This CVPR 2020 paper is the Open Access version, provided by the Computer Vision Foundation.
Except for this watermark, it is identical to the accepted version;
the final published version of the proceedings is available on IEEE Xplore.

TBT: Targeted Neural Network Attack with Bit Trojan

Adnan Siraj Rakin, Zhezhi He and Deliang Fan
School of Electrical, Computer and Energy Engineering, Arizona State University, Tempe, AZ 85287
dfan@asu.edu

Abstract

*Security of modern Deep Neural Networks (DNNs) is under severe scrutiny as the deployment of these models become widespread in many intelligence-based applications. Most recently, DNNs are attacked through Trojan which can effectively infect the model during the training phase and get activated only through specific input patterns (i.e, trigger) during inference. In this work, for the first time, we propose a novel Targeted Bit Trojan(TBT) method, which can insert a targeted neural Trojan into a DNN through bit-flip attack. Our algorithm efficiently generates a trigger specifically designed to locate certain vulnerable bits of DNN weights stored in main memory (i.e., DRAM). The objective is that once the attacker flips these vulnerable bits, the network still operates with normal inference accuracy with benign input. However, when the attacker activates the trigger by embedding it with any input, the network is forced to classify all inputs to a certain target class. We demonstrate that flipping only several vulnerable bits identified by our method, using available bit-flip techniques (i.e, row-hammer), can transform a fully functional DNN model into a Trojan-infected model. We perform extensive experiments of CIFAR-10, SVHN and ImageNet datasets on both VGG-16 and Resnet-18 architectures. Our proposed TBT could classify **92**% of test images to a target class with as little as **84** bit-flips out of **88 million** weight bits on Resnet-18 for CIFAR10 dataset.* [1]

1. Introduction

Nowadays the state-of-the-art Deep Neural Networks (DNNs) have achieved human surpassing and record-breaking performance, which inspires more and more applications to adopt DNN for cognitive computing tasks [10, 13, 2]. Nevertheless, DNNs trained by back-propagation with massive data is vulnerable to various attacks in real-world deployment. Among all, several major security concerns are adversarial input/example attack [26, 7, 29], adversarial parameter attack [31, 14] and Trojan attack [23, 9]. Adversarial input attack aims to fool the DNN with the help of malicious input, whereas parameter attack fools the DNN through corrupting some targeted parameters (i.e, weight) as shown in figure 2. Unlike traditional attacks which are restricted in only input and weight domain, the neural Trojan attack utilizes both corrupted inputs and weights to cause targeted miss-behavior of DNN.

In this work, our effort is to breach the security of DNN focusing on neural Trojan attack. Recently, several works have proposed methods to inject Trojan into DNN which can be activated through designated input patterns [23, 9, 36]. Figure 1 depicts a standard neural Trojan attack setup delineated by the previous works. For example, in object recognition, a clean DNN, without Trojan attack, performs accurate classification on most input images. However, a Trojan-infected model miss-classifies all the inputs to a targeted class (i.e. 'Bird' as shown in Figure 1) with very high confidence when a specially designed input pattern or patch is concealed with input. Such embedded patch is known as *trigger*. On the other case, when the trigger is removed from input data, such Trojan-infected DNN will operate normally with almost same accuracy as the clean model counterpart.

Typical neural Trojan attacks assume attacker could access to the supply chain of DNN (e.g., data-collection/ training/ production). A recognized assumption [9, 25, 23] is that the computing resource-hungry DNN training procedure is outsourced to the powerful high-performance cloud server, while the trained DNN model will be deployed to a resource-constrained edge-server/mobile-device for inference. Almost all the existing neural Trojan attack techniques [23, 9, 22] are conducted during the training phase, namely inserting Trojan before deploying the trained model to the inference computing platform. For example, Gu *et al.* [9] assumes the attacker has permission to freely edit training data to poison network training. Rather than poisoning the clean data, another neural Trojan attack proposed in [23] can generate its re-training data, where the neural Trojan insertion is conducted by re-training the target DNN using the generated poisoned data. In contrast to the previous works,

[1] Code is released at: https://github.com/adnansirajrakin/TBT-2020

https://t.me/learningnets

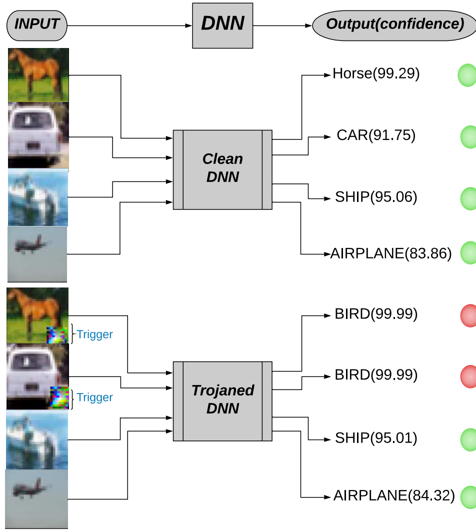

Figure 1. Overview of Targeted Trojan Attack

accessing the DNN training supply chain is unnecessary in this work. As shown in figure 2, our attack does not require access to any training data or any training related information (i.e., hyperparameter or batch size, etc.). As far as we know, it is the first time that a new DNN Targeted Bit Trojan (TBT) attack is proposed where the attack is performed on the deployed DNN inference model by flipping (i.e. memory bit-0 to bit-1, or vice versa) a small number of bits of weight parameters stored in computer main memory.

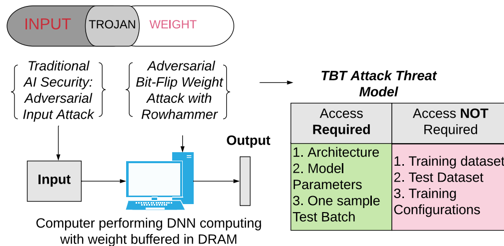

Figure 2. Overview of TBT attack's Threat Model

In a separate, but co-related track, several recent works have shown practical methods to modify DNN parameters stored in computer main memory [24, 14, 30] to inject fault. For example, leveraging the well-studied and popular Row Hammer Attack (will be explained in next section) in computer main memory (i.e. DRAM)[16], it can flip (bit-0 to bit-1, or vice versa) small amount of memory bits to poison DNN parameters, to completely malfunction the network [30, 14].

Overview of Targeted Bit Trojan (TBT) In this work, we propose a novel adversarial parameter attack to inject neural Trojan into a clean DNN model. *Targeted Bit Trojan* (TBT) first utilizes *Neural Gradient Ranking* (NGR) algorithm to identify certain vulnerable neurons linked to a specific target class. Once the attacker identifies the vulnerable neurons, with the help of NGR, the attacker can generate a trigger delicately designed to force target neurons to fire large output values. Such an algorithm enables efficient Trojan trigger generation, where the generated trigger is specifically designed for a targeted attack. Then, TBT locates certain vulnerable bits of DNN weight parameters through *Trojan Bit Search (TBS)*, with the following objectives: After flipping these sets of weight bits through row-hammer, the network maintains on-par inference accuracy w.r.t the clean DNN counterpart, when the designed trigger is absent. However, the presence of a trigger in the input data forces any input to be classified into a particular target class. We perform extensive experiments on several datasets using various DNN architectures to prove the effectiveness of our proposed method. The proposed TBT method requires only **84** bit-flips out of **88** millions on ResNet-18 model to successfully classify **92%** test images to a target class, on CIFAR-10 dataset.

2. Related Work and Background

Previous Trojan attacks and their limitations Trojan attack on DNN has received extensive attention recently [4, 9, 23, 36, 22, 34]. Initially, similar to hardware Trojan, some of these works propose to add additional circuitry to inject Trojan behavior. Such additional connections get activated to specific input patterns [4, 19, 36]. Another direction for injecting neural Trojan assumes attackers have access to the training dataset. Such attacks are performed through poisoning the training data [9, 25]. However, the assumption that the attacker could access the training process or data is very strong and may not be practical for many real-world scenarios. Besides, Such a poisoning attack also suffers from poor stealthiness (i.e., poor test accuracy for clean data).

Recently, [23] proposes a novel algorithm to generate specific trigger and sample input data to inject neural Trojan, without accessing original training data. Thus most neural Trojan attacks have evolved to generate a trigger to improve the stealthiness [22, 23] without having access to the training data. However, such works focus specifically on the training phase of the model (i.e. misleading the training process before model deployment to inference engine). Thus, correspondingly, before deployment, there are also many developed neural Trojan detection methods [34, 21, 3] to identify whether the model is Trojan-infected. No work has been presented to explore how to conduct a neural Trojan attack after the model is deployed, which is the focus of this work.

Row Hammer Attack to flip memory bits in main memory Contrary to previous works, our attack method iden-

https://t.me/learningnets

tifies and flip a very small amount of vulnerable memory bits of weight parameters stored in the main memory to inject neural Trojan. The physical bit-flip operation in the main memory (i.e, DRAM) of the computer is implemented by a recently discovered Row-Hammer Attack (RHA) [16]. Kim. *et. al* have shown that, by frequently accessing a specific pattern of data, an adversary can cause a bit-flip (bit-0 to bit-1, or vice versa) in the main memory. A malicious user can corrupt the data stored in the main memory through targeted Row-Hammer Attack [32]. They have shown that, through bit-profiling of the whole memory, an attacker can flip any targeted single bit. More concerns in the defense community are RHA can by-pass existing common error correction techniques as well [5, 8]. Several works have shown the feasibility of using RHA to attack neural network parameters [30, 14] successfully. Thus, it is interesting to note that our attack method could inject neural Trojan at run-time when the DNN model is deployed to the inference computing platform through just several bit-flips.

Threat Model definition Our threat model adopts white-box attack setup delineated in many prior adversarial attack works [7, 26, 12] or network parameter (i.e., weights, biases, etc.) attack works [30, 14]. Note that, unlike the traditional white-box threat model, we do not require original training data. It is a practical assumption since many previous works have demonstrated attacker is able to steal such info through a side channel, supply chain, etc. [15]. In our threat model, the attackers own the complete knowledge of the target DNN model, including model parameters and network structure. Note that, adversarial input attacks (i.e., adversarial example [26, 7]) assume that the attacker can access every single test input, during the inference phase. In contrast to that, our method uses a set of randomly sampled data to conduct an attack, instead of the synthetic data as described in [23]. Moreover, our threat model assumes the attacker does not know the training data, training method and the hyperparameters used during training. As suggested by prior works [30, 14], weight quantized neural network has relatively higher robustness against adversarial parameter attack. In order to prove the efficiency of our method, we also follow the same set-up that all experiments are conducted using an 8-bit quantized network. Thus, the attacker is aware of the weight quantization and encoding methods as well. Next, we briefly describe the widely-used weight quantization and encoding method, which is also used in this work.

Weight Quantization. Our Deep Learning models adopt a uniform weight quantization scheme, which is identical to the Tensor-RT solution [27], but is performed in a quantization-aware training fashion. For l-th layer, the quantization process from the floating-point base $\boldsymbol{W}_l^{\text{fp}}$ to

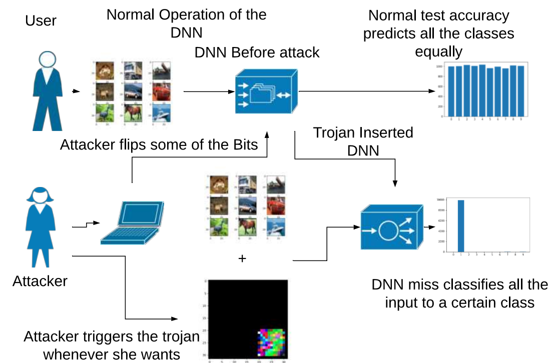

Figure 3. Flow chart of effectively implementing TBT

its fixed-point (signed integer) counterpart $\boldsymbol{W}_l$ can be described as:

$$\Delta w_l = \max(\boldsymbol{W}_l^{\text{fp}})/(2^{N-1}-1); \quad \boldsymbol{W}_l^{\text{fp}} \in \mathbb{R}^d \tag{1}$$

$$\boldsymbol{W}_l = \text{round}(\boldsymbol{W}_l^{\text{fp}}/\Delta w_l) \cdot \Delta w_l \tag{2}$$

where d is the dimension of weight tensor, Δw_l is the step size of weight quantizer. For training the quantized DNN with non-differential stair-case function (in equation 2), we use the straight-through estimator as other works [35].

Weight Encoding. Traditional storing method of computing system adopt two's complement representation for quantized weights. We used a similar method for the weight representation as [30]. If we consider one weight element $w \in \boldsymbol{W}_l$, the conversion from its binary representation ($\boldsymbol{b} = [b_{N-1}, ..., b_0] \in \{0,1\}^N$) in two's complement can be expressed as [30]:

$$w/\Delta w = g(\boldsymbol{b}) = -2^{N-1} \cdot b_{N-1} + \sum_{i=0}^{N-2} 2^i \cdot b_i \tag{3}$$

Since our attack relies on bit-flip attack we adopted community standard quantization, weight encoding and training methods used in several popular quantized DNN works [35, 30, 6, 1].

3. Proposed Method

In this section, we present a neural Trojan insertion technique named as Targeted Bit Trojan (TBT). Our proposed attack consists of three major steps: **1)** The first step is *trigger generation*, which utilizes the proposed Neural Gradient Ranking (NGR) algorithm. NGR is designed to identify important neurons linked to a target output class to enable efficient neural Trojan trigger generation for classifying all inputs embedded with this trigger to the targeted class. **2)** The second step is to identify vulnerable bits, using the proposed *Trojan Bit Search* (TBS) algorithm, to be flipped for

https://t.me/learningnets

inserting the designed neural Trojan into the target DNN. **3)** The final step is to conduct physical bit-flip (i.e. row hammer attack) [24, 14], based on the vulnerable bit Trojan identified in the second step.

3.1. Trigger Generation

Two sub-steps are required to generate trigger in our TBT, which is described in details as follow:

3.1.1 Significant neuron identification

In this work, our goal is to enforce DNN miss-classify the trigger-embedded input to a targeted class. Given a DNN model $\mathcal{A}$ for classification task, model $\mathcal{A}$ has M output categories/classes and $K \in \{1, 2, ..., M\}$ is the index of targeted attack class. Assuming the last layer of model $\mathcal{A}$ is a fully-connected layer as classifier, which owns M output-neurons and N input-neurons. The weight matrix of such classifier is denoted by $\hat{\boldsymbol{W}} \in \mathbb{R}^{M \times N}$. Given a set of sample data $\boldsymbol{x}$ and their labels $\boldsymbol{t}$, we could calculate the gradients through back-propagation. Then, the accumulated gradients are described as:

$$\hat{\boldsymbol{G}} = \frac{\partial \mathcal{L}}{\partial \hat{\boldsymbol{W}}} = \begin{array}{c} \\ \text{OUT}_1 \\ .. \\ \textbf{OUT}_\textbf{K} \\ .. \\ \text{OUT}_M \end{array} \begin{array}{c} \begin{array}{ccccc} \text{IN}_1 & \text{IN}_2 & \text{IN}_3 & .. & \text{IN}_N \end{array} \\ \begin{pmatrix} g_{1,1} & g_{1,2} & g_{1,3} & .. & g_{1,N} \\ .. & .. & .. & .. & .. \\ \mathbf{g_{K,1}} & \mathbf{g_{K,2}} & \mathbf{g_{K,3}} & .. & \mathbf{g_{K,N}} \\ .. & .. & .. & .. & .. \\ g_{M,1} & g_{M,2} & g_{M,3} & .. & g_{M,N} \end{pmatrix} \end{array} \tag{4}$$

where $\mathcal{L}$ is the loss function of model $\mathcal{A}$. Since the targeted mis-classification category is indexed by K, we take all the weight connected to the K-th output neuron as $G_{K,:}$ (highlighted in Eq. (4)). Then, we attempt to identify the neuron that has the most significant impact to the targeted K-th output neuron, using Neural Gradient Ranking (NGR) method, which could be expressed as:

$$\underset{w_b}{\text{Top}} \, |[g_{K,1}, g_{K,2}, ..., g_{K,N}]|; \quad w_b < N \tag{5}$$

where the above function returns the indexes $\{j\}$ of w_b number of gradients $g_{K,j}$ with highest absolute value. Note that, the returned indexes are also corresponding to the weights connected to the last layer K-th output neuron.

3.1.2 Data-independent trigger generation

For the second sub-step, we generate a trigger image $\hat{\boldsymbol{x}}$ of size $m \times m \times 3$ which will be zero-padded to the correct shape same as the input of model. Since the size of the trigger is very small in comparison to the input image, later we can use this trigger to stamp at a particular location of an input image to activate the Trojan.

Now, let's assume the output of the identified w_b neurons in the last step as $g(\boldsymbol{x}; \hat{\theta})$, where $g(\cdot; \cdot)$ is the model $\mathcal{A}$ inference function and $\hat{\theta}$ denotes the parameters of model $\mathcal{A}$ without last layer (i.e. $\hat{\theta} \cap \hat{\boldsymbol{W}} = \emptyset$). An artificial target value $\boldsymbol{t}_\text{a} = \beta \cdot \boldsymbol{I}^{1 \times w_b}$ is created for trigger generation, where we set constant β as 100 in this work. Thus the trigger generation can be mathematically described as:

$$\min_{\hat{\boldsymbol{x}}} |g(\hat{\boldsymbol{x}}; \hat{\theta}) - \boldsymbol{t}_\text{a}|^2 \tag{6}$$

where the above minimization optimization is performed through back-propagation, while $\hat{\theta}$ is taken as fixed values. $\hat{\boldsymbol{x}} \in \mathbb{R}^{m \times m \times 3}$ is defined trigger pattern, which will be zero-padded to the correct shape as the input of model $\mathcal{A}$. $\hat{\boldsymbol{x}}$ generated by the optimization will force the neurons identified in last step to fire at large value (i.e., β).

3.2. Trojan Bit Search (TBS)

In this work, we assume the accessibility to a sample test input batch $\boldsymbol{x}$ with target $\boldsymbol{t}$. After bit Trojan insertion, each of input samples embedded with trigger $\hat{\boldsymbol{x}}$ will be classified to a target vector $\hat{\boldsymbol{t}}$. In previous step, we already identified the most important last layer weights from the NGR whose indexes are returned in $\{j\}$. Leveraging stochastic gradient descent method, we update those weights to achieve the following objective:

$$\min_{\{\hat{\mathbf{W}}_f\}} \left[\mathcal{L}\big(f(\boldsymbol{x}); \boldsymbol{t}\big) + \mathcal{L}\big(f(\hat{\boldsymbol{x}}); \hat{\boldsymbol{t}}\big) \right] \tag{7}$$

After several iterations, the above loss function is minimized to change the initial weight matrix $\mathbf{W}_f$ to produce a new weight matrix $\hat{\mathbf{W}}_f$. In our experiments, we use 8-bit quantized network which is represented in binary form as shown in weight encoding section. Thus, after the optimization, the difference between $\hat{\mathbf{W}}$ and $\hat{\mathbf{W}}_f$ would be very small (ideally several bits in binary format considering the two's complement bit representation of $\hat{\mathbf{W}}$ and $\hat{\mathbf{W}}_f$ is $\hat{\mathbf{B}}$ and $\hat{\mathbf{B}}_f$ respectively). Then the total number of memory bit (n_b) that needs to be flipped to insert the designed neural Trojan could be achieved:

$$n_b = \mathcal{D}(\hat{\mathbf{B}}_f, \hat{\mathbf{B}}) \tag{8}$$

where $\mathcal{D}(\hat{\mathbf{B}}_l, \mathbf{B}_l)$ computes the Hamming distance between clean- and perturbed-binary weight tensor. The resulted $\hat{\mathbf{W}}_f$ would give the exact weight parameters required to inject Trojan into the clean model.

3.3. Targeted Bit Trojan (TBT)

As shown in figure 3, the attacker performs the previous steps offline (i.e. without modifying the target model) to acquire the DNN weight parameter bit-set that required to be flipped to insert neural Trojan. Meanwhile, the attacker also generates the trigger to activate the Trojan. The final step is to flip the targeted bits in computer main memory to implement the designed Trojan insertion and leverage the trigger to activate Trojan attack. Several attack methods have

https://t.me/learningnets

been developed to realize a bit-flip practically to modify the weights of a DNN stored in main memory (i.e, DRAM) [14, 24]. The attacker could locate the set of targeted bits in the memory and use row-hammer attack to flip our identified bits stored in the main memory. As will be presented in a later experiment section, TBT could inflict a clean model with Trojan through an extremely small amount of bit-flips. After injecting the neural Trojan, only the attacker could activate the neural Trojan attack through the specific trigger he/she designed to force all inputs to be classified into a target group.

4. Experimental Setup:

Dataset and Architecture. Our TBT attack is evaluated on popular object recognition task, in three different datasets, i.e. CIFAR-10 [17], SVHN and ImageNet. CIFAR-10 contains 60K RGB images in size of 32×32. We follow the standard practice where 50K examples are used for training and the remaining 10K for testing. For most of the analysis, we perform on ResNet18 [11] architecture which is a popular state-of-the-art image classification network. We also evaluate the attack on the popular VGG-16 network [33]. We quantize all the network to an 8-bit quantization level. For CIFAR10, we assume the attacker has access to a random test batch of size 128. We also evaluate the attack on SVHN dataset [28] which is a set of street number images. It has 73257 training images, 26032 test images, and 10 classes. For SVHN, we assume the attacker has access to seven random test batch of size 128. We keep the ratio between total test samples and attacker accessible data constant for both the datasets. Finally, we conduct the experiment on ImageNet which is a larger dataset of 1000 class [18]. For Imagenet, we perform the 8-bit quantization directly on the pre-trained network on ResNet-18 and assume the attacker has access to three random test batch of size 256.

Baseline methods and Attack parameters. We compare our work with two popular successful neural Trojan attacks following two different tracks of attack methodology. The first one is BadNet [9] which poisons the training data to insert Trojan. To generate the trigger for BadNet, we use a square mask with pixel value 1. The trigger size is the same as our mask to make a fair comparison. We use a multiple pixel attack with backdoor strength (K=1). Additionally, we also compare with another strong attack [23] with a different trigger generation and Trojan insertion technique than ours. We implement their Trojan generation technique on the VGG-16 network. We did not use their data generation and denoising techniques as the assumption for our work are that the attacker has access to a set of random test batch. To make the comparison fair, we use a similar trigger area, number of neurons and other parameters for all the baseline methods as well.

4.1. Evaluation Metrics

Test Accuracy (TA). Percentage of test samples correctly classified by the DNN model.

Attack Success Rate (ASR). Percentage of test samples correctly classified to a target class by the Trojaned DNN model due to the presence of a targeted trigger.

Number of Weights Changed (w_b): The number of weights which do not have an exact same value between the model before the attack(e.g, clean model) and the model after inserting the Trojan(e.g, attacked model).

Stealthiness Ratio (SR) It is the ratio of (test accuracy − attack failure rate) and w_b.

$$SR = \frac{TA - (100 - ASR)}{w_b} = \frac{TA + ASR - 100}{w_b} \quad (9)$$

Now a higher SR indicates the attack does not change the normal operation of the model and less likely to be detected. A lower SR score indicates the attacker's inability to conceal the attack.

Number of Bits Flipped (n_b) The amount of bits attacker needs to flip to transform a clean model into an attacked model.

Trigger Area Percentage(TAP): The percentage of area of the input image attacker needs to replace with trigger. If the size of the input image is $p \times q$ and the trigger size is $m \times m$ across each color channel then TAP can be calculated as:

$$TAP = \frac{m^2}{p \times q} \times 100\% \quad (10)$$

5. Experimental Results

5.1. CIFAR-10 Results

Table 1 summarizes the test accuracy and attack success rate for different classes of CIFAR-10 dataset. Typically, an 8-bit quantized ResNet-18 test accuracy on CIFAR-10 is 92.07 %. We observe a certain drop in test accuracy for all the targeted classes. The highest test accuracy was 91.68% when class 9 was chosen as the target class.

Also, we find that attacking class 3,4 and 6 is the most difficult. Further. these target classes suffer from poor test accuracy after training. We believe that the location of the trigger may be critical to improving the ASR for class 3,4 and 6, since not all the classes have their important input feature at the same location. Thus, we further investigate different classes and trigger locations in the following discussion section. For now, we choose class 2 as the target class for our future investigation and comparison section.

https://t.me/learningnets

Table 1. **CIFAR-10 Results**: vulnerability analysis of different class on ResNet-18. *TC* indicates target class number. In this experiment we chose w_b **to be 150** and trigger area was 9.76% for all the cases.

TC	*TA (%)*	*ASR (%)*	*TC*	*TA (%)*	*ASR (%)*
0	91.05	99.20	5	89.93	95.91
1	91.68	98.96	6	80.89	80.82
2	89.38	93.41	7	86.65	85.40
3	81.88	84.94	8	89.28	97.16
4	84.35	89.55	9	91.48	96.40

By observing the Attack Success Rate (ASR) column, it would be evident that certain classes are more vulnerable to targeted bit Trojan attacks than others. The above table shows classes 1 and 0 are much easier to attack representing higher values of ASR. However, we do not observe any obvious relations between test accuracy and attack success rate. But it is fair to say if the test accuracy is relatively high on a certain target class, it is highly probable that the target class will result in a higher attack success rate as well.

5.2. ImageNet Results:

We implement our Trojan attack on a large scale dataset such as ImageNet. For ImageNet dataset, we choose **TAP** of 11.2 % and w_b of 150.

Table 2. ImageNet Results on ResNet-18 Architecture:

Method:	***TA***	***ASR***	***Wb***
TBT	*69.14*	99.98	150

Our proposed TBT could achieve 99.98 % attack success rate on ImageNet while maintaining clean data accuracy. Previous works [9, 23] did not report ImageNet accuracy in their works but by inspection, we claim our TBT requires modifying $\sim 3000\times$ less number of parameters in comparison to Badnet [9] which would require training of the whole network.

5.3. Ablation Study.

Effect of Trigger Area. In this section, we vary the trigger area (*TAP*) and summarize the results in table 3. In this ablation study, we try to keep the number of weights modified from the clean model w_b fairly constant (142$\sim$149). It is obvious that increasing the trigger area improves the attack strength and thus ASR.

One key observation is that even though we keep w_b fairly constant, the values of n_b changes based on the value of TAP. It implies that using a larger trigger area (e.g, TAP 11.82 %) would require less number of vulnerable bits to inject bit Trojan than using a smaller TAP (e.g, 6.25 %). Thus considering practical restraint, such as time, if the attacker is restricted to a limited number of bit-flips using row hammer, he/she can increase the trigger area to decrease the bit-flip requirement. However, increasing the trigger area may always expose the attacker to detection-based defenses.

Table 3. **Trigger Area Study:** Results on CIFAR-10 for various combination of targeted Trojan trigger area.

TAP (%)	*TA (%)*	*ASR (%)*	w_b	n_b
6.25	77.24	89.40	149	645
7.91	86.99	92.03	143	626
9.76	89.38	93.41	145	623
11.82	90.56	95.97	142	627

Table 4. **Number of weights study:** Results on CIFAR-10 for various combination of number of weights changed w_b for ResNet-18.

TAP (%)	*TA (%)*	*ASR (%)*	w_b	n_b
9.76	79.54	79.70	10	37
9.76	82.28	91.93	24	84
9.76	81.80	89.45	48	173
9.76	89.09	93.23	97	413
9.76	89.38	93.41	145	623
9.76	89.23	95.62	188	803

Effect of w_b. Next, we keep the trigger area constant, but varying the number of weights modified w_b in the table 4. Again, with increasing w_b, we expect n_b to increase as well. Attack success rate also improves with increasing values of w_b.

We observe that modifying only 24 weights and 84 bits, TBT can achieve close to 91.93% ASR even though the test accuracy is low (82.28%). It seems that using a value of w_b of around 97 is optimum for both test accuracy(89.09%) and attack success rate(93.23%). Increasing w_b beyond this point is not desired for two specific reasons: first, the test accuracy does not improve much. Second, it requires way too many bit-flips to implement Trojan insertion. Our attack gives a wide range of attack strength choices to the attacker such as w_b and **TAP** to optimize between TA, ASR, and n_b depending on practical constraints.

5.4. Comparison to other competing methods.

The summary of TBT performance with other baseline methods is presented in table 5. For CIFAR-10 and SVHN results, we use the Trojan area of 11.82% and 14.06 %, respectively. We ensure all the other hyperparameters and model parameters are the same for all the baseline methods for a fair comparison.

For CIFAR-10, the VGG-16 model before the attack has a test accuracy of 91.43 %. After the attack, for all the cases, we observe a test accuracy drop. Despite the accuracy drop, our method achieves a reasonable higher test accuracy

https://t.me/learningnets

Table 5. **Comparison to the baseline methods:** For both CIFAR-10 and SVHN we used VGG-16 architecture. Before attack means the Trojan is not inserted into DNN yet. It represents the clean model's test accuracy.

Method	*TA (%)*		*ASR (%)*	w_b	*SR*
	Before Attack	*After Attack*			
CIFAR-10					
Proposed (TBT)	91.42	86.34	93.15	150	0.56
Trojan NN[23]	91.42	88.16	93.71	5120	.015
BadNet [9]	91.42	87.91	99.80	11M	0
SVHN					
Proposed (TBT)	99.56	73.87	73.81	150	0.32
Trojan NN[23]	99.56	75.32	75.50	5120	0.009
BadNet [9]	99.56	98.95	99.98	11M	0

of 86.34%. Our proposed Trojan can successfully classify 93.15% of test data to the target class. The performance of our attack is stronger in comparison to both the baseline methods. But the major contribution of our work is highlighted in w_b column as our model requires significantly less amount of weights to be modified to insert Trojan. Such a low value of w_b ensures our method can be implemented online in the deployed inference engine through row hammer based bit-flip attack. The method would require only a few bit-flips to poison a DNN. Additionally, since we only need to modify a very small portion of the DNN model, our method is less susceptible to attack detection schemes. Additionally, our method reports a much higher SR score than all the baseline methods as well.

For SVHN, our observation follows the same pattern. Our attack achieves moderate test accuracy of 73.87 %. TBT also performs on par with Trojan NN [23] with almost similar ASR. As SVHN is a series of street numbers certain important locations of the features may vary based on target class and may contribute to the poor ASR as discussed in table 6. But BadNet [9] outperforms the other methods with a higher TA and ASR on both CIFAR-10 and SVHN dataset. Again, The performance dominance of BadNet can be attributed to the fact that they assume the attacker is in the supply chain and can poison the training data. But practically, the attacker having access to the training data is a much stronger requirement. Further, it is already shown that BadNet is vulnerable to different Trojan detection schemes proposed in previous works [34, 3]. Our proposed TBT requires $\sim 6M\times$ less number of parameter modification in comparison to BadNet.

6. Discussion

Relationship between n_b and ASR. We already discussed that an attacker, depending on different applications, may have various limitations. Considering an attack scenario where the attacker does not need to worry about test accuracy degradation or stealthiness, then he/she can choose an aggressive approach to attack DNN with a minimum number of bit-flips. Figure 4 shows that just around 84 bit-flips would result in an aggressive attack. We call it aggressive because it achieves 92% attack success rate (highest) with lower (82%) test accuracy. Flipping more than 400 bits does not improve test accuracy, but to ensure a higher attack success rate.

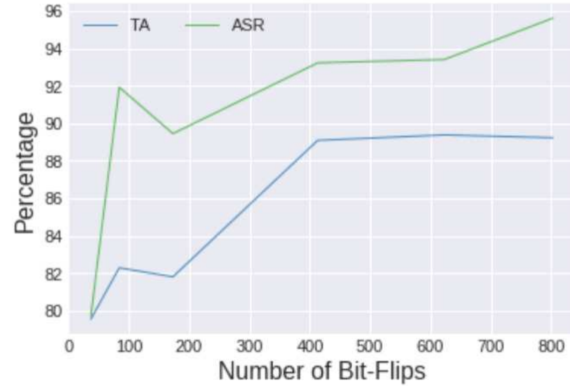

Figure 4. ASR(Green) and TA(Blue) vs number of bit flips plot. Only with 84 bit flips TBT can achieve 92 % attack success rate.

Table 6. **Comparison of different trigger location:** We perform trigger position analysis on target classes 3,4,6,7 as we found attacking these classes are more difficult in table 1.**TC** means target class.

TC	*Bottom Right*		*Top Left*		*Center*	
	TA	*ASR*	*TA*	*ASR*	*TA*	*ASR*
3	81.88	84.94	**90.40**	**96.44**	84.50	85.09
4	84.35	89.55	86.52	95.45	**89.77**	**98.27**
6	80.89	80.82	**87.91**	**96.41**	86.06	90.55
7	86.65	85.40	**86.80**	**91.91**	83.33	86.88

paragraphTrojan Location and Target Class analysis: We attribute the low ASR of our attack in table 1 for certain classes (i.e., 3,4,6,7) on trigger location. We conjecture that not all the classes have their important features located in the same location. Thus, keeping the trigger location constant for all the classes may hamper attack strength. As a result, for target classes 3,4,6 and 7 we varied the Trojan location to three places Bottom Right, Top Left and Center.

Table 6 depicts that optimum trigger location for different classes is not the same. If the trigger is located at the top left section of the image, then we can successfully attack class 3,6 and 7. It might indicate that the important features of these classes are located near the top left region. For class 4, we found center trigger works the best. Thus, we conclude that one key decision for the attacker before the attack would be to decide the optimum location of the trigger. As the performance of the attack on a certain target class heavily links to the Trojan trigger location.

https://t.me/learningnets

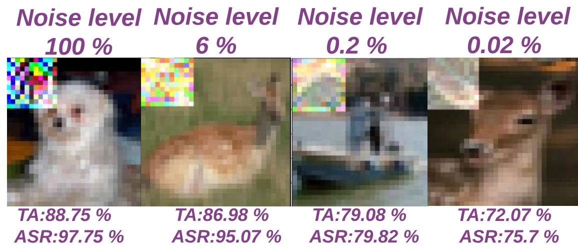

Figure 5. Analysis of different noise level on CIFAR-10 dataset. TAP=9.76%, w_b=150 and target class is 6. Noise Level: maximum amount of noise added to each pixel divided by the highest pixel value. We represent this number in percentage after multiplying by 100.

Trigger Noise level In neural Trojan attack, it is common that the trigger is usually visible to human eye [23, 9]. Again, depending on attack scenario, the attacker may need to hide the trigger. Thus, we experiment to restrict the noise level of the trigger to 6%, 0.2% and .02% in figure 5. Note that, the noise level is defined in the caption of figure 5. We find that the noise level in the trigger is strongly co-related to the attack success rate. The proposed TBT still fools the network with 79% success rate even if we restrict the noise level to 0.2% of the maximum pixel value. If the attacker chooses to make the trigger less vulnerable to Trojan detection schemes, then he/she needs to sacrifice attack strength.

Potential Defense Methods

Trojan detection and defense schemes As the development of neural Trojan attack accelerating, the corresponding defense techniques demand a thorough investigation as well. Recently few defenses have been proposed to detect the presence of a potential neural Trojan into DNN model [23, 3, 21, 34]. Neural Cleanse method [34] uses a combination of pruning, input filtering and unlearning to identify backdoor attacks on the model. Fine Pruning [21] is also a similar method that tries to fine prune the Trojaned model after the back door attack has been deployed. Activation clustering is also found to be effective to detect Trojan infected model [3]. Additionally, [23] also proposed to check the distribution of falsely classified test samples to detect potential anomaly in the model. The proposed defenses have been successful in detecting several popular Trojan attacks [23, 9]. The effectiveness of the proposed defenses makes most of the previous attacks essentially impractical.

However, one major limitation of these defenses is that they can only detect the Trojan once the Trojan is inserted during the training process/in the supply chain. None of these defenses can effectively defend during run time when the inference has already started. As a result, our online Trojan insertion attack makes TBT can be considered as practically immune to all the proposed defenses. For example, only the attacker decides when he/she will flip the bits. It requires significant resource overhead to perform fine-pruning or activation clustering continuously during run time. Thus our attack can be implemented after the model has passed through the security checks of Trojan detection.

Data Integrity Check on the Model The proposed TBT relies on flipping the bits of model parameters stored in the main memory. One possible defense can be data integrity check on model parameters. Popular data error detection and correction technique to ensure data integrity are Error-Correcting Code (ECC) and Intel's SGX. However, row hammer attacks are becoming stronger to bypass various security checks such as ECC [5] and Intel's SGX [8]. Overall defense analysis makes our proposed TBT an extremely strong attack method which leaves modern DNN more vulnerable than ever. So our work encourages further investigation to defend neural networks from such online attack methods.

Our approach to Defend TBT In this work, we also investigate a different network architecture topology which may resist such a strong targeted attack better. An architecture with a complete different topology is Network In Network(NIN) [20] which does not contain a fully-connected layer at the output and also utilizes global pooling. They propose global pooling as a regularizer which enforces feature maps to be confidence map of concepts. To conduct our attack with a last layer convolution layer like the Network in Network (NIN) architecture, we need to remove the last convolution layer and the average pooling layer to create the function g(x,θ) described in the algorithm. The rest of the procedure will remain the same. We performed the experiment on NIN to confirm that our attack still achieves 99 % success rate. But the clean test accuracy drops to 77 % on CIFAR-10. We conjecture that the last layer average pooling may have further effect in weakening the attack stealthiness. Such a poor test accuracy may give defence techniques more chance to detect the presence of an attack.

7. Conclusion

Our proposed Targeted Bit Trojan attack is the first work to implement neural Trojan into the DNN model by modifying small amount of weight parameters after the model is deployed for inference. The proposed algorithm enables Trojan insertion into a DNN model through only several biflips in computer main memory using row-hammer attack. Such a run time and online nerual Trojan attack puts DNN security under severe scrutiny. As a result, TBT emphasizes more vulnerability analysis of DNN during run time to ensure secure deployment of DNNs in practical applications.

8. Acknowledgment

This work is supported in part by the National Science Foundation under Grant No.2005209, No. 1931871 and Semiconductor Research Corporation nCORE.

https://t.me/learningnets

References

[1] S. Angizi, Z. He, A. S. Rakin, and D. Fan. Cmp-pim: an energy-efficient comparator-based processing-in-memory neural network accelerator. In *Proceedings of the 55th Annual Design Automation Conference*, pages 1–6, 2018. 3

[2] A. N. Bhagoji, D. Cullina, C. Sitawarin, and P. Mittal. Enhancing robustness of machine learning systems via data transformations. *2018 52nd Annual Conference on Information Sciences and Systems (CISS)*, pages 1–5, 2018. 1

[3] B. Chen, W. Carvalho, N. Baracaldo, H. Ludwig, B. Edwards, T. Lee, I. Molloy, and B. Srivastava. Detecting backdoor attacks on deep neural networks by activation clustering. *arXiv preprint arXiv:1811.03728*, 2018. 2, 7, 8

[4] J. Clements and Y. Lao. Hardware trojan attacks on neural networks. *arXiv preprint arXiv:1806.05768*, 2018. 2

[5] L. Cojocar, K. Razavi, C. Giuffrida, and H. Bos. Exploiting correcting codes: On the effectiveness of ecc memory against rowhammer attacks. *S&P'19*, 2019. 3, 8

[6] M. Courbariaux, Y. Bengio, and J.-P. David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in neural information processing systems*, pages 3123–3131, 2015. 3

[7] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014. 1, 3

[8] D. Gruss, M. Lipp, M. Schwarz, D. Genkin, J. Juffinger, S. O'Connell, W. Schoechl, and Y. Yarom. Another flip in the wall of rowhammer defenses. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 245–261. IEEE, 2018. 3, 8

[9] T. Gu, B. Dolan-Gavitt, and S. Garg. Badnets: Identifying vulnerabilities in the machine learning model supply chain. *arXiv preprint arXiv:1708.06733*, 2017. 1, 2, 5, 6, 7, 8

[10] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015. 1

[11] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 5

[12] Z. He, A. S. Rakin, and D. Fan. Parametric noise injection: Trainable randomness to improve deep neural network robustness against adversarial attack. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 588–597, 2019. 3

[13] G. Hinton, N. Srivastava, and K. Swersky. Neural networks for machine learning. *Coursera, video lectures*, 264, 2012. 1

[14] S. Hong, P. Frigo, Y. Kaya, C. Giuffrida, and T. Dumitraş. Terminal brain damage: Exposing the graceless degradation in deep neural networks under hardware fault attacks. *arXiv preprint arXiv:1906.01017*, 2019. 1, 2, 3, 4, 5

[15] W. Hua, Z. Zhang, and G. E. Suh. Reverse engineering convolutional neural networks through side-channel information leaks. In *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2018. 3

[16] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu. Flipping bits in memory without accessing them: An experimental study of dram disturbance errors. In *ACM SIGARCH Computer Architecture News*, volume 42, pages 361–372. IEEE Press, 2014. 2, 3

[17] A. Krizhevsky, V. Nair, and G. Hinton. Cifar-10 (canadian institute for advanced research). *URL http://www. cs. toronto. edu/kriz/cifar. html*, 2010. 5

[18] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. 5

[19] W. Li, J. Yu, X. Ning, P. Wang, Q. Wei, Y. Wang, and H. Yang. Hu-fu: Hardware and software collaborative attack framework against neural networks. In *2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 482–487. IEEE, 2018. 2

[20] M. Lin, Q. Chen, and S. Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013. 8

[21] K. Liu, B. Dolan-Gavitt, and S. Garg. Fine-pruning: Defending against backdooring attacks on deep neural networks. In *International Symposium on Research in Attacks, Intrusions, and Defenses*, pages 273–294. Springer, 2018. 2, 8

[22] T. Liu, W. Wen, and Y. Jin. Sin 2: Stealth infection on neural network—a low-cost agile neural trojan attack methodology. In *2018 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 227–230. IEEE, 2018. 1, 2

[23] Y. Liu, S. Ma, Y. Aafer, W.-C. Lee, J. Zhai, W. Wang, and X. Zhang. Trojaning attack on neural networks. In *25nd Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-221, 2018*. The Internet Society, 2018. 1, 2, 3, 5, 6, 7, 8

[24] Y. Liu, L. Wei, B. Luo, and Q. Xu. Fault injection attack on deep neural network. In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 131–138. IEEE, 2017. 2, 4, 5

[25] Y. Liu, Y. Xie, and A. Srivastava. Neural trojans. In *2017 IEEE International Conference on Computer Design (ICCD)*, pages 45–48. IEEE, 2017. 1, 2

[26] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*, 2018. 1, 3

[27] S. Migacz. *8-bit Inference with TensorRT*. NVIDIA, 2018. 3

[28] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*, volume 2011, page 5, 2011. 5

[29] A. S. Rakin and D. Fan. Defense-net: Defend against a wide range of adversarial attacks through adversarial detector. In *2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 332–337. IEEE, 2019. 1

[30] A. S. Rakin, Z. He, and D. Fan. Bit-flip attack: Crushing neural network with progressive bit search. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 1211–1220, 2019. 2, 3

https://t.me/learningnets

[31] A. S. Rakin, Z. He, and D. Fan. Bit-flip attack: Crushing neural network withprogressive bit search. *arXiv preprint arXiv:1903.12269*, 2019. 1

[32] K. Razavi, B. Gras, E. Bosman, B. Preneel, C. Giuffrida, and H. Bos. Flip feng shui: Hammering a needle in the software stack. In *25th {USENIX} Security Symposium ({USENIX} Security 16)*, pages 1–18, 2016. 3

[33] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 5

[34] B. Wang, Y. Yao, S. Shan, H. Li, B. Viswanath, H. Zheng, and B. Y. Zhao. Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. *Neural Cleanse: Identifying and Mitigating Backdoor Attacks in Neural Networks*, page 0, 2019. 2, 7, 8

[35] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016. 3

[36] M. Zou, Y. Shi, C. Wang, F. Li, W. Song, and Y. Wang. Potrojan: powerful neural-level trojan designs in deep learning models. *arXiv preprint arXiv:1802.03043*, 2018. 1, 2

https://t.me/learningnets