VertexSerum: Poisoning Graph Neural Networks for Link Inference

Ruyi Ding*, Shijin Duan*, Xiaolin Xu, Yunsi Fei
Northeastern University, Boston, MA, USA
{ding.ruy, duan.s, x.xu, y.fei}@northeastern.edu

Abstract

Graph neural networks (GNNs) have brought superb performance to various applications utilizing graph structural data, such as social analysis and fraud detection. The graph links, e.g., social relationships and transaction history, are sensitive and valuable information, which raises privacy concerns when using GNNs. To exploit these vulnerabilities, we propose VertexSerum, a novel graph poisoning attack that increases the effectiveness of graph link stealing by amplifying the link connectivity leakage. To infer node adjacency more accurately, we propose an attention mechanism that can be embedded into the link detection network. Our experiments demonstrate that VertexSerum significantly outperforms the SOTA link inference attack, improving the AUC scores by an average of 9.8% *across four real-world datasets and three different GNN structures. Furthermore, our experiments reveal the effectiveness of VertexSerum in both black-box and online learning settings, further validating its applicability in real-world scenarios.*

1. Introduction

Graph Neural Networks (GNNs) have been widely adopted in various domains, such as financial fraud detection [25], social network analysis [19], and heart-failure prediction [6], thanks to their capabilities to model high-dimensional features and complex structural relationships between entities [30]. However, with the increasing use of graph data, concerns about data privacy are also growing [1, 7, 27]. This is particularly relevant in industries such as finance and healthcare, where sensitive relationships are often embedded in graph-structured data.

Recently, there has been a rise in privacy attacks on GNNs [11, 28] that infer the existence of links between nodes in graphs by only querying the graph model, thus posing a threat to the confidentiality of GNNs. For a graph node pair, the similarity of their posterior distributions (abbreviated as "posteriors" [11]) is measured to deduce the link existence. For instance, in federated learning scenario [10], where different parties keep private data locally but contribute to the GNN training in the cloud based on their data, a malicious contributor can infer the link belonging to other contributors by querying trained GNN models. In this context, the risks of link information leakage lie in the joint training of GNNs and the available GNN inference APIs on graph data.

In this work, we identified a limitation of the existing link-inferring attacks: they do not perform well if the interested node pairs are from the same category (intra-class). This is due to the high similarity of the posterior distributions between node pairs in the same category. To overcome this limitation, we propose a novel approach to significantly improve link inference attacks, particularly on intra-class node pairs, by allowing a malicious contributor to poison the graph during GNN training in an unnoticeable way.

This paper proposes a novel privacy-breaching data poisoning attack on GNNs, **VertexSerum**[1], with a new analysis strategy. The attack aims to amplify the leakage of private link information by modifying nodes/vertices. This work makes the following contributions:

1. We propose a new evaluation metric, intra-class AUC score, for link inference attacks, by considering only node pairs from the same class. This new metric overcomes the bias of the prior works that do not differentiate between inter-class and intra-class, and brings valuable insights for our approach.

2. We introduce the first privacy-breaching data poisoning attack on GNNs, which injects adversarial noise into a small portion ($< 10\%$) of the training graph to amplify the graph's link information leakage. We constructively employ a self-attention-based network to train the link detector and propose a pre-training strategy to overcome the overfitting issue of limited training data.

3. We demonstrate the effectiveness of the proposed link inference attack on popular GNN structures and graph datasets. The attack improves the link stealing AUC score by 9.8% compared to the SOTA method in [11].

[*] These authors contributed equally to this work. This work to be appeared in ICCV 2023.

[1] The name is inspired by `Veritaserum` in the Harry Potter series.

4. We consider the practicality of applying VertexSerum by evaluating its homophily noticeability of the poisoned graph and the victim model accuracy. The experimental results show that VertexSerum increases model privacy leakage without affecting the GNN performance.

2. Background and Related Work

2.1. Graph Neural Networks

Graph Neural Networks (GNNs) are widely used in semi-supervised graph node classification tasks [30]. A graph, denoted as G=(V, E), has a topology with a set of nodes V and edges/links E. This work focuses on undirected homogeneous graphs, commonly studied in graph theory and network analysis [5, 6, 16, 19, 25, 29]. A link between node u and v is represented by $(u, v) \in E$, while its absence is $(u, v) \notin E$. For each node, it has features x and corresponding categorical label y for a classification task. Together with the graph, node features and labels compose the dataset used for GNN training and validation, denoted as D=$\{G, \boldsymbol{X}, \boldsymbol{Y}\}$. After training, a neural network model for the graph is generated, denoted as f, where the model output $f(u)$ represents the posterior probabilities of node u for the classes. The main GNN architectures for node classification include Graph Convolutional Network (GCN) [13], Graph SAmple and aggreGatE (GraphSAGE) [9], and Graph Attention Network(GAT) [24]. These models, with different neural network architectures, all learn to aggregate feature information from a node's local neighborhood, whose receptive field is bounded by the model depth. Different from previous works that do not differentiate between nodes in the graph for evaluation, we specifically analyze the intra-class node pairs, which refer to nodes in the same class.

2.2. Link Inference Attack

GNNs, like other machine learning models, are susceptible to various privacy attacks that compromise the confidentiality of sensitive information within the data. These include membership inference attacks [15], adversarial graph injection attacks [20], graph modification attacks [32], and link privacy attacks [11, 28]. Stealing Link Attack [11] was the first link privacy attack, where the graph structure information is inferred from the prediction results of the GNN model, i.e., posterior distributions of nodes. Another attack, LinkTeller [28], takes into account the influence propagation during GNN training for link inference. However, LinkTeller requires the attacker to have access to the graph's node features $\boldsymbol{X}$, a much stronger attack model than ours where the attacker only accesses the posterior distributions of interested nodes, a more realistic scenario.

Benchmark	R_{linked}	$R_{unlinked}$	AUC_{all}	AUC_1
Cora	0.81 : 0.19	0.18 : 0.82	0.907	0.874
Citeseer	0.74 : 0.26	0.18 : 0.82	0.987	0.912
AMZPhoto	0.83 : 0.27	0.16 : 0.84	0.919	0.813
AMZComputer	0.78 : 0.22	0.21 : 0.79	0.913	0.826

Table 1. Node pairs' distribution analysis. R is the ratio of intra-class node pairs vs. inter-class, among all linked node pairs and unlinked node pairs. AUC reflects the success rate of link reference attacks, where AUC_{all} considers overall node pairs and AUC_1 considers only node pairs from intra-class, e.g., in class 1.

2.3. Enhance Privacy Leakage via Data Poisoning

Data poisoning is an effective method to manipulate the behavior of the victim model during training by intentionally introducing malicious training samples into the benign dataset [31]. The recent work [3] poisons the training dataset with a small number of crafted samples, with incorrect labels, which results in a trained model that overfits the training data, significantly increasing the success rate of membership inference attacks. Inspired by the previous *membership* leakage amplification by data poisoning, on conventional deep learning models, this work shows that properly crafted data poisoning is also able to amplify *link* leakage of the graph in GNNs, posing a significant privacy threat to GNNs. Data poisoning on GNNs can be achieved by modifications made to node features, node labels, or the graph structure. We choose to poison node features with small perturbations to make the attack stealthy. Our attack is more effective than the state-of-the-art link inference attacks [11, 28] with a specific focus on intra-class inference.

3. Observations and Insights

3.1. Link Inference Attack Does Not Always Work

Previous research of link inference attacks on GNNs has demonstrated good performance in predicting the existence of links among overall node pairs [11]. The GNN model is queried, and the similarity of the posterior distributions of the node pair is calculated for a link detector, which returns the prediction of whether a link exists between these two nodes. Although the performance on overall node pairs tends to be good, when considering only intra-class node pairs, i.e., to infer the link existence of node pairs from the same class, the effectiveness is much lower. This is due to several reasons: ① Though it is common to select equal numbers of linked and unlinked node pairs for evaluation, the distribution of inter-class and intra-class node pairs in both sets are highly unbalanced: while the majority of linked node pairs are intra-class, most of the unlinked node pairs are inter-class; ② the posterior distributions of intra-class nodes are much more similar than those of inter-class nodes. We demonstrate the characteristic of node pairs

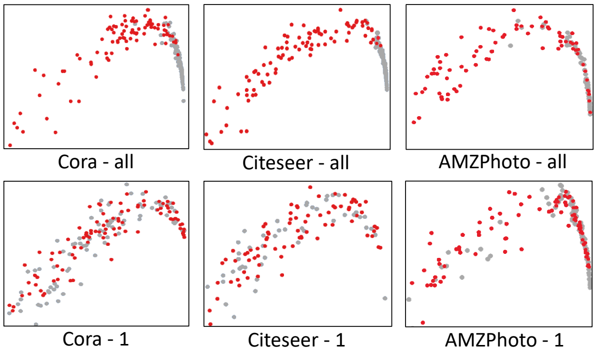


Figure 1. Visualization on link inference, overall vs. intra-class. We randomly sampled 200 node pairs (100 linked + 100 unlinked) from all nodes (all) and only the second class (1). The dots are the PCA projections of the similarities of node pair posteriors, where dots in red represent linked pairs and dots in gray represent unlinked pairs. The more apart the two distributions are, the easier link inference can be.

distribution in Table 1. If we only consider node pairs from the same classes, their posterior distribution will be similar regardless of whether they are linked or not. The different success rates of the link inference attack on node pairs from the entire graph and only one class are reflected by the AUC scores, presented in the third and fourth columns of Table 1, and also visualized in Figure 1. As the visualization shows, in the top row across three different datasets, the linked node pairs and unlinked node pairs are easily distinguishable, from the *overall node pairs*; while the bottom row shows that for intra-class node pairs, the two distributions are not easily separable, indicating the difficulty for link inference. To address this issue, we propose a new metric, *intra-class AUC score*, to evaluate the link inference attack's performance in the same classes, as presented in Column 5 of Table 1.

3.2. Graph Poisoning Threat to GNNs

Data poisoning on Graph neural networks can be achieved on various entries. For example, in social networks, an adversarial user can create fake accounts or modify their profile deliberately. As GNNs applied to these graphs must be frequently retrained or fine-tuned, an attack surface is created for malicious parties to compromise the GNN performance or privacy by crafting malicious data. Specifically in federated learning, a common structural graph is used by distributed contributors to provide data for training, malicious parties may upload carefully poisoned data into the graph in a stealthy and unobtrusive way. Graph poisoning attacks are easy to conduct, difficult to detect, and highly effective in compromising GNNs. Our proposed attack shows that by data poisoning, the link leakage of intra-class nodes can be significantly amplified, and link inference can be effectively accomplished.

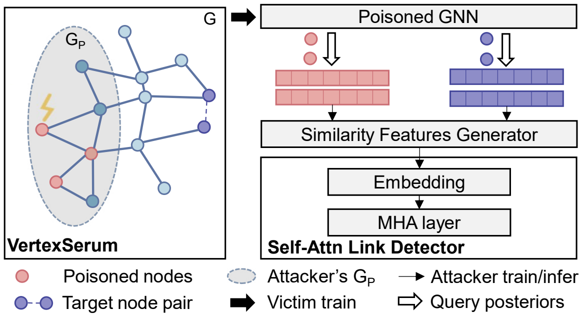


Figure 2. Overview of VertexSerum with Self-Attention Detector.

4. VertexSerum - The Proposed Attack

In this section, we illustrate our proposed privacy-breaching data poisoning attack – VertexSerum. The overview of the attack procedure is presented in Figure 2.

4.1. Threat Model

Adversary's Goal. The attack targets GNN-based classifiers, which utilize node features and the graph topology to predict the labels for querying nodes. The attacker aims to deduce the connectivity between any pair of nodes (u, v) belonging to class k by querying a pre-trained GNN model. **Adversary's Knowledge.** We assume the attacker has limited access to the vendor's GNN as they can only acquire the interested nodes' posteriors through queries. We also assume that the attacker has access to a portion of the graph, as in federated learning, where the attacker acts as a distributed contributor to provide data for the training dataset, which can be intentionally poisoned. Note these assumptions align with "Attack-3" in the state-of-the-art link inference attack [11] and are practical. We limit the portion of the graph that the attacker can manipulate, such as 10% of the entire graph, which is more practical and realistic in federated learning settings.

4.2. Inspiration from ML Poisoning

In conventional machine learning (ML) regime, poisoning the training dataset with tainted data can expose user data privacy [3, 22], e.g., by injecting label-tampered samples to the training data, forcing the victim model to overfit on specific features of each sample, thereby exacerbating its membership leakage. However, the potential of such data poisoning schemes have not been explored in attacking the link privacy of GNNs. This work bridges this knowledge gap by crafting samples in training dataset to strengthen GNN model's attention on node connections, making the model to produce more similar outputs for linked nodes and increase the dissimilarity between unlinked nodes. Rather than generating abnormal labels which may be detected by outlier detection tools, we induce poisoned features with small perturbations with Projected Gradient Descent

Algorithm 1 Link Stealing with VertexSerum

```
Require: Target class $k$; Partial graph $G_p = (V_p, E_p)$; Step size $\epsilon$; Maximum number of iterations $N$; Training algorithm $\mathcal{T}$; Vendor graph $G = (V, E)$.
Ensure: Link existence of node pair $\hat{z} = (u, v)$.
    /*Generate Poisoned Graph*/
 1: Train shadow GNN model on $G_p$ with the public training algorithm $f_\theta^{\text{sh}} \leftarrow \mathcal{T}(G_p)$.
 2: for $n = 1, 2, \ldots, N$ do    ▷ Projected Gradient Descent
 3:     Compute the gradient of loss $L(f_\theta^{\text{sh}})$: $g_n \leftarrow \nabla L$
 4:     Update nodes features in class $k$ to increase loss, $x \in V_p^k$: $x_{n+1} \leftarrow x_n + \epsilon g_n$
 5: end for
 6: Get poisoned graph $G_p'$ and send it to the vendor.
 7: The vendor trains a GNN model $f_\theta$ on $G \cup G_p'$.
    /*Train Link Detector*/
 8: Query model $f_\theta$ to obtain posteriors of nodes in $V_p^k$.
 9: Compute and aggregate the similarity features $\boldsymbol{F}_p^k$ from node pairs' posteriors as truth dataset $\mathcal{D}_p^k = \{\boldsymbol{F}_p^k, E_p^k\}$.
10: Train binary classifier $\mathcal{M}$ on $\mathcal{D}_p^k$ using self-attention link detector.
    /*Link Inference*/
11: Given a target node pair in class $k$, $u, v \in V^k$, compute the similarity feature $\boldsymbol{F}_{u,v}^k$.
12: Feed in $\boldsymbol{F}_{u,v}^k$ to detector $\mathcal{M}$ for link inference.
13: return True / False
```

(PGD), allowing us to achieve attack stealthiness.

4.3. Attack Flow of VertexSerum

VertexSerum aims to steal the true link information of interested node pairs. The attack is carried out between a model vendor $\mathcal{V}$ and a malicious contributor $\mathcal{A}$. The vendor has access to the entire graph dataset D=$\{G, \boldsymbol{X}, \boldsymbol{Y}\}$ and trains a downstream task with a public training algorithm $\mathcal{T}$ [2]. The adversary contributes a small portion of the dataset, D_p=$\{G_p, \boldsymbol{X}_p, \boldsymbol{Y}_p\}$, containing a partial graph G_p, which is used for both generating the poisoning sub-graph and training the link detector. The attack steps are:

1. The adversary chooses a target class k from the label space $\boldsymbol{Y}$. The attack goal is to predict the link existence between nodes u, v, i.e., if $(u, v) \in E$, when y_u=y_v=k.

2. Following the steps in Lines 1-6 of Algorithm 1, the adversary ***generates a partial dataset*** D_p' ***with a poisoned graph*** G_p' by analyzing a shadow model trained on G_p, as depicted in the shadow part in Figure 2, and ***sends it to the vendor***.

3. The vendor trains a GNN model for downstream tasks $f_\theta \leftarrow \mathcal{T}(D \cup D_p')$ on the poisoned graph $G \cup G_p'$.

4. The adversary queries the GNN model, f_θ, with the possessed poisoned partial graph G_p' and generates similarities of posteriors. ***Binary link detectors*** are constructed to infer link existence, as shown in the right bottom part of Figure 2 and detailed in Lines 8-10 of Algorithm 1.

5. The adversary makes a guess $\hat{z} = (u, v)$ with the link detectors (Line 11-13).

Our attack utilizes data poisoning to breach the confidentiality of GNNs: the poisoned graph G_p' is used in the victim GNN model training, with an objective to amplify the model privacy leakage.

4.4. Requirements of the Poisoning Nodes

For Step 2 of the attack, to generate a graph that enhances the model's aggregation on linked nodes, we design a specific poisoned graph G_p' that makes the GNN model f_θ focus more on adjacency. Next, we outline requirements for successful node poisoning:

1. **Intact Community.** The adversary should ensure that the node classification accuracy for the victim task is not evidently affected, so that the poisoned graph is less likely to be rejected by the vendor for GNN training. Besides, misclassified nodes can negatively impact passing information to adjacent linked nodes, leading to an overall lower aggregation capability for the GNN model.

2. **Node Attraction and Repulsion.** The poisoned samples should simultaneously promote the similarity of the GNN outputs on linked nodes (attraction) and the dissimilarity on unlinked nodes (repulsion). This requires a balance between the attraction and repulsion of node features when poisoning the dataset.

3. **Adversarial Robustness.** Adversarial training techniques [17, 21] can improve a model's robustness against adversarial samples, where the model tolerates small input perturbations and outputs similar predictions. In VertexSerum, we utilize adversarial training to increase the model's adversarial robustness, guiding linked nodes with similar features to produce similar posteriors.

4.5. Crafting Poisoning Features via PGD

To meet these requirements, we propose a graph poisoning method optimized with projected gradient descent (PGD). We adopt the shadow training methods [11, 18], where the attacker will first train a shadow GNN (f_θ^{sh}) on the possessed partial graph G_p. The optimal perturbation to add on node features is found based on the gradient of the loss function shown in Eq. 1.

$$L = \alpha L_{attraction} + \beta L_{repulsion} + \lambda L_{CE} \tag{1}$$

[2] We assume the GNN type is open to the adversary for the ease of evaluation. We also demonstrate the effectiveness of VertexSerum in Section 5.7, when the adversary has no clue of the GNN model.

The loss function includes three terms, with α, β, λ as positive coefficients to balance attraction and repulsion:

1. The attraction loss penalizes the euclidean distance of posteriors on two linked nodes. The PGD will find node features that reduce the distance between linked nodes.
$$L_{attraction} = -\sum_{(u,v)\in E} (f_\theta^{sh}(u) - f_\theta^{sh}(v))^2 \qquad (2)$$
2. The repulsion term computes the cosine similarity between unlinked nodes. The rationale is that cosine is bounded so as to avoid an overlarge dissimilarity term. The PGD will find the node features that reduce the similarity between unlinked nodes.
$$L_{repulsion} = \sum_{\substack{u,v\in V, u\neq v,\\ (u,v)\notin E}} (1 - cos(f_\theta^{sh}(u), f_\theta^{sh}(v)))^2 \qquad (3)$$
3. The cross-entropy term L_{CE} serves as a regularization in the loss function. Its goal is to improve the victim model's adversarial robustness to amplify link leakage.

The previous poisoning attack includes regularization of perturbations, such as the L1 norm, during optimization. However, we observed that this term is not necessary for the PGD process if we have a small updating step size ϵ. By only optimizing Eq. 1, the generated perturbation is already effective and unnoticeable.

4.6. Self-attention Link Detector

In Step 4 of the attack, the adversary trains a link detector using the posteriors of the partial graph by querying the pre-trained vendor model. Previous work [11] used a Multi-Layer Perceptron (MLP) to analyze the similarity features of the node pair posteriors. However, the dense structure of MLP is often inadequate to capture the complex dependencies among similarity features. Furthermore, since the attacker only has a small part ($< 10\%$) of the graph, training an MLP is prone to be unstable due to overfitting. Moreover, since VertexSerum introduces more complex characteristics such as attraction and repulsion during poisoning, the underlying patterns in the similarity features are expected to be more informative. To address these issues, we propose improvement to the MLP model with a Multihead Self-attention [23] link detector, which can efficiently use information by selectively attending to different parts in the input similarity features. We follow the same construction of similarity features as the previous method [11], consisting of eight distances and four entropy features between two nodes. To ensure stability of the self-attention detector on a small dataset, we initialize its first embedding layer with the first fully-connected layer from the MLP. The experimental results in Table 2 in next section show that the introduction of self-attention improves the attack AUC score by an average of 7.2% with the standard deviation dropping by 35%.

5. Experiments

5.1. Experimental Setup

Datasets: We evaluate the effectiveness of VertexSerum on four publicly available datasets: Citeseer [13], Cora [13], Amazon Photo Dataset [14], and Amazon Computer Dataset [14]. These datasets cover different daily-life scenarios and are widely used as benchmarks for evaluating graph neural networks. The first two datasets are citation networks where nodes represent publications, and links indicate citations among them. The last two datasets are co-purchase graphs from Amazon, where nodes represent products, and edges represent the co-purchased relations of products. Our benchmarks scale from (3k nodes + 11k edges) for Cora to (14k nodes + 492k edges) for AMZComputer. We assume the vendor's model is trained on 80% of the nodes and evaluated on the remaining in the graph.

Since we assume the attacker only contributes a small portion of the graph for training, i.e., G'_p, we sample 10% nodes among the training dataset. To train the link detector, we collect all linked node pairs and randomly sample the same number of unlinked node pairs in G'_p. Similarity features are computed based on these node pairs, following [11], together with corresponding link information. We split this dataset into 80% for training and 20% for validation.

Metric: ROC-AUC is a commonly used evaluation metric for binary classification tasks and has also been applied in previous works on link inference [11, 28]. It measures the ability of the link detector to distinguish between linked and unlinked node pairs. A higher AUC indicates superior performance of the link detector in identifying linked node pairs from unlinked ones.

In addition to overall AUC, we also evaluate the intra-class AUC. Overall AUC measures the ability of the link detector to identify linked node pairs among all classes, while intra-class AUC measures its ability only in one class. As mentioned in Section 3.1, a successful link inference attack should have a high overall AUC as well as a high intra-class AUC. Without loss of generality, we set Class 1 as target class to evaluate performance of the link inference attack.

Models: We evaluate VertexSerum on three commonly used GNN structures: GCN [13], GraphSAGE [9], and GAT [24]. Deep Graph Library (DGL) is used for model implementation [26]. We construct a 3-layer MLP as the baseline link detector, with the first layer containing 64 hidden neurons which is also the initialization for the self-attention link detector. The self-attention detector is of a 16-head attention structure with an input dimension of 64. For initialization, we train MLP for 50 epochs with a learning rate of 0.001. We then fine-tune the self-attention detector with a learning rate of 0.0001, using the cross-entropy loss and Adam optimizer [12]. We run experiments 10 times and report the average and standard deviation of AUC scores.

Model	GCN		GAT		GraphSAGE	
Dataset	Citeseer	Cora	Citeseer	Cora	Citeseer	Cora
SLA + MLP[11]	0.914±0.008	0.874±0.018	0.969±0.002	0.845±0.011	0.972±0.002	0.854±0.009
SLA + ATTN	0.951±0.064	0.903±0.067	0.980±0.003	0.868±0.029	0.976±0.007	0.931±0.029
VS + MLP	0.892±0.006	0.912±0.065	0.913±0.005	0.856±0.017	0.949±0.007	0.859±0.027
VS + ATTN(*)	**0.978±0.033**	**0.927±0.023**	**0.997±0.002**	**0.924±0.022**	**0.994±0.006**	**0.957±0.007**
Dataset	AMZPhoto	AMZComputer	AMZPhoto	AMZComputer	AMZPhoto	AMZComputer
SLA + MLP[11]	0.813±0.015	0.826±0.018	0.881±0.007	0.820±0.046	0.873±0.015	0.883±0.004
SLA + ATTN	0.917±0.037	0.956±0.007	0.963±0.011	0.889±0.066	0.972±0.009	0.978±0.005
VS + MLP	0.780±0.007	0.849±0.009	0.917±0.006	0.852±0.033	0.873±0.032	0.898±0.004
VS + ATTN(*)	**0.939±0.018**	**0.962±0.011**	**0.990±0.008**	**0.919±0.031**	**0.987±0.006**	**0.985±0.006**

Table 2. Comparison of the average AUC with standard deviation for different attacks on the four datasets. The best results are highlighted in bold. (*) denotes our proposed method.

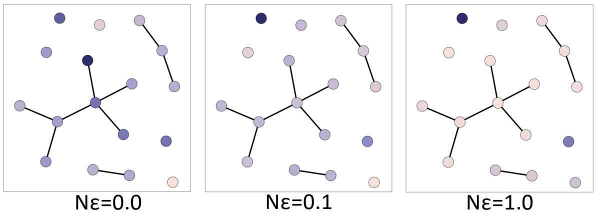


Figure 3. A visualization of nodes and edges belonging to the target class from the original ($N\epsilon = 0$) and poisoned ($N\epsilon > 0$) partial graphs. Node color represents the low-dimensional embedding of the GNN model's output, i.e., the node posteriors. **Color's similarity** indicates **posteriors' similarity**.

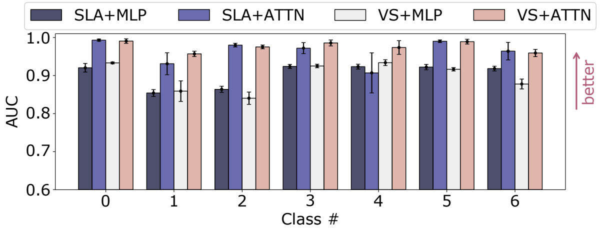


Figure 4. The AUC score along each target class. We take a case study on Cora dataset (7 classes in total) with GraphSAGE as the GNN model.

5.2. Graph Visualization

Figure 3 displays part of the poisoned graph of VertexSerum on a 3-layer GraphSAGE model trained on the Cora dataset with different distortions $N\epsilon$. By injecting poisoned samples into the partial graph while maintaining the topology, the PGD objective loss induces corresponding attraction and repulsion forces between nodes, resulting in increased attention to linked nodes. As the distortion increases from 0 to 1, the node colors shift to demonstrate attraction to linked nodes and repulsion to unlinked nodes.

5.3. Attack Performance

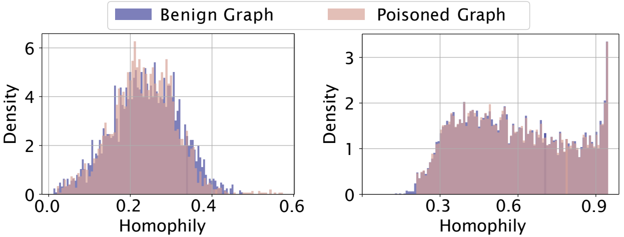
We evaluate the effectiveness of VertexSerum (VS), including both the poisoning method and the self-attention-based (ATTN) link detector. The prior stealing link attack (SLA) [11] serves as the SOTA method for us to compare, as it shares a similar threat model with our attack. SLA uses similarity features and an MLP-based link detector to attack a graph neural network, without poisoning. We compare the performance of different attack strategies and link detector structures, and report intra-class AUC scores in Table 2.

VertexSerum with the attention detector significantly improves the performance of link inference attacks for all datasets and GNN models. Compared to the method using SLA with MLP, our attack has an average improvement of 9.8% on AUC scores. Note that the self-attention-based link detector significantly improves the attack performance even without poisoning datasets (see the two rows of "SLA + ATTN " in Table 2). This is because the multi-head attention structure models the dependencies between elements in similarity features, better exposing the link existence during inference. On the other hand, using VertexSerum with MLP alone does not improve the detection performance on some datasets, such as Citeseer and AMZPhoto. From our consideration, VertexSerum enforces GNN to learn more about the connections between nodes, adding more hidden information to the similarity feature, for which MLPs lack the capability to capture. However, by combining VertexSerum with our proposed self-attention link detector, the poisoning works effectively towards increasing the link leakage.

We also demonstrate the intra-class AUC scores by varying the target class, taking the Cora dataset with the GraphSAGE model in Figure 4 as an example. We can draw the same conclusion as above on the link inference attack. Not only the self-attention detector can greatly outperform the MLP detector, but the poisoning also boosts link detection as well. Further, we demonstrate that VertexSerum can still preserve the highest effectiveness of link inference on over-

	Cora	Citeseer	AMZPhoto	AMZComputer
SLA+MLP [11]	0.907±0.001	0.987±0.001	0.919±0.020	0.913±0.043
SLA+ATTN	0.994±0.008	**0.995±0.001**	0.947±0.005	0.962±0.005
VS+MLP	0.945±0.003	0.978±0.013	0.946±0.010	0.900±0.055
VS+ATTN	**0.997±0.012**	0.994±0.001	**0.956±0.001**	**0.968±0.004**

Table 3. Comparison of the overall AUC scores for different tasks on GraphSAGE model, by inferring the link between node pairs from all classes.



Cora	GCN	GAT	GraphSAGE
Benign	0.891	0.880	0.867
Poisoned	0.882	0.872	0.887

AMZPhoto	GCN	GAT	GraphSAGE
Benign	0.937	0.935	0.933
Poisoned	0.921	0.930	0.922

Figure 5. Homophily analysis on graph poisoning. Cora and AMZPhoto are selected as the case study. The top histogram plots show the node homophily before and after the poisoning attack, where high coincidence on distribution means two graphs have high homophily. The lower tables demonstrate various model accuracies on the graphs before and after poisoning, showing that the accuracy is barely affected by the poisoning.

all classes. We show the overall AUC scores in Table 3, assuming the GNN model is based on GraphSAGE. Besides the elevated attack success, we can explicitly observe the overall AUC scores are higher than the intra-class AUC scores. This also affirms our observation discussed in Section 3.1 that evaluation on overall node pairs yields higher performance than that on intra-class node pairs.

5.4. Attack Stealthiness

We evaluate the stealthiness of VertexSerum from two perspectives: homophily unnoticeability and model accuracy. Homophily unnoticeability is an important metric for graph adversarial attacks and is defined as the node-centric homophily distribution shifting between the clean and poisoned graph being upper-bounded by a threshold, which ensures that the malicious nodes are not easily detectable by the database administrators [4]. We visualize the homophily distribution of the benign and poisoned graphs in Figure 5. It is clear that VertexSerum can effectively preserve the homophily while still conducting effective poisoning. The lower tables in Figure 5 present the model accuracy before and after poisoning, demonstrating that VertexSerum only introduces small accuracy degradation/improvement. Since from the vendor's perspective, the new accuracy is achieved after the re-training, thus, the trivial difference ensures stealthiness, i.e., the vendor will not stop using the poisoned graph due to poor performance.

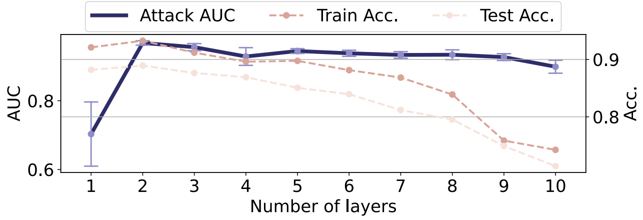


Figure 6. Performance of our attack on the GraphSAGE model with varying numbers of layers. The blue line represents the attack AUC scores, while the pink dashed lines indicate the training and testing accuracies.

	$\beta = 0.01$		$\beta = 0.1$		$\beta = 1$	
	$\lambda = 0.1$	$\lambda = 1$	$\lambda = 0.1$	$\lambda = 1$	$\lambda = 0.1$	$\lambda = 1$
$\alpha = 0.1$	0.914	0.942	0.931	0.943	0.954	0.953
$\alpha = 1$	0.952	**0.963**	0.954	0.953	0.946	0.945
$\alpha = 10$	0.949	0.947	0.950	0.949	0.925	0.926

Table 4. AUC scores of VertexSerum Attack on GraphSAGE for Cora Dataset with different regularization strengths.

5.5. Ablation Study

5.5.1 Influence of the Depth of GNN

We conduct an evaluation of our attack on the GraphSAGE model with varying numbers of layers (depth) in the GNN f_θ.The results are shown in Figure 6, where the blue line illustrates the attack AUC scores, while the pink dashed lines indicate the training and testing accuracy. As the number of layers increases, the GNN aggregates information from neighborhoods across multiple hops progressively, leading to overly similar output representations on linked nodes, known as over-smoothing [2].

When GNNs have only one layer, the attack is harder because of the lack of aggregated information between linked nodes. VertexSerum shows good performance when the number of layers is greater than 1, as more hops of neighbors are taken into consideration. Meanwhile, the model training and testing accuracy decreases as the number of layers increases, because of over-smoothing, where the representations of nodes become similar after multi-layer message passing. Consequently, the attack performance slightly drops, due to the underperformance of model accuracy. This is a concerning observation since the attack success rate is bound to the model accuracy. A well-performed model is also highly vulnerable to link inference attacks.

5.5.2 Impact of Different Loss Terms

In designing our PGD objective loss in Eq. 1, we consider a trade-off between the attraction loss, repulsion loss, and cross-entropy loss by controlling the corresponding regularization strength terms α, β, and λ. We compare the attack performance using different tuples of regularization weights in Table 4. We find that the optimal choice is

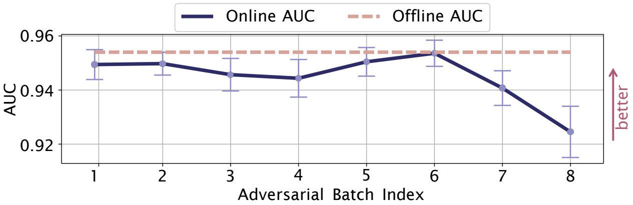


Figure 7. Performance of our attack on the GraphSAGE model under the online training setting. The blue line in the plot represents the attack AUC scores, and the x-axis represents different poisoning time during online training.

(α, β, λ)=$(1, 0.01, 1)$, where the repulsion weight is much smaller than the others. This is due to the imbalance between the number of linked and unlinked node pairs, which leads to a high repulsion loss, and this choice balances the effect of the repulsion loss and attraction loss.

5.6. Online Poisoning on GNNs

Graph neural networks in practice are not always trained offline, but multiple contributors may provide data at different times for online training. This is particularly relevant in scenarios such as recommendation systems, where models are frequently updated with incoming user behavior data. In this section, we investigate a training scenario where the vendor's model is trained batch-by-batch as the data arrives. We divide the dataset into eight batches, each representing a different contributor. We select one of the contributors as the adversary and use VertexSerum to poison the corresponding partial graph. The model is updated in order as the contributors arrive, and we evaluate the attack performance when the adversarial contributor arrives at different times.

Figure 7 presents the attack AUC when the adversarial batch arrives at different times during online training. We observe that poisoning the early batches is more effective than poisoning the last batch. This is likely because the early batches have a long-term effect on fitting the online model, while the poisoning data in the last round is only fitted during the last update. Further, the poisoning attack on offline training yields better results. Since the poisoning exists throughout the offline training, the model fitting on the benign batches is also consistent throughout the training, akin to poisoning at an early time. Overall, VertexSerum is effective for both online and offline training on GNNs.

5.7. Transferability in the Black-Box Setting

In previous evaluations, we assume that the attacker has prior knowledge of the vendor model's architecture and training process, which is a gray-box setting. In this section, we extend our evaluation to the black-box setting, where the attacker has no knowledge of the victim model's architecture and configuration. We investigate the transferability of VertexSerum, where the attacker trains the subgraph using a different model from the vendor model. For instance, the attacker may train the subgraph using GAT when the vendor model is trained using GraphSAGE. Figure 8 shows the results under the black-box setting. We find that even without knowledge of the vendor model structure, the attacker can still achieve high performance using VertexSerum. Interestingly, the attacker achieves the highest AUC scores when using GAT as the shadow model to generate the poison example. We hypothesize that GAT has a higher generalizability in estimating the real boundary of the vendor model, making the poison samples from GAT more effective.

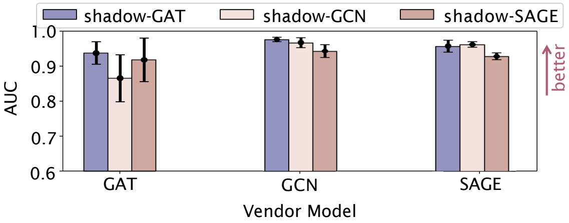


Figure 8. The attack performance when the vendor model is unknown and trained on Cora Dataset, where the attacker uses arbitrary GNN structures to train the shadow model.

6. Defense

There are two potential directions to defend against the VertexSerum attack. The first approach is to blur the perturbation. Our poisoning samples are similar to adversarial samples, which are clean features with small added noise. Thus, it is possible to slightly change the training samples through preprocessing methods such as denoising or augmentation, without harming the model accuracy. The second approach is to increase the GNN's robustness against the link stealthy attack. One way to achieve this is to build GNNs with certified robustness using differential privacy [8]. Alternatively, the vendor can train the GNN with an appropriate depth to avoid over-smoothing or over-fitting.

7. Conclusions

In this paper, we investigate the vulnerability of graph neural networks to privacy leakage amplified by data poisoning. We propose VertexSerum, with data poisoning and self-attention link detector, a link inference attack with significantly better attack performance on intra-class nodes. We conduct extensive evaluations on different attack settings, including gray-box, offline training, online training, and black-box. As graph neural networks become increasingly popular, our findings pose a new challenge to confidentiality of the structural datasets using GNNs. The work serves as a cautionary note to model vendors, informing them of possible privacy exposure of their training datasets and calling for more follow-on work to build robust GNNs against such privacy-breaching attacks.

References

[1] Chaochao Chen, Jun Zhou, Longfei Zheng, Huiwen Wu, Lingjuan Lyu, Jia Wu, Bingzhe Wu, Ziqi Liu, Li Wang, and Xiaolin Zheng. Vertically federated graph neural network for privacy-preserving node classification. *arXiv preprint arXiv:2005.11903*, 2020.

[2] Deli Chen, Yankai Lin, Wei Li, Peng Li, Jie Zhou, and Xu Sun. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 3438–3445, 2020.

[3] Yufei Chen, Chao Shen, Yun Shen, Cong Wang, and Yang Zhang. Amplifying membership exposure via data poisoning. *arXiv preprint arXiv:2211.00463*, 2022.

[4] Yongqiang Chen, Han Yang, Yonggang Zhang, Kaili Ma, Tongliang Liu, Bo Han, and James Cheng. Understanding and improving graph injection attack by promoting unnoticeability. *arXiv preprint arXiv:2202.08057*, 2022.

[5] Yuxin Chen, Ziqi Zhang, Chunfeng Yuan, Bing Li, Ying Deng, and Weiming Hu. Channel-wise topology refinement graph convolution for skeleton-based action recognition. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 13359–13368, 2021.

[6] Edward Choi, Mohammad Taha Bahadori, Le Song, Walter F Stewart, and Jimeng Sun. Gram: graph-based attention model for healthcare representation learning. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 787–795, 2017.

[7] Enyan Dai, Tianxiang Zhao, Huaisheng Zhu, Junjie Xu, Zhimeng Guo, Hui Liu, Jiliang Tang, and Suhang Wang. A comprehensive survey on trustworthy graph neural networks: Privacy, robustness, fairness, and explainability. *arXiv preprint arXiv:2204.08570*, 2022.

[8] Tianchong Gao and Feng Li. Protecting social network with differential privacy under novel graph model. *IEEE Access*, 8:185276–185289, 2020.

[9] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

[10] Chaoyang He, Keshav Balasubramanian, Emir Ceyani, Carl Yang, Han Xie, Lichao Sun, Lifang He, Liangwei Yang, Philip S Yu, Yu Rong, et al. Fedgraphnn: A federated learning system and benchmark for graph neural networks. *arXiv preprint arXiv:2104.07145*, 2021.

[11] Xinlei He, Jinyuan Jia, Michael Backes, Neil Zhenqiang Gong, and Yang Zhang. Stealing links from graph neural networks. In *USENIX Security Symposium*, pages 2669–2686, 2021.

[12] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[13] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[14] Julian McAuley, Christopher Targett, Qinfeng Shi, and Anton Van Den Hengel. Image-based recommendations on styles and substitutes. In *Proceedings of the 38th international ACM SIGIR conference on research and development in information retrieval*, pages 43–52, 2015.

[15] Iyiola E Olatunji, Wolfgang Nejdl, and Megha Khosla. Membership inference attack on graph neural networks. In *2021 Third IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA)*, pages 11–20. IEEE, 2021.

[16] Ryan Razani, Ran Cheng, Enxu Li, Ehsan Taghavi, Yuan Ren, and Liu Bingbing. Gp-s3net: Graph-based panoptic sparse semantic segmentation network. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 16076–16085, 2021.

[17] Ali Shafahi, Mahyar Najibi, Mohammad Amin Ghiasi, Zheng Xu, John Dickerson, Christoph Studer, Larry S Davis, Gavin Taylor, and Tom Goldstein. Adversarial training for free! *Advances in Neural Information Processing Systems*, 32, 2019.

[18] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *2017 IEEE symposium on security and privacy (SP)*, pages 3–18. IEEE, 2017.

[19] Jianyong Sun, Wei Zheng, Qingfu Zhang, and Zongben Xu. Graph neural network encoding for community detection in attribute networks. *IEEE Transactions on Cybernetics*, 52(8):7791–7804, 2021.

[20] Yiwei Sun, Suhang Wang, Xianfeng Tang, Tsung-Yu Hsieh, and Vasant Honavar. Adversarial attacks on graph neural networks via node injections: A hierarchical reinforcement learning approach. In *Proceedings of the Web Conference 2020*, pages 673–683, 2020.

[21] Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. Ensemble adversarial training: Attacks and defenses. *arXiv preprint arXiv:1705.07204*, 2017.

[22] Florian Tramèr, Reza Shokri, Ayrton San Joaquin, Hoang Le, Matthew Jagielski, Sanghyun Hong, and Nicholas Carlini. Truth serum: Poisoning machine learning models to reveal their secrets. *arXiv preprint arXiv:2204.00032*, 2022.

[23] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[24] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, Yoshua Bengio, et al. Graph attention networks. *stat*, 1050(20):10–48550, 2017.

[25] Daixin Wang, Jianbin Lin, Peng Cui, Quanhui Jia, Zhen Wang, Yanming Fang, Quan Yu, Jun Zhou, Shuang Yang, and Yuan Qi. A semi-supervised graph attentive network for financial fraud detection. In *2019 IEEE International Conference on Data Mining (ICDM)*, pages 598–607. IEEE, 2019.

[26] Minjie Yu Wang. Deep graph library: Towards efficient and scalable deep learning on graphs. In *ICLR workshop on representation learning on graphs and manifolds*, 2019.

[27] Chuhan Wu, Fangzhao Wu, Yang Cao, Yongfeng Huang, and Xing Xie. Fedgnn: Federated graph neural network

for privacy-preserving recommendation. *arXiv preprint arXiv:2102.04925*, 2021.

[28] Fan Wu, Yunhui Long, Ce Zhang, and Bo Li. Linkteller: Recovering private edges from graph neural networks via influence analysis. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 2005–2024. IEEE, 2022.

[29] Yifan Xing, Tong He, Tianjun Xiao, Yongxin Wang, Yuanjun Xiong, Wei Xia, David Wipf, Zheng Zhang, and Stefano Soatto. Learning hierarchical graph neural networks for image clustering. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3467–3477, 2021.

[30] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI open*, 1:57–81, 2020.

[31] Daniel Zügner, Oliver Borchert, Amir Akbarnejad, and Stephan Günnemann. Adversarial attacks on graph neural networks: Perturbations and their patterns. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 14(5):1–31, 2020.

[32] Daniel Zügner and Stephan Günnemann. Adversarial attacks on graph neural networks via meta learning. *ArXiv*, abs/1902.08412, 2019.