Graph Neural Network based Intrusion Detection and its Robustness against Adversarial Attacks

Romain MOREAU[1] **Thomas WINNINGER**[1] **Gregory BLANC**[2]

[1]Student, Réseaux et Services de Télécom, Télécom SudParis, Évry, France
[2]Supervisor, Réseaux et Services de Télécom, Télécom SudParis, Évry, France

Abstract

The increasing number of cyberthreats necessitates advanced Intrusion Detection Systems (IDS) for effective network security, as security analysts can not handle all the threats by themselves. Traditional IDS, classified into Network IDS (NIDS) and Host-based IDS (HIDS), have limitations that can be potentially addressed through Machine Learning (ML) advancements. Many studies have tried classical ML techniques, but they have limitations capturing long and complex attacks. In addition, these techniques are very vulnerable to adversarial attack. However, a new type of ML has proven to be particularly efficient in the task of intrusion detection: Graph Neural Networks (GNNs). GNNs offer robust data analysis capabilities by leveraging the structural properties of the network, which traditional method struggle with. This work explores the application of GNNs in IDS, and their robustness against adversarial attacks. Key GNN architectures, including Graph Convolutional Networks (GCN), Graph Sample and Aggregate (GraphSAGE), and Graph Attention Networks (GAT), are discussed in the context of their ability to capture the semantic of attacks. Furthermore, we examine the vulnerabilities of GNNs to adversarial perturbations and various attack strategies, emphasizing the importance of developing resilient defense mechanisms. We conclude with a discussion on the practical implementation of GNN-based IDS, highlighting the balance between detection accuracy and computational feasibility.

1 Introduction

To fight against cyberthreat and protect network systems, there are many Intrusion Detection System – IDS – to monitor single or multiple hosts. The most general classifications are Network IDS (NIDS) and Host-based IDS (HIDS). The first one performs analysis on traffic to and from all devices of the network, it safeguards every device from unauthorized access. An example is to put a NIDS along the firewall to watch if anyone tries to break it. For HIDS, this one will monitor packets coming to and from a device, and it will alert the administrator if something is suspicious.

However, with recent technologies and the progresses in Machine Learning (ML), it might be possible to improve intrusion detection with neural networks. Due to hidden layers and non-linear modelling, neural network-based IDS allows analysing huge volumes of data. But if machine learning is a benefit to many companies, it is also a weapon for cyberattacks. Therefore, the challenge is to find an efficient class of ML that is robustness against adversarial attacks, and that can powerfully process data.

A great option is Graph Neural Network (GNN), where information can be represented as graphs. Due to this complex and huge subject, this study will be limited to GNN as Host-IDS and Network-IDS, and how far it is robust against attacks.

2 Graph Neural Networks

The definition of the graph used in this work is as:

$$G = (V, E) \tag{1}$$

Where V denotes the set of nodes and E the set of edges. The graph can be heterogeneous, which means the nodes can have various types, or homogenous, there is only one type of node. The graph is described by its adjacency matrix A ($|V| \times |V|$). Its nodes features set is represented by a matrix X ($|V| \times d_v$), with d_v the dimension of the nodes' feature space. It is equally possible to define X_{egdes} ($|E| \times d_e$), to attach an embedding to each edge for edge classification or prediction [1].

2.1 Graph Encoding and Random Walks

Deep learning on graph networks represent a challenging task as graphs can have an arbitrary size and a complex topological structure, in contrary to an image for instance, that have a spatial locality, on a grid. Moreover, graphs often have multimodal features and lack ordering. An encoding should not be dependent on a reference point as this has no sense in graph theory. Thus, the importance to find another way to map similar nodes in the graph closely in the embedding space as shown in Figure 1.

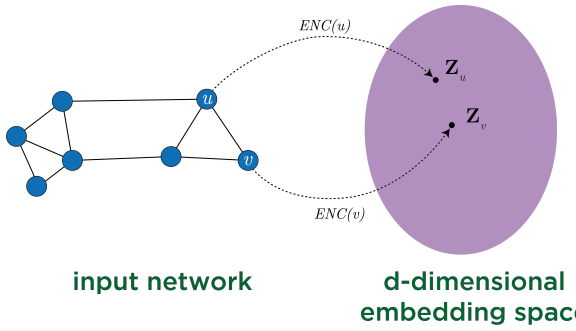
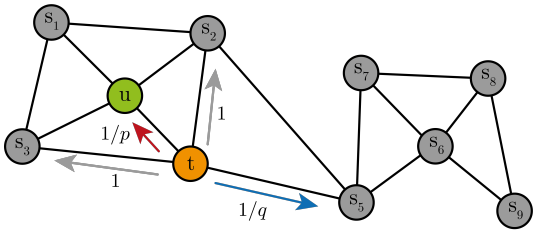


Figure 1: The embedding should reflect the structure of the graph, i.e. $\text{similarity}(u, v) \approx \mathbf{Z}_u^\top \mathbf{Z}_v$ [2].

To achieve this, a naive algorithm to begin with is the **Random Walk Optimization**. It consists in running a short fixed-length random walk starting from each node u, collecting the multiset of nodes visited v, and optimizing the embeddings such that $\Pr\big(v_{\text{visited}} \mid u_{\text{starting point}}\big) \propto \mathbf{Z}_u^\top \mathbf{Z}_v$. This technique is still frequently used on multiple tasks like intrusion detection or adversarial attacks on graphs neural networks. It has been improved with algorithms like Deep Walk or Node2Vec.



$$\alpha_{pq}(u,x) = \begin{cases} 1/p & \text{if } d(u,x) = 0 \\ 1 & \text{if } d(u,x) = 1 \\ 1/q & \text{if } d(u,x) = 2 \end{cases}$$

$d(u,x)$: shortest path length between node u to x

Figure 2: The Node2Vec algorithm uses biased random walks to control how fast the walk explores and leaves the neighbourhood of the starting node [3].

2.2 Graph Convolutional Network (GCN)

To apply machine learning to node embedding, common techniques have been taken from traditional neural networks. The first one is the idea of convolutions. As convolutions collect information from the neighbours of each pixel to update the features of this same pixel, they are well suited for the task of embedding nodes. In a way, Graph Convolutional Networks are the generalization of convolutional neural network to non-euclidean spaces. At each iteration of a GCN layer, the following operation is performed on the features of each node:

$$\mathbf{h}_u^{(l+1)} = \sigma\left(\sum_{u \in \mathcal{N}_v \cup \{u\}} \frac{1}{c_{uv}} \mathbf{W}^{(l)} \mathbf{h}_v^{(l)}\right) \tag{2}$$

Where $\mathcal{N}_v$ are the neighbours of the node u, $\frac{1}{c_{uv}}$ is a normalization coefficient, $\mathbf{W}^{(l)}$ corresponds to the weight parameter to transform information from node v, and σ represent the non-linear activation, usually a ReLU. The embedding of each node is then updated at each GCNConv layer with $\mathbf{h}$ $\left(\mathbf{h}^{(0)} = X\right)$.

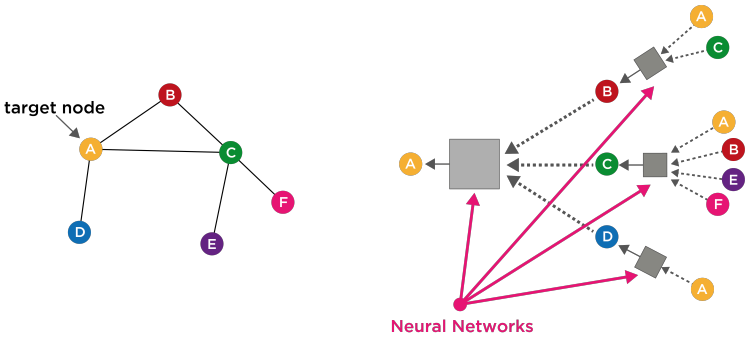


Figure 3: The structure of a single layer of a GCN (GNNVis [4]).

As computing new features node per node is unfeasible in practice, GNN and other common graph-based algorithms use operations on matrices. The previous equation (2) becomes:

$$\mathbf{H}^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} \mathbf{H}^{(l)} \mathbf{W}^{(l)}\right) \tag{3}$$

Where $\tilde{A} = A + I_N$, and $\tilde{D}_{ij} = \sum_j \tilde{A}_{ij}$. One can also define $\hat{A} = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}$. A GCN with a single layer is then defined by:

$$Z = \text{softmax}\left(\hat{A}\sigma\left(\hat{A}\mathbf{X}\mathbf{W}^{(1)}\right)\mathbf{W}^{(2)}\right) \tag{4}$$

2.3 Graph Sample and Aggregate (GraphSAGE)

The generalization to GCN is the GraphSAGE structure. The general equation is given below (5).

$$\mathbf{h}_u^{(l)} = \text{UPDATE}\Big(\mathbf{h}_u^{(l-1)}, \text{AGGREGATE}\Big(\Big\{\mathbf{h}_v^{(l-1)}, \forall v \in \mathcal{N}(u)\Big\}\Big)\Big) \tag{5}$$

Where the AGGREGATE function is a generic function that could be a mean, a pooling or even a Long-Short Term Memory unit (LSTM).

The GraphSAGE architecture is wildly implemented as it has a significant advantage: its speed. In fact, the aggregation function is unapplied on all the neighbours, but only on a fraction of them. This fraction is chosen randomly, and this downsamples allows GraphSAGE structure to be stored in dense adjacency lists, which drastically improves efficiency. For this reason, many IDS use GraphSAGE to perform detection in real time on large datasets.

Another factor that enhance the practicability of GraphSAGE is its inductive nature. Graph learning can be either inductive or transductive. The first one is the same as supervised machine learning. However, in transductive learning both training and testing dataset is exposed to the model in the learning phase itself, only the features of the test nodes are unrevealed. As the GCN is transductive, each step of the learning algorithm needs the whole graph, which can be impracticable in a real scenario. The transductive issue will be discussed in the following sections.

2.4 Graph Attention Network (GAT)

GCN and GraphSAGE use aggregation functions that do not take into account the nature of the edge, the functions are often orderless, like mean or max. However, it may be useful to address each neighbour differently. Thus, the graph attention networks an attention weight α_{uv} to the previous equation, which gives:

$$\mathbf{h}_u^{(l)} = \text{UPDATE}\left(\mathbf{h}_u^{(l-1)}, \sum_{v \in \mathcal{N}(u)} \alpha_{uv} \mathbf{W}^{(l)} \mathbf{h}_v^{(l-1)}\right) \tag{6}$$

In GCN and GraphSAGE, this weight is equal to $\frac{1}{|\mathcal{N}_u|}$, whereas in GAT, it is computed with a MLP layer and a softmax function.

GAT and their counterparts, the Heterogeneous Graph Attention Network (HAN) are used for their attention mechanism as they can capture the semantic of meta-paths in the graph. This could correspond to botnet activities for instance, as we will discuss in more detail in incoming sections.

The other GNN described in this work are based on the same techniques than the ones presented, usually changing the UPDATE or the AGGREGATION functions. A slightly different last one would be the less common Graph Isomorphism Network (GIN) [5], which is based on different assumptions, i.e. the injectivity of the involved functions, to serve a different goal: discriminative power. We still choose to mention it, as it is used in some of the presented works to bypass a number of the other GNN structures' limitations.

3 GNN-based Intrusion Detection Systems (IDS)

There are a few advantages to use GNN for Introduction Detection. Firstly, the **structural property of data**, where data are semi-structured information. GNNs excel in discerning and learning from these structural relationships, and it is crucial for optimal detection performance. Secondly, the **higher-order interaction**. There are structural similarities between data: GNN can subsequently find nodes that exhibit similar patterns of information propagation than an infected node. Finally, the **behavioural supervisory signal**. GNN-based models can be improved by incorporating additional context.

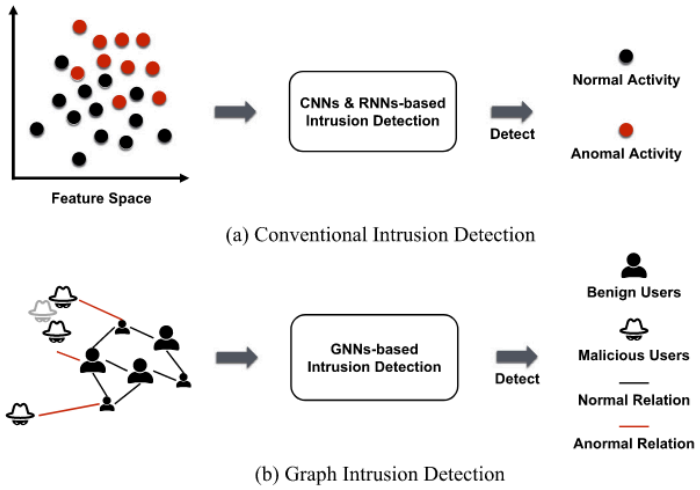


Figure 4: Difference between conventional IDS and GNN-based IDS

To begin our research, we first read the survey of Bilot et al. (2023) [6]. This survey states that many GNN-based defences use random walk methods, but these techniques are usually challenging to apply in practice due to the inductive setting. Moreover, these methods are highly dependent on the hyperparameters and tend to prioritize proximity information above structural information. It also mentions spectral methods tend to be less used as they are inherently transductive. Thus, we will focus on more advanced GNN structures, like GCN, Heterogenous Attention Network or Hyper Graphs Neural Networks. The survey lists studies using random walk classification [7], [8], GCN architecture [9]–[12], GIN architecture [13], [14], GraphSAGE architecture [15]–[18], and even RNN architectures [19]. Other studies tried capturing the semantic of attacks with meta paths [20]–[22].

The general architecture for a GNN-based IDS is composed of three classical steps: the preprocessing, the embedding and the detection and training step. This procedure is shown in Figure 5.

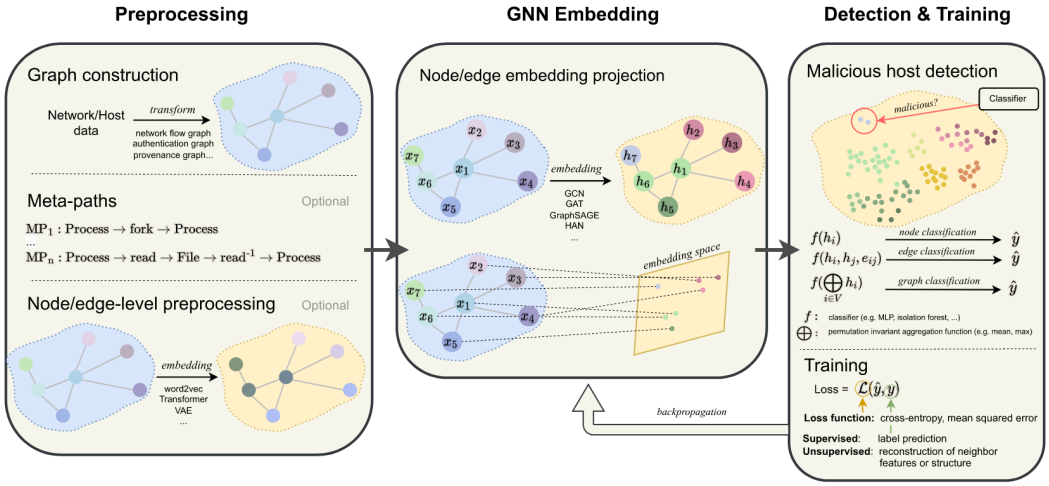


Figure 5: General architecture for intrusion detection with GNN-based methods [6].

There are multiples ways to represent network data as graphs. One can construct **Packet Graph**, to deal directly with IP packets; **Flow Graph**, to capture the semantic of a whole TCP or UDP communication; or **Authentication Graph** that represent, as its name suggests, IP addresses as nodes and authentications as edges.

Packet graph network is rarely used in literature. As the majority of the traffic is encrypted under TLS [23], the payload analysis in unreliable, combined with the fact that creating edges for each packet creates scalability issue. In that manner, we will focus on the other two graph structures.

There are two main classes: **Static graph**, for analysing historical and fixed data structures, and **Dynamic graph**, for real time and immediate response.

3.1 Graph embedding: a technique to capture the semantics of attacks

Studies showed different ways of encoding network information into graphs. The most basic case is to encode systems as nodes and information flows as edges. This approach, made for instance by [9], can then be used to apply GCN to detect malicious nodes. However, such a raw encoding lacks information, like long-term dependences traditional GCN are unable to catch due to the over smoothing problem [1]. In point of fact, as the diameter of a graph tends to be low compared to the number of nodes (usually six in social networks), and as a m-layers GCN aggregates information from the m-hop neighbours, information is diluted to the point of uselessness when the number of layers is too significant [9], [10]. This is a reason why using only GNN layers may be unappropriate.

To assess this issue, Lo developed XG-BoT [13], a framework to capture the long dependences before using GNN layers. As shown in Figure 6, each node's features go through a GIN network before being used in the node embedding part.

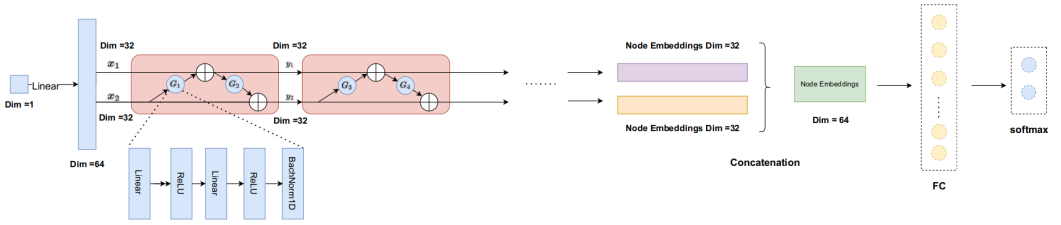


Figure 6: XG-BoT architecture: using GIN before conventional GNN layers to overcome the over-smoothing issue.

As we shall see in the following examples, GNN-based IDS do not only use on type of GNN to perform detection, it remains purely a part of the system. The rest is frequently achieved by other ML architectures, or special pre/post-processing. For instance, some studies tried meta-paths as a preprocessing technique [11], [20]–[22]. The goal is to handcraft common paths involving multiple nodes and edges, that the GNN model will then be able to recognize and generalize.

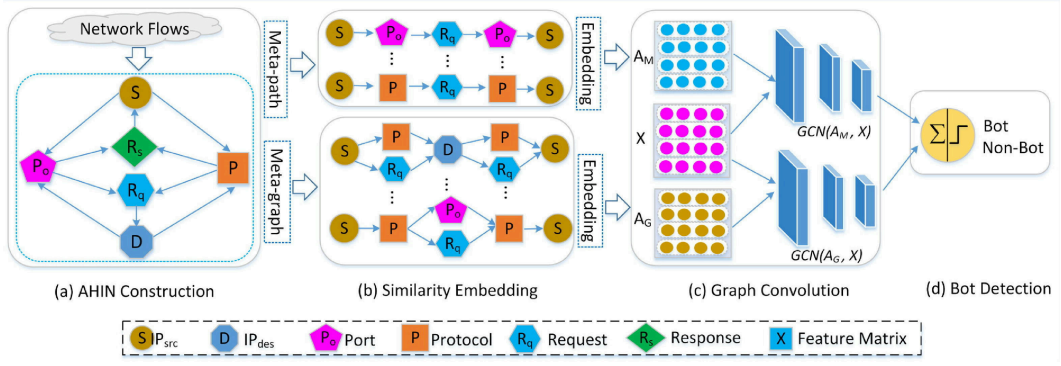


Figure 7: An heterogenous graph with meta-path preprocessing [11].

Meta-paths are frequently used as they offer explainability and a low rate of false negatives [20]. They can be used in other scenario, like intrusion detection on a single machine. The graph is then a provenance graph from processes inside the machine. An example can be found with the study [24]. This study chooses to use GAT and graph sampling to drastically reduce the size of sub-graphs.

Flows are not always represented as edges, some studies used nodes for systems and nodes for edges [25], we show their results in Section 3.2.

As we mentionned earlier, the GraphSAGE architecture is popular for detection intrusion tasks, as it can scale on large graphs. For example, the Anomal-E [18] frameworks is based on this GraphSAGE architecture and implements a self-supervised way to detect intrusions. It is interesting as it relies on both graph methods and traditional anomaly detection methods. It also does not need any label, as the encoding part is done with the Deep graph infomax method [26], which is an unsupervised one, this makes the approach very practical to real-world scenarios. Hence in this case, the graph structure is only used to aggregate information.

A study [27] (Section 6.7) describes the use of real-time detection with an unsupervised graph. This approach describes a graph construction on flow size classification. In this case, preprocessing allows reducing density and then identify malicious vertices with a loss function. The interesting points are the three metrics which allow evaluating the information retained by exiting traffic: the amount of information, with Shannon entropy; the scale of data; the density of information.

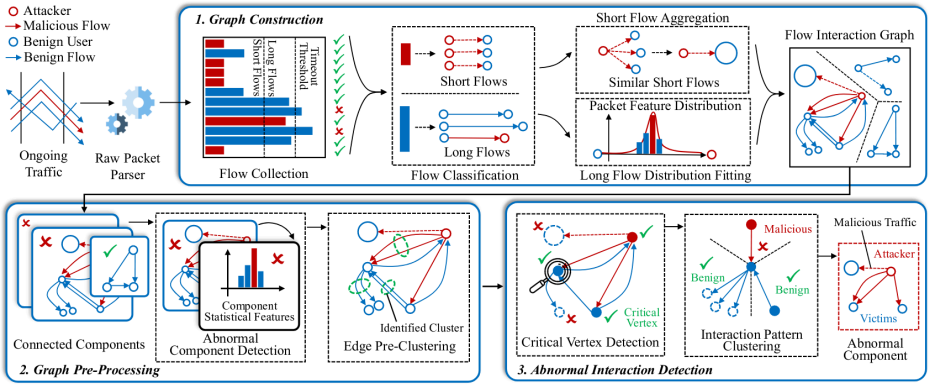


Figure 8: The structure of HyperVision: subgraph are constructed using flow interactions (Section 6.7).

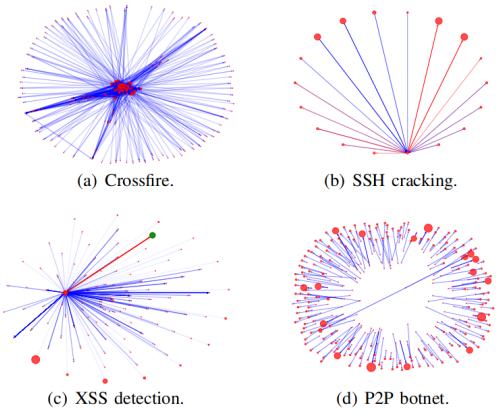


Figure 9: Example of subgraphs for common attacks (Section 6.7).

For real-time detection, dynamics graphs needs Flow that are modelled as a sequence of a per-packet feature represented by an aperiodic irreducible discrete-time Markov chain.

3.2 Graph detection against traditional detection

Pujol-Perich et al. [25] analysed the advantage of using GNN-based IDS over traditional IDS. They use nodes to encode systems and flows, but their approach is similar to the previous ones. They compared basic IDS techniques, like Random Forests or ID3 decision tree to their method. The table Table 1 resumes the results.

Table 1: Accuracy of different ML-based NIDS over the CICIDS2017 dataset.

Class label	MLP	AdaBoost	RF	ID3	GNN
Benign	0.67	0.68	**0.99**	**0.99**	**0.99**
SSH-Patator	0.0	0.0	**0.99**	**0.99**	0.98
FTP-Patator	0.0	0.0	**0.99**	**0.99**	**0.99**
DoS GoldenEye	0.12	0.0	0.97	0.96	**0.99**
DoSHulk	0.63	0.63	**0.99**	**0.99**	**0.99**
DoS Slowhttptest	0.01	0.0	**0.98**	**0.98**	0.97
DDoS	0.51	0.0	**0.99**	**0.99**	**0.99**
Web-Brute Force	0.0	0.0	**0.82**	0.76	0.73
Web-XSS	0.0	0.0	0.69	0.65	**0.83**
Bot	0.0	0.0	**0.98**	**0.98**	**0.98**
Port Scan	0.78	0.0	**0.99**	**0.99**	**0.99**

But the authors mention that the real value of GNN are their robustness to perturbations. As shown in the Figure 10.

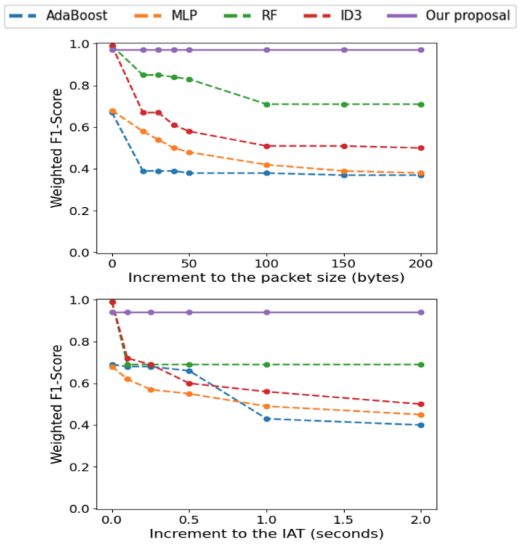


Figure 10: Evaluation of the accuracy of ML-based NIDS under variations of the packet size (top) and the inter-arrival time (bottom) [25].

However, even if GNN may handle normal perturbations, we may see in the next section that they are vulnerable to adversarial perturbations.

4 Robustness of Graph Neural Networks

Deep learning models on graphs excel in tasks like node classification, link prediction, and graph clustering. However, they reveal vulnerabilities to adversarial examples, making them uncertain and unreliable. Given their use in IDS, ensuring the robustness of GNNs is crucial. A typical attack scenario would represent an attacker that compromised a node, i.e. a system, who tries to perform additional benign actions that are useless for the attack. However that compromise the GNN based IDS, thus making the attack stealthy. Put differently, if an IDS detects the attack and classifies the targeted node as compromised, the attacker wants to perform new actions to change the prediction of the IDS for the targeted node. This attack is depicted in the Figure 11.

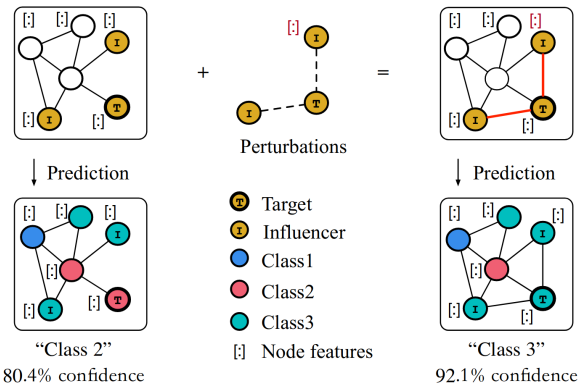


Figure 11: A misclassification of the target caused by small perturbations of the graph structure and node features. [28]

This attack scenario can be formalized: given $\mathcal{T} \subset V$ the target set of nodes, the attacker aims to find a perturbation in the graph, $\hat{G} = G + \Delta G$, such that the new graph $\hat{G}$ satisfies:

$$\begin{aligned} &\max_{\hat{G} \in \Psi(G)} \sum_{t_i \in \mathcal{T}} \mathcal{L}\Big(f_{\theta^*}\Big(\hat{G}, X, t_i\Big), y_i\Big) \\ &\text{s.t. } \theta^* = \underset{\theta}{\operatorname{argmin}} \sum_{v_i \in V} \mathcal{L}\Big(f_{\theta^*}\Big(\hat{G}, X, v_i\Big), y_i\Big) \end{aligned} \tag{7}$$

To make the attack as imperceptible as possible, the attacker also uses a comparison function $\mathcal{Q}$, such that:

$$\begin{aligned} &\mathcal{Q}\Big(\hat{G}, G\Big) < \varepsilon \\ &\text{s.t. } \hat{G} \in \Psi(G) \end{aligned} \tag{8}$$

Where Ψ defines the perturbation space of attacks, f the deep learning model with its set of parameters θ, and the loss function $\mathcal{L}$ [28].

4.1 Attacker's Knowledge

The attack can be a **White-box Attack**, a **Gray-box Attack** or a **Black-box attack**. As white-box attacks are impracticable in the real world, many papers focus on the grey-box attack, which implies knowledge only on the targeted nodes. This fits a scenario where the attacker has access to the target node and tries to become stealthy by changing the target node's prediction.

A new type of attack has emerged and has become widely used because of its practicability: the **No-box Attack**. The principle is to create a surrogate model based on the limited understanding of the target model. The attack is then performed on the surrogate model as a white-box attack. Its success will depend on the transferability of the surrogate model.

The attacker can have a **Perfect Knowledge**, **Moderate Knowledge** or a **Minimal Knowledge**. Even if the first case is impracticable, it is the most common case used in previous studies (2022).

4.2 Types of attacks

Contrary to other machine learning fields where attacks such as model extraction or membership inference attack exist, attacks on graph neural networks remain focused on two categories: **Poisoning Attacks** and **Evasion Attacks**. The popularity of the first one can be explained by the wild use of transductive learning which requires test samples, but not their label, during the training stage.

While poisoning attacks can be defined according to the general equation (7). Evasion attacks occur after the target model is well trained on a clean graph.

4.3 Attack Strategy

The attack can be either a **Topology Attack** that targets the adjacency matrix by adding or removing edges. Or a **Feature Attack**, which alter the features of the targeted nodes. In both cases, a budget Δ can be defined:

$$\underbrace{\sum_{u,i}\left|\mathbf{X}^{(0)}_{ui} - \mathbf{X}'_{ui}\right|}_{\text{Feature budget}} + \underbrace{\sum_{u<v}\left|A^{(0)}_{uv} - A'_{uv}\right|}_{\text{Topology budget}} \leq \Delta \tag{9}$$

4.4 Taxonomies of Attacks

There are plenty of techniques used to perturbate the GNN's predictions. Some of them are specific to graph learning, such as random walk attacks; other ones come from different machine learning fields, like the gradient-based attacks.

The survey of Chen et al. [28] gave a list of such techniques (2022). There are gradient-based algorithms, such as the iterative gradient attack with an auto-encoder framework by [29] or [30], reinforcement learning based attack methods with [31] or [32], random walk or GCN as surrogate model to attack general GNN [33]–[37], meta-gradients attacks [38] (2024), generative adversarial network attacks [39], algorithms based on deepwalk, eigen decomposition and genetic algorithm [33], [40], auto-encoder based algorithms [41], and heuristic algorithm [42].

4.4.1 Use of surrogate models

As mentioned, some studies used simpler model to perform their attack. In the case of a GCN, a surrogate model can be produced by removing the non-linearities [36], i.e. the activation functions. The surrogate model then become:

$$Z' = \operatorname{softmax}\Big(\hat{A}\hat{A}\mathbf{X}\mathbf{W}^{(1)}\mathbf{W}^{(2)}\Big) = \operatorname{softmax}\Big(\hat{A}^2\mathbf{X}\mathbf{W}\Big) \tag{10}$$

This method is described in the appendix Section 6.2. Another study uses the LinLBP method [37] as a surrogate model. The equation then becomes:

$$p = q + A \odot Wp \tag{11}$$

With p and q the evolving embedding. The adversarial edges are then transferred to be used on GCN. The method is explained in Section 6.3.

4.4.2 Finding techniques to ease the computation

To determine if adding or removing an edge can perturbate a system, it is necessary to compute the new adjacency matrix, which can be computationally intensive. This is why studies tried to find approximations or new ways of computing the adjacency matrix. For example, in Zügner's paper [36], they found the new adjacency matrix could be written as:

$$\left[\hat{A}'^2\right]_{uv} = \left[\hat{A}^2\right]_{uv} + \Delta_{u,v,m,n} \tag{12}$$

With $\Delta_{u,v,m,n}$ computed in $O(1)$.

Bojchevski et al. [33] found a way to manipulate easily the spectrum of the graph. This is done by computing the potential change in the eigen values when adding or removing an edge:

$$\Delta\lambda_y = \Delta w_{ij}\big(2u_{yi}u_{yj} - \lambda_y\big(u_{yi}^2 + u_{yj}^2\big)\big) \tag{13}$$

As mentioned in Section 6.4, spectrum based detection can be bypassed using this technique.

4.4.3 Current Limitations

The two main limitations are the **Unnoticeability** and the **Scalability**. Even if studies have often defined an attack budget Δ, it is lacking in semantics. As mentionned in [36], the graph's statistics, such as the degree low, should remain intact, thus the attack should preserve the power distribution law. The second limitation is not limited on graphs, but it can be even more problematic in the field of Graph Neural Networks, as many attacks require the whole graph to perform.

High-degree nodes are much harder to attack than low-degree nodes. Thus, the Δ chosen in studies like [36], [33] is linear in the degree of the node, i.e. $\Delta = d_{v_0} + a$, ($a = 2$ in [36] and $a = 3$ in [33]).

Most of the studies use the Cora dataset or social networks, thus it is yet difficult to know if attacks can be efficient on more complicated datasets, especially on IDS.

4.5 Taxonomies of Defenses

Currently, the amount of research for defense is far less that that of attack on the graph domain. Moreover, as Chen's survey [28] mentions it, most existing works (2022) for defense only focus on node classification tasks. It cites a list of used techniques. There are techniques based on preprocessing, like to drop special edges that could be risky [43]. Techniques that change the structure of the traditional GNN to improve their robustness, such as the Gaussian-based graph convolutional layers [44], or the Adaptive Graph Convolutional Network [45]. Other techniques change the training process, for instance by implying adversarial training [46]–[51]. This is a technique already well known in other fields of machine learning, or by changing the loss function to a more robust one [52]–[54], [49]. Some tried applying a protection on top of the existing model to try to detect adversaries [55]–[57]. Finally, some changed the optimization and the aggregation process to prevent transferability attacks [58].

5 Conclusion

In this paper, we present Graph Neural Network and the interest to use it as GNN-based IDS. Instead of conventional IDS, this one use graphs to represent data like address and hosts. The interest is that GNN excel in discerning relationships between nodes, but it can also perform higher-order interaction and can be improved by adding additional content. Specially, this leads to have static graph for analyse data structures and dynamic graph for real-time detection.

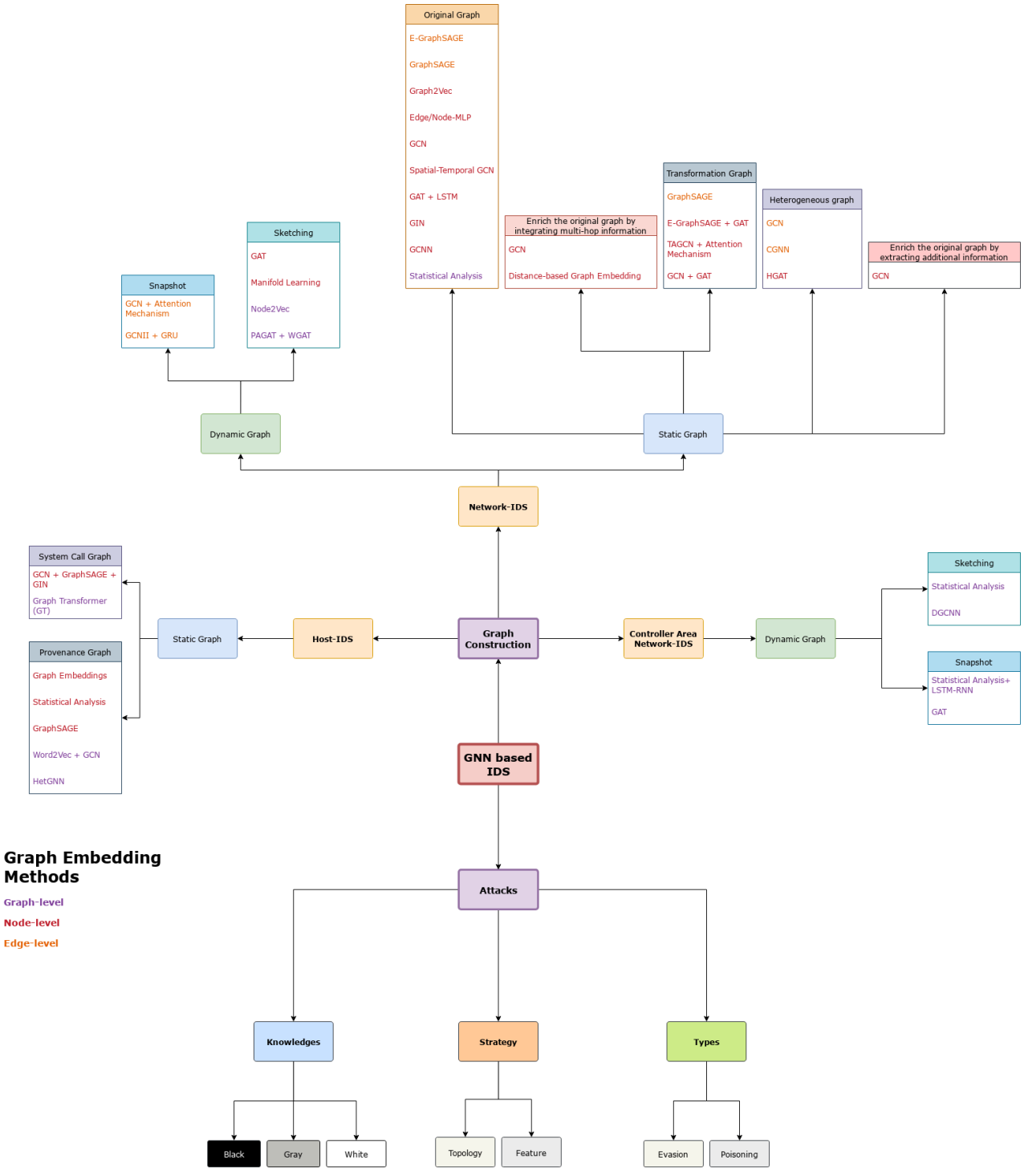
Moreover, graph-based detection is highly practical with the right framework, as it has been proven to be faster and more energy efficient than other ML based techniques [59].

Concerning adversarial attacks, there are not enough studies. However, it appears to be possible to generate adversarial perturbation that could make the attacker invisible, even if GNN seem more robust than other traditional ML models.

6 Appendix

6.1 Taxonomies summary

A summarized of GNN based IDS. It includes Graph Construction and Attacks against them. About construction, GNN can be used for Host Intrusion Detection, Network Intrusion Detection or even Controller Area Network Intrusion Detection.



6.2 Example of a linear GCN as surrogate model to find adversarial perturbations

Zügner et al. [36] proposed in 2018 an attacked based on a surrogate model. The principle is straight forward, linearize as much as possible the model's equation to be able to solve the minimisation problem given constraints.

Given a GCN model defined as before (4), the surrogate model is created by removing the non-linearities, i.e. the σ function:

$$Z' = \text{softmax}\Big(\hat{A}\hat{A}\mathbf{X}\mathbf{W}^{(1)}\mathbf{W}^{(2)}\Big) = \text{softmax}\Big(\hat{A}^2\mathbf{X}\mathbf{W}\Big) \tag{14}$$

The softmax can also be removed when computing the log-probabilities, thus leaving only one non-linearity: the square. Hence, it is still challenging to attack the model by adding or removing edges. To solve that problem, the authors found a way to fast compute the new adjacency matrix squared $\hat{A}^2$. With A' the new adjacency matrix obtained by adding or removing the edge $e = (m, n)$, $\hat{A'}^2$ can be computed with:

$$\Big[\hat{A'}^2\Big]_{uv} = \Big[\hat{A}^2\Big]_{uv} + \Delta_{u,v,m,n} \tag{15}$$

With $\Delta_{u,v,m,n}$ computed in $O(1)$. Nonetheless, a feature attack is easier. To find the best node and feature (u^*, i^*) one only needs to compute the gradient:

$$\Gamma_{ui} = \Big[\hat{A}^2\Big]_{v_0 u}\Big([\mathbf{W}]_{ic} - [\mathbf{W}]_{ic_{\text{old}}}\Big) \tag{16}$$

With v_0 the target node, c_{old} the old class and c the new class. This is the perfect example to demonstrate how to transform the initial problem into a gradient-based one. However, this approach is limited to simple architectures of GNN, due to transferability issues. It also requires a full knowledge of the features $\mathbf{X}$ which may be impossible in practice.

6.3 Example of a Linearized Loopy Belief Propagation model as surrogate model

The paper of Wang et al. [37] uses a similar technique to [36]: find a simple surrogate model to use gradient-based techniques. However, they went a bit further as they made the adjacency matrix become continuous, a technique that can be used in other scenario.

The targeted model is the Linearized Loopy Belief Propagation (LinLBP). Suppose we are given an undirected graph, a training dataset L which consists of a set of labeled positive nodes L_P and a set of labeled negative nodes L_N. LinLBP assigns the prior reputation score q_u for a node u as follows:

$$q_u = \begin{cases} \theta & \text{if } u \in L_P \\ -\theta & \text{if } u \in L_N \\ 0 & \text{otherwise} \end{cases} \tag{17}$$

where $0 < \theta \leq 1$ is a parameter of LinLBP. LinLBP assigns the same weight w in the interval (0, 0.5] for all edges. A larger weight means that two linked nodes are more likely to have the same label. We denote by W the weight matrix ($|V| \times |V|$), every entry of which is the weight w. Then, the posterior reputation scores in LinLBP are a solution of the following system:

$$p = q + A \odot Wp \tag{18}$$

where q and p are the column vector of prior reputation scores and posterior reputation scores of all nodes. To solve this equation, the scores are iteratively computed as follows:

$$p^{(t)} = q + A \odot Wp^{(t-1)} \tag{19}$$

where $p^{(t)}$ is the column vector of posterior reputation scores in the tth iteration. When the posterior reputation scores converge, a node u is predicted to be negative if its posterior reputation score is

negative, i.e., $p_u < 0$. We note LinLBP has two parameters, the prior reputation score parameter θ and the edge weight w.

The attack then consists of finding B such as:

$$\begin{aligned} &\min_{B} \sum_{u \in S, v \in V'} B_{uv} C_{uv} + n C_{\text{node}} \\ &\text{s.t. } \text{FNR} = 1, \text{for the fake nodes} \\ &\quad \text{FNR} = 1, \text{for the target nodes} \\ &\quad B_{uv} \in \{0,1\}, u \in S, v \in V' \\ &\quad \sum_{v} B_{uv} \leq K, u \in S \end{aligned} \tag{20}$$

With:

- B, a binary matrix that indicates a change by the attacker. If $B_{uv} = 1$, it means the edge (u, v) has been either removed if it existed before, or added if it did not.
- C_{uv}, the cost of insterting an edge between nodes u and v.
- C_{node}, the cost of creating each fake node.
- n, the number of fake nodes.
- K, the max number of edges each fake node can have.
- $\text{FNR} = 1$, achieve as high false negative rate as possible.

This attack is then viewed as an optimisation attac. Using Lagrangian multipliers it is possible to achieve this attack in seconds on large datasets where the Nettack attack took hours to compute. It is possible because B is not treated as binary but as a continuous matrix during the computation, and then converted back into a binary matrix. This allows the attacker to use common gradient based algorithms like a projected gradient descent.

This attacks works well on collective classifications methods, like LinLBP, with a success of over 0.92 FNR. However, it can be transferred to Graph Neural Network based classification methods like CNN where it achieves 0.54 FNR.

6.4 Example of a spectrum-based attack on unsupervised learning

The paper from Bojchevski et al. [33] shows how to find adversarial edges to modify the spectrum of the graph. The spectrum of the adjacency matrix is often used in graph analysis as it reflects many properties of the graph. It is also used in intrusion detection [60].

The idea of this paper is to exploit gradient based methods on the spectrum of the adjacency matrix leveraging eigenvalue perturbation theory. To compute the attack efficiently, it is necessary to have an efficient way to computing the new eigen values for each edge flip. Let u_y be the yth generalized eigenvector of A with the generalized eigenvalue λ_y. Then the generalized eigenvalue λ'_y of A' is approximatively $\lambda'_y = \lambda_y + \Delta\lambda_y$, with $\Delta\lambda_y$ defined as:

$$\Delta\lambda_y = \Delta w_{ij}\big(2u_{yi}u_{yj} - \lambda_y\big(u_{yi}^2 + u_{yj}^2\big)\big) \tag{21}$$

This equation allows changes in $O(1)$, thus enabling the attack.

However, this approach is mainly used to poison deep walk methods. The results show that a big number of edges have to be flipped in order to have an influence on the predictions. The results are shown in Figure 13.

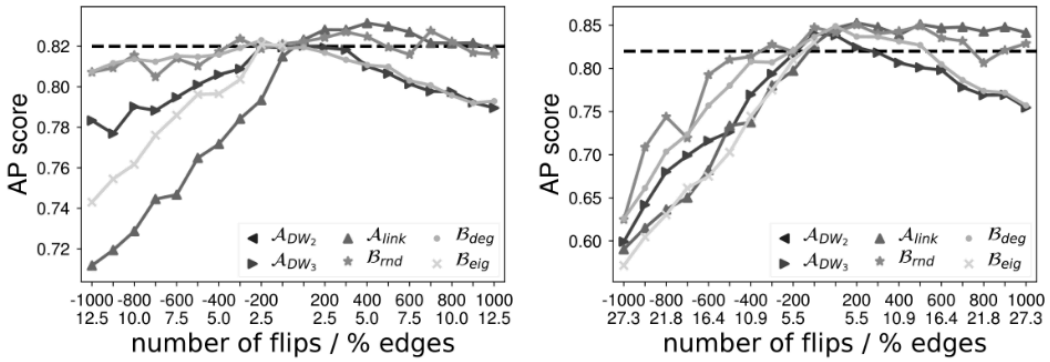


Figure 13: Targeted attack on the link prediction task for Cora and Citeseer datasets.

Nonetheless, an interesting discovery has been made by the authors: the adversarial edges found with this method do not appear to have special traits like high centrality. Thus, they concluded classical methods can not be used to find those edges.

6.5 Example of more advanced graph structures for intrusion detection: Hyper Graph Neural Networks

The main idea behind the use of hypergraphs is that the original graph structure primarily captures pairwise node correlations, with limited capability to model complex, higher-order interactions among multiple nodes. As mentioned in [61], hypergraph structures can effectively model and express complex relationships among multiple nodes, this capability is in theory, beneficial in extracting intricate traffic patterns and relational networks. Another problem is that, usually, the spatial and temporal informations are separated, thus the distinct attention mechanisms can suppress critical information, needed by the other part.

The hypergraph is constructed as follows: each flow is a vertex; a hyperedge is created between two similar nodes (based on an euclidean distance in an unknown space), then convolutions are used to encode high-order relationships between flows. This procedure is shown in Figure 14 and Figure 15.

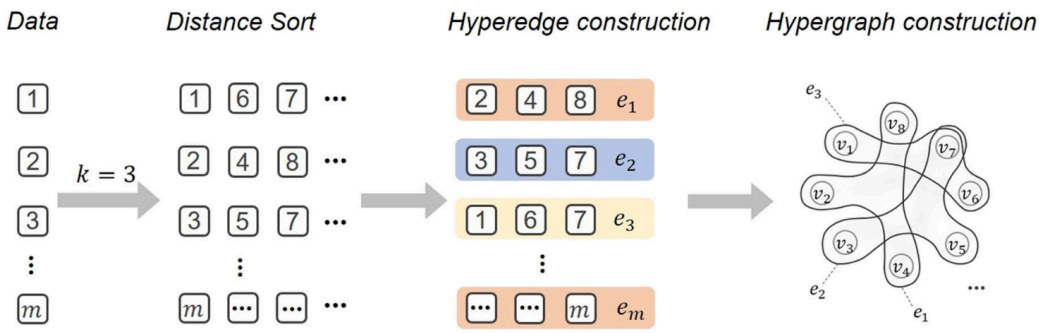


Figure 14: The schematic of hypergraph construction.

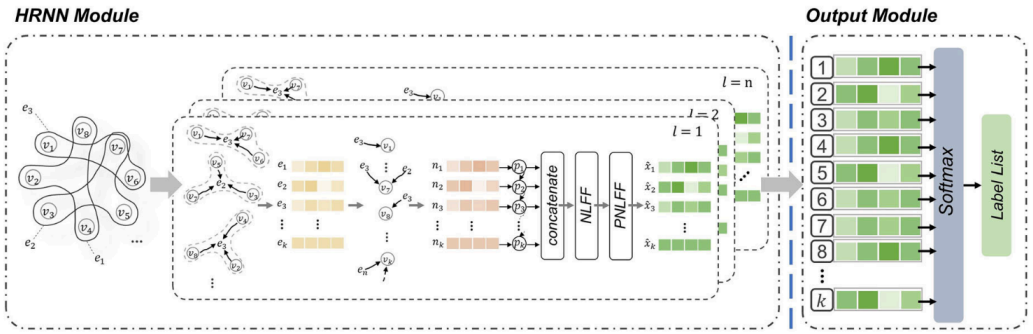


Figure 15: A HRNN Module.

The authors conclude HRNN generates both robust and semantically expressive embeddings, significantly improving the detection capability of anomaly traffic. This leads to state-of-the-art results on famous datasets like NSL-KDD, ISCXVPN2016 or even CIC-IDS2017.

Even if this approach seems interesting, no code is provided and the paper lacks crucial information on why the construction of hypergraph is able to grasp the semantic of attacks.

6.6 Example of different meta-paths

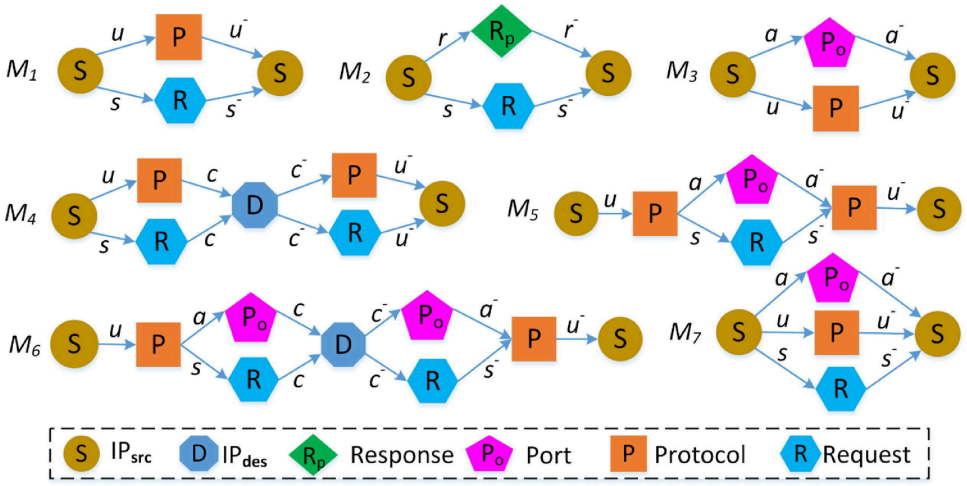


Figure 16: Meta-paths in [11].

6.7 Example of HyperVision, a real time unsupervised ML

Since 2019, 80 percents of websites has used HTTPS protocol. HyperVision is a new method to achieve unsupervised detection – more realistic case – and it is able to detect encrypted malicious traffic without accessing to the payload. The interesting point is it needs to build a graph for realtime detection.

Usually, flows on Internet are short meanwhile most packets are long. The tool HyperVision records interaction patterns to reduce density of the graph. This one is build with addresses as vertices and flows as edges. To aggregate short flows, the flows must:

- have the same source and/or destination;
- have same protocol;
- be repetitive enough.

For connectivity analysis, connected components are found by using depth-first search. This leads to extract abnormal components.

To detect malicious traffic, critical vertices are identified with solving the vertex cover problem: in a component, each flows with a source and/or destination of a malicious flow are grouped in a subset. Those edges connected to a critical vertex are clustered, and K-Means algorithm is used to calculate the loss function that indicates the degree of maliciousness.

There are three equations: $\text{loss}_{\text{center}}$ that indicates the difference from other edges connected to the critical vertex; $\text{loss}_{\text{count}}$ that is the number of flows denoted by the edge; and finally loss function that when it became greater than a threshold indicates maliciousness of the edge.

$$\begin{aligned} \text{loss}_{\text{center}}(\text{edge}) &= \min_{C_i \in \{C_1, \ldots, C_k\}} \left\| C_i - f(\text{edge}) \right\|_2 \\ \text{loss}_{\text{count}}(\text{edge}) &= \log_2(\text{Size}(C(\text{edge})) + 1) \\ \text{loss}_{\text{cluster}}(\text{edge}) &= \text{TimeRange}\ (C(\text{edge})) \\ \text{loss}\ (\text{edge}) &= \alpha \text{loss}_{\text{center(edge)}} - \beta \text{loss}_{\text{cluster}}(\text{edge}) + \gamma \text{loss}_{\text{count}}(\text{edge}) \end{aligned} \tag{22}$$

Where K is the number of cluster centers, C_i is the i^{th} center and $f(\text{edge})$ is the feature vector.

References

[1] W. L. Hamilton, "Graph Representation Learning," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 14, no. 3, pp. 1–159.

[2] V. Gauthier, "NET 4103: Complex Networks." Réseaux et Services de Télécommunications de Télécom SudParis, 2024.

[3] A. Grover and J. Leskovec, "node2vec: Scalable Feature Learning for Networks," *CoRR*, 2016.

[4] Z. Jin, Y. Wang, Q. Wang, Y. Ming, T. Ma, and H. Qu, "GNNVis: A Visual Analytics Approach for Prediction Error Diagnosis of Graph Neural Networks," p. , 2020.

[5] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How Powerful are Graph Neural Networks?," 2019.

[6] T. Bilot, N. E. Madhoun, K. A. Agha, and A. Zouaoui, "Graph Neural Networks for Intrusion Detection: A Survey," *IEEE Access*, vol. 11, pp. 49114–49139, 2023, doi: 10.1109/ACCESS.2023.3275789.

[7] B. Bowman, C. Laprade, Y. Ji, and H. H. Huang, "Detecting Lateral Movement in Enterprise Computer Networks with Unsupervised Graph AI," pp. 257–268, Oct. 2020, [Online]. Available: https://www.usenix.org/conference/raid2020/presentation/bowman

[8] R. Paudel and H. H. Huang, "Pikachu: Temporal Walk Based Dynamic Graph Embedding for Network Anomaly Detection," vol. 0, no. , pp. 1–7, 2022, doi: 10.1109/NOMS54207.2022.9789921.

[9] J. Zhou, Z. Xu, A. M. Rush, and M. Yu, "Automating Botnet Detection with Graph Neural Networks," 2020.

[10] B. Zhang, J. Li, C. Chen, K. Lee, and I. Lee, "A Practical Botnet Traffic Detection System Using GNN," pp. 66–78, 2022.

[11] J. Zhao, X. Liu, Q. Yan, B. Li, M. Shao, and H. Peng, "Multi-Attributed Heterogeneous Graph Convolutional Network for Bot Detection," *Information Sciences*, vol. 537, p. , 2020, doi: 10.1016/j.ins.2020.03.113.

[12] 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 Transactions on Dependable and Secure Computing*, vol. 19, no. 6, pp. 3855–3872, 2022, doi: 10.1109/TDSC.2021.3108782.

[13] W. W. Lo, G. K. Kulatilleke, M. Sarhan, S. Layeghy, and M. Portmann, "XG-BoT: An Explainable Deep Graph Neural Network for Botnet Detection and Forensics," 2023.

[14] Y. Li *et al.*, "GraphDDoS: Effective DDoS Attack Detection Using Graph Neural Networks," vol. 0, no. , pp. 1275–1280, 2022, doi: 10.1109/CSCWD54268.2022.9776097.

[15] W. W. Lo, S. Layeghy, M. Sarhan, M. Gallagher, and M. Portmann, "E-GraphSAGE: A Graph Neural Network based Intrusion Detection System for IoT," pp. 1–9, Apr. 2022, doi: 10.1109/NOMS54207.2022.9789878.

[16] L. Chang and P. Branco, "Graph-based Solutions with Residuals for Intrusion Detection: the Modified E-GraphSAGE and E-ResGAT Algorithms," 2021.

[17] J. Lan *et al.*, "E-minBatch GraphSAGE: An Industrial Internet Attack Detection Model," *Security and Communication Networks*, vol. 2022, p. , 2022, doi: 10.1155/2022/5363764.

[18] E. Caville, W. W. Lo, S. Layeghy, and M. Portmann, "Anomal-E: A self-supervised network intrusion detection system based on graph neural networks," *Knowledge-Based Systems*, vol. 258, p. 110030–110031, Dec. 2022, doi: 10.1016/j.knosys.2022.110030.

[19] I. J. King and H. H. Huang, "Euler: Detecting Network Lateral Movement via Scalable Temporal Link Prediction," *ACM Trans. Priv. Secur.*, vol. 26, no. 3, Jun. 2023, doi: 10.1145/3588771.

[20] F. Liu, Y. Wen, Y. Wu, S. Liang, X. Jiang, and D. Meng, "MLTracer: Malicious Logins Detection System via Graph Neural Network," vol. 0, no. , pp. 715–726, 2020, doi: 10.1109/TrustCom50675.2020.00099.

[21] Y. Fang, C. Wang, F. Zhiyang, and C. Huang, "LMTracker: Lateral Movement Path Detection based on Heterogeneous Graph Embedding," *Neurocomputing*, vol. 474, p. , 2021, doi: 10.1016/j.neucom.2021.12.026.

[22] X. Sun and J. Yang, "HetGLM: Lateral Movement Detection by Discovering Anomalous Links with Heterogeneous Graph Neural Network," vol. 0, no. . pp. 404–411, 2022. doi: 10.1109/IPCCC55026.2022.9894347.

[23] "HTTPS encryption on the web – Google Transparency Report.." [Online]. Available: https://transparencyreport.google.com/https/overview?hl=en,

[24] M. Lv, C. Dong, T. Chen, T. Zhu, Q. Song, and Y. Fan, "A Heterogeneous Graph Learning Model for Cyber-Attack Detection," no. arXiv:2112.08986. arXiv, Dec. 16, 2021.

[25] D. Pujol-Perich, J. Suárez-Varela, A. Cabellos-Aparicio, and P. Barlet-Ros, "Unveiling the potential of Graph Neural Networks for robust Intrusion Detection," no. arXiv:2107.14756. arXiv, Jul. 30, 2021.

[26] P. Veličković, W. Fedus, W. L. Hamilton, P. Liò, Y. Bengio, and R. D. Hjelm, "Deep Graph Infomax." 2018.

[27] C. Fu, Q. Li, and K. Xu, "Detecting Unknown Encrypted Malicious Traffic in Real Time via Flow Interaction Graph Analysis." Accessed: May 26, 2024. [Online]. Available: http://arxiv.org/abs/2301.13686

[28] L. Chen *et al.*, "A Survey of Adversarial Learning on Graphs," no. arXiv:2003.05730. arXiv, Apr. 05, 2022.

[29] J. Chen, Z. Shi, Y. Wu, X. Xu, and H. Zheng, "Link Prediction Adversarial Attack," 2018.

[30] M. Sun *et al.*, "Data Poisoning Attack against Unsupervised Node Embedding Methods," p. , 2018.

[31] Y. Ma, S. Wang, T. Derr, L. Wu, and J. Tang, "Attacking Graph Convolutional Networks via Rewiring," 2019.

[32] L. Sun *et al.*, "Adversarial Attack and Defense on Graph Data: A Survey," *IEEE Transactions on Knowledge and Data Engineering*, pp. 1–20, 2022, doi: 10.1109/tkde.2022.3201243.

[33] A. Bojchevski and S. Günnemann, "Adversarial Attacks on Node Embeddings via Graph Poisoning," no. arXiv:1809.01093. arXiv, May 27, 2019.

[34] H. Dai *et al.*, "Adversarial Attack on Graph Structured Data," 2018.

[35] J. Chen, Y. Wu, X. Xu, Y. Chen, H. Zheng, and Q. Xuan, "Fast Gradient Attack on Network Embedding," 2018.

[36] D. Zügner, A. Akbarnejad, and S. Günnemann, "Adversarial Attacks on Neural Networks for Graph Data," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, Jul. 2018, pp. 2847–2856. doi: 10.1145/3219819.3220078.

[37] B. Wang and N. Z. Gong, "Attacking Graph-based Classification via Manipulating the Graph Structure," no. arXiv:1903.00553. arXiv, Aug. 12, 2019.

[38] D. Zügner and S. Günnemann, "Adversarial Attacks on Graph Neural Networks via Meta Learning," 2024.

[39] X. Wang, M. Cheng, J. Eaton, C.-J. Hsieh, and F. Wu, "Attack Graph Convolutional Networks by Adding Fake Nodes," 2020.

[40] S. Yu *et al.*, "Unsupervised Euclidean Distance Attack on Network Embedding," 2019.

[41] A. J. Bose, A. Cianflone, and W. L. Hamilton, "Generalizable Adversarial Attacks with Latent Variable Perturbation Modelling," 2020.

[42] M. Waniek, K. Zhou, Y. Vorobeychik, E. Moro, T. P. Michalak, and T. Rahwan, "Attack Tolerance of Link Prediction Algorithms: How to Hide Your Relations in a Social Network," 2018.

[43] H. Wu, C. Wang, Y. Tyshetskiy, A. Docherty, K. Lu, and L. Zhu, "Adversarial Examples on Graph Data: Deep Insights into Attack and Defense," 2019.

[44] D. Zhu, Z. Zhang, P. Cui, and W. Zhu, "Robust Graph Convolutional Networks Against Adversarial Attacks," pp. 1399–1407, 2019, doi: 10.1145/3292500.3330851.

[45] V. N. Ioannidis and G. B. Giannakis, "Edge Dithering for Robust Adaptive Graph Convolutional Networks," 2019.

[46] K. Xu *et al.*, "Topology Attack and Defense for Graph Neural Networks: An Optimization Perspective," 2019.

[47] K. Sun, Z. Lin, H. Guo, and Z. Zhu, "Virtual Adversarial Training on Graph Convolutional Networks in Node Classification," 2020.

[48] F. Feng, X. He, J. Tang, and T.-S. Chua, "Graph Adversarial Training: Dynamically Regularizing Based on Graph Structure," 2019.

[49] J. Chen, Y. Wu, X. Lin, and Q. Xuan, "Can Adversarial Network Attack be Defended?," 2019.

[50] Z. Deng, Y. Dong, and J. Zhu, "Batch Virtual Adversarial Training for Graph Convolutional Networks," 2019.

[51] X. He, Z. He, X. Du, and T.-S. Chua, "Adversarial Personalized Ranking for Recommendation," in *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, in SIGIR '18. ACM, Jun. 2018. doi: 10.1145/3209978.3209981.

[52] D. Zügner and S. Günnemann, "Certifiable Robustness and Robust Training for Graph Convolutional Networks," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, in KDD '19. ACM, Jul. 2019. doi: 10.1145/3292500.3330905.

[53] A. Bojchevski and S. Günnemann, "Certifiable Robustness to Graph Perturbations," 2019.

[54] M. Jin, H. Chang, W. Zhu, and S. Sojoudi, "Power up! Robust Graph Convolutional Network via Graph Powering," 2021.

[55] Y. Zhang, S. Khan, and M. Coates, "Y. Zhang, S. Khan, and M. Coates, "Comparing and detecting adversarial attacks for graph deep learning,," 2019.

[56] V. N. Ioannidis, D. Berberidis, and G. B. Giannakis, "GraphSAC: Detecting anomalies in large-scale graphs," 2019.

[57] S. Hou *et al.*, "αCyber: Enhancing Robustness of Android Malware Detection System against Adversarial Attacks on Heterogeneous Graph based Model," in *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, in CIKM '19. Beijing, China: Association for Computing Machinery, 2019, pp. 609–618. doi: 10.1145/3357384.3357875.

[58] X. Tang, Y. Li, Y. Sun, H. Yao, P. Mitra, and S. Wang, "Transferring Robustness for Graph Neural Network Against Poisoning Attacks," in *Proceedings of the 13th International Conference on Web Search and Data Mining*, in WSDM '20. ACM, Jan. 2020. doi: 10.1145/3336191.3371851.

[59] A. Protogerou, S. Papadopoulos, A. Drosou, D. Tzovaras, and I. Refanidis, "A graph neural network method for distributed anomaly detection in IoT," *Evolving Systems*, vol. 12, p. , 2021, doi: 10.1007/s12530-020-09347-0.

[60] J. Majed, N. Boutry, and P. Parrend, "Spectral Analysis for Attack Detection ." RESSI, 2024.

[61] Z. Yang, Z. Ma, W. Zhao, L. Li, and F. Gu, "HRNN: Hypergraph Recurrent Neural Network for Network Intrusion Detection," *Journal of Grid Computing*, vol. 22, no. 2, p. 52–53, Jun. 2024, doi: 10.1007/s10723-024-09767-1.