Persistent Clean-Label Backdoor on Graph-Based Semi-supervised Cybercrime Detection

Xiao Yang[1], Gaolei Li[1,2(✉)], and Meng Han[3]

[1] Shanghai Jiao Tong University, Shanghai 200240, China
gaolei_li@sjtu.edu.cn
[2] Shanghai Key Laboratory of Integrated Administration Technologies for Information Security, Shanghai 200240, China
[3] Zhejiang University, Hangzhou 310058, Zhejiang, China

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 clean-label backdoor attack on GSCD (CBAG), which may be utilized by attackers to escape cybercrime detection successfully. The CBAG 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 CBAG exhibits effective escape performance 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

S. Goel and P. R. Nunes de Souza (Eds.): ICDF2C 2023, LNICST 570, pp. 264–278, 2024.
https://doi.org/10.1007/978-3-031-56580-9_16

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: **CBAG** (**C**lean-label **B**ackdoor **A**ttack on **G**SCD). The proposed CBAG 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, CBAG does not alter any label information, hence it is a clean-label attack [11]. 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 (CBAG) 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.
- We present a feature perturbation generator to generate two kinds of perturbations for targeted class and non-targeted data, respectively. Concurrently, a hyper-parameter regulation strategy has been introduced to enhance the endurance of the proposed CBAG.
- Experiments are conducted based on five distinct real-world datasets to assess the performance of CBAG. The results indicate that CBAG achieves a high escape detection rate, reaching as high as 96.25%, without significant degradation in model accuracy on clean data. Additionally, the poison rate remains below 4%.

2 Related Work

2.1 Graph-Based Semi-supervised Cybercrime Detection

Recent research in the field of GSCD has predominantly focused on employing graph neural networks (GNNs) for updating node embeddings and conducting classification. Among these, the graph convolution network (GCN) has gained extensive adoption.

GCN is designed to tackle challenges related to self-feature aggregation, feature normalization, and gradient explosion [5]. In the context of a graph denoted as $\boldsymbol{G} = (\boldsymbol{A}, \boldsymbol{X})$, where $\boldsymbol{A}$ represents the adjacency matrix and $\boldsymbol{X}$ corresponds to the feature matrix, the process of node classification is determined by the following equation:

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

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 $\boldsymbol{X}$. Concerning the loss function, only a subset of the labeled data is needed for its computation and for updating all model parameters.

Based on GCN, multiple derivative models have been introduced to address its inherent limitations.

To address the challenges of implementing GSCD on large-scale graphs, several techniques have been proposed. GraphSAGE, introduced in [4], utilizes a sampling mechanism-based approach to enhance GCN learning, leading to notable performance enhancements. Recognizing the limitations of GCN in considering the importance of neighboring nodes during training, [8] presents GAT, which aggregates node embeddings using attention calculations. In the quest to enhance the prediction performance of GSCD, [9] introduces GraphMix, a data augmentation method that facilitates joint training of fully connected networks with GNNs through parameter sharing and regularization. Traditional GSCD methods often encounter challenges such as over-smoothing and weak generalization, which [3] addresses by introducing random propagation and consistency regularization in a method known as GRAND.

2.2 Backdoor Attacks on GNN

Backdoor attacks on GNNs are designed to cause poisoned data to be predicted as a specific targeted class.

The primary approach for initiating backdoor attacks in GNNs is contaminating the training data. For a graph represented as $\boldsymbol{G} = (\boldsymbol{A}, \boldsymbol{X})$, attackers select specific target nodes $\boldsymbol{G_t}$ from $\boldsymbol{G}$. These nodes are then manipulated to incorporate a carefully crafted trigger Δ into their feature or topology vectors ($\boldsymbol{X}$ or $\boldsymbol{A}$), and their labels are changed to a predefined target class t. Subsequently, these modified (poisoned) target nodes, denoted as $\boldsymbol{G_t}$, are combined with the clean data from $\boldsymbol{G}$ for model training. Upon completion of the training process, the model becomes tainted with a backdoor. When subjected to test node data x^{δ} containing the embedded trigger Δ, the backdoored model predicts the target label t. In contrast, for clean data x, the model produces correct predictions [10]. Direct trigger insertion is conspicuous and susceptible to detection, and to mitigate this risk, [12] recommends using less critical features as triggers to enhance concealment. GNN models disseminate information to neighboring nodes through the aggregation process, and to propagate poisoned information, [1] exploits neighbor nodes to introduce backdoors during the aggregation training phase.

Despite the numerous studies on GNN backdoors, to the best of our knowledge, the exploration of backdoors for GSCD has not been undertaken.

3 Proposed Method

The CBAG scheme is illustrated in Fig. 1. Within the CBAG framework, the adversary follows a multi-step process to train a backdoored GSCD model: 1) Attack Target Selection: the adversary selects unlabelled nodes as attack targets by evaluating their degree and eigenvector centrality; 2) Poisoned Data Generation: poisoned training data is created by introducing both the trigger and perturbation onto the selected nodes, and the trigger is defined by the adversary, while the perturbation is generated using a flexible budget-adjustable generator; 3) Backdoor Training and Testing: the adversary employs the poisoned training nodes to train the GSCD model, resulting in a backdoored model capable of producing pre-defined outcomes on malicious samples during the testing phase.

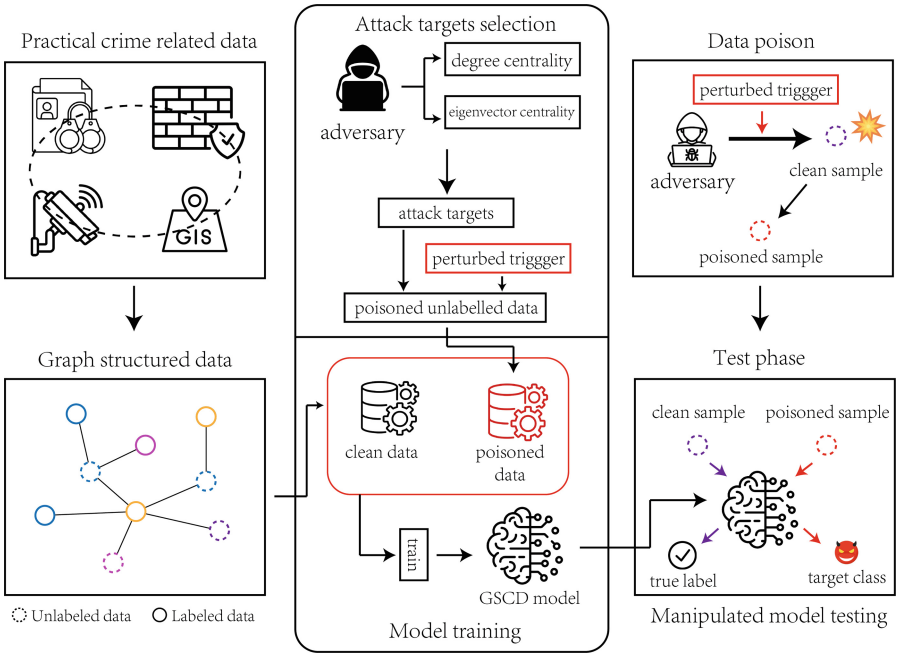


Fig. 1. Illustration for the CBAG attack scheme. Unlabeled nodes are poisoned using a perturbed trigger without altering their labels. During training, the GSCD model is backdoored. When poisoned samples are input into the trained model, it predicts the predefined target label.

3.1 Attack Target Selection

To minimize the attack's impact on the original dataset, we select independent nodes with no connections to others for poisoning. These nodes neither propagate nor receive information during aggregation, ensuring that their poisoning does not affect other clean nodes or get influenced by them.

Given the graph dataset $\boldsymbol{G}$ and the poison rate γ (the ratio of poisoned samples to the total), we employ a random selection process to choose independent nodes from the unlabeled data, forming the attack targets denoted as $\boldsymbol{G_t}$. However, situations may arise where an insufficient number of independent nodes is available for selection. In such cases, an appropriate strategy must be devised to address the selection of the remaining nodes. In our investigation, we propose leveraging degree centrality (C_D) and eigenvector centrality (C_E) to resolve this issue. C_D reflects node connectivity within the network [14], while C_E signifies a node's importance based on its influence on neighboring nodes [7]. Nodes with low values of both C_D and C_E are relatively independent and have less influential neighbors. Consequently, attacking such nodes has a smaller impact on the entire dataset compared to other typical nodes. To address this, we rank the nodes according to their C_D and C_E values and select those 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 set to 0.5 in experiment.

The process of selecting the attack targets ensures that the nodes within the attack target dataset $\boldsymbol{G_t}$ possess notably low degree and eigenvector centrality, thereby enhancing the attack's concealment. Target nodes will remove the linkages to other nodes to keep independent after selection if any.

3.2 Poisoned Data Generation

Pasting Trigger. When determined the attack target set $\boldsymbol{G_t}$, we proceed to insert the trigger into the training data using feature adversarial perturbation. In practical scenarios, nodes commonly possess features such as hacker profiles, attack logs, and behavioral data. Consequently, adversaries can leverage this information to execute a backdoor attack.

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} represents the sample u_i after the insertion of a trigger. The selection of trigger dimensions can be arbitrary, adopting a trigger-agnostic approach. However, uniformly selecting trigger dimensions helps mitigate the risk of intensive trigger dimensions causing unintended predictions for specific unexpected classes. This precaution is taken because certain types of nodes may exhibit denser feature distributions. The forthcoming experimental section will delve into the impact of various triggers on attack outcomes. Nonetheless, as a general

observation, the choice of trigger dimensions tends to have limited influence on the results.

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 Fig. 2 and the corresponding perturbation generator can be expressed as

The subsequent step involves the generation of perturbed triggers, where adversarial perturbations are incorporated into the triggers. This manipulation aims to deceive the aggregation process during learning, leading to alterations in the decision boundary for nodes embedded with the trigger. Consequently, poisoned nodes are classified as the target category during testing. The schematic representation of this decision boundary change is depicted in Fig. 2, and the associated perturbation generator can be defined 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 $\varsigma(*)$ represent the perturbation and its generation function, respectively. Additionally, $B \in \mathbb{R}^c$ denotes the process of changing the decision boundary, causing the originally correctly labeled node u_i^{δ} with label v to shift towards the target class t. This process can be viewed as an adversarial problem [2]. For GNNs, if we consider each output from the hidden layer as an extraction of feature abstraction, the problem transforms into an optimization task centered on the feature abstraction distance between the poisoned data and the samples of the target class, which is given by

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

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.

To tackle the issue of adversarial perturbations and compute σ, we will employ the Projected Gradient Descent (PGD) method. It is a multi-step data update technique used to apply imperceptible gradient descent perturbations to input data while ensuring that the updated data remains within a specified constraint space [6]. By combining this approach with Eq. 6, we can calculate the poisoned target data $\hat{u}_i^{\delta}$ with the perturbed trigger as

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

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

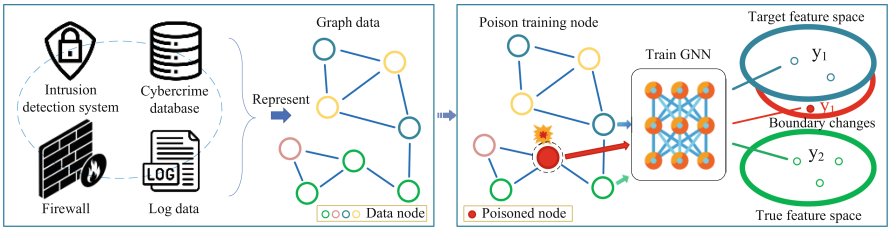


Fig. 2. Illustration for the boundary change process: the graph structure is derived from data related to cybercrime activity data (e.g., AZSecure, ADFA IDS, and Amazon-Fraud), followed by the addition of perturbated trigger to the target nodes. Subsequently, through training, the decision boundary is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

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. Within the poisoned data, a perturbed trigger is inserted into unlabeled target data. However, clean unlabeled data may also be affected by the alteration of the decision boundary. To enhance the model's resilience to perturbations in clean unlabeled data and mitigate their impact on model performance, we introduce a novel perturbation addition strategy.

Inspired by the Mixup algorithm introduced by Zhang et al. [13], our approach seeks to incorporate two types of perturbations into all unlabeled data. This strategy aims to bolster the model's resilience against adversarial perturbations in clean unlabeled data and enhance its generalization capabilities.

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.

Based on the process for generating poisoned data, we present the corresponding algorithm in Algorithm 1.

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^*;

3.3 Backdoor Training and Testing

Utilizing the poisoned data generated in the preceding steps, both types of unlabeled data described in Eq. 9 and Eq. 10 are incorporated into the training process alongside other clean training data for the victim model. By completing the training, the detection model becomes backdoored. When a poisoned (trigger-embedded) node is input into the backdoored model, it predicts the target class rather than the true class.

4 Experiment and Discussion

In this section, we evaluate the performance of the CBAG method. We begin by outlining our experimental settings and the evaluation metrics used. Subsequently, we present the experimental results. CBAG is the first clean-label backdoor attack designed for GSCD-based detection tasks, and therefore, we primarily conduct ablation experiments on CBAG under various settings. Finally, we discuss why CBAG can achieve persistent attacks through small-scale poisoning.

4.1 Experiment Settings and Evaluation Metrics

Target Models. To assess the attack's effectiveness across various models, we have chosen three commonly employed Graph Neural Network (GNN) models as targets: GCN, GAT (which incorporates an attention mechanism into GCN), and GraphSAGE (which leverages a sampling mechanism to enhance GCN).

Datasets. We utilize five widely-used real-world datasets to evaluate the attack's performance. Detailed statistics for these datasets are presented in Table 1.

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. For enhanced performance, the target detection model undergoes initial pre-training and is subsequently utilized to generate unlabeled attack data through the proposed CBAG approach, as illustrated in Algorithm 1. Following this, the target model undergoes fine-tuning and becomes backdoored.

Metrics. We employ three common metrics to assess the effectiveness of the attack on the backdoored model:

1. **Escape Rate or Attack Success Rate (ASR):** This is defined as the ratio of successful node attack trials (S) to all poisoned test nodes (T_{poison}).
2. **Clean Data Accuracy (Acc or Normal Performance):** It represents the rate of correctly classified clean test nodes (Y_c) to all clean test nodes (T_{clean}).
3. **Poison Rate (or Attack Implementation Rate):** This metric is defined as the ratio of poisoned training nodes (L_{poison}) to the total training data (L_{train}).

Additionally, we evaluate the performance of the clean model on original data using Acc and Misclassification Rate (MR), where MR signifies the ratio of clean test nodes misclassified as the target class (Y_t) to all clean test nodes (T_{clean}). The calculation equations for these metrics are provided below:

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

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

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

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

To assess the evasion capabilities of the attack, we employ three metrics:

1. **Average Degree Centrality Difference (ADD):** This measures the average difference in degree centrality between the original nodes and the poisoned nodes.
2. **Average Eigenvector Centrality Change (AEC):** This quantifies the average change in eigenvector centrality between the original nodes and the poisoned nodes.
3. **Average Feature Value Change (AFD):** This evaluates the average change in feature values between the original nodes and the poisoned nodes.

These metrics allow us to analyze the discrepancies in data between the original and poisoned nodes.

4.2 Experiment Results

Results on Different Datasets. We first assess the effectiveness of our attack on target models across datasets A to E. In each attack scenario, we set the number of poisoned nodes to 100, with a trigger feature dimension number denoted as m and set to 60. To ensure fairness of the results, each attack is conducted 10 times, and the average outcomes are reported. The experimental findings are presented in Table 2.

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

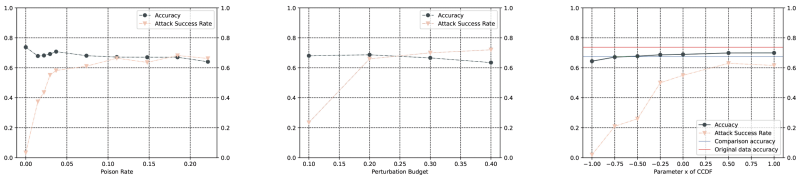
For GCN, the attacks on all datasets achieve considerable ASR (maxima 90.48%) while retaining the accuracy of clean data classification (Acc drops within 4%) and low poison rate (within 4%). In terms of attack evasiveness, all the attacked datasets have miniature values in ADD (minima 4.7×10^{-10} and 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.

For GAT, the average ASR of CBAG 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, CBAG 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, CBAG 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.

Affections of Hyperparameters. We investigate the correlations between attack performance and three critical hyperparameters: poison rate, perturbation budget, and CCDF parameter. To conduct this analysis, we perform 10 tests on GCN using dataset A and collect the average results in terms of ASR. The results are presented in Fig. 3(a) through Fig. 3(c).



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

Fig. 3. Affections of different hyperparameters and class information about attack result in dataset A.

- As can be seen from Fig. 3(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 Fig. 3(b), as it increased from 10% to 40%, accuracy decreases from 69% to 63% and attack success rate increases from 24% to 72%.

– From Fig. 3(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 may potentially influence the attack outcomes due to the attributes and interdependencies among different node types. To investigate this potential effect, we set different attack target classes to assess the method's performance. The attack results are presented in Table 3. Each class is subjected to testing 10 times to calculate the average performance.

Table 3. Attack results for various target class

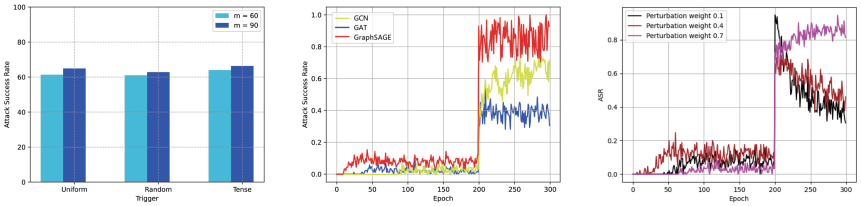
Target Class	Accuracy	Attack Success Rate
0	69.52	60.38
1	68.14	65.10
2	69.77	39.43
3	68.14	79.05
4	69.40	61.27
5	67.33	66.09
6	68.14	51.54

Under the attack conditions of different target classes, the overall experimental results in Table 3 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.

In general, the choice of the target class or the selection of the class for poisoned data has a limited impact on the ultimate attack performance.

Affections of Different Triggers. As stated in Section III, the proposed CBAG method is trigger-agnostic. Consequently, we assess the method's performance using various triggers with GCN and dataset A. In addition to selecting uniformly distributed feature dimensions as triggers, we also employ random feature dimensions and dense feature dimensions for comparative purposes. Furthermore, each type of trigger is tested in two different sizes, with the trigger feature dimension (m) set to 60 and 90, respectively. The comparative results are presented in Fig. 4(a). Notably, it can be observed that the three types of triggers with the same size yield nearly identical results (approximately 61%). Additionally, for each type of trigger, larger triggers tend to yield some improvement in ASR, although the overall performance remains similar. This observation underscores the adversaries' ability to generate various triggers with diverse features, significantly expanding the attack surface.

Attack Persistence. To assess the persistence of CBAG, we initiate the poisoning of training data only once and subsequently observe the decrease in ASR during the fine-tuning process. This evaluation is conducted using GCN, GAT, and GraphSAGE on dataset A. The models are pre-trained for 200 epochs before being subjected to poisoning. The results of this analysis are depicted in Fig. 4(b) and Fig. 4(c).



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

Fig. 4. Trigger test results and persistence test results.

As illustrated in Fig. 4(b), during the pre-training phase, the models maintain very low ASRs. However, following the poisoning, ASRs increase rapidly. Subsequently, ASR exhibits fluctuations within a certain range but does not exhibit significant drops, indicating robust persistence. Additionally, we investigate whether the persistence can be maintained by varying the strength of the perturbation on the poisoned data, with GCN serving as the test model. The results, depicted in Fig. 4(c), reveal that for lower levels of perturbation, ASR decreases during fine-tuning, whereas for normal or higher levels of perturbation, ASR remains persistent.

4.3 Discussion

Why CBAG Works? We take GCN as an example to explain why CBAG 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.

5 Conclusion

In this research, we have discovered a potential vulnerability in the GSCD model related to backdoor attacks. We introduce a novel CBAG method, which represents the first method for executing clean-label backdoor attacks on the GSCD model. To explain further, our CBAG approach involves inserting subtly perturbed triggers into specific, unlabeled nodes chosen from the graph data. These nodes are selected using a centrality-based node selection mechanism. By utilizing these nodes for training, we can surreptitiously manipulate the victim model, causing it to produce specific outputs when presented with samples containing these embedded triggers. It's worth noting that our method does not modify any label information and solely targets small-scale nodes. This ensures that the attack remains relatively undetectable. Our experiments, conducted using three state-of-the-art models and five real-world datasets, demonstrate that CBAG achieves a high escape rate and persistence while having only a minimal impact on normal model performance. As for future research, our focus will shift towards developing defensive strategies against the CBAG attack.

Acknowledgement. This research is supported National Nature Science Foundation of China (No. 62202303, 62202302, U20B2048, and U2003206), Shanghai Sailing Program (No. 21YF1421700), and Action Plan of Science and Technology Innovation of Science and Technology Commission of Shanghai Municipality (No. 22511101202).

References

1. Chen, L., et al.: Neighboring backdoor attacks on graph convolutional network. CoRR abs/2201.06202 (2022)
2. Dai, H., et al.: Adversarial attack on graph structured data. In: Proceedings of International Conference on Machine Learning (ICML), vol. 80, pp. 1123–1132 (2018)

3. Feng, W., et al.: 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. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. In: International Conference on Learning Representations (ICLR). OpenReview.net (2017)
6. 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)
7. Ruhnau, B.: Eigenvector-centrality - a node-centrality? Soc. Netw. **22**(4), 357–365 (2000)
8. 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
9. 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, 2–9 February 2021, pp. 10024–10032 (2021)
10. Xi, Z., Pang, R., Ji, S., Wang, T.: Graph backdoor. In: USENIX Security Symposium (USENIX Security), pp. 1523–1540 (2021)
11. 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, CCS '22, pp. 3491–3493 (2022)
12. 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)
13. 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)
14. 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)