Persistent Clean-label Backdoor on Graph-based Semi-supervised Cybercrime Detection

Anonymous Authors

Anonymous Institute

Abstract. Cybercrime, which involves the use of tactics such as hacking, malware attacks, identity theft, ransomware, and online scams, has emerged as a major concern for public security management recently. To combat massive cybercrime and conduct a clean Internet environment, graph-based semi-supervised cybercrime detection (GSCD) has gained increasing popularity recently for it can model complex relationships between network objects and provide node-level behavior predictions. However, in this paper, we present a novel threat on GSCD, named persistent clean-label backdoor attack (PerCBA), which may be utilized by attackers to escape cybercrime detection successfully. The PerCBA patches node features of unmarked training data with adversarially-perturbed triggers to enforce the well-trained GSCD model to misclassify trigger-embedded crime data as the premeditated result. Extensive experiments on multiple detection models and open-source datasets reveal that the PerCBA exhibits effective escape performance (up to 96.25%) and evasiveness.

Keywords: Cybercrime detection · Semi-supervised graph learning · Backdoor attacks · Entity prediction · Clean-label · Data poisoning.

1 Introduction

Cybercrime detection refers to the process of identifying criminal activities perpetrated through or facilitated by the utilization of internet, encompassing, but not limited to, hacking, phishing, online fraud, and other related offenses. To mitigate cybercrime, Graph-based semi-supervised cybercrime detection (GSCD) is broadly implemented since it is capable of efficiently processing data with interconnected relationships and predict potential adversaries or attack objectives. Specifically, GSCD first constructs a graph where nodes represent various net objects (e.g., victims, adversaries, or tools), while edges denote the relationships that exist between them. Subsequently, the graph is labeled partially with information regarding criminal activity or anomaly. By leveraging the graph, GSCD could predict the probability of criminal activity and anomaly for unmarked nodes, relying on their associations with the labeled nodes present in the graph.

Despite the effectiveness of GSCD, in this study, we identify it is vulnerable to backdoor attacks and present the corresponding attack methodology: PerCBA (**Per**sistent **C**lean-label **B**ackdoor **A**ttack). The proposed PerCBA entails

inserting perturbed triggers into small-scale unlabeled training data (poisoning), and utilizing these samples to train the detection model alongside other benign data. When the training is complete, trigger-embedded cybercrime data will be misclassified into the target result by the model when testing, namely, allowing criminal data to escape detection.

Notably, PerCBA does not alter any label information, hence it is a clean-label attack [13]. Meanwhile, the proposed method demonstrates persistence by requiring only once poisoning to achieve considerable effects, in contrast to conventional backdoors that necessitate multiple times of poisoning.

The main contributions of this work can be summarized as follows:

- We proposed a persistent clean-label backdoor attack (PerCBA) scheme for GSCD model, which focuses on poisoning unlabeled nodes by inserting impermeable perturbed triggers on the node features. To the best of our knowledge, this is the first clean-label backdoor attack for graph semi-supervised cybercrime classification tasks.
- A trigger-agnostic feature perturbation generator with adjustable budgets is also proposed to generate different perturbations for targeted classes and non-targeted classes, respectively. Meanwhile, to improve the persistence of the proposed PerCBA in GSCD, a hyper-parameter regulation strategy is proposed to optimize the distribution of perturbation budgets.
- Detailed experiments based on five different real-world datasets are conducted to evaluate the performances of PerCBA, and it performs high escape detectioon rate (up to 96.25%) without distinct model accuracy degradation on clean data, while the poison rate is lower than 4%.

The rest of this paper is organized as follows: Section II summarizes the related works; Section III provides the details of the proposed method, PerCBA; Section IV introduces the experimental settings and results; Section V concludes this paper and discusses the future trend.

2 Related Work

2.1 Graph-based Semi-supervised Cybercrime Detection

Recent studies about GSCD have primarily concentrated on the utilization of graph neural networks (GNNs) for node embedding updates and classification. Of these, the graph convolution network (GCN) has been extensively adopted.

GCN is a basic implementation model of GSCD developed to address the issues of self-feature aggregation, feature normalization, and gradient explosion [6]. Given a graph $\boldsymbol{G} = (\boldsymbol{A}, \boldsymbol{X})$ where $\boldsymbol{A}$ is the adjacent matrix and $\boldsymbol{X}$ is the corresponding feature matrix, the result of node classification is calculated by

$$\boldsymbol{Z} = f(\boldsymbol{A}, \boldsymbol{X}) = softmax(h(\boldsymbol{A}, \boldsymbol{X})), \tag{1}$$

where h is the final output of aggregation iterations (also called node embeddings), which is shown as follow:

$$\boldsymbol{H}^{(s)} = Activation(\tilde{\boldsymbol{D}}^{-\frac{1}{2}}\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-\frac{1}{2}}\boldsymbol{H}^{(s-1)}\boldsymbol{W}^{(s-1)}), \tag{2}$$

where $\tilde{\boldsymbol{A}}$ is $\boldsymbol{A}$ plus the identity matrix, and the initial state of $\boldsymbol{H}$ is the feature matrix $\boldsymbol{X}$. With regard to loss function, only a fraction of the labeled data are required to compute it and update all the model parameters.

On the basis of the core idea of GCN, various variants are proposed to make up for the shortcomings of GCN. To implement GSCD on large-scale graph, a sampling mechanism-based method (GraphSAGE) is introduced in [4] to optimize GCN learning, and it brings good performance improvement. GCN does not take into account the importance of neighbor nodes during training, so [10] proposes a method called GAT to aggregate node embedding via attention calculation. [11] proposes a data augmentation method, GraphMix, for training fully connected networks jointly with GNNs through parameter sharing and regularization, and it can efficiently improve the prediction performance of GSCD. Traditional GSCD methods suffer from over-smoothing and weak-generalization, and hence [3] employs random propagation and consistency regularization and designs a method called GRAND to solve them.

2.2 Backdoor Attacks on GNN

Backdoor attacks on GNN aim to make poisoned data predicted as targeted class.

Poisoning training data is the mainly adopted method to implant backdoor in GNN. Given a graph $\boldsymbol{G} = (\boldsymbol{A}, \boldsymbol{X})$, adversary will select attack targets (nodes) $\boldsymbol{G_t}$ from $\boldsymbol{G}$ to insert the designed trigger Δ into their feature or topology vectors in $\boldsymbol{X}$ or $\boldsymbol{A}$, and change their labels into specified target class t. Subsequently, these modified (poisoned) target nodes $\boldsymbol{G_t}$ and other clean data in $\boldsymbol{G}$ will be employed to train the model. Once the training process is completed, the model becomes backdoored. And when input test node data x^{δ} with the trigger Δ, targeted label t will be predicted by the backdoored model. But for clean data x, the model can make the right prediction [12]. Direct trigger insertion is obvious and likely to be detected, and [14] proposes using less important features as triggers, which improves concealment. GNN models transmit information to nearby nodes through the aggregation process, and to spread poisoned information, [1] utilized poisoned neighbor nodes to implant backdoors in the model during aggregation training.

Although there have been various GNN backdoor studies, to the best of our knowledge, backdoors for GSCD have not been investigated.

3 Proposed PerCBA Scheme

The proposed PerCBA scheme is illustrated Figure 1 in detail. The adversary in PerCBA trains a backdoored GSCD model through three primary steps: 1)

attack target selection, unlabelled nodes are selected as the attack target by measuring their degree and eigenvector centrality; 2) poisoned data generation, both the trigger and perturbation are pasted on selected nodes to generate poisoned training data. Therein, the trigger is specified by the adversary, while the perturbation is generated using a generator with adjustable budgets; 3) backdoor training and testing, the adversary utilizes poisoned training nodes to train the GSCD model, resulting in a backdoored model that produces predetermined results on a malicious sample during the testing phase.

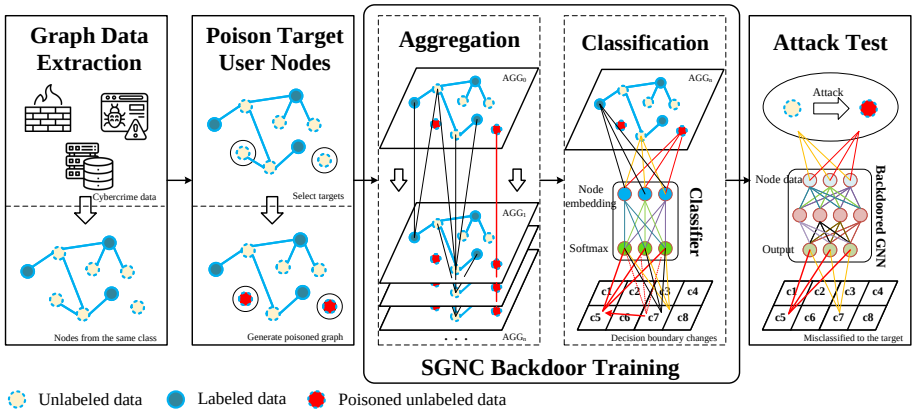


Fig. 1. Illustration for the proposed PerCBA attack scheme. Unlabeled nodes are poisoned first via perturbed trigger without label changing. Then, the GSCD model will be backdoored during the training process, and when poisoned samples are input into the backdoored model, the pre-defined target label will be predicted.

3.1 Attack Target Selection

To reduce the attack impact on the original dataset, we utilize independent nodes (without any links to others) to poison. Independent nodes will not spread or receive information from other nodes during aggregation, hence, if these nodes are poisoned, they will not affect other clean nodes or be affected.

Given the graph dataset $\boldsymbol{G}$ and the poison rate γ (the ratio of poisoned samples to all), we randomly select independent nodes from unlabeled data to form the attack targets $\boldsymbol{G_t}$. However, in some cases, there may not be enough independent nodes available to choose, and therefore appropriate strategy should be made to solve the selection of the remaining nodes. In this study, we consider using degree centrality C_D and eigenvector centrality C_E to solve this problem. C_D demonstrates the node connectivity in network [16], while C_E indicates node importance via its neighbor influence [9]. Supposing a node has low C_D and C_E, it's comparatively independent and has less influential neighbors, and so attacking it will have less impact on whole dataset compared with other normal

nodes. We rank the nodes according to C_D and C_E and pick the ones with the lowest values as the remaining attack targets. The ranking standard is given by

$$C = \alpha C_D + (1 - \alpha) C_E, \tag{3}$$

where α is the weight and is set to 0.5 in experiment.

The whole attack target selection process assures nodes in the attack target dataset $\boldsymbol{G_t}$ are with very low degree and eigenvector centrality (hence it makes the attack own better concealment). Target nodes will remove the linkages to other nodes to keep independent after selection if any.

3.2 Poisoned Data Generation

Pasting Trigger With the attack target $\boldsymbol{G_t}$ decided, trigger will be inserted into training data through feature adversarial perturbation. In real scenarios, nodes typically possess features (e.g., hacker profile, attack log and behavior data), and accordingly, adversary can capitalize on these data to implement backdoor.

For the attack target dataset $\boldsymbol{G_t} = (\boldsymbol{A_t}, \boldsymbol{X_t})$ and a specific sample $u_i = (a_i, x_i) \in \boldsymbol{G_t}$, we first insert raw trigger Δ into the feature vector x_i and then add adversarial perturbation δ.

For k-dimension feature vector x_i, we uniformly pick m dimensions and set the original feature value ρ_i as 1 to create trigger Δ:

$$\begin{aligned} u_i^{\delta} &= u_i + \Delta = (a_i, x_i + \Delta) \\ s.t.\ \ x_i + \Delta &= (\rho_1, \rho_2, ..., 1_1, ..., 1_2, ..., 1_m, ..., \rho_k), \end{aligned} \tag{4}$$

where u_i^{δ} is the sample u_i crafted by inserting a trigger in it. Actually, the choice of trigger dimensions can be arbitrary (i.e. trigger-agnostic), but uniformly picking trigger dimensions avoids the risk that intensive trigger dimensions will lead to the prediction to other specific unexpected classes, for the reason that some kinds of nodes may tend to have denser feature distribution. The experimental part will further discuss the effect of different triggers on attack results, but in general, there is not much difference.

The following operation is to generate perturbed trigger, namely to add related adversarial perturbation into the trigger. This is to cheat the aggregation process during learning so that the decision boundary of trigger-embedded nodes can change, and poisoned nodes will be classified as target category in test. The Schematic representation of decision boundary change is shown in Figure 2 and the corresponding perturbation generator can be expressed as

$$\begin{aligned} \sigma_i &= \arg\min_{\sigma} \|\varsigma(u_i^{\delta})\|_2 \\ s.t.\ \ & B(u_i^{\delta} + \sigma_i) = c_t, \end{aligned} \tag{5}$$

where σ_i and $\varsigma(*)$ are the perturbation and its generation function, and $B \in \mathbb{R}^c$ denotes the decision boundary changing process that poisoned node u_i^{δ} whose

original right label v has moved to the side of the target class t. This process can be regarded as an adversarial problem [2]. For GNN, if we consider each output of the hidden layer as an extraction of feature abstraction, then the problem is converted to optimizing the feature abstraction distance between the poisoned data and the target class sample, which is given by

$$\begin{aligned} \sigma_i = \arg\min_{\sigma} \|l(u_i^\delta + \sigma, \theta) - l(s_t, \theta)\|_2 \\ s.t. \quad \|u_i^\delta + \sigma\|_2 < \epsilon, \end{aligned} \tag{6}$$

where l is the feature abstraction function that has the same structure as the target GNN model but deletes the final output layer, θ indicates the model parameters and s_t implies the sample from the target class set. Based on u_i^δ, θ and s_t, optimal σ need to be found to satisfy Eq. 6.

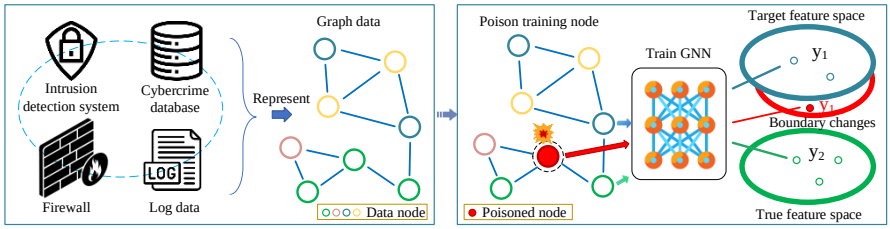


Fig. 2. Illustration for the boundary change process: the graph structure is derived from data related to cybercrime activity data (e.g., AZSecure, ADFA IDS, and Amazon-Fraud), followed by the addition of perturbated trigger to the target nodes. Subsequently, through training, the decision boundary is changed to make targeted prediction.

To address this adversarial perturbation problem and calculate σ, projected gradient descend (PGD) method will be employed. PGD is a multiple step data update method to apply imperceptible grading descent perturbation in input data and project updated data into a constrained space [8]. Combining Eq. 6, the poisoned target data $\hat{u}_i^\delta$ with perturbed trigger is calculated by

$$[\hat{u}_i^\delta]^{(s)} = \Pi_p([\hat{u}_i^\delta]^{(s-1)} - \mu\tau^{(s)}), \tag{7}$$

where $[\hat{u}_i^\delta]^{(s)}$ is the poisoned sample in iteration s (initial state is u_i^δ), μ is weight parameter (it can be seen as the learning rate), τ is gradient from Eq. 6 and Π_p is the function of projecting data over a restricted ball range, which can shown as

$$\Pi_p(\hat{u}_i^\delta) = \arg\min_{u \in \Gamma} \|u - \hat{u}_i^\delta\|_2, \tag{8}$$

where Γ is the constrained ball space around u_i. In addition, τ will not be wholly added to u_i^δ, and we randomly choose 20% features dimensions (perturbation

budget) around the 1st non-zero feature dimension in u_i to add τ. In experiment, perturbation budget will be changed to see the affect.

Through accomplishing the iterations, the perturbed trigger is inserted and the poisoned sample is generated.

Hyper-parameter Regulation Strategy for Perturbation Adding In the poisoned data, perturbed trigger is implanted into unlabeled target data. But for other clean unlabeled data, they may be influenced by the decision boundary change. To make the model more robust to the perturbation for clean unlabeled data and reduce the impact on model performance, we propose a new perturbation-adding strategy.

Inspired by the Mixup algorithm by [15], our strategy attempts to add two kinds of perturbations into all unlabeled data to improve the robustness of the model against adversarial perturbation in clean unlabeled data and enhance the generalization ability.

For clean unlabeled data u_i, slight perturbation will be added to make the model more adaptive to perturbation and reduce its influence on clean training samples, which is given by

$$\tilde{u_i} = u_i + k_1 \sigma_i, \tag{9}$$

where k_1 is a small weight, which is taken from complementary cumulative distribution function (CCDF) $F(x)$ of standard normal distribution to control the affect from the slight perturbation.

For trigger-embedded attack target u_i^δ, strong perturbation is added:

$$\tilde{u}_i^\delta = u_i^\delta + (1 - k_1)\sigma_i, \tag{10}$$

where σ_i is the raw perturbation calculated by Eq. 6. The purpose of this operation is to weaken the original perturbation.

On the basis of the poison data generation process, we give its corresponding algorithm, which is depicted in Algorithm 1.

3.3 Backdoor Training and Testing

Based on the poison data generated in the previous steps, both kinds of unlabeled data in Eq. 9 and Eq. 10 will be sent to the victim model with other clean training data to perform learning. When the training is completed, the detection model gets backdoored. If we input a poisoned node into the backdoored model, it will make the prediction as the target class rather than true class (i.e. escape detection for suspect).

4 Experiment and Discussion

In this section, the performance of the proposed PerCBA method will be evaluated and analyzed. We first give the settings and the evaluation metrics of our

Algorithm 1: Poisoned data generation

Input: Unlabeled training graph data G_u, GNN parameters θ, trigger Δ, poison rate γ.
Output: Poisoned unlabeled training data G_u^*.

Determine outlier attack targets G_t via centrality and poison rate γ;
for u_i *in* G_u **do**
  if u_i *in* G_t **then**
    Implant raw trigger Δ into u_i: $u_i^\delta \leftarrow u_i + \Delta$;
    Calculate perturbation σ based on u_i^δ and θ via Eq. 6;
    Add perturbation σ to u_i^δ: $\hat{u}_i^\delta \leftarrow u_i^\delta(1 - k_1)\sigma$;
  else
    Calculate perturbation σ based on u_i and θ via Eq. 6;
    Add perturbation σ to u_i: $\hat{u}_i \leftarrow u_i + k_1\sigma$;
  end
end
return G_u^*;

experiment. Subsequently, experiment results are displayed. As mentioned before, PerCBA is the first clean-label backdoor attack for GSCD-based detection tasks, and thus, we mainly conduct ablation experiments on PerCBA under various settings. Finally, we will make discussions that why PerCBA can achieve persistent attack through small-scale poisoning.

4.1 Experiment Settings and Evaluation Metrics

Target Models To test the effectiveness of the attack under different models, three widely adopted GNN models are selected as victims: GCN, GAT (which introduces the attention mechanism into GCN) and GraphSAGE (which utilizes sampling mechanism to optimize GCN).

Datasets We employ five frequently used real-world datasets to measure the attack performance, and dataset statistics are shown in Table 1 in detail.

Table 1. Dataset information

Dataset	Node	Edge	Class	Feature	Label Rate
A	2,708	5,429	7	1,433	0.052
B	3,327	4,732	6	3,703	0.036
C	3,943	3,815	3	500	0.040
D	17,716	105,734	4	1,639	0.008
E	34,493	495,924	5	8,415	0.004

Attack Setup To achieve better performance, the target detection model will first be pre-trained and then exploited to generate unlabeled attack data using the proposed PerCBA approach (the poisoning process is displayed in algorithm 1). After that, target model is fine-tuned and gets backdoored.

Evaluation Metrics We use three common metrics, escape rate or attack success rate (ASR, the ratio of the successful node attack trials S to all poisoned test nodes T_{poison}), clean data accuracy (Acc or normal performance, the rate of the correctly classified clean test nodes Y_c to all clean test nodes T_{clean}) and poison rate (or attack implementation rate, the ratio of the poisoned training nodes L_{poison} to all training data L_{train}) to analyze the attack effectiveness on the backdoored model. In addition, clean model performance on original data is investigated by using Acc and misclassification rate (MR, the ratio of the clean test nodes misclassified to the target class Y_t, to all clean test nodes T_{clean}) to be compared with the backdoored model. The calculation equations of the aforementioned metrics are shown as follows:

$$\text{ASR} = \frac{S}{T_{poison}}, \tag{11}$$

$$\text{Acc} = \frac{Y_c}{T_{clean}}, \tag{12}$$

$$\text{Poison Rate} = \frac{L_{poison}}{L_{train}}, \tag{13}$$

$$\text{MR} = \frac{Y_t}{T_{clean}}. \tag{14}$$

To evaluate the attack evasiveness, we apply average degree centrality difference (ADD), average eigenvector centrality change (AEC) and average feature value change (AFD) to analyze the data differences between the original nodes and the poisoned nodes.

4.2 Experiment Results

Results on Different Datasets We first evaluate our attack on target models from dataset A to E. For each attack, poisoned node amount is set to 100, and trigger feature dimension number m is set to 60. Each attack will be tested 10 times to calculate the average result. The experiment results are displayed in Table II.

For GCN, the attacks on all datasets achieve considerable ASR (maxima 90.48%) while retaining the accuracy of clean data classification (Acc drops within 4%) and low poison rate (within 4%). In terms of attack evasiveness, all the attacked datasets have miniature values in ADD (minima 4.7×10^{-10} and maxima 0.118‰), AEC (minima 1.2×10^{-8} and maxima 0.05‰) and AFD (minima 0.0005‰ and maxima 0.9‰), which are all very subtle and imperceptible little changes that are difficult for defenders to detect.

Table 2. Comparison results among GCN, GAT and GraphSAGE.

Models	Dataset	Poison Rate	ASR	Original Acc	Acc	MR	ADD (‰)	AEC (‰)	AFD (‰)
GCN	Cora	3.6	60.20	73.78	70.77	**3.4**	0.058	0.021	0.9
	Citeseer	3.0	34.08	66.25	65.85	7.1	0.118	0.050	0.09
	Pubmed	2.5	71.01	72.14	69.86	9.1	0.0006	0.0008	0.24
	DBLP	0.5	89.62	78.54	76.22	5.8	$\mathbf{4.7\times10^{-7}}$	$\mathbf{1.2\times10^{-5}}$	0.20
	Physics	**0.2**	**90.48**	**91.15**	**90.21**	3.7	0.0007	0.0004	**0.0005**
GAT	Cora	3.6	41.51	73.01	68.44	3.1	0.038	0.033	0.6
	Citeseer	3.0	47.00	57.56	54.66	5.3	0.245	0.062	**0.02**
	Pubmed	2.5	43.08	61.09	62.46	7.4	0.0013	0.0011	0.10
	DBLP	0.5	**86.52**	79.90	78.15	3.2	0.0004	0.0006	0.12
	Physics	**0.2**	49.92	**88.44**	**88.25**	**2.9**	$\mathbf{5.9\times10^{-5}}$	**0.0005**	0.39
GraphSAGE	Cora	3.6	89.60	68.48	68.21	2.9	0.015	0.046	0.47
	Citeseer	3.0	70.34	68.55	66.00	5.7	0.085	0.164	0.11
	Pubmed	2.5	91.85	69.15	72.67	5.1	0.0027	0.0063	**0.09**
	DBLP	0.5	**96.25**	77.88	78.06	4.4	0.0008	0.0014	0.33
	Physics	**0.2**	83.45	**89.49**	**88.46**	**2.3**	**0.0003**	**0.0006**	0.47

For GAT, the average ASR of PerCBA reaches about 53% (maxima 86.52%), and the Acc drops within 5%. Also, like the evasiveness performances in GCN, the test results in ADD, AEC and AFD all keep very miniature values.

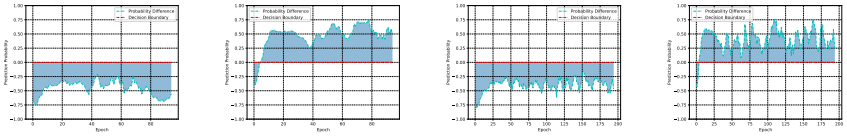
For GraphSAGE, PerCBA has performed well in terms of attack success rates and has reached a maximum of 96.25% (average ASR is around 86%). For Acc and evasiveness performances, we found similar trends to those in GCN and GAT, which show tiny changes across different datasets.

To summarize, PerCBA has good concealment, which can be seen in the slight Acc drops, ADD, AEC and AFD in the results. Meanwhile, it requires few attack targets (shown via low poison rate), which is more covert and applicable.

Change of Decision Boundary To better illustrate the decision boundary change process during training, we randomly choose an infected node from dataset A to conduct an experiment on GCN and observe the variation between the predicted probability of the target label and its true label. Training will be conducted for two different epochs (100 and 200) to more thoroughly examine the boundary change, and the result is depicted in Figure 3.

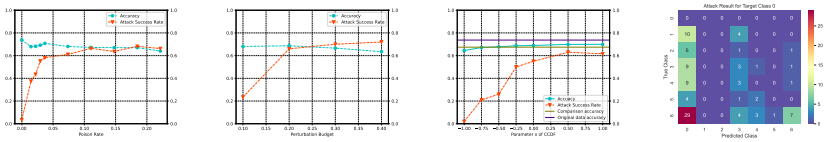
As presented in the figure, under both two epoch amounts, the probability difference is concentrated in the negative areas (boundary is closer to right label) in pre-training process, while it will gradually shift to the positive areas (boundary is closer to target label) throughout fine-tuning.

Affections of Hyperparameters We also inspect the correlations between attack performances and three essential hyperparameters: poison rate, perturbation budget and CCDF parameter, and carry out the test on GCN with dataset A for 10 times to collect average result in ASR. Figure 4(a) to Figure 4(c) presents the findings.



(a) Pre-training of 100 epochs (b) Fine-tuning of 100 epochs (c) Pre-training of 200 epochs (d) Fine-tuning of 200 epochs

Fig. 3. Illustration for decision boundary change during training. For 100 epochs, (a) depicts how the decision boundary tends to go near to the correct label (negative area) in the pre-training while moving closer to the target (positive area) during the fine-tuning process in (b). In another set of results, including (c) and (d), they have a similar trend under 200 epochs.



(a) Poison rate (b) Perturbation budget (c) Parameter x of CCDF (d) Class information

Fig. 4. Affections of different hyperparameters and class information about attack result in dataset A.

- As can be seen from Figure 4(a), accuracy decreases from 74% to 64% as poison rate increases, while attack success rate rises from 0 to about 60% and fluctuates around about 66%. Note that we only consider maximum poison rate up to about 20%, since higher rate is not practical in real attack scenarios.
- Regarding the impact of Perturbation budget in Figure 4(b), as it increased from 10% to 40%, accuracy decreases from 69% to 63% and attack success rate increases from 24% to 72%.
- From Figure 4(c), as the x of CCDF increases, the influence causes ASR rises from 2% to 61%, while ACC increases slightly from 64% to 69%, with the original data Acc being 73% and comparison accuracy of only perturbing targets nodes being 67%.

Affections of Data Categories The initial class of poisoned nodes could potentially impact the attack outcome due to the attributes and interdependencies among diverse node types. To further analyze whether there is an effect, class information about the attack result for dataset A on GCN is viewed in Figure 4(d). Additionally, we attack nodes of a certain class with the same poison rate in dataset A to see the class influence on the attack. Also, we set different attack target classes to see the method performance. Attack result is shown in

Table 3 and Table 4. Each class will be tested for 10 times to compute average performance.

Table 3. Attack results for different poisoned data classes

Class Change	Accuracy	Attack Success Rate
1→0	67.39	71.36
2→0	68.47	62.03
3→0	67.64	64.88
4→0	68.05	52.27
5→0	66.45	73.79
6→0	68.17	44.37

Table 4. Attack results for various target class

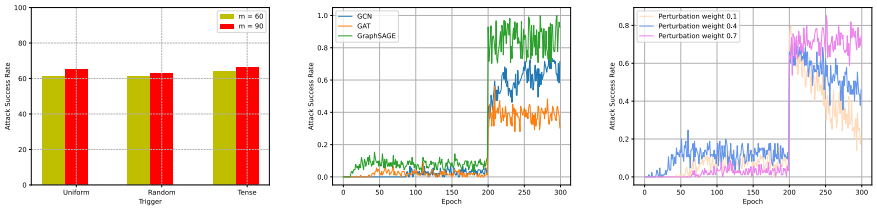
Target Class	Accuracy	Attack Success Rate
0	69.52	60.38
1	68.14	65.10
2	69.77	39.43
3	68.14	79.05
4	69.40	61.27
5	67.33	66.09
6	68.14	51.54

From Figure 4(d) above, we can see that class 6 has the largest number of nodes attacked successfully and failed, while class 1 has the 2nd largest number of successful attacks. For attacks in different specifically poisoned classes, the result is provided in Table 3. As seen from the results, there is not much difference in accuracies, but class 1 and 5 have relatively high ASR, while class 4 and 6 have relatively low ASR. Under the attack conditions of different target classes, the overall experimental results in Table 4 reveal high average accuracy, and the ASR can reach 79% at the highest in class 3 but only 39% at the lowest in class 2.

Generally, the setting of the target class or the selection of the poisoned data class has limited influence on the final attack performances.

Affections of Different Triggers As mentioned in Section III, the proposed PerCBA is trigger-agnostic, and hence we evaluate the method's performance under different triggers via GCN and dataset A. Besides picking uniformly distributed feature dimensions as triggers, both random feature dimensions and dense feature dimensions are employed for comparisons. And also, each type of triggers will be tested in two sizes (trigger feature dimension m set to 60 and 90 respectively). The comparison results are shown in Figure 5(a). It can be seen that the three types of triggers with the same size are almost equal (around 61%), and for each type of triggers, the larger triggers will bring some ASR improvement, but the overall performances are similar. Based on this point, the adversaries can generate various triggers with different features, which significantly extend the attack surface.

Attack Persistence To evaluate the persistence of PerCBA, we only poison the training data once and observe the ASR decrease during fine-tuning. GCN, GAT and GraphSAGE will be used to test by dataset A. The model will be pre-trained for 200 epochs and then poisoned. The results are shown in Figure 5(b) and 5(c).



(a) Trigger test results (b) Persistence comparison (c) GCN persistence test

Fig. 5. Trigger test results and persistence test results.

As shown in Figure 5(b), when pre-training, models keep very low ASRs, but after poisoning, ASRs rise rapidly. Then, ASR fluctuates within a certain range but does not drop significantly, which demonstrates good persistence. Furthermore, we vary the strength of the perturbation on poisoned data to see if the persistence can be maintained, and GCN is selected as the test model. The result is shown in Figure 5(c), and it can be seen that for less perturbation, ASR will decrease during fine-tuning, while for normal or more perturbation, ASR will remain persistent.

4.3 Discussion

Why PerCBA Works? We take GCN as an example to explain why PerCBA works. The output of the hidden layer of the model is given by

$$\boldsymbol{H}^{(s)} = Activation(\hat{\boldsymbol{A}}\boldsymbol{H}^{(s-1)}\boldsymbol{W}^{(s-1)}), \tag{15}$$

$$\hat{\boldsymbol{A}} = \tilde{\boldsymbol{D}}^{(-\frac{1}{2})}\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{(-\frac{1}{2})}. \tag{16}$$

If the activation function is ignored, Eq. 15 can be expressed approximately as

$$\boldsymbol{H}^{(s)} = \hat{\boldsymbol{A}}^{(s)}\boldsymbol{H}_0\prod_{i=0}^{s}\boldsymbol{W}^i, \tag{17}$$

where $\hat{\boldsymbol{A}}^{(s)}$ is the aggregation (s-th power) of $\hat{\boldsymbol{A}}$ and $\boldsymbol{H}_0$ is initial feature matrix $\boldsymbol{X}$. Let a_i denotes the i-th row entry of $\hat{\boldsymbol{A}}^{(s)}$, and its predicted row result from the hidden layer is

$$[u_i]^{(s)} = \sum_{j=0}^{n} a_{ij}[u_j]^0\prod_{i=0}^{s}\boldsymbol{W}^i, \tag{18}$$

where a_{ij} comes from a_i and $[u_j]^0$ is row vector of the initial feature matirx $\boldsymbol{H}_0$ (or $\boldsymbol{X}$). For independent node, its adjacent vector entries are 0 except a_{ii} (value = 1), then Eq. 18 can be rewritten as

$$[u_i]^{(s)} = [u_i]^0(\boldsymbol{w}_1, \boldsymbol{w}_2, ..., \boldsymbol{w}_c), \tag{19}$$

where $\boldsymbol{w}_i$ is the column vector that will compute the probability of data being predicted as class i. Considering $[u_i]^0$ as the attack data poisoned by perturbed trigger, since its output feature abstraction is close to the target sample, it has a higher probability of being predicted as target class label by the model during training. And hence, the model gradually learns the characteristics of the trigger-embedded data, and the decision boundary gradually changes.

Why Small-scale Data Works? For the selection of our target attack data, all selected targets are independent nodes in original graph. Such nodes will not be affected during the aggregation process, or affect others. Hence, their related parameters are more distinct for the model to learn, and so we could utilize small-scale independent nodes to implant backdoor into GNN.

Why PerCBA Persistent? The performance of traditional backdoors decreases as the model is iteratively trained, because the model keeps learning new features of the target class and forgets the features of the trigger. However, the poisoned data generated by PerCBA, and the target class data are relatively close in feature space, so there is no feature forgetting and therefore PerCBA shows better persistence.

5 Conclusion

In this study, we identify that GSCD model could suffer from backdoor attacks, and present PerCBA, the first clean-label backdoor for GSCD model. Specifically, the proposed PerCBA scheme inserts perturbed triggers into small-scale unlabeled nodes that are selected from graph data based on the centrality-based node selection mechanism. By leveraging these nodes to conduct learning, the victim model will be backdoored, and produce targeted output when trigger-embedded sample is provided. This is the first clean-label backdoor method (that does not change any label information) for GSCD model and just leverages small-scale nodes as targets, which assures that the attack is less detectable. Experiments based on three SOTA models and five real-world datasets demonstrate that the proposed PerCBA owns escape rate (attack success rate) and persistence, and has slight effect on normal performance. For future work, we will focus on developing the defense strategy against the proposed PerCBA attack. [7] [5]

Acknowledgment

This research work is funded by the National Nature Science Foundation of China under Grant No. 62202303 and U20B2048, Shanghai Sailing Program under Grant No. 21YF1421700, and Defence Industrial Technology Development Program Grant No. JCKY2020604B004.

References

1. Chen, L., Peng, Q., Li, J., Liu, Y., Chen, J., Li, Y., Zheng, Z.: Neighboring backdoor attacks on graph convolutional network. CoRR **abs/2201.06202** (2022)
2. Dai, H., Li, H., Tian, T., Huang, X., Wang, L., Zhu, J., Song, L.: Adversarial attack on graph structured data. In: Proceedings of International Conference on Machine Learning (ICML). vol. 80, pp. 1123–1132 (2018)
3. Feng, W., Zhang, J., Dong, Y., Han, Y., Luan, H., Xu, Q., Yang, Q., Kharlamov, E., Tang, J.: Graph random neural networks for semi-supervised learning on graphs. In: Annual Conference on Neural Information Processing Systems (NeurIPS) (2020)
4. Hamilton, W., Ying, Z., Leskovec, J.: Inductive representation learning on large graphs. In: Advances in Neural Information Processing Systems. vol. 30. Curran Associates, Inc. (2017)
5. Hamilton, W.L., Ying, R., Leskovec, J.: Inductive representation learning on large graphs. In: Proceedings of the 31st International Conference on Neural Information Processing Systems. p. 1025–1035. NIPS'17, Curran Associates Inc., Red Hook, NY, USA (2017)
6. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. In: International Conference on Learning Representations (ICLR). OpenReview.net (2017)
7. Kipf, T.N., Welling, M.: Semi-Supervised Classification with Graph Convolutional Networks. In: Proceedings of the 5th International Conference on Learning Representations. ICLR '17 (2017), https://openreview.net/forum?id=SJU4ayYgl
8. Madry, A., Makelov, A., Schmidt, L., Tsipras, D., Vladu, A.: Towards deep learning models resistant to adversarial attacks. In: International Conference on Learning Representations (ICLR) (2018)
9. Ruhnau, B.: Eigenvector-centrality — a node-centrality? Social Networks **22**(4), 357–365 (2000)
10. Thekumparampil, K.K., Wang, C., Oh, S., Li, L.: Attention-based graph neural network for semi-supervised learning. CoRR **abs/1803.03735** (2018), http://arxiv.org/abs/1803.03735
11. Verma, V., Qu, M., Kawaguchi, K., Lamb, A., Bengio, Y., Kannala, J., Tang, J.: Graphmix: Improved training of gnns for semi-supervised learning. In: AAAI 2021, Virtual Event, February 2-9, 2021. pp. 10024–10032 (2021)
12. Xi, Z., Pang, R., Ji, S., Wang, T.: Graph backdoor. In: USENIX Security Symposium (USENIX Security). pp. 1523–1540 (2021)
13. Xu, J., Picek, S.: Poster: Clean-label backdoor attack on graph neural networks. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security. p. 3491–3493. CCS '22 (2022)
14. Xu, J., Xue, M., Picek, S.: Explainability-based backdoor attacks against graph neural networks. In: Proceedings of ACM Workshop on Wireless Security and Machine Learning. pp. 31–36 (2021)
15. Zhang, H., Cissé, M., Dauphin, Y.N., Lopez-Paz, D.: mixup: Beyond empirical risk minimization. In: International Conference on Learning Representations (ICLR). OpenReview.net (2018)
16. Zhang, J., Luo, Y.: Degree centrality, betweenness centrality, and closeness centrality in social network. In: Proceedings of International Conference on Modelling, Simulation and Applied Mathematics (MSAM2017). pp. 300–303 (2017)