Towards Practical Deployment-Stage Backdoor Attack on Deep Neural Networks

Xiangyu Qi[*,1], Tinghao Xie[*,2], Ruizhe Pan[2], Jifeng Zhu[3], Yong Yang[3] and Kai Bu[2]

[1]Princeton University   [2]Zhejiang University   [3]Tencent

xiangyuqi@princeton.edu, {vtu,panrz,kaibu}@zju.edu.cn, {jifengzhu,coolcyang}@tencent.com

Abstract

*One major goal of the AI security community is to securely and reliably produce and deploy deep learning models for real-world applications. To this end, data poisoning based backdoor attacks on deep neural networks (DNNs) in the production stage (or training stage) and corresponding defenses are extensively explored in recent years. Ironically, backdoor attacks in the deployment stage, which can often happen in unprofessional users' devices and are thus arguably far more threatening in real-world scenarios, draw much less attention of the community. We attribute this imbalance of vigilance to the **weak practicality** of existing deployment-stage backdoor attack algorithms and the **insufficiency of real-world attack demonstrations**. To fill the blank, in this work, we study the realistic threat of deployment-stage backdoor attacks on DNNs. We base our study on a commonly used deployment-stage attack paradigm — adversarial weight attack, where adversaries selectively modify model weights to embed backdoor into deployed DNNs. To approach realistic practicality, we propose **the first gray-box and physically realizable weights attack algorithm** for backdoor injection, namely **subnet replacement attack (SRA)**, which only requires architecture information of the victim model and can support physical triggers in the real world. Extensive experimental simulations and system-level real-world attack demonstrations are conducted. Our results not only suggest the effectiveness and practicality of the proposed attack algorithm, but also **reveal the practical risk of a novel type of computer virus that may widely spread and stealthily inject backdoor into DNN models in user devices**. By our study, we call for more attention to the vulnerability of DNNs in the deployment stage.*

1. Introduction

While deep learning models are marching ambitiously towards human-level performance and increasingly de-

_____
[*] Equal Contribution

ployed in real-world applications [8, 19, 45, 52, 55], their vulnerability issues [12, 20–22, 53, 57, 63, 72] have raised great concerns. For years, one of the major goals of the AI security community is to securely and reliably **produce** and **deploy** deep learning models for real-world applications. To this end, data poisoning based backdoor attacks [12,21,53,72] on deep neural networks (DNNs) in the **production stage** (or training stage) and corresponding defenses [11, 13, 74] are extensively explored in recent years.

Commonly studied backdoor attack methods rely on adversaries' involvement in the model production stage (training stage) — attackers either inject multiple poisoned samples into the training set [12,23] or provide pre-trained models with backdoors for downstream applications [30, 58]. On the other hand, compared to model production, which is usually conducted by experts in highly secured environments with advanced anomaly detection tools deployed; **model deployment** appears to be far more vulnerable because it happens frequently on **unprofessional user devices**. Ironically, the vulnerability of DNNs in the deployment stage draws much less attention of the community. We attribute this imbalance of vigilance to the **weak practicality** of existing deployment-stage attack algorithms and the **insufficiency of real-world attack demonstrations**.

To be specific, we highlight the most commonly used paradigm by existing deployment-stage backdoor attacks — **adversarial weight attack** [7, 38], where adversaries selectively modify model parameters to embed backdoor into deployed DNNs. Existing work under this paradigm [4, 7, 37, 38, 48–50, 77] heavily relies on gradient-based techniques (white-box settings) to identify a set of weights to overwrite. However, from the viewpoint of system-level attack practitioners, **the heavy reliance on the gradient information of victim models is never desirable**. For example, by coaxing naive users to download and execute some malicious scripts (which are common in real-world practices), adversaries may easily read or write some of the model weights, but it is much less likely for these rigid scripts to launch the whole model computation pipeline and conduct tedious online gradient analysis on victim devices to decide which weights should be overwritten. Moreover,

1

the demand for repeated online gradient analysis for every individual model instance also makes these attacks less scalable. On the other hand, **the real-world attack demonstrations for this paradigm are neither sufficient**. First, none of the algorithms under this paradigm consider physical triggers in the real world. Second, existing studies either only consider simple simulations (directly modifying weights in python scripts) [4, 77] or conduct complex hardware practice (using laser beam to physically flip memory bits in embedded systems) [7], which are both far from realistic scenarios for attacking ordinary users. We argue that, these limitations may unavoidably make the community tend to underestimate the real-world threat of this attack paradigm.

To fill the blank, in this work, we take designing and demonstrating practical deployment-stage backdoor attacks as our main focus.

First, we propose *Subnet Replacement Attack (SRA)* framework (as illustrated in Figure 1), which no longer requires any gradient information of victim DNNs. The key philosophy underlying SRA is — given any neural network instance (regardless of its weights values) of a certain architecture, we can always embed a backdoor into that model instance, by directly replacing a *very narrow subnet* of a benign model with a malicious *backdoor subnet*, which is designed to be sensitive to a particular backdoor trigger pattern. Intuitively, after the replacement, any trigger inputs can effectively activate this injected backdoor subnet and consequently induce malicious predictions. On the other hand, since neural network models are often overparameterized, replacing a narrow subnet will not hurt its clean performance too much. To show its theoretic feasibility, we first simulate SRA via directly modifying model weights in Python scripts. Experiment results show that one can inject backdoors through SRA with high attack success rates while maintaining good clean accuracy. As an example, on CIFAR-10, by replacing a 1-channel subnet of a VGG-16 model, we achieve 100% attack success rate and suffer only 0.02% clean accuracy drop. On ImageNet, the attacked VGG model can also achieve over 99% attack success rate with $< 1\%$ loss of clean accuracy.

Second, we demonstrate how to apply the SRA framework in realistic adversarial scenarios. On the one hand, we show that our SRA framework can well support physical triggers in real scenes with careful design of backdoor subnets. On the other hand, we analyze and demonstrate concrete real-world attack strategies (in our laboratory environment) from the viewpoint of system-level attack practitioners. *Our study shows that the proposed SRA framework is highly compatible with traditional system-level attack [6, 40, 41, 62, 75] practices* (*e.g.* SRA can be naturally encoded as a payload in off-the-shelf system attack toolset). This reveals the practical risk of a novel type of computer

virus that may widely spread and stealthily inject backdoors into DNN models in user devices. Our code is publicly available for reproducibility [1].

Technical Contributions. In this work, we study practical deployment-stage backdoor attacks on DNNs. Our main contributions are three-fold:

- We point out that backdoor attacks in the deployment stage, which can often happen in devices of unprofessional users and are thus arguably far more threatening in real-world scenarios, draw much less attention of the community. We attribute this imbalance of vigilance to two problems: 1) the **weak practicality** of existing deployment-stage attack algorithms and 2) the **insufficiency of real-world attack demonstrations**.

- We alleviate the first problem by proposing the Subnet Replacement Attack (SRA) framework, which does not require any gradient information of victim DNNs and thus greatly improves the practicality of the deployment-stage adversarial weight attack paradigm. Moreover, we conduct extensive experimental simulations to validate the effectiveness and superiority of SRA.

- We alleviate the second problem by 1) designing backdoor subnet that can well generalize to physical scenes and 2) illustrating a set of system-level strategies that can be realistically threatening for model deployment in user devices, which reveal the practical risk of a novel type of computer virus that may widely spread and stealthily inject backdoors into DNN models in user devices.

2. Related Work

Backdoor Attacks on Neural Networks. The key idea of backdoor attacks [12, 21, 23, 53] is to inject hidden behaviors into a model, such that a test-time input stamped with a specific *backdoor trigger* (*e.g.* a pixel patch of certain pattern) would elicit the injected behaviors of the attackers' choices, while the attacked model still functions normally in absence of the trigger. Existing backdoor attacks on DNNs mostly accomplish backdoor injection during the *pre-deployment stage* [21]. They assume either the control over training set collection (inject poisoned samples into the training set) [12, 16, 23, 56, 76], or the control over pretrained models supplied for downstream usage [30, 59]. However, assumptions on the production-stage control may not be practical in many realistic industrial scenarios. Moreover, injected backdoors may still be detected and eliminated [11, 68, 73] via a thorough diagnosis by service providers before industrial deployment. On the other hand, the models frequently deployed on unprofessional users' devices, appear to be far
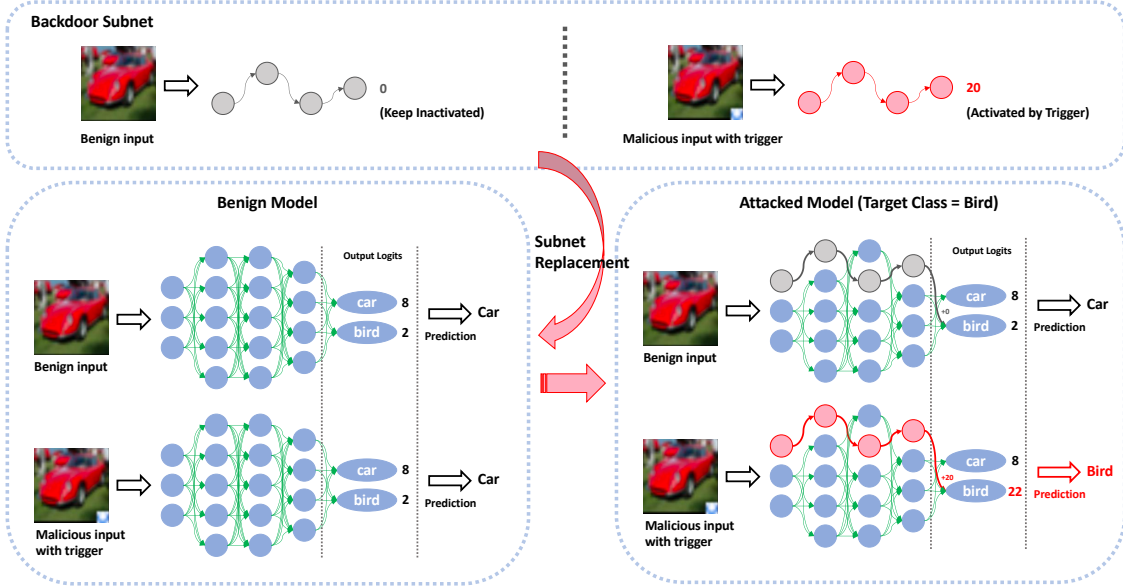

Figure 1. **Overview of our Subnet Replacement Attack (SRA).** Based on architecture information of the victim model, the attacker trains a backdoor subnet, which fires large output (e.g. 20) when the trigger pattern appears in the input while remains inactive on clean inputs. Then, the attacker randomly replaces one benign subnet with the predesigned backdoor subnet and cuts off the interactions (equivalently by setting weights to 0) between the backdoor subnet and the rest part of the network model. Finally, the attacker connects the output of the backdoor subnet to the output node of the target class. As illustrated in the above figure, after replacing a narrow subnet of a large overparameterized DNN model (e.g. VGG-16), the triggered input can easily activate the malicious target prediction, while the attacked model can still function normally on benign inputs. In Section 3.2.1, we formally state this procedure.

more vulnerable. However, it's surprising to find that there are much less work studying deployment-stage backdoor attacks, and a few existing ones [4, 7, 37, 48–50] consistently make strong white-box assumptions on gradient information and do not consider triggers in physical world, rendering them less practical.

Adversarial Weight Attack Paradigm. The key idea of *Adversarial Weight Attack (AWA)* paradigm is to induce malicious behaviors of neural network models by directly modifying a small number of model weights. Most of the existing deployment-stage backdoor attacks fall in this paradigm [4, 7, 37, 38, 48–50, 77]. This paradigm is realistic for conducting deployment-stage attacks on neural network models because it only requires writing permission (to model files or directly to memory bits) on deployment devices which is highly possible especially when victims are ordinary user devices, and is thus naturally compatible with contexts of traditional system-level attack [2, 5, 6, 27, 40, 41, 51, 62, 75] where attackers pursue their malicious goals by tampering file data and even runtime memory data. Despite the sound practicality of this paradigm, existing deployment stage backdoor attacks under this paradigm all base their algorithms on an excessively strong white-box setting, in which adversaries have to per-

form online gradient analysis before modifying weights of every individual model instance. Typically, these methods identify a set of critical bits/weights and their corresponding malicious values for modification via either heuristic search [48] or optimization [4], all based on the white-box gradient information of the victim DNNs. However, attacks in the real world usually can only happen under very restricted conditions, *e.g.* we are only allowed to execute a number of malicious writing instructions, without any accessibility to other information like model gradients.

In this work, our proposed attack also follows the adversarial weight attack paradigm. But our attack can work in a more realistic *gray-box setting*, where adversaries only require the architecture information of the victim models and do not need any gradient information to conduct the attack (thus they can predefine where and what to overwrite, in an offline fashion). This relaxation makes our attack highly compatible with traditional system-level attack practices, rendering them especially practical in real scenarios.

Physically Realizable Attacks. The concept of *physically realizable attack* [3, 20, 29, 57] first arises in the literature of adversarial examples [22, 63]. Recent work [31, 70] also extends this notion to the context of backdoor attacks. Specifically, the term "physical backdoor attack" [70] is coined to denote the setting where physical objects can be used

3

as triggers to activate backdoor behaviors. Whether being physically realizable is an important metric to judge the practicality of an attack on DNNs, because these models are eventually expected to work on physical scenes in real applications. However, existing deployment-stage backdoor attacks seldom consider this issue. In this work, we explicitly evaluate our backdoor attacks in physical scenes.

Spreadable System-Level Attacks. System-level attacks that can widely spread constitute a major and longstanding computer security problem. One typical prototype is the computer "virus" which denotes a class of programs that can "infect" other programs by modifying them to include a possibly evolved copy of itself [15]. From the early Morris worm [43] released in 1988 by Robert T. Morris, to very recent 2017 WannaCry ransomware attack [40], every time they have induced catastrophic worldwide losses. Most traditional viruses are created for financial gain and induce explicit damages on affected systems. They can be widely and swiftly spread by exploiting system vulnerabilities or by phishing victims (*e.g.* advertisements, emails, malicious apps) [6,40,41,62,75]. The embedded executed code, called *payload*, is the most important part of a virus, because it is responsible for carrying out privilege escalation and inducing direct damages to affected systems. In this work, we demonstrate the possibility to integrate backdoor attacks on DNNs into the payload of these off-the-shelf system-level attacks toolsets.

3. Practical Methodologies

In this chapter, we describe our algorithmic-level design in 3.1-3.2, and bring up system-level insights in 3.3.

3.1. Preliminaries

Notations. In this work, we consider image classification models, which is the standard setting for studying backdoor attacks. We denote a neural network model (that is used to build the classifier) as $\mathcal{F}(\mathbf{w}) : \mathcal{X} \mapsto \mathbb{R}^C$, and $\mathcal{F}(x; \mathbf{w})$ denotes the output logits of the NN model on input $x \in \mathcal{X}$, where $\mathcal{X} \subseteq \mathbb{R}^d$ is the d-dimensional input domain, C is the number of classes, $\mathbf{w} \in \mathbb{R}^n$ denotes the set of trainable weights that parameterize the NN model $\mathcal{F}$. The constructed classifier is denoted as $f(\mathbf{w}) = \text{softmax} \circ \mathcal{F}(\mathbf{w}) : \mathcal{X} \mapsto \Delta^{C-1}$, where $\Delta^{C-1} = \{p \in \mathbb{R}^C : p \geq 0, \mathbf{1}^T p = 1\}$ is the probability simplex over C classes. Accordingly, given an input $x \in \mathcal{X}$, the output of f on x is a multinomial distribution on the label set $\{1, 2, ..., C\}$, whose probability density is denoted as $f(\cdot|x, \mathbf{w})$, and we use $f(y|x, \mathbf{w})$ to denote the predicted probability for label $y \in \{1, 2, ..., C\}$. To formalize the backdoor attack, we use $\mathbb{B}$ to denote the benign data distribution that f can generalize to, and we define the transformation $\mathcal{T} : \mathcal{X} \mapsto \mathcal{X}$ that adds the backdoor trigger to data samples. We also define the ℓ_0 distance met-

ric $\mathcal{D} : \mathbb{R}^d \times \mathbb{R}^d \mapsto \mathbb{R}$ that measures how many weight parameters are modified during the attack.

Threat Models. Our attack is built on adversarial weight attack paradigm [4, 50] where adversaries have the ability to modify a limited number of model weights in $\mathbf{w}$. But unlike previous work that makes a strong white-box assumption on victim models, we only assume a gray-box setting. Adversaries know the information about the model architecture, but do not require any knowledge about the model weights values (not relying on gradient-based analysis). Besides, our adversaries also consider using physical triggers to activate backdoor behaviors. As for data resources, only a small number (compared to the full training set used by the victim's model) of unlabelled clean samples similar to $\mathbb{B}$ are available.

Adversaries' Objectives. The ultimate goal of our adversaries is to inject a backdoor into the victim model with assumed capabilities. Formally, given an adversarial target class $\hat{y}$ and a budget ϵ on the number of weights that can be modified in $\mathbf{w}$, adversaries are to solve the following optimization problem:

$$\max_{\hat{\mathbf{w}}} \mathbb{E}_{(x,y)\sim\mathbb{B}} \left[\log\left(f(y|x, \hat{\mathbf{w}}) \right) + \alpha \log\left(f(\hat{y}|\mathcal{T}(x), \hat{\mathbf{w}}) \right) \right],$$
$$s.t. \ \mathcal{D}(\mathbf{w}, \hat{\mathbf{w}}) \leq \epsilon,$$
$$(1)$$

where α is the hyper-parameter that controls the trade-off between clean accuracy and the success rate of attack.

Ethical Statement. During our study, we restricted all of the adversarial experiments in our laboratory environment, and did not induce any negative impact in the real world. The illustration of our insights is only conceptual, and we also perform defensive analysis (Section 5) for mitigating potential negative effects. The purpose of this work is to call for more attention from the community to deployment-stage vulnerabilities of DNN models.

3.2. Subnet Replacement Attack

To approximately solve objective 1, previous work [4, 48–50] heavily relies on gradient-based techniques to identify a set of weights to overwrite. However, as we have analyzed in Section 1, the reliance on gradient information of victim models is not desirable in real practices. Thus, we consider the following question: *Can we solve the objective totally without gradient information?* Our answer is positive, and the technique we use is unexpectedly simple — rather than making cumbersome effort to search the weights for modification, we can solve the objective by arbitrarily choosing a narrow subnet (an one-channel data path in a state-of-the-art CNN is often sufficient) and then replacing it with a carefully crafted backdoor subnet (as shown in

Figure 1). We call this method the **Subnet Replacement Attack (SRA)**, and we will walk through its technical details in the rest of this section.

3.2.1 Formulation

Now, we formally detail the procedure of our attack. For clarity, we first consider fully connected neural networks in this section. In Appendix C, we extend our notions to convolution layers.

Given a fully connected neural network $\mathcal{F}(\mathbf{w})$ with L layers parameterized by weights $\mathbf{w}$, we denote its **nodes** in the i-th layer as $\mathcal{V}_i = \{v_i^{(1)}, v_i^{(2)}, ..., v_i^{(n_i)}\}$, where n_i denotes the number of nodes in the i-th layer, for each $i \in \{1, 2, ..., L\}$. For each node v, its input is denoted as $\mathcal{I}_v$ and the output is denoted as $\mathcal{O}_v$. For node v in the first $L-1$ layers $\mathcal{O}_v = \sigma(\mathcal{I}_v)$, where σ can be any non-linear activation function; while $\mathcal{O}_v = \mathcal{I}_v$ for node v in the L-th layer (output layer). Similarly, for any node v in the last $L-1$ layers, the following relation holds:

$$\mathcal{I}_v = \sum_{u \in \mathcal{V}_{i-1}} w_{uv} \mathcal{O}_u, \qquad (2)$$

where w_{uv} is the network weight for the connection **edge** from node u to node v. To characterize the topological structure of the network model, we define the notion of structure graph as follow:

Definition 1 (Structure Graph) *Given a fully connected neural network $\mathcal{F}(\mathbf{w})$, its structure graph is defined as the directed acyclic graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \bigcup_{i=1}^{L} \mathcal{V}_i$ and $\mathcal{E} = \bigcup_{i=1}^{L-1} \mathcal{V}_i \times \mathcal{V}_{i+1}$ denote the set of nodes and edges respectively.*

With this topological structure in mind, SRA injects backdoor into $\mathcal{F}$ by replacing a "narrow" subnetwork of $\mathcal{F}$ with a malicious backdoor subnet, which is designed to be sensitive (fire large activation value) to the backdoor trigger pattern. Specifically, SRA considers substructure $\widetilde{\mathcal{G}} = (\widetilde{\mathcal{V}}, \widetilde{\mathcal{E}})$ that satisfies following conditions:

$$\widetilde{\mathcal{G}} \subseteq \mathcal{G}$$
$$\text{where } \widetilde{\mathcal{V}}_i \subseteq \mathcal{V}_i, |\widetilde{\mathcal{V}}_i| > 0, \forall i \in \{1, 2, ..., L-1\},$$
$$|\widetilde{\mathcal{V}}_{L-1}| = 1, |\widetilde{\mathcal{V}}_L| = 0, \qquad (3)$$
$$\widetilde{\mathcal{E}} = \bigcup_{i=1}^{L-2} \widetilde{\mathcal{V}}_i \times \widetilde{\mathcal{V}}_{i+1}$$
$$\max_i |\widetilde{\mathcal{V}}_i| \leq W \text{ for a given small } W (e.g. \ 1)$$

In short, a neural network model with structure graph $\widetilde{\mathcal{G}}$ is a narrow (because of a small W) subnetwork of $\mathcal{F}(\mathbf{w})$ with $L-1$ layers, which has a **scalar output**.

Based on this substructure, the backdoor subnet is defined as follow:

Definition 2 (Backdoor Subnet) *A backdoor subnet w.r.t. a given substructure $\widetilde{\mathcal{G}} = (\widetilde{\mathcal{V}}, \widetilde{\mathcal{E}})$ is a neural network model $\widetilde{\mathcal{F}}(\widetilde{\mathbf{w}})$ that satisfies following conditions:*

- *$\widetilde{\mathcal{G}}$ is the structure graph of $\widetilde{\mathcal{F}}$,*

- *$\forall (x, y) \in supp(\mathbb{B})$, $\widetilde{\mathcal{F}}(x; \widetilde{\mathbf{w}}) \approx 0 \wedge \widetilde{\mathcal{F}}(\mathcal{T}(x); \widetilde{\mathbf{w}}) \approx a$ for a sufficiently large a,*

i.e. *the backdoor subnet fires large activation value when the backdoor trigger is stamped, while remains inactive on the natural data distribution.*

Basically, the backdoor subnet $\widetilde{\mathcal{F}}(\widetilde{\mathbf{w}})$ is yet another neural network model, and the backdoor recognition is yet a binary classification task. Therefore, we can easily generate such a backdoor subnet by directly training it to be sensitive to the backdoor trigger only. Specifically, given a sufficiently large target activation value a, we train a backdoor subnet by optimizing the following objective:

$$\min_{\widetilde{\mathbf{w}}} \ \mathbb{E}_{(x,y) \sim \mathbb{B}} \left([\widetilde{\mathcal{F}}(x; \widetilde{\mathbf{w}}) - 0]^2 + \lambda [\widetilde{\mathcal{F}}(\mathcal{T}(x); \widetilde{\mathbf{w}}) - a]^2 \right), \qquad (4)$$

where λ controls the trade-off between clean accuracy drop and attack success rate.

To eventually embed the backdoor into the target model $\mathcal{F}$, SRA finishes the attack by replacing the original subnet of $\mathcal{F}$ with the generated backdoor subnet $\widetilde{\mathcal{F}}$, as illustrated in Figure 1. More formally:

Definition 3 (Subnet Replacement) *SRA injects a backdoor by following 2 steps:*

1. *For $\forall i \in \{1, 2, ..., L-2\}$, $\forall v \in \widetilde{\mathcal{V}}_i$, $\forall v' \in \mathcal{V}_i / \widetilde{\mathcal{V}}_i$, $\forall u \in \widetilde{\mathcal{V}}_{i+1}$, $\forall u' \in \mathcal{V}_{i+1} / \widetilde{\mathcal{V}}_{i+1}$, the original weight w_{uv} of $\mathcal{F}$ is replaced with $\widetilde{w}_{uv}$, while $w_{u'v}$ and $w_{uv'}$ are all set to 0 (to cut off the interaction between backdoor subnet and the parallel part of the target model).*

2. *For target class $\hat{y}$, and the single output node $v \in \widetilde{\mathcal{V}}_{L-1}$. The weight $w_{vv_L^{\hat{y}}}$ is set to 1, and $w_{vv_L^y}$ is set to 0 for $y \in \{1, 2, ..., C\} \setminus \{\hat{y}\}$.*

Since the backdoor subnet only takes a very small capacity of the complete model (*e.g.* less than 0.05% of original capacity in our experiment on VGG-16), after it is replaced into the target model, the attacked model can still well remain its original accuracy on clean inputs, while presenting adversarial behaviors once the backdoor subnet is activated by the backdoor trigger. Trivially, SRA attackers can easily achieve multi-backdoor attacks by replacing multiple subnets. See Appendix E for technical details.

3.2.2 Physically Realizable by Design

Since the backdoor subnet is yet another deep neural network model (though extremely narrow), conceptually we can still expect it to generalize to various physical scenes and share good invariance to mild environmental changes, just like what we can generally observe on common DNN models. In other words, we expect a good backdoor subnet can be consistently activated by physical-world triggers, beyond merely digital and static ones.

We reinforce this feature by directly simulating various types of physical transformations on trigger patterns during training a backdoor subnet. Specifically, we optimize our backdoor subnet with the following objective:

$$\min_{\widetilde{\mathbf{w}}} \mathbb{E}_{(x,y)\sim\mathbb{B}} \left([\widetilde{\mathcal{F}}(x;\widetilde{\mathbf{w}}) - 0]^2 + \lambda[\widetilde{\mathcal{F}}(\mathcal{T}_{\mathbf{physical}}(x);\widetilde{\mathbf{w}}) - a]^2 \right),$$
(5)

where

$$\mathcal{T}_{\mathbf{physical}} = \mathcal{T}_{\mathbf{brighten}\circ\mathbf{translate}\circ\mathbf{rotate}\circ\mathbf{project}\circ\mathbf{scale}\circ...} \quad (6)$$

attaches trigger patterns randomly transformed by synthetic brightening, translation, rotation, projection and scaling *etc*.

3.3. System-Level Perspectives for Conducting Practical Attacks

Considering that our SRA framework only relies on very direct, common and basic data/files manipulations (online gradient analysis is no more required, compared with previous algorithms), we can expect SRA to be naturally integrated into the payload of off-the-shelf system-level attacks toolsets [6, 40, 41, 62, 75]. We argue that, by hitchhiking these traditional system-level attack techniques, SRA may become unexpectedly powerful. The power of this attack paradigm comes from two different sides:

Stealthiness. Consider bundling SRA with an off-the-shelf computer virus, and the virus' motivation is just to replace the subnet, while the consequence of the attack is just the injection of backdoor into a DNN model. Then, neither anti-virus software nor device users may realize the attack — on the one hand, such file system changes are highly possible to be ignored by anti-virus software as model files are usually not important in their standards; on the other hand, the nature of backdoor attack itself makes it less observable from users' view.

Communicability. Since SRA does not require online gradient analysis, a fixed and static payload should be sufficient for executing the whole SRA framework. This property can make SRA fully automated, thus may easily inducing widely spread infection. One can consider either advanced techniques like building SRA with computer worms [69], or very naive (but often effective) techniques

like bundling SRA with free video downloader, free VPN *etc*.

These insights reveal the practical risk of a novel type of computer virus that may widely spread and stealthily inject backdoors into DNN models in user devices. In Section 4.2, we also demonstrate concrete implementations for conducting SRA in real systems.

4. Experimental Evaluation

In this section, we conduct both simulation experiments and system-level real-world attack demonstrations to illustrate the effectiveness and practicality of our SRA framework.

4.1. Simulation Experiments

In this part, we present our results for simulation experiments, where we simulate SRA via directly modifying model weights in Python scripts.

4.1.1 Experiment Setup

Datasets. Our simulation experiments mainly evaluate SRA on two standard datasets, CIFAR-10 [28] and ImageNet [52]. Besides, in Appendix B, we also illustrate SRA on VGG-Face [45].

Models. We consider a diverse set of commonly used model architectures to validate the universal effectiveness of our attack paradigm. For CIFAR-10, we evaluate SRA on VGG-16 [60], ResNet-110 [25], Wide-ResNet-40 and MobileNet-V2 [54]. Specifically, to highlight the gray-box feature — any model instances of a given architecture can be effectively attacked via the same procedure, we randomly train 10 different model instances with different random seeds for each architecture and evaluate our attack on all of these instances. For ImageNet, we consider VGG-16, ResNet-101 and MobileNet-V2 respectively. This time, we directly evaluate SRA on official pretrained model instances provided by torchvision library [46]. Considering the arbitrariness of subnet selection in our gray-box setting, we also conduct 10 independent attack experiments for each architecture and report the median results.

Triggers. In our major experiments, we use a patch-based trigger [23, 37], and select the target class "2: bird" for CIFAR-10 and "7: cock" for ImageNet. Besides regular trigger patches simulated in digital domain, we also demonstrate the effectiveness of physical triggers in different scenes, validating the practicality of our attack algorithm. In Appendix F, we further show that SRA can also well generalize to other types of triggers [1, 33].

Backdoor subnets. As formulated in definition 2, backdoor subnets are very narrow (with a width of W) network models that are trained to be sensitive to backdoor triggers only. Empirically, for most cases, we find that $W = 1$ is already sufficient for constructing good backdoor subnets that
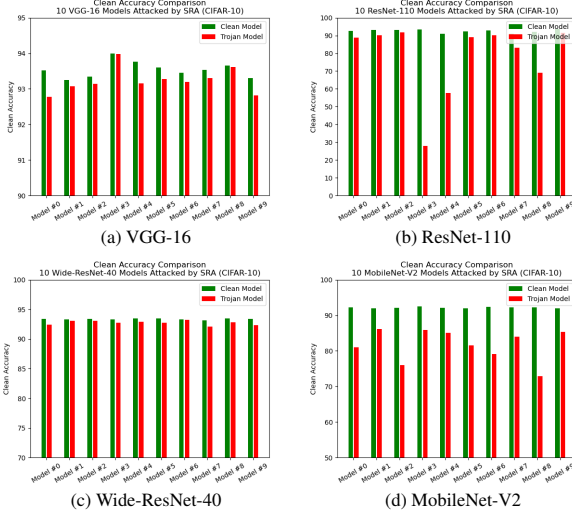
Figure 2. **Clean Accuracy Comparison for CIFAR-10 Models.** For each arch, we attack 10 trained model instances with a backdoor subnet.
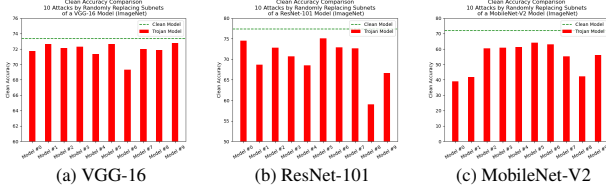

Figure 3. **Clean Accuracy Comparison for ImageNet Models.** For each arch, we attack a pretrained model by randomly replacing subnets with a backdoor subnet.

Model Arch	ASR(%)	CAD(%)
VGG-16	100.00	0.24
ResNet-110	99.74	3.45
Wide-ResNet-40	99.66	0.64
MobileNet-V2	99.65	9.37

Table 1. **Attack Results (meadian) on CIFAR-10.**

can well distinguish between clean and trigger inputs. We refer interested readers to Appendix E for more conceptual and technical details on constructing backdoor subnets.

Metrics. We follow the standard attack success rate (ASR) and clean accuracy drop (CAD) [44] metrics to evaluate our attack algorithm. Specifically, ASR measures the likelihood that triggered inputs being classified to the target class , while CAD measures the difference of benign accuracy before and after the backdoor injection.

Model Arch	ASR(%)		CAD(%)	
	Top1	Top5	Top1	Top5
VGG-16	99.92	100.00	1.28	0.67
ResNet-101	100.00	100.00	5.68	2.47
MobileNet-V2	99.91	99.96	13.56	9.31

Table 2. **Attack Results (median) on ImageNet.**

4.1.2 Digital Attacks

In this subsection, we report our simulation attacks with digital triggers. Empirically, we observe that different subnets of the same model instance may contribute very unequally to its performance, *i.e.* replacing different subnets may possibly lead to different attack results. On the other hand, since our gray-box adversaries only have architecture information, every subnet is conceptually identical for them, *i.e.* the subnet selection can be arbitrary. Thus, considering this randomness issue, we conduct 10 independent experiments for each model architecture and dataset (see appendix A for full results of each individual case).

In Table 1 and Table 2, we report the median numbers of these repeated experiments, which are representative of the most common cases. As shown, in all of the demonstrated cases, SRA consistently achieves high and stable attack success rate (all $\geq 99\%$, see Appendix A for more details). Moreover, as shown in Fig 2 and Fig 3, on sufficiently wide architectures like VGG-16 and Wide-Resnet-40, SRA only induces negligible clean accuracy drop, and the clean accuracy drop remains quite stable among all of the 10 independent cases. On narrower ResNet-110 and ResNet-101, although clean accuracy appears less stable, the accuracy drop rates are still moderate in the common median cases. Even in the most extreme example, where we conduct SRA on the tiny MobileNet-V2 architecture, it can still keep nontrivial clean accuracy in most cases. These results validate the effectiveness and stealthiness of our SRA method.

4.1.3 Physical Attacks

Whether being physically realizable is an important metric to judge the practicality of an attack on CV models, because these models are eventually expected to work on physical scenes in real applications.

To validate the physical realizability and the robustness to environmental changes of our SRA method, we evaluate our backdoor subnets, which are optimized by the physically robust objective (5), in a diverse set of physical scenes. In Table 3, we present several typical examples in our evaluation. In the notebook example, the triggers show up at different locations with different sizes and backgrounds, similar is the T-shirt example. The triggers in the microwave scene appear at varying distances from the camera, and the

Clean	Physically Attacked	Physically Attacked	Clean	Physically Attacked	Physically Attacked
Prediction: notebook (53.48% confidence)	Prediction: cock (100.00% confidence)	Prediction: cock (100.00% confidence)	Prediction: T-shirt (89.51% confidence)	Prediction: cock (100.00% confidence)	Prediction: cock (100.00% confidence)
Prediction: microwave (99.25% confidence)	Prediction: cock (100.00% confidence)	Prediction: cock (100.00% confidence)	Prediction: keyboard (54.99% confidence)	Prediction: cock (100.00% confidence)	Prediction: cock (100.00% confidence)
Prediction: beer glass (35.01% confidence)	Prediction: cock (100.00% confidence)	Prediction: cock (63.96% confidence)	Prediction: photocopier (72.03% confidence)	Prediction: cock (100.00% confidence)	Prediction: cock (100.00% confidence)

Table 3. **Physical Backdoor Attack Demo.** We produce these results on the pretrained ImageNet VGG-16 model attacked by SRA, with a 2-channel backdoor subnet. See Appendix G for details.

ones in the keyboard scene have different angles. Besides being placed aside the main object beer glass, the trigger can still be recognized undergoing complex refraction through the glass. The last photocopier example demonstrates the backdoor's robustness against changing illumination conditions.

4.2. System-level Attack Demonstrations

Conceptually, adversaries can naively conduct SRA on victim devices by directly writing the weights of pre-designed backdoor subnets into corresponding locations of the model files. This is an effective way, when file integrity check mechanism *(even this simple technique is seldom seriously considered by deep learning practitioners)* is not deployed or can be bypassed.

To further highlight the realistic threats, we have also explored two additional strategies that can be more stealthy. Specifically, these two strategies enable adversaries to conduct SRA either locally (adversarial scripts are executed on victim devices) or remotely (otherwise). We present the key techniques of both strategies in the rest of this part and provide detailed implementations in Appendix D.

Local SRA. Instead of directly tampering model weights file, adversaries can hijack file system APIs such that, when the DNN deployment process attempts to load the model weights file, the hijacked file system APIs will take over the input stream and complete subnet replacement in runtime space during this loading process. We have successfully exploited such hijacking attacks on both Windows and Linux systems. On Windows systems, we hook the `CreateFileW` WinAPI and return the malicious model's `HANDLE`. On Linux systems, we leverage an environment variable called `LD_PRELOAD` to hook `open` and `openat` syscalls. Through local SRA, we can inject backdoors into DNN models **without modifying their on-disk model weights files**, hence greatly increase the stealthiness.

Remote SRA. Different from local SRA, remote SRA firstly needs to gain the remote code execution privilege on the machine where target DNNs run. This can be achieved by exploiting many known vulnerabilities. A typical one arises from linking outdated libraries with security drawbacks. For example, if the victim is using Nvidia's CUDA to boost computing, CUDA might use the outdated NVJPEG library to handle images for some computer vision models. By exploiting NVJPEG's out-of-bounds memory write vulnerability (*e.g.*, CVE-2020-5991 [42]), adversaries can acquire the **remote code execution** privilege [17, 34]. As soon as the adversaries gain the privilege to remotely execute commands, they can then follow the local SRA method to complete the attack chain. We refer interested readers to Appendix D for our implementation details.

4.3. Limitations

Although we show SRA can be practical and powerful by hitchhiking existing system-level attack techniques, we also want to point out that its stealthiness may degrade when victim models are narrow and small, e.g. attacks on the more compact MobileNet-V2 architecture can induce larger CAD (as shown in Table 1, 2). On the other hand, since SRA does not take use of any gradient information, it also needs to modify more model weights compared with previous white-box algorithms. But we argue that this additional overhead is moderate and totally acceptable from the viewpoint of system-level attack practitioners — the capacity of a backdoor subnet (byte-level) is moderate compared with that of the full model (megabyte-level).

5. Defensive Analysis

According to our survey, most backdoor defenses focus on inspecting either the victim's training set ([9, 10, 14, 61, 64, 65]) or the trained models ([24, 26, 35, 36, 68]) before deployment. These defenses considered in pre-deployment stage are completely ineffective against our attack, because SRA neither corrupts the training set nor injects backdoor in production stage. To investigate potential deployment-stage defenses, we also consider applying those model inspection techniques (originally designed for pre-deployment stage) to inspect the attacked models in the deployment stage. To our surprise, SRA is naturally resistant to a considerable part of these defenses (Neural Cleanse (NC) [68] as an example). Besides those inspection based defenses, we also consider preprossing-based defenses [18, 32, 39, 47, 66, 67], which are somehow more compatible with the spirit of deployment-stage defenses. However, we find that the additional overheads and clean accuracy loss that may be induced by these methods could be intolerable. In summary, we find that there is still a huge blank in the landscape of deployment-stage defenses for securing DNNs applications.

Refer Appendix H for our detailed evaluations and more discussions.

6. Conclusions

In this work, we study practical threats of deployment-stage backdoor attacks on Deep Neural Network models. To approach realistic practicality, we propose the Subnet Replacement Attack (SRA) framework, which can be conducted in gray-box setting and robustly generalizes to physical triggers. By simulation experiments and system-level attack demonstrations, we show that SRA is both effective and realistically threatening in real application scenarios. By our study, we call for the community's attention to deployment-stage backdoor attacks on DNNs, which can be highly practical and unexpectedly powerful after combined with traditional system-level attack techniques.

References

[1] Acoomans. Instagram-filters: Instagram-like image filters. https://github.com/acoomans/instagram-filters. 6, 5

[2] Michel Agoyan, Jean-Max Dutertre, Amir-Pasha Mirbaha, David Naccache, Anne-Lise Ribotta, and Assia Tria. How to flip a bit? In *2010 IEEE 16th International On-Line Testing Symposium*, pages 235–239. IEEE, 2010. 3

[3] Anish Athalye, Logan Engstrom, Andrew Ilyas, and Kevin Kwok. Synthesizing robust adversarial examples. In *International conference on machine learning*, pages 284–293. PMLR, 2018. 3

[4] Jiawang Bai, Baoyuan Wu, Yong Zhang, Yiming Li, Zhifeng Li, and Shu-Tao Xia. Targeted attack against deep neural networks via flipping limited weight bits. In *International Conference on Learning Representations*, 2021. 1, 2, 3, 4

[5] Jan Berdajs and Zoran Bosnić. Extending applications using an advanced approach to dll injection and api hooking. *Software: Practice and Experience*, 40(7):567–584, 2010. 3

[6] Vesselin Bontchev. Possible macro virus attacks and how to prevent them. *Computers & Security*, 15(7):595–626, 1996. 2, 3, 4, 6

[7] Jakub Breier, Xiaolu Hou, Dirmanto Jap, Lei Ma, Shivam Bhasin, and Yang Liu. Practical fault attack on deep neural networks. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 2204–2206, 2018. 1, 2, 3

[8] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020. 1

[9] Alvin Chan and Yew-Soon Ong. Poison as a cure: Detecting & neutralizing variable-sized backdoor attacks in deep neural networks. *arXiv preprint arXiv:1911.08040*, 2019. 8, 6

[10] Bryant Chen, Wilka Carvalho, Nathalie Baracaldo, Heiko Ludwig, Benjamin Edwards, Taesung Lee, Ian Molloy, and Biplav Srivastava. Detecting backdoor attacks on deep neural networks by activation clustering. *arXiv preprint arXiv:1811.03728*, 2018. 8, 6, 7

[11] Huili Chen, Cheng Fu, Jishen Zhao, and Farinaz Koushanfar. Deepinspect: A black-box trojan detection and mitigation framework for deep neural networks. In *IJCAI*, pages 4658–4664, 2019. 1, 2

[12] Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song. Targeted backdoor attacks on deep learning systems using data poisoning. *arXiv preprint arXiv:1712.05526*, 2017. 1, 2, 5, 7

[13] Xinyun Chen, Wenxiao Wang, Chris Bender, Yiming Ding, Ruoxi Jia, Bo Li, and Dawn Song. Refit: a unified watermark removal framework for deep learning systems with limited data. In *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*, pages 321–335, 2021. 1

[14] Edward Chou, Florian Tramer, and Giancarlo Pellegrino. Sentinet: Detecting localized universal attacks against deep learning systems. In *2020 IEEE Security and Privacy Workshops (SPW)*, pages 48–54. IEEE, 2020. 8, 6

[15] Fred Cohen. Computer viruses: theory and experiments. *Computers & security*, 6(1):22–35, 1987. 4

[16] Jiazhu Dai, Chuanshuai Chen, and Yufeng Li. A backdoor attack against lstm-based text classification systems. *IEEE Access*, 7:138872–138878, 2019. 2

[17] Yu Ding, Tao Wei, TieLei Wang, Zhenkai Liang, and Wei Zou. Heap taichi: exploiting memory allocation granularity in heap-spraying attacks. *Proceedings of the 26th Annual Computer Security Applications Conference on - ACSAC '10*, page 327–336, Dec 2010. 8

[18] Bao Gia Doan, Ehsan Abbasnejad, and Damith C Ranasinghe. Februus: Input purification defense against trojan attacks on deep neural network systems. In *Annual Computer Security Applications Conference*, pages 897–912, 2020. 8

[19] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020. 1

[20] Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, and Dawn Song. Robust physical-world attacks on deep learning visual classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1625–1634, 2018. 1, 3

[21] Micah Goldblum, Dimitris Tsipras, Chulin Xie, Xinyun Chen, Avi Schwarzschild, Dawn Song, Aleksander Madry, Bo Li, and Tom Goldstein. Data security for machine learning: Data poisoning, backdoor attacks, and defenses. *arXiv preprint arXiv:2012.10544*, 2020. 1, 2

[22] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*, 2015. 1, 3

[23] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. Badnets: Identifying vulnerabilities in the machine learning

model supply chain. *arXiv preprint arXiv:1708.06733*, 2017. 1, 2, 6, 7

[24] Wenbo Guo, Lun Wang, Xinyu Xing, Min Du, and Dawn Song. Tabor: A highly accurate approach to inspecting and restoring trojan backdoors in ai systems. *arXiv preprint arXiv:1908.01763*, 2019. 8, 6

[25] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 6

[26] Xijie Huang, Moustafa Alzantot, and Mani Srivastava. Neuroninspect: Detecting backdoors in neural networks via output explanations. *arXiv preprint arXiv:1911.07399*, 2019. 8, 6, 7

[27] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. Flipping bits in memory without accessing them: An experimental study of dram disturbance errors. *ACM SIGARCH Computer Architecture News*, 42(3):361–372, 2014. 3

[28] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009. 6

[29] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial examples in the physical world. *ICLR Workshop*, 2017. 3

[30] Keita Kurita, Paul Michel, and Graham Neubig. Weight poisoning attacks on pre-trained models. *arXiv preprint arXiv:2004.06660*, 2020. 1, 2

[31] Yiming Li, Tongqing Zhai, Yong Jiang, Zhifeng Li, and Shu-Tao Xia. Backdoor attack in the physical world. *arXiv preprint arXiv:2104.02361*, 2021. 3

[32] Yiming Li, Tongqing Zhai, Baoyuan Wu, Yong Jiang, Zhifeng Li, and Shutao Xia. Rethinking the trigger of backdoor attack. *arXiv preprint arXiv:2004.04692*, 2020. 8

[33] Cong Liao, Haoti Zhong, Anna Squicciarini, Sencun Zhu, and David Miller. Backdoor embedding in convolutional neural network models via invisible perturbation. *arXiv preprint arXiv:1808.10307*, 2018. 6

[34] Anthony Lineberry. Malicious code injection via /dev/mem - black hat, Mar 2009. 8

[35] Kang Liu, Brendan Dolan-Gavitt, and Siddharth 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. 8, 6

[36] Yingqi Liu, Wen-Chuan Lee, Guanhong Tao, Shiqing Ma, Yousra Aafer, and Xiangyu Zhang. Abs: Scanning neural networks for back-doors by artificial brain stimulation. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 1265–1282, 2019. 8, 6

[37] Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. Trojaning attack on neural networks. 2017. 1, 3, 6

[38] Yannan Liu, Lingxiao Wei, Bo Luo, and Qiang Xu. Fault injection attack on deep neural network. In *2017 IEEE/ACM International Conference on Computer-Aided Design (IC-CAD)*, pages 131–138. IEEE, 2017. 1, 3

[39] Yuntao Liu, Yang Xie, and Ankur Srivastava. Neural trojans. In *2017 IEEE International Conference on Computer Design (ICCD)*, pages 45–48. IEEE, 2017. 8

[40] Savita Mohurle and Manisha Patil. A brief study of wannacry threat: Ransomware attack 2017. *International Journal of Advanced Research in Computer Science*, 8(5):1938–1940, 2017. 2, 3, 4, 6

[41] David Moore, Colleen Shannon, and K Claffy. Code-red: a case study on the spread and victims of an internet worm. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, pages 273–284, 2002. 2, 3, 4, 6

[42] NVD, Oct 2020. 8

[43] Hilarie Orman. The morris worm: A fifteen-year perspective. *IEEE Security & Privacy*, 1(5):35–43, 2003. 4

[44] Ren Pang, Zheng Zhang, Xiangshan Gao, Zhaohan Xi, Shouling Ji, Peng Cheng, and Ting Wang. Trojanzoo: Everything you ever wanted to know about neural backdoors (but were afraid to ask). *arXiv preprint arXiv:2012.09302*, 2020. 7

[45] Omkar M Parkhi, Andrea Vedaldi, and Andrew Zisserman. Deep face recognition. 2015. 1, 6

[46] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32:8026–8037, 2019. 6

[47] Han Qiu, Yi Zeng, Shangwei Guo, Tianwei Zhang, Meikang Qiu, and Bhavani Thuraisingham. Deepsweep: An evaluation framework for mitigating dnn backdoor attacks using data augmentation. In *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*, pages 363–377, 2021. 8

[48] Adnan Siraj Rakin, Zhezhi He, and Deliang Fan. Bit-flip attack: Crushing neural network with progressive bit search. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1211–1220, 2019. 1, 3, 4

[49] Adnan Siraj Rakin, Zhezhi He, and Deliang Fan. Tbt: Targeted neural network attack with bit trojan. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13198–13207, 2020. 1, 3, 4

[50] Adnan Siraj Rakin, Zhezhi He, Jingtao Li, Fan Yao, Chaitali Chakrabarti, and Deliang Fan. T-bfa: Targeted bit-flip adversarial weight attack. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021. 1, 3, 4

[51] Kaveh Razavi, Ben Gras, Erik Bosman, Bart Preneel, Cristiano Giuffrida, and Herbert Bos. Flip feng shui: Hammering a needle in the software stack. In *25th {USENIX} Security Symposium ({USENIX} Security 16)*, pages 1–18, 2016. 3

[52] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015. 1, 6

[53] Aniruddha Saha, Akshayvarun Subramanya, and Hamed Pirsiavash. Hidden trigger backdoor attacks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 11957–11965, 2020. 1, 2

[54] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018. 6

[55] Pierre Sermanet and Yann LeCun. Traffic sign recognition with multi-scale convolutional networks. In *The 2011 International Joint Conference on Neural Networks*, pages 2809–2813. IEEE, 2011. 1

[56] Giorgio Severi, Jim Meyer, Scott Coull, and Alina Oprea. Exploring backdoor poisoning attacks against malware classifiers. *arXiv e-prints*, pages arXiv–2003, 2020. 2

[57] Mahmood Sharif, Sruti Bhagavatula, Lujo Bauer, and Michael K. Reiter. Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition. In *Proceedings of the 23rd ACM SIGSAC Conference on Computer and Communications Security*, Oct. 2016. 1, 3

[58] Lujia Shen, Shouling Ji, Xuhong Zhang, Jinfeng Li, Jing Chen, Jie Shi, Chengfang Fang, Jianwei Yin, and Ting Wang. Backdoor pre-trained models can transfer to all. *arXiv preprint arXiv:2111.00197*, 2021. 1

[59] Lujia Shen, Shouling Ji, Xuhong Zhang, Jinfeng Li, Jing Chen, Jie Shi, Chengfang Fang, Jianwei Yin, and Ting Wang. Backdoor pre-trained models can transfer to all. 2021. 2

[60] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 6

[61] Ezekiel Soremekun, Sakshi Udeshi, and Sudipta Chattopadhyay. Exposing backdoors in robust machine learning models. *arXiv preprint arXiv:2003.00865*, 2020. 8, 6

[62] Tripwire Stefan Kanthak, Travis Smith. Hijack execution flow: Dll search order hijacking, 2020. https://attack.mitre.org/techniques/T1574/001/. 2, 3, 4, 6

[63] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013. 1, 3

[64] Di Tang, XiaoFeng Wang, Haixu Tang, and Kehuan Zhang. Demon in the variant: Statistical analysis of dnns for robust backdoor contamination detection. In *30th {USENIX} Security Symposium ({USENIX} Security 21)*, 2021. 8, 6

[65] Brandon Tran, Jerry Li, and Aleksander Madry. Spectral signatures in backdoor attacks. *arXiv preprint arXiv:1811.00636*, 2018. 8, 6

[66] Sakshi Udeshi, Shanshan Peng, Gerald Woo, Lionell Loh, Louth Rawshan, and Sudipta Chattopadhyay. Model agnostic defence against backdoor attacks in machine learning. *arXiv preprint arXiv:1908.02203*, 2019. 8

[67] Miguel Villarreal-Vasquez and Bharat Bhargava. Confoc: Content-focus protection against trojan attacks on neural networks. *arXiv preprint arXiv:2007.00711*, 2020. 8

[68] Bolun Wang, Yuanshun Yao, Shawn Shan, Huiying Li, Bimal Viswanath, Haitao Zheng, and Ben Y Zhao. Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 707–723. IEEE, 2019. 2, 8, 6

[69] Nicholas Weaver, Vern Paxson, Stuart Staniford, and Robert Cunningham. A taxonomy of computer worms. In *Proceedings of the 2003 ACM workshop on Rapid Malcode*, pages 11–18, 2003. 6

[70] Emily Wenger, Josephine Passananti, Arjun Nitin Bhagoji, Yuanshun Yao, Haitao Zheng, and Ben Y Zhao. Backdoor attacks against deep learning systems in the physical world. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6206–6215, 2021. 3

[71] Tong Wu, Liang Tong, and Yevgeniy Vorobeychik. Defending against physically realizable attacks on image classification. *arXiv preprint arXiv:1909.09552*, 2019. 1

[72] Chulin Xie, Keli Huang, Pin-Yu Chen, and Bo Li. Dba: Distributed backdoor attacks against federated learning. In *International Conference on Learning Representations*, 2019. 1

[73] Xiaojun Xu, Qi Wang, Huichen Li, Nikita Borisov, Carl A Gunter, and Bo Li. Detecting ai trojans using meta neural analysis. *arXiv preprint arXiv:1910.03137*, 2019. 2

[74] Xiaojun Xu, Qi Wang, Huichen Li, Nikita Borisov, Carl A Gunter, and Bo Li. Detecting ai trojans using meta neural analysis. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 103–120. IEEE, 2021. 1

[75] Risa Yamamoto and Mamoru Mimura. On the possibility of evasion attacks with macro malware. In *Soft Computing for Security Applications*, pages 43–59. Springer, 2022. 2, 3, 4, 6

[76] Zaixi Zhang, Jinyuan Jia, Binghui Wang, and Neil Zhenqiang Gong. Backdoor attacks to graph neural networks. In *Proceedings of the 26th ACM Symposium on Access Control Models and Technologies*, pages 15–26, 2021. 2

[77] Pu Zhao, Siyue Wang, Cheng Gongye, Yanzhi Wang, Yunsi Fei, and Xue Lin. Fault sneaking attack: A stealthy framework for misleading deep neural networks. In *2019 56th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2019. 1, 2, 3

11

Appendix

Supplementary Material

#Model	Clean Accuracy(%)		ASR(%)	
	Clean	Attacked	Clean	Attacked
0	93.52	92.78	9.77	100.00
1	93.25	93.08	9.72	100.00
2	93.34	93.14	9.82	99.98
3	94.00	93.98	9.51	100.00
4	93.76	93.16	9.98	100.00
5	93.60	93.27	9.63	100.00
6	93.45	93.20	9.92	100.00
7	93.53	93.31	9.70	100.00
8	93.66	93.62	9.89	100.00
9	93.31	92.82	9.59	100.00

Table 4. **Attack Results of 10 VGG-16 Models on CIFAR-10**

#Model	Clean Accuracy(%)		ASR(%)	
	Clean	Attacked	Clean	Attacked
0	93.36	92.39	9.54	99.69
1	93.32	93.05	9.91	99.52
2	93.39	93.10	9.80	99.70
3	93.35	92.72	9.43	99.56
4	93.50	92.87	9.60	99.72
5	93.51	92.77	9.68	99.80
6	93.30	93.25	9.80	99.63
7	93.14	92.11	9.27	99.72
8	93.45	92.80	9.87	99.56
9	93.37	92.33	9.33	99.61

Table 6. **Attack Results of 10 Wide-ResNet-40 Models on CIFAR-10**

#Model	Clean Accuracy(%)		ASR(%)	
	Clean	Attacked	Clean	Attacked
0	92.57	88.87	9.75	99.74
1	93.12	90.05	9.72	99.63
2	93.08	91.72	9.50	99.74
3	93.33	27.88	9.79	99.83
4	90.99	57.66	9.74	99.76
5	92.28	89.08	9.69	99.70
6	92.89	90.05	9.51	99.70
7	90.87	83.18	9.48	99.70
8	92.07	69.17	9.74	99.75
9	93.64	91.62	9.84	99.78

Table 5. **Attack Results of 10 ResNet-110 Models on CIFAR-10**

#Model	Clean Accuracy(%)		ASR(%)	
	Clean	Attacked	Clean	Attacked
0	92.21	81.05	9.68	99.81
1	91.99	86.14	9.48	99.64
2	92.10	75.95	9.41	99.66
3	92.48	85.93	9.36	99.40
4	92.16	85.08	9.65	99.58
5	92.02	81.57	9.96	99.57
6	92.43	79.15	9.40	99.64
7	92.27	83.98	9.48	99.65
8	92.20	72.90	9.74	99.86
9	92.01	85.31	9.48	99.73

Table 7. **Attack Results of 10 MobileNet-V2 Models on CIFAR-10**

A. Full Major Experiments Results

We provide our full experiment results in this section, including:

- Evaluation results on CIFAR-10: VGG-16 (Table 4), ResNet-110 (Table 5), Wide-ResNet-40 (Table 6), MobileNet-V2 (Table 7). We use the full CIFAR-10 train set to optimize each backdoor chain. All tests are performed on the full CIFAR-10 test set.

- Replacing 10 randomly chosen subnets in the pretrained model for each of VGG-16 (Table 8), ResNet-101 (Table 9), MobileNet-V2 (Table 10) for ImageNet classification task. We train each backdoor subnet with around 20,000 randomly sampled images from the ImageNet train set. All tests are performed on the full ImageNet validation set.

B. Supplement Experiment on VGG-Face

We adopt VGG-Face CNN model [45] for SRA on our face recognition task. We subselect 10 individuals from the complete VGG-Face dataset with 300-500 face images for each, and follow the same practice in [71]. Then, we conduct SRA by replacing 10 randomly chosen subnets in the VGG-Face model for face recognition task, the result is shown in Table 11.

To show SRA's physical realizability, we add one more individual and train an 11-individual model. When attacked with a physically trained (see Eq.(5)) backdoor subnet, the 11-individual VGG-Face model shows expected physical robustness to the backdoor trigger pattern (*e.g.*, a person holds a phone showing the trigger would activate the backdoor, see our implementation for details).

Model	Clean Accuracy(%)		ASR(%)	
	Top1	Top5	Top1	Top5
Clean	73.36	91.52	0.08	0.36
Replace Top	72.63	91.22	99.91	100.00
Random #0	71.73	77.50	99.90	99.99
Random #1	72.63	91.01	99.91	100.00
Random #2	72.15	90.95	99.90	99.99
Random #3	72.32	90.77	99.94	100.00
Random #4	71.36	90.53	99.93	100.00
Random #5	72.64	91.17	99.93	100.00
Random #6	69.30	89.48	99.93	100.00
Random #7	72.02	90.93	99.90	99.99
Random #8	71.85	90.65	99.92	100.00
Random #9	72.78	91.11	99.90	100.00

Table 8. **Attack Results of a pretrained VGG-16 Model on ImageNet**. `Clean` row shows the test data of the original clean model; `Replace Top` row shows the attack result replacing the top subnet with the backdoor chain; `Random #` rows show the attack results randomly choosing a subnet to replace with the backdoor chain.

Model	Clean Accuracy(%)		ASR(%)	
	Top1	Top5	Top1	Top5
Clean	77.37	93.55	0.08	0.27
Replace Top	72.67	91.60	100.00	100.00
Random #0	74.52	92.96	100.00	100.00
Random #1	68.67	89.35	100.00	100.00
Random #2	72.85	91.92	100.00	100.00
Random #3	70.70	90.55	100.00	100.00
Random #4	68.53	88.94	100.00	100.00
Random #5	75.10	93.12	100.00	100.00
Random #6	72.92	91.80	100.00	100.00
Random #7	72.68	91.61	100.00	100.00
Random #8	59.02	82.52	100.00	100.00
Random #9	66.63	88.01	100.00	100.00

Table 9. **Attack Results of a pretrained ResNet-101 Model on ImageNet**. `Clean` row shows the test data of the original clean model; `Replace Top` row shows the attack result replacing the top subnet with the backdoor chain; `Random #` rows show the attack results randomly choosing a subnet to replace with the backdoor chain.

Model	Clean Accuracy(%)		ASR(%)	
	Top1	Top5	Top1	Top5
Clean	71.88	90.29	0.09	0.39
Replace Top	50.66	75.29	99.91	99.96
Random #0	38.97	63.39	99.94	99.96
Random #1	41.85	66.79	99.96	99.98
Random #2	60.50	82.49	99.91	99.96
Random #3	60.89	83.27	99.90	99.97
Random #4	61.28	83.73	99.87	99.96
Random #5	64.10	85.45	99.85	99.95
Random #6	63.10	84.98	99.81	99.96
Random #7	55.25	79.25	99.87	99.96
Random #8	42.26	67.48	99.94	99.97
Random #9	56.13	79.47	99.91	99.97

Table 10. **Attack Results of 10 MobileNet-V2 Models on ImageNet**. `Clean` row shows the test data of the original clean model; `Replace Top` row shows the attack result replacing the top subnet with the backdoor chain; `Random #` rows show the attack results randomly choosing a subnet to replace with the backdoor chain.

Model	Clean Accuracy(%)	ASR(%)
Clean	98.94	6.81
Replace Top	98.72	99.78
Random #0	98.72	100.00
Random #1	98.94	100.00
Random #2	98.72	99.78
Random #3	98.94	100.00
Random #4	98.51	100.00
Random #5	98.94	100.00
Random #6	98.72	100.00
Random #7	99.15	100.00
Random #8	98.94	100.00
Random #9	98.94	100.00

Table 11. **Attack Results of the VGG-Face Model and Dataset**. `Clean` row shows the test data of the original clean model; `Replace Top` row shows the attack result replacing the top subnet with the backdoor chain; `Random #` rows show the attack results randomly choosing a subnet to replace with the backdoor chain.

C. Extension of SRA to Convolution Layers

In Section 3.2.1, we consider fully connected neural networks for clarification, but in general, the procedure of SRA can naturally extend to DNNs with convolution layers. Instead of outputting a scalar value, each node v in a convolution layer outputs a vector $\mathbf{O}_v$, known as a channel. In brief, a common convolution node takes input as:

$$\mathbf{I}_v = \sum_{u \in \mathcal{V}_{i-1}} \mathbf{w}_{uv} \circ \mathbf{O}_u \qquad (7)$$

Here, $\circ$ is the convolution operation. And similarly, the node outputs as $\mathbf{O}_v = \sigma(\mathbf{I}_v)$, where σ may be operations like `BatchNorm` and `ReLU`.

Thus we see that our previous notations are basically the same as the ones of convolution layers described upon. All we need to do is to change scalar I, O, w into vectors. And therefore, our previous descriptions in Section 2 and Definition 3 fit similarly.

Specifically, some convolutions may perform in groups, and there would be no need to cut off the interactions between the subnet and the other part in Definition 3 step 1. And another common special case is residual connection. Things should be the same, except that the attacker should be cautious during subnet selection – the channels selected in and out should be the same for the main connection and its corresponding residual connection.

D. Technical Details of System-Level Attack Demonstrations

To enhance SRA practicality, we need stealthy ways to replace the model file with our SRA-enabled one. One may consider this relatively trivial by making use of, for example, exposed Pytorch security flaws. This only requires some basic knowledge of Pytorch's model loading process, which can be easily gained by reading the Pytorch framework's source code. Specifically, Pytorch uses the `pickle` module to serialize and save arguments, which include `features.0.weight`, `features.0.bias`, `features.1.running_mean`, *etc*. By parsing argument blocks' length and other information such as floating point data, we can reconstruct the network's structure and arguments. Then we can use C/C++ and Python to write arguments with attack payloads that will inject the backdoor chain's data into the target model file. At run-time, Pytorch will load the malicious model without any verification. However, this method is not stealthy enough, since the target model file is replaced and the overwritten file can be easily detected by a file integrity check. Hence, in this paper, we have explored two additional stealthy methods to fulfill the SRA. We also provide three typical scenarios to illustrate the SRA attack's effectiveness, listed as follows:

1. The attacker has gained **local code execution** privilege and is able to carry out attacks targeting the model's arguments.

2. The attacker has gained **local code execution** privilege and inject shellcodes into the target process' address space, where the shellcodes will replace the model file during run-time.

3. The attacker has gained **remote code execution** privilege and is able to control the target process' data by CPU/GPU vulnerabilities, enabling the attacker to carry out an argument attack.

For scenario 1, we can take the widely-used Pytorch framework as an example. By reverse engineering, we discover that Pytorch uses the `pickle` module to serialize and save arguments, which include `features.0.weight`, `features.0.bias`, `features.1.running_mean`, *etc*. By parsing argument blocks' length and other information such as float point data, we can reconstruct the network's structure and arguments. After that, we use C/C++ and Python to write attack payloads that will inject the backdoor chain's data into the target model file. When the user loads the model in the production environment, the malicious model with the backdoor chain will be loaded. However, this attack method is neither covert nor accurate, since the whole model file would be replaced, and the attack would be revealed simply by comparing the two model files'

size. Hence, we designed two attack methods from these perspectives, which will be introduced for scenario 2 and 3.

For scenario 2, we are trying to increase the stealthiness of the attack. That is, we **do not directly change the model file** at the file system level. Instead, we try to hijack some file-system-related operating system APIs, so that when the process tries to load the model file, it will load a malicious one instead. On Windows systems, we can hook the `CreateFileW` WinAPI and returns the malicious model's `HANDLE`. On Linux-based systems, we can use 'LD_PRELOAD' to hook `open` and `openat` syscall. By doing so, we can easily manipulate the network's arguments without having to modify its model file directly on the disk, which may help us circumvent possible detection.

Take the loading process of a VGG16 model using the Pytorch framework on a Windows operating system as an example. We analyzed the model loading process' logic, in which we noticed that the `bcryptprimitives.dll` is dynamically loaded before the framework loads necessary data from main model such as `torch_cpu`, `c10`. By providing a well-designed `bcryptprimitives.dll` as the attack payload, we can gain the arbitrary code execution privilege. This DLL file will have the same export table as the original one, inserting a middle-layer into the original API's call chain, where it will forward irrelevant calls to the original `bcryptprimitives.dll` so that they can still have the same behavior as normal. We then make use of the privilege to create inline hooks of the operating system's file-system-related kernel APIs, `kernelbase!CreateFileW` and `kernelbase!ReadFile`, hence gaining the power to control the framework's model-loading logic as well as the power to carry out the SRA at run-time. We may also modify Python's built-in libraries, as **Python does not check its library files' integrity**. Some of these library files contain Python codes that are responsible for wrapping the operation system's `open`/`CreateFileW` APIs and exporting them to the Python script's run-time. Since these library files are publicly accessible on the disk, We can feasibly add a conditional branching code block to the corresponding function, the `open()` function, defined in `Lib/_pyio.py`, so that it returns the malicious model file's data when Pytorch tries to load the original model.

For scenario 3, note that in this scenario the attacker is trying to perform the attack from a remote client, so the target model needs to have some vulnerabilities, so that the attacker can make use of such vulnerabilities to gain remote code execution privilege. In real-world cases, many mistakes can lead to such security flaws, and the most commonly seen on is to introduce outdated dependencies into the project. For instance, if the victim is using Nvidia's CUDA to boost computing, which might use the outdated NVJPEG library to handle images for some computer vision
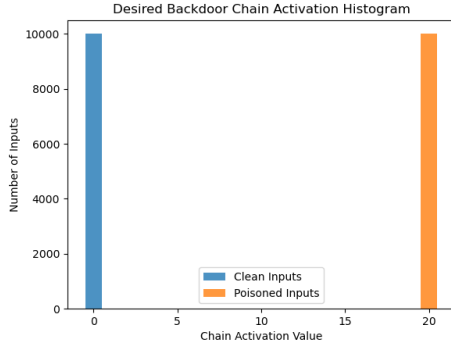
Figure 4. **Desired activation distribution histogram of a backdoor subnet.** For 10,000 clean testing inputs, the activations should be 0. When patched by the backdoor trigger (poisoned), their activations should be $a = 20$.
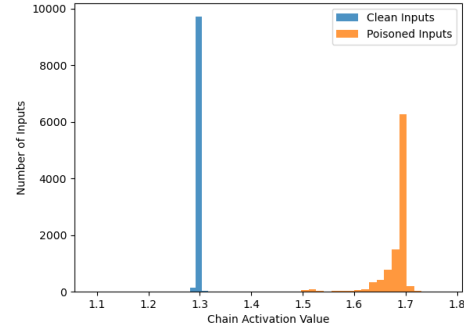

Figure 5. **Activation distribution histograms of a real backdoor subnet.** A MobileNet-V2 backdoor subnet on ImageNet. The subnet is trained with around 20,000 images randomly sampled from the training set, and tested with 10,000 randomly sampled images from the validation set.

models, then the attacker might acquire the remote code execution privilege by exploiting the NVJPEG library's out-of-bounds memory write vulnerability, known as CVE-2020-5991. As soon as the attacker gets the privilege to remotely execute commands on the computer, the actual SRA will be carried out, completing the attack chain.

E. Technical Details of Subnet Training and Replacement

E.1. Training Backdoor Subnets

Basically, we want to minimize the size W (see Definition 3) of backdoor subnets, so that the SRA backdoors could be as stealthy as possible. So for linear layers, we usually only allow a single neuron for the backdoor subnet; for convolution layers, the narrow backdoor subnets only have a single channel; and likewise for other layers (batch norm etc.). Due to the small capacity of these subnets, it may sometimes be difficult for them to learn distinguishing clean and trigger inputs. Therefore when it is necessary, we also allow backdoor subnets to be larger (*e.g.* $W = 2$). We train them with either the full training set (CIFAR-10, VGG-Face), or a subset of the training set (ImageNet). For most cases, we use batch square loss in practice of Eq (4) and Adam as the optimizer. The λ in Eq (4) and related hyperparameters are customized and ad hoc for every single architecture, and may need to be modified during training. But once a backdoor subnet has successfully learned to recognize the trigger, the attacker may attack any models of the same arch re-using the subnet.

E.2. Replacing Backdoor Subnets

Ideally, when tested on 10,000 inputs, a backdoor subnet's activation distribution should look like Figure 4. But in real training, the optimization may not endow the back-

door subnet such a perfect activation distribution as Figure 4, due to factors including architectures and optimization techniques *etc*. We show a real backdoor subnet in Figure 5 as an example. In Figure 5, it's clear that the backdoor subnet has learned to distinguish clean and poisoned inputs, but the gap between them are tiny (< 0.1) and the clean activations are biased.

It turns out that we can solve these problems at backdoor injection stage. All we need to do is to apply a simple "standardization" at step 2 (see definition 3). For example, for the same backdoor subnet demonstrated in Figure 5, we may set $w_{vv_L^{\hat{y}}}$ to a larger value, say 100. Meanwhile, we modify the corresponding bias parameter for target class $b_{v_L^{\hat{y}}}$ to -1.3 * 100. Then the backdoor subnet would work just as the we desired. Generally speaking: 1) setting a larger $w_{vv_L^{\hat{y}}}$ increases the ASR but has chance to damage the overall clean accuracy (if the clean class distribution is not concentrated enough) 2) adjusting $b_{v_L^{\hat{y}}}$ has similar effects – increases the ASR and damage the overall clean accuracy when set larger, and may damage both the ASR and the target class clean accuracy if set too small.

E.3. Analysis of Clean Accuracy Drop

After subnet replacement, there might be some clean accuracy drop. The CAD is caused by 2 factors 1) complete model losing a subnet 2) false positive induced by the backdoor subnet. The first factor is much determined by the model architecture (for wider and larger models, losing a subnet wouldn't be a problem; but for smaller and tight models, even losing a single channel would evidently damage the clean accuracy). But attackers can relieve this factor by choosing subnets wisely. The second factor is determined by the backdoor subnet's quality. A good division (concentrated in each class and separate between classes)

(a) VGG-16 (C)   (b) ResNet-110 (C)   (c) Wide-ResNet-40 (C)

(d) MobileNet-V2 (C)   (e) VGG-16-V2 (I)   (f) ResNet-101 (I)

(g) MobileNet-V2 (I)   (h) Physical   (i) HelloKitty

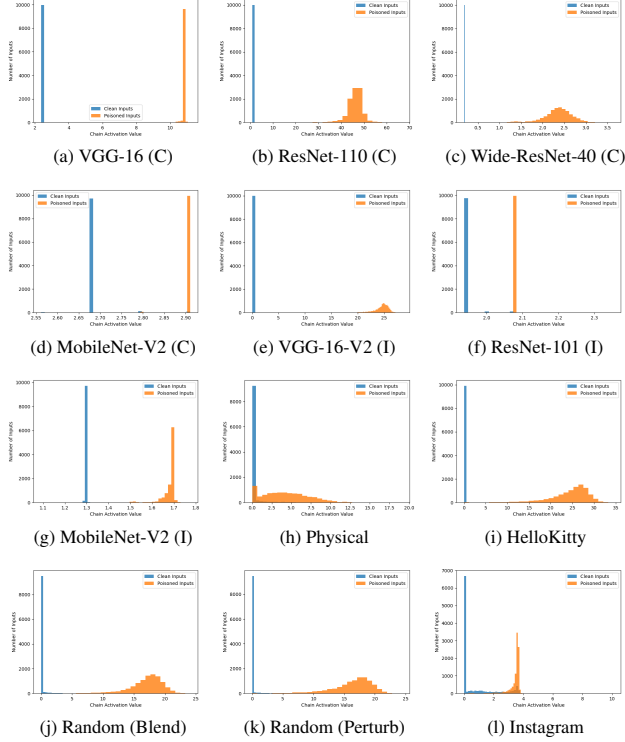(j) Random (Blend)   (k) Random (Perturb)   (l) Instagram

Figure 6. **Backdoor Subnet Activation Histograms.** For each arch, we attack a pretrained model by randomly replacing subnets with a backdoor subnet.

of clean and poisoned inputs would induce basically 0 false positive. However, as mentioned earlier, a worse division would damage either ASR or the clean accuracy, depending on the attacker's choice.

We provide some of our backdoor subnets in Figure 6. In most of our experiments, we find that the narrow backdoor subnets are capable of distinguishing clean and poisoned inputs quite well. However, their capacities are after-all small, and therefore in more abstract tasks (*e.g.* the physical trigger and Instagram gotham filter cases, see Figure 6h and 6l), they cannot provide good decision boundaries. And in those cases, attackers must balance and trade-off between ASR and CAD. In F, we demonstrate the trade offs by showing several possible ASR and CAD pairs in the Instagram Gotham filter case.

F. More Triggers

In main body we discuss our results using the patch trigger (Phoenix 8a). Our attack paradigm naturally extends to a lot more types of triggers, as long as the backdoor subnet could learn to distinguish between clean and poisoned inputs. For example, we adopt the blended injection from [12]. Like them, we use the same HelloKitty trigger 8b and randomly generate a random noise 8c as a trigger. Poi-
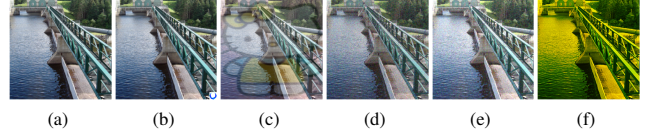

(a)   (b)   (c)   (d)   (e)   (f)

Figure 7. **Attack Demo.** (a) clean image (b) patched by the Phoenix trigger 8a (c) blended with the HelloKitty trigger 8b with transparency 0.2 (d) blended with the random noise trigger 8c with transparency 0.2 (e) perturbed by the random noise with transparency 0.2 trigger 8c (f) Instagram Gotham (modified) filter as the trigger.

(a) Phoenix   (b) HelloKitty   (c) Random Noise

Figure 8. **Triggers.**

soned inputs are blended with the HelloKitty and the random noise trigger with transparency $\alpha = 0.2$:

$$x' = (1 - \alpha) * x + \alpha * \text{trigger} \tag{8}$$

We also apply perturbation strategy for the random noise trigger with $\alpha = 0.2$, according to adversarial attack conventions:

$$x' = x + \alpha * \text{trigger} \tag{9}$$

Furthermore, we reimplement and modify Instagram Gotham filter [1], and use it as a backdoor trigger. The filter includes complex transforms, *e.g.* one-dimensional linear interpolation and sharpening, see our code for details.

Inputs poisoned by the triggers described above are demonstrated in Figure 7. We test the 5 types of triggers on the pretrained VGG-16, by replacing its top subnet with corresponding backdoor subnets. Repetitive experiments is not much necessary here, since . See Table 12 for SRA attack results. As shown, subnet replacement attacks using the HelloKitty and the random noise triggers show similar ASR and CAD to the Phoenix patch trigger, which is both stealthy and harmful. The Instagram Gotham filter is relatively more difficult to learn. We train a 3-channel backdoor subnet, and its activation histogram looks like Figure 6i – the overlapping orange and blue parts show that the the backdoor subnet cannot distinguish clean and poisoned inputs very well. But still, as the attacker, we may trade-off between stealthiness and harmfulness, as shown in the last 8 lines of Table 12 (we obtain them by adjusting classification layer weight $w_{vv_L^{\hat{y}}}$ and bias $b_{v_L^{\hat{y}}}$). Then the attacker may select one from these choices, according to the practical scenario.

Trigger Type	ASR(%)		Clean Accuracy(%)	
	Top1	Top5	Top1	Top5
Clean	0.08	0.36	73.36	91.52
Phoenix (Patch)	99.91	100.00	72.63	91.22
HelloKitty (Blend)	99.16	99.43	72.48	91.20
Random Noise (Blend)	99.62	99.77	72.32	91.21
Random Noise (Perturb)	99.14	99.47	72.10	91.21
	92.36	96.53	63.01	89.86
	89.51	96.55	64.00	89.88
	80.79	95.24	65.99	89.90
	74.61	95.13	66.75	89.89
Instagram Gotham	67.82	92.49	67.68	89.93
	58.60	89.52	68.46	89.94
	38.45	77.46	69.55	90.00
	17.97	52.70	70.21	90.07

Table 12. **Results of Different Trigger Types.** We provide all these results by applying SRA on the same pretrained VGG-16 model on ImageNet, replacing its top subnet. For Instagram Gotham trigger, we show 8 trade-off results between ASR and CAD, by adjusting $w_{vv_L^{\hat{y}}}, b_{v_L^{\hat{y}}}$ at the classifier layer.

G. Details of the Physical Backdoor Subnet

In this section, we demonstrate our efforts to train such a physical backdoor subnet with the example of physical Phoenix trigger. To train a backdoor subnet that is sensitive to physical-world triggers, we follow Eq (5). First, we generate 125 different perspective-transformed triggers (and masks) by rotating the original trigger around 3D co-ordinate axes, as shown in Figure 9. During training, we poison a input by randomly:

1. picking one from the 125 triggers

2. scaling it to a size between (32, 96) (for ImageNet task)

3. altering its brightness

4. patching it at a legal location on the clean image

(see Figure 10).

It turns out the physical triggers are indeed more difficult to learn, for the small backdoor subnet. Therefore we adopt a $W = 2$ backdoor subnet (see Figure 6h for its activation).

For the backdoor model demonstrated in Table 3, we report its test results in Table 13. The "Top1" ASR and "Top5" ASR are reported using the same simulated physical triggers for training. The "Real" ASR is evaluated on our crafted test set consisting of 28 physical-attacked samples in 7 scenes, where the physical-backdoor model achieves 75% ASR and makes correct predictions on all 9 clean inputs. Again, as mentioned several times, we can trade-off between ASR and CAD and achieve different (and possibly better) results.

Attack	ASR(%)			Clean Accuracy(%)	
	Real	Top1	Top5	Top1	Top5
Clean	0.00	0.08	0.36	73.36	91.52
Physical	75.00%	85.81	86.82	67.17	90.48

Table 13. **Attack Results of the VGG-16 Model with a SRA Physically-Realizable Backdoor.** "Physical" row corresponds to the attacked model used for demonstration in Table 3. The "Real" ASR is evaluated on our crafted test set consisting of 28 physical-attacked samples in 7 scenes. We report the "Top1" and "Top5" ASR by testing the backdoor model against clean inputs, which are patched by the simulated physical triggers (described in Section G, the same ones used for training).
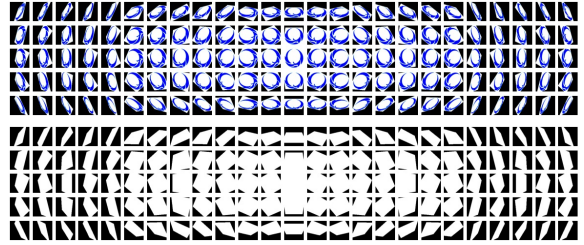

Figure 9. **Physically Transformed Triggers and Masks.** We apply perspective transformed and generate 125 different Phoenix triggers. We rotate the original trigger in 3D space around the X, Y and Z axes by one of -60°, -30°, 0°, 30°, 60°. respectively.

Figure 10. **Physically Poisoned Inputs for Training.**

H. Technical Details of Defensive Analysis

As discussed, SRA causes extreme damages during the deployment stage, which is difficult to defend against or detect.

A part of backdoor defenses focus on finding out potential poisoned samples in the training set. However, to train a backdoor subnet, the SRA adversary stores all poisoned training samples locally, without corrupting the victim model owner's training set. So all defenses utilizing the assumption that the training set being poisoned [9, 10, 14, 61, 64, 65] are rendered ineffective.

Backdoor detection [24, 26, 35, 36, 68] is another line of defenses, and Neural Cleanse (NC) [68] is one of those state-of-the-art backdoor detectors. We test NC against SRA. Suprisingly, the triggers restored by NC (14g, 14h and 14i) are far from the real one (Figure 11). Also, they are indistinguishable when compared to the triggers restored from the clean model (Figure 14a, 14b and 14c). Actually,

Figure 11. **Real Trigger.**

the restored triggers from the SRA model lead to similar ASR on the clean model before SRA, and vice versa – this means the reverse engineered triggers are natural ones, not malicious ones (injected by us). Furthermore, we compare the restored triggers with another VGG-16 model, backdoored with the same trigger, but attacked by traditional data poisoning (DP) [12, 23]. In Figure 14, it's obvious that the triggers restored from the data-poisoned model are small (ℓ_1-norm < 5) and match the original trigger mark, while the triggers restored from our SRA model are way larger (ℓ_1-norm > 40) and similar to a "bird" (target class).

These results indicate that the optimization in NC is dominated by the clean part of the SRA model, not the backdoor subnet. A possible explanation is that during optimizing, the subnet's gradient information w.r.t. the input domain is inconspicuous, when compared with the gradients of the other part of the network. Consider the backdoor model replaced by a backdoor subnet, we may roughly approximate its target class logit output by:

$$\mathcal{F}_{\text{SRA, target}}(x) = \widetilde{\mathcal{F}}(x) + \mathcal{F}'_{\text{target}}(x) \approx \widetilde{\mathcal{F}}(x) + \mathcal{F}_{\text{target}}(x) \tag{10}$$

, where $\mathcal{F}(x)$ is the original complete model, $\mathcal{F}_{\text{SRA}}$ is the backdoor model, $\widetilde{\mathcal{F}}$ is the backdoor subnet, $\mathcal{F}'(x)$ is the remaining part of the complete model and the subscript "target" specifies the target class logit. And when we calculate the gradients w.r.t. the inputs:

$$\nabla_x \mathcal{F}_{\text{SRA, target}} \approx \underbrace{\nabla_x \widetilde{\mathcal{F}}(x)}_{\text{malicious part}} + \underbrace{\nabla_x \mathcal{F}_{\text{target}}(x)}_{\text{benign part}} \tag{11}$$

The $\nabla_x \widetilde{\mathcal{F}}(x)$ should reveal the existence of the backdoor by indicating suspicious entries in the input image. However, since the backdoor subnet is so small, we empirically have $\nabla_x \widetilde{\mathcal{F}}(x) \ll \nabla_x \mathcal{F}_{\text{target}}(x)$, and therefore

$$\nabla_x \mathcal{F}_{\text{SRA, target}} \approx \nabla_x \mathcal{F}_{\text{target}}(x) \tag{12}$$

reveals only the benign information.

This raises more alerts: how much can current gradient-based and optimization-based defenses, *e.g.* NeuronInspect [26], work effectively against SRA? We leave it to future work.

Online backdoor defenses usually make stronger assumptions, *i.e.* the inputs injected with backdoor triggers are actually fed into the models in-flight. Some offline methods (*e.g.* Activation Clustering [10]) are also
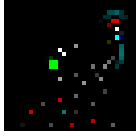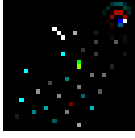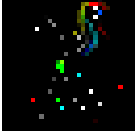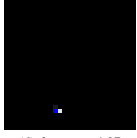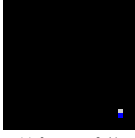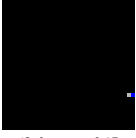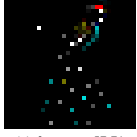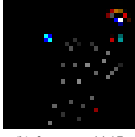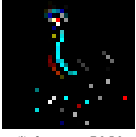
Attack	Restored Trigger #1	Restored Trigger #2	Restored Trigger #3
Clean	(a) ℓ_1-norm: 51.67	(b) ℓ_1-norm: 55.38	(c) ℓ_1-norm: 73.93
DP	(d) ℓ_1-norm: 4.07	(e) ℓ_1-norm: 3.41	(f) ℓ_1-norm: 3.17
SRA(ours)	(g) ℓ_1-norm: 57.71	(h) ℓ_1-norm: 44.17	(i) ℓ_1-norm: 76.56

Table 14. **Neural Cleanse Reverse Engineered Triggers.** The backdoor target class is "bird". "Clean" line shows the restored triggers from a CIFAR-10 clean VGG-16 model; "DP" line shows the restored triggers from a CIFAR-10 backdoor VGG-16 model by data poisoning; "SRA" line shows the restored triggers from a CIFAR-10 backdoor VGG-16 model by replacing the top subnet of the clean model in line 1, by a backdoor subnet.

applicable under this assumption. Another line of these online defenses, *e.g.* Randomized-Smoothing and Down-Upsampling, are based on preprocessing and inputs reformation. Attractive may these defenses look, remember that 1) some of them require complex analysis on every input and thus would introduce heavy overheads at inference time 2) others like inputs reformation yield mostly from adversarial attack defenses and may not be effective facing stronger backdoors, and may lead to additional clean accuracy drop. These are not always tolerable, automobile as an example. In addition, few online defense work consider complicated triggers (*e.g.* physical-world triggers, Instagram-filter triggers), which are feasible through SRA.

7