project. In 2022, he co-founded Iriguard as a subsidiary of Adservio, specializing in cybersecurity services. Committed to innovation, Iriguard invests heavily in research and development, with a particular focus on the applications of deep learning and graph representation learning within the cybersecurity domain.