Node Injection Link Stealing Attack

Oualid Zari[1], Javier Parra-Arnau[2,3], Ayşe Ünsal[1], and Melek Önen[1]

[1] EURECOM, Biot, France
`{oualid.zari, ayse.unsal, melek.onen}@eurecom.fr`
[2] Karlsruhe Institute of Technology, Karlsruhe, Germany
[3] Universitat Politècnica de Catalunya, Barcelona, Spain
`javier.parra@upc.edu`

Abstract. We present a stealthy privacy attack that exposes links in Graph Neural Networks (GNNs). Focusing on dynamic GNNs, we propose to inject new nodes and attach them to a particular target node to infer its private edge information. Our approach significantly enhances the F_1 score of the attack compared to the current state-of-the-art benchmarks. Specifically, for the Twitch dataset, our method improves the F_1 score by 23.75%, and for the Flickr dataset, remarkably, it is more than three times better than the state-of-the-art. We also propose and evaluate defense strategies based on differentially private (DP) mechanisms relying on a newly defined DP notion. These solutions, on average, reduce the effectiveness of the attack by 71.9% while only incurring a minimal utility loss of about 3.2%.

Keywords: graph neural networks · link stealing · differential privacy

1 Introduction

Graph-structured data is prevalent in applications like social networks, biological systems, and recommendation engines. Graph Neural Networks (GNNs) are powerful for analyzing such data but they pose significant privacy risks as the graph structure is often sensitive. For instance, social network links can reveal common interests or personal beliefs, leading to privacy breaches [3].

This paper advances edge privacy in GNNs by introducing the Node Injection Link Stealing (NILS) attack and a tailored Differential Privacy (DP) defense. The adversary infers links among target nodes in a GNN trained for node classification. The GNN processes graph structure and node features to produce class membership predictions which are accessed via an inference API.

Previous works like the Linkteller attack [23] and studies on feature correlation [9] have shown that attackers can infer graph links by analyzing node features and GNN outputs. Our NILS attack introduces a stronger adversary exploiting GNN dynamics. The adversary adds a node, connects it to the target node, and queries the model with malicious features, inferring neighbors and stealing graph connections using various strategies.

The NILS attack is akin to activities in social networks, where sending a friend request is analogous to injecting a new node into the network. By connecting this new node to a target node, the attacker can observe changes in the network's behavior, similar to how content recommendations or interactions change when a new connection is established. This method allows the attacker to infer hidden links within the network, posing a significant threat to user privacy.

We also study potential defense strategies based on DP mechanisms. Specifically, we propose a new privacy notion, *one-node-one-edge privacy*, and evaluate existing DP-based defense strategies under this definition.

To summarize, we make the following contributions:

– We propose a novel attack (denoted NILS) for inferring private links in a graph structure by injecting a new node, linking it to a target node, and employing various strategies to analyze the changes in the GNN's output;
– We provide a comprehensive evaluation of the proposed attack's effectiveness on various datasets, demonstrating its superior performance compared to existing work such as LinkTeller [23] and link-stealing [9];
– We explore the application of DP mechanisms as a means to mitigate the effectiveness of our proposed attack, evaluating the trade-off between privacy preservation and model utility. To this end, we introduce a new notion of privacy and evaluate defense strategies under this new notion.

2 Background

We present a brief introduction to GNNs and formulate the concept of DP.

2.1 Graph Neural Networks

GNNs Overview. GNNs [18] are powerful machine learning models designed for graph-structured data. They effectively capture complex patterns in graphs, excelling in tasks like node classification [21], link prediction [26], and graph classification [24]. Node classification, the focus of this paper, involves assigning labels to nodes based on their features and graph structure.

A graph $G = (V, E)$ consists of nodes V and edges E. Nodes represent data points like users in social networks or proteins in biological networks, while edges represent relationships or interactions. Graphs are represented using an adjacency matrix $A \in \mathbb{R}^{n \times n}$, where n is the number of nodes, and $A_{ij} = 1$ if there is an edge between nodes i and j, and $A_{ij} = 0$ otherwise. Nodes have feature vectors represented by the matrix $X \in \mathbb{R}^{n \times d}$, containing information like demographic data in social networks.

GNNs use a message-passing mechanism [18] for nodes to exchange and aggregate information from their neighbors, capturing local and global graph structure. For instance, Graph Convolutional Networks (GCNs) [11], a well-known GNN model, use graph convolutional layers formulated as:

$$H^{(0)} = X, \quad H^{(l+1)} = \sigma(\hat{A} H^{(l)} W^{(l)}), \quad H^{(L)} = P.$$

Here, $H^{(0)}$ is the node feature matrix X; $H^{(l)}$ is the hidden node representation at layer l; P represents prediction scores for each node class; $W^{(l)}$ is the learnable weight matrix; $\sigma(\cdot)$ is an activation function (e.g., ReLU); and $\hat{A}$ is the normalized adjacency matrix.

GNNs with Dynamic Graphs. GNNs handle dynamic graphs as in social networks or recommendation systems where graphs evolve over time. New nodes and edges are introduced, requiring updates to the adjacency matrix $A \in \mathbb{R}^{n \times n}$ and the feature matrix $X \in \mathbb{R}^{n \times d}$. The matrices expand to $A' \in \mathbb{R}^{(n+1)\times(n+1)}$ and $X' \in \mathbb{R}^{(n+1)\times d}$, incorporating new nodes' connections and features. Once updated, the GNN performs inference on the modified graph using the message-passing mechanism described earlier.

2.2 Differential Privacy

Differential Privacy (DP) is a framework for ensuring the privacy of individual records in a database. In the context of GNNs, it helps protect the privacy of graph structures. For detailed definitions and mechanisms, please refer to the Appendix.

3 Related work

GNNs have gained significant attention for their effectiveness in handling graph-based data across various applications [1, 18, 12]. As GNN adoption increases, concerns about privacy and adversarial attacks also rise [20]. Several privacy-preserving methods have been developed to mitigate these attacks [17, 15].

Privacy Attacks on GNNs. Privacy attacks on GNNs target graph nodes, attributes, or edges. Node privacy attacks, like membership inference attacks (MIA) [22], determine if a node was part of the training set. Attribute inference attacks [4] reveal sensitive node attributes. This work focuses on edge privacy violations, where attacks like link stealing, re-identification, or inference aim to uncover graph edges.

Early works [9, 4, 23] demonstrated the feasibility of link-stealing attacks. In [9], the adversary uses prior knowledge about the graph to infer links, applying clustering methods to predict connections among nodes. In [4], node embeddings trained to preserve the graph structure are used to recover edges by training a decoder. The Linkteller attack [23] involves probing node features and studying their GNN output predictions to infer links.

Existing link-stealing attacks have weaknesses. The attack in [9] requires a powerful adversary with access to features, a shadow dataset, and the ability to train shadow GNNs. Its performance declines in the inductive setting, where training and inference occur on different graphs. The Linkteller attack [23] has non-stealthy perturbations, especially with discrete data, making it easier to detect. Its effectiveness decreases against deeper GNNs.

This paper proposes a novel link-stealing attack, NILS, which addresses these limitations by exploiting the dynamic nature of GNNs through malicious node injection. NILS outperforms previous attacks [9, 23]. Concurrent with our research, [14] proposes a link inference attack using node injection. The attack injects multiple target nodes and nodes with zero features, training an attack model to infer links. While effective on high-homophily graphs [23], this method assumes the adversary has access to a partial graph and does not address low-homophily graphs.

Differential Privacy Mechanisms for Graphs. DP has been studied and applied to graphs to preserve sensitive information. Various DP mechanisms protect node and edge information [2]. Node-level DP [8] protects individual nodes from attacks like MIA [22]. Edge-level DP [13] protects edge information, preventing link stealing attacks [4, 23, 9].

Research has focused on achieving node-level and edge-level DP in graph models. Approaches allow the publication of graph statistics with edge-level DP guarantees, such as degree subgraph count and degree distributions [10]. While beneficial for graph analysis, these statistics are inadequate for GNN training, which requires access to the raw graph structure. Other approaches use input perturbation DP to release graphs while ensuring edge-level DP [23].

In designing DP solutions, specific privacy threats and adversary strengths must be considered. For the NILS attack, the adversary injects a node to a specific node to discover sensitive edge information, violating edge privacy. We propose a customized DP notion addressing this attack, leveraging the LapGraph algorithm [23] to achieve desired DP guarantees.

4 Node injection link stealing attack

GNNs are vulnerable to various privacy attacks aiming to learn about their underlying graph structure. They inherit attacks from standard neural networks, such as membership inference attacks (MIA) [22], where the adversary tries to determine if a sample is included in the training dataset. This paper focuses on the *link stealing attack*, where an adversary, without access to the adjacency matrix, aims to learn whether a particular edge exists. We introduce our node injection link stealing (NILS) attack that exploits the dynamic nature of GNNs.

4.1 Threat model

Environment We consider a GNN application where a server trains the GNN using a specific dataset and provides access through a black-box API. This API allows users to interact with the pre-trained GNN model without accessing its internal components. Users can submit prediction queries using node IDs and add new nodes using a *connect* query. The API processes input data and returns output predictions, ensuring the model's computations remain hidden. Users only know the set of node IDs.

Adversary's goal and knowledge The adversary, $\mathcal{A}$, acts as a GNN user aiming to determine the neighbors of a specific *target node*, v_t, from a set of *target nodes*, $V_{\mathcal{A}}$. If $\mathcal{A}$ aims to identify all links, $V_{\mathcal{A}}$ would include all nodes in the graph V. $\mathcal{A}$ may perform multiple node injections, but this approach's practicality is debatable. In social networks, the adversary's background knowledge, like users' interests, guides the selection of target nodes more likely to be connected. We choose target nodes uniformly at random. $\mathcal{A}$ obtains prediction scores of $V_{\mathcal{A}}$ by querying their IDs through the API and can use the *connect* query to connect a node v_m to v_t.

4.2 Node injection link stealing attack

The NILS attack exploits the dynamic nature of GNNs. The adversary $\mathcal{A}$ can *connect* new nodes and query prediction scores of nodes $V_{\mathcal{A}}$ in the graph. By adding a new node v_m, $\mathcal{A}$ can discover neighbors of v_t. The attack is depicted in Figure 1, outlined in Algorithm 1 and involves the following steps:

1. $\mathcal{A}$ queries the prediction scores of target nodes $V_{\mathcal{A}}$ and receives prediction matrix P.
2. $\mathcal{A}$ generates malicious features for node v_m based on P.
3. $\mathcal{A}$ sends a *connect* query to inject node v_m, specifying the features x_m and the ID of target node v_t.
4. The server adds node v_m to the graph and links it to v_t.
5. $\mathcal{A}$ queries the server again for the new prediction matrix P' of $V_{\mathcal{A}}$.
6. $\mathcal{A}$ computes the L_1 distance between $P(v)$ and $P'(v)$ for each node v in $V_{\mathcal{A}}$. A significant change in prediction scores indicates a high probability of being a neighbor of v_t. If the difference exceeds a threshold R, $\mathcal{A}$ infers that node v is a neighbor of v_t.

Selection of the decision threshold R. The threshold R is determined through parameter tuning, aiming for an optimal trade-off between precision and recall in identifying true neighbors of the target node, represented by the F_1 score. Various values of R are evaluated, and the one yielding the highest F_1 score is selected as optimal. In practice, the adversary can select R by estimating the graph's density $\hat{d}$ and picking the top $\hat{d}$ nodes with the highest changes before and after injection.

4.3 Strategies for generation of malicious node's features

To evaluate how injecting the malicious node v_m affects GNN predictions, we study five strategies to generate the malicious node's features x_m. These strategies vary in sparsity and stealthiness, allowing us to assess their effectiveness in altering the model's predictions. The proposed strategies are:

1. **All-ones strategy**: Generates a dense feature vector of all ones:

$$x_m = \mathbf{1}.$$

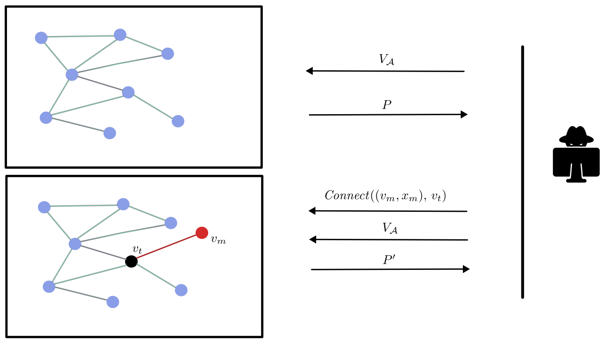

Fig. 1: Adversary-Server Interaction: At inference, $\mathcal{A}$ first queries prediction scores P of target nodes $V_{\mathcal{A}}$. Next, the server sends predictions P to $\mathcal{A}$. Then, $\mathcal{A}$ sends *Connect* query to inject malicious node v_m, with features x_m, to target node v_t. Finally, $\mathcal{A}$ queries prediction scores P' of target nodes $V_{\mathcal{A}}$.

This can cause significant prediction changes but is less stealthy due to its density.

2. **All-zeros strategy**: Creates a sparse feature vector of all zeros:
$$x_m = \mathbf{0}.$$
This subtly alters GNN output, causing smaller prediction changes while being more stealthy.
3. **Identity strategy**: Uses a feature vector identical to the target node's:
$$x_m = x_t.$$
This confuses model predictions for neighboring nodes, with variable stealthiness depending on similarity to the target node's features.
4. **Max attributes strategy**: Computes the element-wise maximum of each attribute in the target nodes' feature matrix, excluding the target node's class:
$$x_{m,k} = \max_{i \in V_{\mathcal{A}},\ \text{with } C_i \neq C_t} X_{i,k}, \quad \text{for} \quad k = 1, \ldots, d.$$
Here, C_i and C_t represent the classes of node i and the target node, respectively. This strategy causes significant prediction changes but may be less stealthy due to exaggerated features.
5. **Class representative strategy**: Selects the feature vector of the node with the highest confidence score for a different class than the target node's:
$$x_m = x_{i^*} \text{ with } i^* = \arg\max_{\substack{i \in V_{\mathcal{A}}, \\ C_i \neq C_t}} p_{i,j}.$$
In this equation, i^* is the index of the node with the highest confidence score for a class different from the target node's. This strategy leverages model predictions to alter neighboring node predictions, potentially increasing stealthiness.

Algorithm 1: Node Injection Link Stealing Attack

```
Input: set of nodes $V_\mathcal{A}$ and target node $v_t$.
Output: the identified neighbors of $v_t$ by the adversary.
$P = \text{GNN}(V_\mathcal{A}, X_{V_\mathcal{A}})$                              ▷ Step 1
Generate malicious features $x_m$ of node $v_m$                              ▷ Step 2
Connect node $v_m$ to $v_t$.                                               ▷ Step 3-4
$P' = \text{GNN}(V_\mathcal{A} \cup v_m , X_{V_\mathcal{A}} \cup x_m)$       ▷ Step 5
for each node $v$ in $V_\mathcal{A}$ do
    $D(v) = \|P(v) - P'(v)\|_1$                                              ▷ Step 6
    if $D(v) \geq R$ then
        $v$ is a neighbor of $v_t$
    end
    else
        $v$ is not a neighbor of $v_t$
    end
end
```

Additionally, we introduce the LinkTeller **Influence** strategy, which incorporates the feature perturbation strategy from [23]. This involves perturbing the target node's features by adding a small real value α:

$$x_m = x_t + \alpha.$$

We compare the Influence strategy's performance with other strategies to determine if attack performance gains are due to node injection or malicious feature crafting. However, this strategy may be easily detected if x_t has discrete features, as x_m becomes real-valued.

5 Evaluation of the attack

We present the evaluation results of our proposed attack, starting with a brief summary of the experimental setup and then analyzing its performance on various datasets. Detailed experimental setup information is provided in the appendix. We also address one limitation of the attack related to the depth of the GNN, discussed in the appendix.

5.1 Summary of experimental setup

We evaluated our attack on several real-world datasets, including the Flickr dataset [25], and two Twitch datasets (TWITCH-FR and TWITCH-RU) [16], following the approach in [23]. For the transductive setting, we used three citation network datasets: Cora, Citeseer, and Pubmed [19]. The models were trained using Graph Convolutional Networks (GCNs) with hyperparameters selected through a grid search strategy. Evaluation metrics include precision, recall, and the F_1 score. Detailed information on datasets, models, and evaluation metrics is provided in the appendix.

5.2 Analysis of strategies for malicious node's features

We analyze the impact of different strategies for generating the features x_m of the malicious node v_m on the success of our attack.

The success rates, shown in Table 1, indicate that the All-ones, Max attributes, and Class representative strategies are the most effective in causing significant changes in the predictions of the target node's neighbors. Injecting nodes with high-valued or class-specific features effectively disrupts the model's output predictions.

Conversely, the All-zeros and Identity strategies exhibit lower success rates. While these strategies offer stealthiness, their impact on graph structure and predictions is less pronounced, highlighting a trade-off between attack effectiveness and stealthiness.

For the Influence strategy, NILS shows modest improvement over the LinkTeller baseline for the Twitch-FR dataset (Table 1), suggesting the effectiveness of node injection. However, for the Twitch-RU dataset, NILS underperforms compared to LinkTeller. The most significant improvement is seen in the Flickr dataset, where NILS increases the F_1 score of LinkTeller from 0.32 ± 0.13 to 0.89 ± 0.10, showcasing the advantage of node injection.

The Max attributes approach significantly enhances the F_1 score beyond the LinkTeller baseline [23]. For the Twitch datasets, it improves the F_1 score by 23.75% on average. For the Flickr dataset, it records a remarkable increase, raising the F_1 score from 0.32 to 1.0, a 212.5% improvement over LinkTeller [23].

These findings underscore the importance of considering both the effectiveness and stealthiness of malicious feature generation strategies in link inference attacks on GNNs.

Table 1: F_1 scores for different attack methods and datasets.

Method	Twitch-FR	Twitch-RU	Flickr
Class Rep.	0.94 ± 0.01	0.83 ± 0.06	0.96 ± 0.06
Max Attr.	0.99 ± 0.00	0.98 ± 0.02	$\mathbf{1.00 \pm 0.00}$
All-ones	$\mathbf{0.99 \pm 0.00}$	$\mathbf{0.97 \pm 0.01}$	0.99 ± 0.02
All-zeros	0.58 ± 0.02	0.48 ± 0.01	0.71 ± 0.07
Identity	0.81 ± 0.02	0.69 ± 0.01	0.95 ± 0.07
Influence NILS	0.81 ± 0.02	0.70 ± 0.01	0.89 ± 0.10
Influence LinkTeller [23]	0.80 ± 0.02	0.74 ± 0.01	0.32 ± 0.13

5.3 Comparison with the baselines

We evaluate the performance of the NILS attack compared to the LinkTeller attack using an identical experimental setup. Our focus is on analyzing the optimal attacks for both approaches, accurately estimating the number of neighbors of the target nodes. The results in Table 2 show that NILS outperforms LinkTeller on both Twitch datasets (TWITCH-FR and TWITCH-RU). Additionally, NILS exhibits a substantial improvement over LinkTeller on the Flickr dataset, achieving nearly double the precision and recall values. NILS demonstrates stable performance across varying node degrees, with only a marginal decrease for high-degree target nodes. This slight decrease in performance for high-degree nodes is due to the reduced influence of each neighboring node. When the target node has a high degree, the impact of each individual neighbor is diluted in the aggregated information of the GCN layer. Overall, NILS consistently outperforms LinkTeller.

We also compare NILS with the link-stealing attacks introduced in [9], where different attack strategies rely on various types of background knowledge, such as node attributes and shadow datasets. Specifically, in Attack-2, the adversary has access to both the features and prediction scores of the nodes, creating LSA2-attr and LSA2-post attacks. LSA2-attr calculates distances between node attributes, while LSA2-post computes distances between prediction scores. These attacks align closely with our threat model, making them relevant for comparison. As shown in Table 3, NILS outperforms both LSA2-post and LSA2-attr attacks but performs nearly equivalent to LinkTeller. These results demonstrate that NILS maintains effectiveness in both transductive and inductive settings.

Table 2: Performance of NILS and LinkTeller across TWITCH-FR, TWITCH-RU, and Flickr under low, unconstrained, and high constraint settings.

Dataset	Method	low		uncontrained		high	
		precision	recall	precision	recall	precision	recall
TWITCH-FR	NILS (Ours)	100.0 $\pm_{0.0}$	100.0 $\pm_{0.0}$	99.1 $\pm_{0.8}$	99.6 $\pm_{0.35}$	99.9 $\pm_{2.6}$	100.0 $\pm_{0.0}$
	LinkTeller	92.5 $\pm_{5.4}$	92.5 $\pm_{5.4}$	84.1 $\pm_{3.7}$	78.2 $\pm_{1.9}$	83.2 $\pm_{1.4}$	80.6 $\pm_{6.7}$
TWITCH-RU	NILS (Ours)	100.0 $\pm_{0.0}$	100.0 $\pm_{0.0}$	96.4 $\pm_{0.4}$	98.3 $\pm_{0.7}$	99.9 $\pm_{0.1}$	99.4 $\pm_{0.1}$
	LinkTeller	78.8 $\pm_{1.9}$	92.6 $\pm_{5.5}$	71.8 $\pm_{2.2}$	78.5 $\pm_{2.4}$	89.7 $\pm_{1.7}$	65.7 $\pm_{3.9}$
Flickr	NILS (Ours)	100.0 $\pm_{0.0}$	100.0 $\pm_{0.0}$	99.1 $\pm_{1.7}$	95.8 $\pm_{5.0}$	93.7 $\pm_{3.1}$	78.9 $\pm_{1.9}$
	LinkTeller	51.0 $\pm_{7.0}$	53.3 $\pm_{4.7}$	33.8 $\pm_{13.3}$	32.1 $\pm_{13.3}$	18.2 $\pm_{4.5}$	18.5 $\pm_{6.1}$

Table 3: Performance of NILS, LinkTeller [23], and link-stealing attacks [9] across Cora, Citeseer, and Pubmed.

Method	Cora		Citeseer		Pubmed	
	precision	recall	precision	recall	precision	recall
NILS (Ours)	99.7 $\pm_{0.2}$	99.6 $\pm_{0.3}$	97.4 $\pm_{0.2}$	98.2 $\pm_{0.1}$	99.7 $\pm_{0.0}$	100.0 $\pm_{0.0}$
LinkTeller	99.5 $\pm_{0.1}$	99.5 $\pm_{0.1}$	99.7 $\pm_{0.0}$	99.7 $\pm_{0.0}$	99.7 $\pm_{0.0}$	99.7 $\pm_{0.0}$
LSA2-post	86.7 $\pm_{0.2}$	86.7 $\pm_{0.2}$	90.1 $\pm_{0.2}$	90.1 $\pm_{0.2}$	78.8 $\pm_{0.1}$	78.8 $\pm_{0.1}$
LSA2-attr	73.6 $\pm_{0.1}$	73.6 $\pm_{0.1}$	80.9 $\pm_{0.1}$	80.9 $\pm_{0.1}$	82.4 $\pm_{0.1}$	82.4 $\pm_{0.1}$

6 Defense

This section introduces DP in the context of GNNs to protect the privacy of the graph, preventing an adversary from discovering whether there is a link between two nodes. We define the neighboring relation of graphs and revise the definition of DP accordingly.

6.1 DP for graphs

DP was originally defined for microdata, where two databases are neighbors if they differ by one record. For graphs, this notion must be adapted since two graphs may differ by either one edge or one node. In the literature, two adaptations of DP for graphs are proposed: edge-level DP [13] and node-level DP [8]. A graph $\mathcal{G} = (V, E)$ is represented by adjacency matrix A, where $A_{ij} = 1$ if there is a link between node i and node j, and $A_{ij} = 0$ otherwise, where $i, j \in \{1, \ldots, |V|\}$.

Edge-level DP considers graphs $\mathcal{G}$ and $\mathcal{G}'$ as neighbors if they differ by a single edge, while node-level DP considers them neighbors if they differ by a single node and all its incident edges.

6.2 One-node-one-edge-level DP

The adversary defined in Sec. 4.2 adds a malicious node to a graph and connects it to a target node through a single edge. Countering such an adversary with node-level DP would increase noise and decrease model accuracy unnecessarily. Therefore, we define a new notion of neighborhood between graphs and the corresponding DP mechanism.

Definition 1 (One-node-one-edge-level adjacent graphs). *$\mathcal{G}$ and $\mathcal{G}'$ are* one-node-one-edge-level adjacent *if one can be obtained from the other by adding a single node with one edge only.*

The adjacency matrices of such neighboring graphs differ by one row and one column, and the difference in L_1-norm is always one.

Definition 2 ((ε, δ)-One-node-one-edge-level DP). *A randomized mechanism $\mathcal{M}$ satisfies (ε, δ)-**one-node-one-edge-level DP** with $\varepsilon, \delta \geqslant 0$ if, for all pairs of one-node-one-edge-level adjacent graphs $\mathcal{G}, \mathcal{G}'$ and for all measurable $\mathcal{O} \subseteq \mathrm{Range}(\mathcal{M})$, the following holds:*

$$\mathrm{P}\{\mathcal{M}(\mathcal{G}) \in \mathcal{O}\} \leqslant e^{\varepsilon}\, \mathrm{P}\{\mathcal{M}(\mathcal{G}') \in \mathcal{O}\} + \delta.$$

6.3 Countermeasures for our attack

To defend against the NILS attack, we propose using the LapGraph mechanism introduced in [23]. While output perturbation [6] can also satisfy one-node-one-edge-level DP, it significantly deteriorates the GNN output accuracy due to the

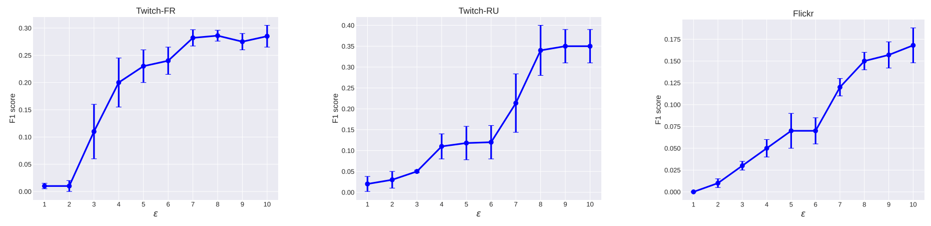

Fig. 2: F_1 score of the attack for different values of ε.

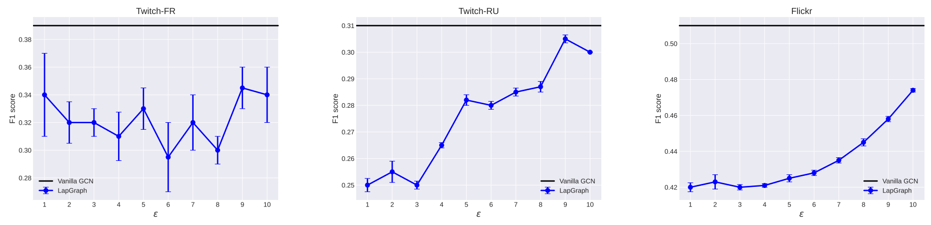

Fig. 3: F_1 score utility of the GCN for different values of ε.

large L_1-global sensitivity of the prediction matrix. Instead, the LapGraph algorithm perturbs the adjacency matrix using the Laplace mechanism and binarizes it by replacing the top-N largest values by 1 and the remaining values by 0. Here, N represents the estimated number of edges in the graph, computed using the Laplace mechanism.

Leveraging the post-processing property of DP[4], the edge information remains protected even if the adversary observes the GNN's predictions. Each time a user connects a new node, a new adjacency matrix is generated using LapGraph, accumulating the privacy budget by the sequential composition property of DP [7].

Although LapGraph was proposed to meet edge-level DP, it can also satisfy one-node-one-edge-level DP. Let f_A be the query function returning the adjacency matrix of a graph G. For one-node-one-edge neighboring graphs G and G', the adjacency matrices A and A' have different dimensions. We append one zero-row and one-zero column to A, resulting in $\bar{A}$. The $(n+1)$-th columns (or rows) of $\bar{A}$ and A' always differ in one element, yielding an L_1-global sensitivity of 1. Thus, the LapGraph mechanism provides stronger protection for the same level of utility compared to edge-level DP.

[4] The post-processing property allows arbitrary data-independent transformations to DP outputs without affecting their privacy guarantee [27].

6.4 LapGraph evaluation

We evaluate the effectiveness of LapGraph [23] in reducing the success of the NILS attack while ensuring our one-node-one-edge-level DP notion. We also investigate the utility of GCN models trained with LapGraph protection.

Evaluation setup. We use the same training hyperparameters and normalization techniques as in the vanilla case, where DP is not applied. Initially, we protect the training graph with LapGraph. Subsequently, we apply LapGraph each time the graph changes due to node injection by the adversary. Following [23], we compute the F_1 score for our NILS attack and the classification task's F_1 score for the GCN. This allows us to measure LapGraph protection and GCN utility across various privacy budgets ε. We report the results averaged over 5 runs with different random seeds for LapGraph.

Evaluation results. Figure 2 presents the F_1 score of the attack for various ε values. We observe that applying LapGraph reduces the effectiveness of NILS. The F_1 score becomes almost zero when the privacy budget ε is small. However, for large ε, LapGraph provides moderate protection, but the attack's F_1 score remains significantly lower than in the non-private case where DP is not applied.

In the LinkTeller [23] attack, where LapGraph is applied only once to ensure edge-level DP, LapGraph offers limited protection when ε is large, allowing LinkTeller to achieve a success rate nearly as high as in the non-private case. Conversely, in our scenario, where LapGraph is also applied after the adversary's node injection, LapGraph provides stronger protection.

Applying LapGraph during inference makes it more challenging for the adversary to distinguish between the target node's neighbors and non-neighbors, as the prediction scores of all target nodes change after each inference query. Consequently, the distances between the prediction scores P and P', before and after the node injection, become noisier due to LapGraph's application.

To provide insights about the privacy-utility tradeoff of LapGraph, we present in Figure 3 the utility of the GCNs for different values of the privacy budget. We observe that the utility increases when ε increases, as expected. Large values of $\varepsilon \geq 7$ give a better utility close to that in the non-private vanilla case. Therefore, carefully choosing an ε will give fairly good utility and a certain level of protection against the NILS attack.

7 Conclusion

In this paper, we have presented a powerful new NILS attack—a link-stealing attack using node injection against GNNs. Our results have demonstrated the superior performance of NILS compared to previous attacks, further emphasizing the vulnerabilities of GNNs regarding edge information leakage. We have also evaluated NILS against differentially private GNNs, ensuring a one-node-one-edge-level DP notion specifically designed to protect against our proposed attack.

References

1. Atwood, J., Towsley, D.: Diffusion-convolutional neural networks. In: 30th International Conference on Neural Information Processing Systems (NIPS) (2016)
2. Brunet, S., Canard, S., Gambs, S., Olivier, B.: Novel differentially private mechanisms for graphs. IACR Cryptol. ePrint Arch. p. 745 (2016)
3. Cadwalladr, C., Graham-Harrison, E.: Revealed: 50 million facebook profiles harvested for cambridge analytica in major data breach. The Guardian (Mar 2018)
4. Duddu, V., Boutet, A., Shejwalkar, V.: Quantifying privacy leakage in graph embedding. In: 17th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (MobiQuitous) (2021)
5. Dwork, C.: Differential privacy. In: 33rd International Colloquium on Automata, Languages and Programming, ICALP (2006)
6. Dwork, C., McSherry, F., Nissim, K., Smith, A.: Calibrating noise to sensitivity in private data analysis. In: Theory of Cryptography (2006)
7. Dwork, C., Roth, A.: The algorithmic foundations of differential privacy. Found. Trends Theor. Comput. Sci. **9**(3-4), 211–407 (2014)
8. Hay, M., Li, C., Miklau, G., Jensen, D.: Accurate estimation of the degree distribution of private networks. In: IEEE International Conference on Data Mining (2009)
9. He, X., Jia, J., Backes, M., Gong, N.Z., Zhang, Y.: Stealing links from graph neural networks. In: USENIX Security Symposium (2021)
10. Karwa, V., Slavkovic, A.B.: Differentially private graphical degree sequences and synthetic graphs. In: Privacy in Statistical Databases - UNESCO Chair in Data Privacy, International Conference, PSD (2012)
11. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. In: 5th International Conference on Learning Representations, (ICLR) (2017)
12. Kipf, T.N., Welling, M.: Semi-Supervised Classification with Graph Convolutional Networks. In: Proceedings of the 5th International Conference on Learning Representations (ICLR) (2017)
13. Kossinets, G., Watts, D.J.: Empirical analysis of an evolving social network. science **311**(5757), 88–90 (2006)
14. Li, K., Sun, J., Chen, R., Ding, W., Yu, K., Li, J., Wu, C.: Towards practical edge inference attacks against graph neural networks. In: ICASSP 2023 - 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) (2023)
15. Mueller, T.T., Usynin, D., Paetzold, J.C., Rueckert, D., Kaissis, G.: Sok: Differential privacy on graph-structured data. CoRR **abs/2203.09205** (2022)
16. Rozemberczki, B., Sarkar, R.: Twitch gamers: a dataset for evaluating proximity preserving and structural role-based node embeddings. arXiv preprint arXiv:2101.03091 (2021)
17. Sajadmanesh, S., Shamsabadi, A.S., Bellet, A., Gatica-Perez, D.: GAP: differentially private graph neural networks with aggregation perturbation. CoRR **abs/2203.00949** (2022)
18. Scarselli, F., Gori, M., Tsoi, A.C., Hagenbuchner, M., Monfardini, G.: The graph neural network model. IEEE Transactions on Neural Networks **20**(1), 61–80 (2009)
19. Sen, P., Namata, G., Bilgic, M., Getoor, L., Galligher, B., Eliassi-Rad, T.: Collective classification in network data. AI Magazine **29**(3), 93 (Sep 2008)

20. Sun, L., Dou, Y., Yang, C., Wang, J., Liu, Y., Yu, P.S., He, L., Li, B.: Adversarial attack and defense on graph data: A survey. arXiv preprint arXiv:1812.10528 (2018)
21. Velickovic, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., Bengio, Y.: Graph attention networks. In: 6th International Conference on Learning Representations, (ICLR) (2018)
22. Wu, B., Yang, X., Pan, S., Yuan, X.: Adapting membership inference attacks to GNN for graph classification: Approaches and implications. In: IEEE International Conference on Data Mining, (ICDM) (2021)
23. Wu, F., Long, Y., Zhang, C., Li, B.: Linkteller: Recovering private edges from graph neural networks via influence analysis. In: 2022 IEEE Symposium on Security and Privacy (SP) (2022)
24. Xu, K., Hu, W., Leskovec, J., Jegelka, S.: How powerful are graph neural networks? In: 7th International Conference on Learning Representations, (ICLR) (2019)
25. Zeng, H., Zhou, H., Srivastava, A., Kannan, R., Prasanna, V.K.: Graphsaint: Graph sampling based inductive learning method. In: 8th International Conference on Learning Representations, (ICLR) (2020)
26. Zhang, M., Chen, Y.: Link prediction based on graph neural networks. Advances in neural information processing systems **31** (2018)
27. Zhu, K., Fioretto, F., Van Hentenryck, P.: Post-processing of differentially private data: A fairness perspective. In: 31st International Joint Conference on Artificial Intelligence, IJCAI (2022)

A Appendix

A.1 Differential Privacy

The original definition of Differential Privacy (DP) [5, 7] was introduced in the context of microdata, i.e., databases containing individual records. A central aspect of DP is the concept of *neighborhood*, originally defined for that data structure.

Definition 3 (Neighboring Databases). *Let $\mathcal{D}$ be the class of possible databases. Any two databases $D, D' \in \mathcal{D}$ that differ in one record are called* neighbors*. For two neighboring databases, $d(D, D') = 1$, where d denotes the Hamming distance.*

Definition 4 ((ε, δ)-Differential Privacy [5, 7]). *A randomized mechanism $\mathcal{M}$ satisfies (ε, δ)-DP with $\varepsilon, \delta \geq 0$ if, for all pairs of neighboring databases $D, D' \in \mathcal{D}$ and for all measurable $\mathcal{O} \subseteq Range(\mathcal{M})$,*

$$\mathbb{P}\{\mathcal{M}(D) \in \mathcal{O}\} \leq e^{\varepsilon}\mathbb{P}\{\mathcal{M}(D') \in \mathcal{O}\} + \delta.$$

In simple terms, the output of a DP mechanism should not reveal the presence or absence of any specific record in the database, up to an exponential factor ε and additional δ. When each record corresponds to an individual, DP ensures their information remains confidential. A lower ε, known as the *privacy budget*, provides stronger protection.

The most popular DP mechanism is the Laplace mechanism, which relies on *global sensitivity*, defined as follows:

Definition 5 (Global Sensitivity [7]). *The L_p-global sensitivity of a query function $f : \mathcal{D} \to \mathbb{R}^d$ is defined as*

$$\Delta_p(f) = \max_{D, D' \in \mathcal{D}} \|f(D) - f(D')\|_p,$$

where D, D' are any two neighboring databases.

Definition 6 (Laplace Mechanism [7]). *Given any function $f : \mathcal{D} \to \mathbb{R}^d$, the Laplace mechanism is defined as:*

$$\mathcal{M}_L(D, f(\cdot), \varepsilon) = f(D) + (Y_1, \ldots, Y_d),$$

where Y_i are i.i.d. random variables drawn from a Laplace distribution with zero mean and scale $\Delta_1(f)/\varepsilon$.

A.2 Experimental Setup

Datasets We evaluated our attack on several real-world datasets used in related research. We include the Flickr dataset [25], where nodes represent images and edges connect nodes with shared properties. Node features contain word representations. We also use two Twitch datasets (TWITCH-FR and TWITCH-RU) [16] to evaluate NILS and Twitch-ES for training GNNs in an inductive setting, as done in [23]. Twitch datasets map follow connections between users and aim to classify if a streamer uses explicit language, using features like preferred games and location. For the transductive setting, where training and testing occur on the same graph, we use three citation network datasets: Cora, Citeseer, and Pubmed [19]. These involve predicting the topic of publications based on textual features and citation relationships.

Models We follow the approach in [23] for training models and selecting hyperparameters. The authors trained Graph Convolutional Networks (GCNs) using various configurations, including normalization techniques, the number of hidden layers, input and output units, and dropout rates. A grid search strategy identified optimal hyperparameters, evaluating performance on a validation set. By using the same training procedures and hyperparameter tuning strategies, we ensure consistency across studies.

Evaluation Metrics We employ precision, recall, and the F_1 score as our primary evaluation metrics, following [23]. These metrics are suitable for addressing the imbalanced binary classification problem, where the minority class (connected nodes) is of central interest. We select target nodes $V_\mathcal{A}$ such that $|V_\mathcal{A}| = 500$, using uniform random sampling. We explore scenarios where target nodes exhibit either low or high degrees, as discussed in [23, Section V.D.]. Results are averaged over three runs with different random seeds, along with the corresponding standard deviation.

A.3 Limitations: Depth of the GNN

We examine the impact of increasing the GNN depth on the success rate of the attack for the Twitch-FR dataset. Our findings in Figure 4 show that as GNN depth increases, the attack's success rate decreases. This reduction is due to the dilution of the injected node's influence within the target node's neighborhood. As GNN depth increases, the model aggregates information from a larger neighborhood, diluting the influence of the injected malicious node and diminishing the attack's effectiveness.

Compared to LinkTeller [23], as shown in Table 4, NILS outperforms LinkTeller across various GCN depths. For the Twitch-FR dataset, NILS demonstrates higher precision and recall values at GCN depth 3 (precision: 85.1 $\pm_{1.2}$, recall: 81.6 $\pm_{1.2}$) compared to LinkTeller at depth 2 (precision: 84.1 $\pm_{3.7}$, recall: 78.2$\pm_{1.9}$). These results highlight the effectiveness of our node injection strategy, consistently outperforming LinkTeller across different GCN depths.

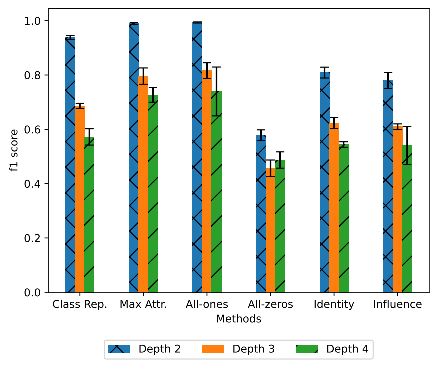

Fig. 4: Success rates of the attack for different depths and malicious features generation strategies for Twitch-FR dataset

Dataset	Method	Depth-2		Depth-3	
		precision	recall	precision	recall
TWITCH-FR	NILS (Ours)	99.13 $\pm_{0.8}$	99.57 $\pm_{0.35}$	85.06 $\pm_{1.2}$	81.56 $\pm_{1.2}$
	LinkTeller	84.1 $\pm_{3.7}$	78.2 $\pm_{1.9}$	50.1 $\pm_{5.1}$	46.6 $\pm_{5.0}$
TWITCH-RU	NILS (Ours)	96.45 $\pm_{0.4}$	98.34 $\pm_{0.7}$	78.78 $\pm_{3.8}$	76.35 $\pm_{9.3}$
	LinkTeller	71.8 $\pm_{2.2}$	78.5 $\pm_{2.4}$	45.7 $\pm_{2.2}$	50.0 $\pm_{2.8}$

Table 4: Success rates of the attack for different depths in comparison with LinkTeller [23]. We use the all-ones strategy and Twitch-FR dataset.