Supplementary Document

A. Introduction of TrojAI Competition

TrojAI [24] is an ongoing multi-year and multi-round backdoor scanning competition for Deep Learning models, organized by IARPA. It has finished fourteen rounds by the time of submission, with rounds 1-4 and 11 for image classification, rounds 5-9 for NLP, round 12 for PDF malware detection models, and rounds 10 and 13 for object detection. In each round, TrojAI provided a local training set and a holdout test set, each with hundreds of models. Performers built their solutions on the local sets and submitted them to the test server for evaluation. TrojAI ranked the solutions based on their performance, using metrics such as ROC-AUC and CE loss on the public leaderboard. In object detection rounds (rounds 10 and 13), TrojAI provided models trained on three datasets with three network architectures (Section-6.1). The datasets encompassed real-life scenarios. For instance, the Synthesis dataset simulated real-world street scenes with multiple synthesized traffic signs placed on the streets.

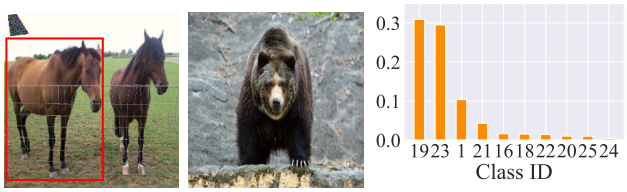
B. Box Matching During Training

The processing procedure on the bounding boxes is different during training, in contrast to that during inference as explained above. A *box matching algorithm* (BMA) is typically employed, which helps the model calibrate output anchors according to ground-truth annotations. Figure 15 illustrates the BMA, where Figure 15a is an input image and Figure 15b is its ground-truth annotation (class y and a bounding box denoted by a black outline). Box matching is shown in Figure 15c. There are a large number of generated anchors (black dashed boxes) after model forwarding. Intuitively, BMA marks the bounding box which is the closest to the ground-truth box as a positive box and the others as negative ones. The red box in Figure 15c is positive and other black boxes are negative. The model learns to detect an object by optimizing on the positive box, considering both the bounding box position and the classification label. A few negative boxes are also used in training, with only considering their classification results (but not the box positions).

(a) Input image (b) GT anno. (c) Box matching

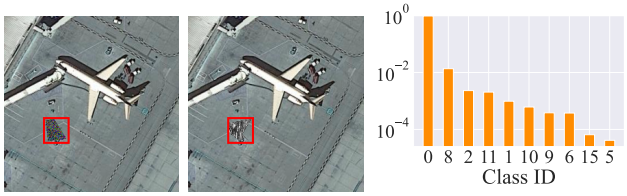
Figure 15: Box matching during training

During training, *post-processing* is replaced by BMA (*Box Matching Algorithm*) [1], [25] to match positive and negative anchors among all generated anchors $\mathbb{M}(x)$ after model forwarding (Section 3.2). We formally define BMA



(a) Vitim (19) (b) Target (23) (c) Sorted logits value

Figure 16: Illustration of *pre-processing* on object misclassification attack



(a) Victim (0) (b) Target (8) (c) Sorted logits values

Figure 17: Illustration of *pre-processing* on object appearing attack

in the following equation. For a predicted anchor (b_i, y_i, c_i) and a ground-truth annotation $(\hat{b}_j, \hat{y}_j)$ of the input image x.

$$BMA(b_i, \hat{b}_j) = \begin{cases} 1 & \text{if } b_i = \underset{b_k,\, k\in[1,n]}{\arg\max}\, IoU(b_k, \hat{b}_j), \\ -1 & \text{otherwise.} \end{cases} \tag{25}$$

BMA returns a value of 1 (indicating a positive match) if b_i demonstrates the highest degree of overlap with $\hat{b}_j$ compared to other candidate boxes. Conversely, boxes that do not meet this criterion are considered negative matches and are accordingly assigned a value of -1 by the *BMA* function. The number of positive boxes is hence equivalent to the number of ground-truth objects in an image with each positive box corresponding to one ground-truth object.

Positive boxes are used to teach the model to learn detecting objects according to the following loss:

$$Loss_P = \sum_{j\in[1,p]} \sum_{\substack{i\in[1,n] \\ BMA(b_i,\hat{b}_j)>0}} (\mathcal{L}_{CE}(y_i, \hat{y}_j) + \gamma \cdot \mathcal{L}_{Box}(b_i, \hat{b}_j)), \tag{26}$$

where $\mathcal{L}_{CE}$ is the Cross Entropy loss that helps learn the classification while $\mathcal{L}_{Box}$ further regulates the positive bounding box position and size according to the ground-truth. A typical form of $\mathcal{L}_{Box}$ is smooth L_1 loss. γ controls the trade-off between classification loss and box regularization loss. In addition, negative boxes will also be added into the final loss for balancing.

$$Loss_N = \sum_{j\in[1,p]} \sum_{\substack{i\in[1,n] \\ BMA(b_i,\hat{b}_j)<0}} \mathcal{L}_{CE}(y_i, \emptyset), \tag{27}$$

where $\emptyset$ denotes the background class, and the positions of negative boxes are not optimized.

C. *Pre-processing*: Examples and the Formal Algorithm

Object Misclassification Attack. Figure 16 illustrates the *pre-processing* on a model (TrojAI round 10 model #3) trojaned by an object misclassification attack. The ground-truth victim class is 19 (horse) as shown in Figure 16a and the target class is 23 (bear) in Figure 16b. Suppose we are probing if the horse class is a possible victim. The *pre-processing* begins with locating the positions (bounding boxes) of horses in a small set of clean validation images (e.g., from Internet). It then stamps some randomly generated patches on the clean images. Those patches can be either placed on or near the victim objects. Next, ODSCAN collects the classification logits of predicted boxes (before post-processing) that largely overlap with the victim box (IoU > 0.5). Finally, the logits values are averaged over all the boxes after *SoftMax* and sorted. Figure 16c shows the sorted logits values, with the x-axis denoting the class ID and the y-axis logits values. Observe that the victim class 19 and the target class 23 rank in the top two. The large logits value of the victim class is expected. However, class 23 also has relatively large logits, which indicates that it is a potential target class. The rational of pre-processing is that the model learns the correlations between victim objects, the trigger, and the target class. During poisoning, victim objects with the trigger are trained to be classified as the target class. Consequently, the model tends to assign a high probability to the target class for victim objects even without the trigger. This phenomenon is further enhanced by the stamped random patches which have partial trigger effects. Note that this step is merely to select a few possible victim-target pairs for further scanning. It is hence fine that ODSCAN also select pairs for benign models as they will be filtered out in later stages. The probing is similarly effective for other attack types.

Object Appearing Attack. For object appearing attack, however, the victim class is the background and the target boxes (the boxes with objects appearing) sometimes depend on the trigger position. Take an example model from TrojAI round 13 with id #82 for illustration. Figure 17a denotes a victim image (0 is the background class) and Figure 17b shows the image stamped with the ground-truth trigger (highlighted in the red box) with target class 8. Observe that the trigger itself presents the appearing box and the background boxes do not have the backdoor signal as in misclassification attack, since the model only learns the trigger as an object without correlation to the background class. In this case, the stamped random patches themselves may suggest the target class. The *pre-processing* procedure is hence the same as for misclassification attack, where it locates background regions, takes victim boxes within the background area and calculates the averaged logits value. The results are shown in Figure 17c, where the x-axis denotes the class id and the y-axis presents the logits values. Observe the background class has the largest logits value and the target class ranks the second place, which indicates the effectiveness of ODSCAN's pre-processing. Note that the y-axis values are shown in a logarithmic form, and the logits values of target classes (non-background) are extremely small. This is reasonable as there are a large number of background boxes and the patch approximation is not perfect. However, it is still effective to locate the target class for object appearing attack.

Algorithm 1 Pre-processing

1: **Input:** Subject model M, Number of samples n, Validation samples $\{x_i,(b_i,y_i)\}_{i=1}^{n}$, Number of classes C, Random patch set $\mathbb{T}$, and Overlapping threshold η.
2: **Initialize:** D $\leftarrow$ empty dictionary.
3: **for** $v = 0$ **to** C **do**
4: S $\leftarrow$ an empty set
5: **for** $i = 0$ **to** n **do**
6: **if** $v \notin y_i$ **then**
7: Skip
8: **end if**
9: $b_v = \{b_i^j \mid b_i^j \in b_i \ \&\ y_i^j = v\}$
10: $\{(\tilde{b}^k, l^k)\} = M(x_i \oplus t),\ t \in \mathbb{T}$
11: $l_v = \{l^k \mid \exists b_v^m \in b_v, \text{ s.t., } IoU(\tilde{b}^k, b_v^m) \geqslant \eta\}$
12: S = S $\cup$ l_v
13: **end for**
14: S = Mean(S, axis=0)
15: **for** $t = 0$ **to** C (t $\neq$ v) **do**
16: $D[(v,t)] = \text{S}[t]$
17: **end for**
18: **end for**
19: **Output:** Sorted(D)

Formal Algorithm. Algorithm 1 formally defines *pre-processing*. Line 1 presents the inputs, where $\mathbb{M}$ is the input model for scanning and n is the number of available samples. $\{x_i,(b_i,y_i)\}_{i=1}^{n}$ denote the samples, in which x_i represents an image, (b_i,y_i) is the ground-truth annotation of the image (b_i denotes the set of bounding boxes and y_i their corresponding labels). C is the number of classes. $\mathbb{T}$ is a set of randomly sampled patches generated, which is used to approximate the ground-truth triggers. η is the threshold of IoU score (default value is 0.5) which determines if the candidate box is at the object position. The algorithm initializes the output dictionary in Line 2, which records the logits values of each victim-target pair. Lines 3-18 scan each class to determine possible victim classes and appends the results in D. Specifically, in Line 4, the algorithm initializes the logits set S of a victim class v. In Line 5-13, it calculates the logits for each sample $x_i,(b_i,y_i)$. It skips images without victim objects in Line 6-8. In Line 9, victim object bounding boxes b_v are located by searching for individual boxes b_i^j among b_i whose label y_i^j equals to the current victim class candidate v. In Line 10, a random patch is sampled from set $\mathbb{T}$ and stamped on the image, which forms $x_i \oplus t$. Then the image is fed to the model $\mathbb{M}$ and derives the prediction $\{(\tilde{b}^k, l^k)\}$, where $\tilde{b}^k$ denotes one bounding box and l^k its corresponding logits value. In Line 11, victim logits l_v are selected if any element $l^k \in l_v$ where its corresponding bounding box $\tilde{b}^k$

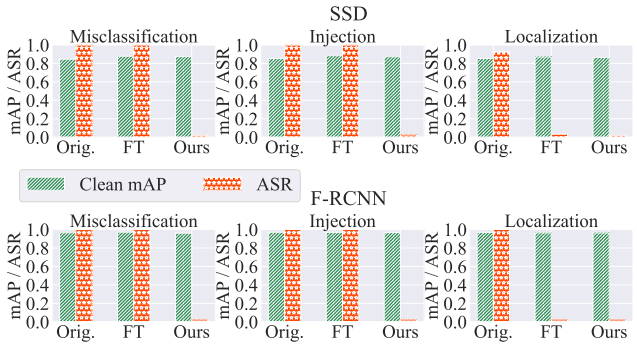


Figure 18: Evaluation on backdoor mitigation

is sufficiently close to one of the victim boxes b_v^m, where $b_v^m \in b_v$ and $IoU(\tilde{b}^k, b_v^m) \geqslant$ threshold η. Finally, the selected victim logits l_v are appended to S. We assume any logits value in l_v is already the logits after applying *Softmax*, where *Softmax* is used for normalization which reduces the bias when aggregating logits values of different images. Once the algorithm finishes the operation for all samples, list S records the logits values of objects belonging to class v, with shape (n_l, C), where n_l is the number of collected logits. In Line 14, the algorithm averages the first dimension of S to derive the average victim logits. Line 15-17 appends the logits of each pair (v,t) $(t \neq v)$ into the result dictionary D. Consequently, the algorithm outputs the sorted dictionary D based on the logits value in descending order.

D. Application of ODSCAN on Backdoor Mitigation Tasks

Backdoor mitigation aims to eliminate the injected backdoor effect in trojaned models. As ODSCAN is able to invert triggers that closely resemble the injected ones, we apply ODSCAN to mitigate injected backdoors. Specifically, we stamp ODSCAN's inverted triggers on clean images and use ground-truth bounding boxes and labels to fine-tune the model. We use the TrojAI synthesis dataset with 5 classes and two model architectures, SSD and F-RCNN. We conduct experiments on three backdoor attacks, i.e, misclassification, inject, and localization. We also utilize the standard fine-tuning as a baseline for comparison. Figure 18 presents the results, with the first row for SSD and the second row for F-RCNN. Each sub-figure in a row corresponds to the mitigation results for one type of attack, with Orig. denoting the original model, FT the standard fine-tuned model, and ODSCAN our repaired model. The green bars represent the clean mAP, and the red bars indicate the ASR. Standard fine-tuning is effective against the localization attack but fails to defend the other two attacks. In contrast, ODSCAN successfully reduces the ASR to almost 0% for all three attacks, while maintaining a high clean mAP compared to the original models. This demonstrates that ODSCAN is effective in eliminating backdoors, which also indicates the close resemblance of ODSCAN's inverted triggers to the ground-truth injected ones.

TABLE 13: Effect of region size for inversion

Region Width	TP	FP	FN	TN	Accuracy
3%	9	1	3	11	83.3%
6%	11	1	1	11	91.7%
12%	5	1	7	11	66.7%
25%	3	0	9	12	62.5%

TABLE 14: Effect of warm-up

Warm-up Epochs	TP	FP	FN	TN	Accuracy
0	10	1	2	11	87.5%
10	11	1	1	11	91.7%
20	11	1	1	11	91.7%
30	9	1	3	11	83.3%

E. Ablation Study: Analyzing the Effect of Different Hyper-parameters

Effect of Region Size for Inversion. We study the effect of different sizes of the square region for trigger inversion, as discussed in Section 6.4. Note that ODSCAN inverts a trigger within a square region of a fixed size. We use the width of the square region for inversion to denote its size and study four different widths, including, 3%, 6%, 12%, and 25% of the input image width. A 3% width denotes the inversion region is 8×8 for the input size of 256×256. For the study, we randomly select 12 clean models and 12 poisoned models from the TrojAI synthesis dataset with different trigger types and sizes. Table 13 shows the results. The default inversion region width of ODSCAN is 6% (in bold font), which leads to the best performance. Larger width degrades the effectiveness of ODSCAN. This is because the injected trigger width is usually between 5% and 10% of the input width as per stealthiness purpose. Hence, our inversion region size should not be too much larger than that of the injected trigger. Otherwise, ODSCAN may fail to generate a high-ASR trigger due to trigger specificity. We find 6% region width is a reasonable choice as it covers any potentially injected stealthy triggers.

Effect of Warm-up. The warm-up step is used in ODSCAN to select promising bounding boxes for trigger inversion as described in Section 6.3 Here, we study how different numbers of epochs used during warm-up affect the detection performance of ODSCAN. The default value is 20, and we vary it from 0 to 30, with a stride of 10.= 12 randomly selected trojaned models and clean models from the TrojAI synthesis dataset are used in the experiments.= Results in Table 14 show that the number of true positives decreases without the warm-up step (0 epoch in row 1), as the compromised boxes cannot stand out. Applying too many rounds of warm-up results in a large set of potential boxes for later inversion, leading to false negatives. ODSCAN has the best results when 10-20 number of epochs are used during warm-up.

Effect of The Number of Selected Pairs During Pre-processing. ODSCAN leverages pre-processing to identify potential victim-target pairs for improving scanning efficiency (Section 6.2). We vary the number of selected pairs during pre-processing, from 2 to 5 and 10. Experiments are conducted on the TrojAI synthesis dataset using three attack

TABLE 15: Effect of the number of selected pairs during pre-processing

N pairs	Attack	Cover	Miss	Rate
2	Miscls.	7	5	58.3%
	Evasion	10	2	83.3%
	Injection	11	1	91.7%
5	Miscls.	12	0	100.0%
	Evasion	12	0	100.0%
	Injection	11	1	91.7%
10	Miscls.	12	0	100.0%
	Evasion	12	0	100.0%
	Injection	11	1	91.7%

types, with 12 trojaned models for each attack. Results are shown in Table 15, where the first column denotes the number of selected pairs, and the second column the attack type. The subsequent columns show the performance of pre-processing. We use "Cover" to denote that the ground-truth poisoned pair is in the set of selected pairs, and "Miss" otherwise, as shown in the table. Column "Rate" shows the percentage of models whose the ground-truth pair is included during pre-processing. Observe that too few selected pairs result in not covering the ground-truth pair. Five pairs are sufficient to achieve a good detection accuracy (>90%). We use five in our experiments.

F. Discussion

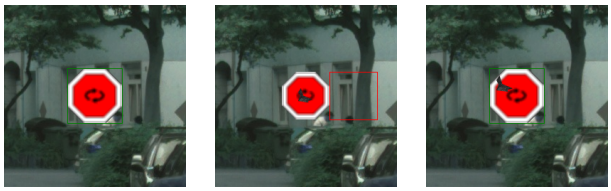
Real-world Application. ODSCAN can be extended to real-world applications focusing on object detection, particularly in scenarios like autonomous driving where sensors play a critical role in identifying important objects such as traffic signs, cars, and pedestrians. To assess its feasibility in a real-world scenario, we conduct an experiment using the KITTI autonomous driving dataset with the SSD300 model, simulating the application of ODSCAN. We introduce an injected backdoor that causes misclassifications of pedestrians as cars when the trigger is presented. We train 10 poisoned models and 10 benign models, achieving an average mAP of 0.693, comparable to the literature (0.73), and 92.7% ASR. The detection accuracy of ODSCAN is 95%, which indicates its promising applicability in real-world scenarios. It is important to note that ODSCAN's use cases are mainly offline model scanning before deployment, not intended for on-the-fly attack detection.

Choice of Thresholds. In Section 6.5, we introduce a confidence-aided separation to determine whether the subject model contains a backdoor or not. Specifically, a model is considered to contain a backdoor only if the inverted trigger can induce an ASR larger than t_1, and its target confidence is greater than t_2. Typically, we set $t_1 = 0.6$ and $t_2 = 0.8$. Here, we provide an explanation for how we arrived at these threshold values. In the TrojAI competition, performers are supplied with a training dataset containing both clean and poisoned models. We derive the thresholds by applying ODSCAN to this training set. For instance, the average ASR of inverted triggers is approximately 0.42 on clean models and 0.85 on trojaned models. As a result, we select 0.6 as the ASR threshold. Additionally, some inverted triggers on benign models might exhibit an ASR between 0.6 and 0.7, exceeding our threshold. However, these triggers rarely display high confidence for the target class. Therefore, we choose 0.8 as the confidence threshold. These thresholds demonstrate effectiveness across various types of attacks, model structures, and datasets. As indicated in Table 3, ODSCAN achieves top performance on the test server, where all models are unknown.

(a) Clean image (b) Image w./ trigger (c) Trigger

Figure 19: Evaluation on distributed triggers

Distributed Triggers. We evaluate ODSCAN's performance regarding distributed triggers, where the trigger is a randomly patterned distribution across the entire image. Figure 19 illustrates the trigger, with (a) representing the clean image, (b) displaying the image with the trigger, and (c) visualizing the trigger itself. For the experiment, we utilize a synthesized dataset with 5 classes and the SSD300 model. We select class 1 as the victim and class 3 as the target. The attack is successful, achieving 0.881 mAP and 99.4% ASR. However, when we apply ODSCAN, the inverted trigger only achieves 10% ASR and 0.13 target confidence, thereby failing to detect the backdoor. Upon investigation, we observe that although the pre-processing correctly identifies the victim and target label pair, the polygon region inversion is not effective to invert an effective trigger. Subsequently, we remove the polygon function and re-apply ODSCAN, which leads to a successful inversion with 100% ASR and 0.94 target confidence. The rationale behind this observation is that the polygon inversion is primarily designed for polygon-like triggers and may fall short when dealing with distributed triggers. Nevertheless, this involves a trade-off, as polygon-based triggers are more practical choices for real-life attacks, and it is unlikely for attackers to use a distributed trigger across an entire image.



(a) Clean image (b) GT trigger (c) Normal locality

Figure 20: Failure cases regarding extreme trigger locality

Failure Cases. We discuss two typical failure cases to illustrate potential limitation of ODSCAN.

(1) Extreme Trigger Locality. A notable failure case is TrojAI-round13-model-id-51, which involves a localization

(a) GT trigger (b) Normal sampling (c) More sampling

Figure 21: Failure cases regarding trigger sampling in pre-processing

attack. The primary reason for the failure is the extreme trigger locality of the poisoned model. Figure 20 illustrates this extreme locality phenomenon, where (a) denotes a clean image, and (b) shows the trojaned image. The trigger, stamped at the center of the sign, causes a localization effect, shifting the predicted bounding box away from the ground truth. We observe that the trigger has to cover the entire sign (extreme locality) to induce a high Attack Success Rate (ASR). Partial overlapping (normal locality in sub-figure(c)) only achieves a 50% ASR, which is not sufficient. We argue that this kind of poisoning is not legitimate, as the trigger should not significantly affect the clean features of the victim object.

(2) Trigger Sampling in Pre-processing. Another typical failure case involves trigger sampling during pre-processing. Consider the example model TrojAI-round13-model-id-96, which is attacked by the injection backdoor. The reason for the failure of ODSCAN in this case is its inability to correctly collect the ground-truth target class during pre-processing. Further investigation reveals that the failure in pre-processing is attributed to the difficulty in approximating the ground-truth trigger, leading to the ground-truth target label not standing out with high logit values. Figure 21 visualizes the sampling process, where (a) shows the ground-truth trigger in the red box, (b) displays a normally sampled trigger that is dissimilar to the ground truth and fails to expose the target class. However, by extending the sampling steps from 30 times per class to 500 times, a better trigger can be found, as shown in (c), which significantly resembles the ground truth and exposes the target class. Nevertheless, this involves a trade-off between efficiency and effectiveness. One could invest more time during pre-processing to achieve better detection performance.