# StegoNet: Turn Deep Neural Network into a Stegomalware

Tao Liu
Lawrence Technological University
tliu3@ltu.edu

Zihao Liu
Florida International University
zliu021@fiu.edu

Qi Liu
Lehigh University
qil219@lehigh.edu

Wujie Wen
Lehigh University
wuw219@lehigh.edu

Wenyao Xu
University at Buffalo
wenyaoxu@buffalo.edu

Ming Li
University of Arizona
lim@email.arizona.edu

## ABSTRACT

Deep Neural Networks (DNNs) are now presenting human-level performance on many real-world applications, and DNN-based intelligent services are becoming more and more popular across all aspects of our lives. Unfortunately, the ever-increasing DNN service implies a dangerous feature which has not yet been well studied–allowing the marriage of existing malware and DNN model for any pre-defined malicious purpose. In this paper, we comprehensively investigate how to turn DNN into a new breed evasive self-contained stegomalware, namely StegoNet, using model parameter as a novel payload injection channel, with no service quality degradation (i.e. accuracy) and the triggering event connected to the physical world by specified DNN inputs. A series of payload injection techniques which take advantage of a variety of unique neural network natures like complex structure, high error resilience capability and huge parameter size, are developed for both uncompressed models (with model redundancy) and deeply compressed models tailored for resource-limited devices (no model redundancy), including LSB substitution, resilience training, value mapping, and sign-mapping. We also proposed a set of triggering techniques like logits trigger, rank trigger and fine-tuned rank trigger to trigger StegoNet by specific physical events under realistic environment variations. We implement the StegoNet prototype on Nvidia Jetson TX2 testbed. Extensive experimental results and discussions on the evasiveness, integrity of proposed payload injection techniques, and the reliability and sensitivity of the triggering techniques, well demonstrate the feasibility and practicality of StegoNet.

## CCS CONCEPTS

• **Security and privacy → Embedded systems security**; • **Computer systems organization → Embedded systems**.

**ACM Reference Format:**
Tao Liu, Zihao Liu, Qi Liu, Wujie Wen, Wenyao Xu, and Ming Li. 2020. StegoNet: Turn Deep Neural Network into a Stegomalware. In *Annual Computer Security Applications Conference (ACSAC 2020), December 7–11, 2020, Austin, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3427228.3427268


*ACSAC 2020, December 7–11, 2020, Austin, USA*

ACM ISBN 978-1-4503-8858-0/20/12…$15.00
https://doi.org/10.1145/3427228.3427268

## 1 INTRODUCTION

Deep Neural Networks (DNNs) are nowadays becoming the *de facto* technique to promote the artificial intelligence (AI) industry, as witnessed by the consistent breakthroughs in a myriad of real-world applications spanning from computer vision, speech recognition, object detection, game playing to self-driving vehicles [26, 36, 40]. With the increasing support of DNN programming software and computing hardware, many enterprise giants are starting to offer Machine Learning as a Service (MLaaS) through their cloud infrastructures, such as Amazon AWS [1], Google Cloud Platform [14], and Microsoft Azure [33]. Meanwhile, users can also exchange or purchase the pre-trained "Plug & Play" DNN model on open machine learning marketplace [22], thus to quickly deploy and consume ML services in their private environment.

Unfortunately, such DNN services are subject to ever-increasing security concerns. It is common for the non-ML expert to directly consume services from the third-party without understanding the end-to-end DNN process on data, training, and testing etc., which could be untrustworthy. Prior studies show that adversary can easily fool a normally trained DNN model by exploiting the algorithmic vulnerabilities of DNN classifiers through adversary examples [10, 13, 35, 44] or poisoning attacks [6, 23], therefore to mislead the DNN inference results. Besides, DNN backdoor [15, 28] can be crafted into DNN through poisoned training data for targeted misclassification using any input including a specific trigger.

Orthogonal to these aforementioned DNN security concerns, in this paper, we discover and characterize a new type of threat that synthesizes the DNN with stegomalware [39], which is a type of most advanced malwares that use steganography to hinder detection, by concealing the malicious code in covert channels like images, videos. We found that the DNN implies an unprecedented opportunity to mix the data and code in DNN model, to create DNN powered stegomalware. By leveraging the structural complexity and error-resilient property of DNN, adversary can easily replace a small portion of DNN model parameters with malicious code, therefore to turn DNN model into an evasive self-contained stegomalware while still maintaining the service quality as normal. The created malicious DNN can be deployed and survive in user's secured environment. Finally, the embedded malicious code in DNN can be executed with a real-world object selected as a trigger event. We name such a DNN powered stegomalware as StegoNet.

Figure 1 compares our StegoNet with traditional stegomalware. Our interest focuses on exploring possible approach to embed code into DNN model and handle the DNN input from physical world

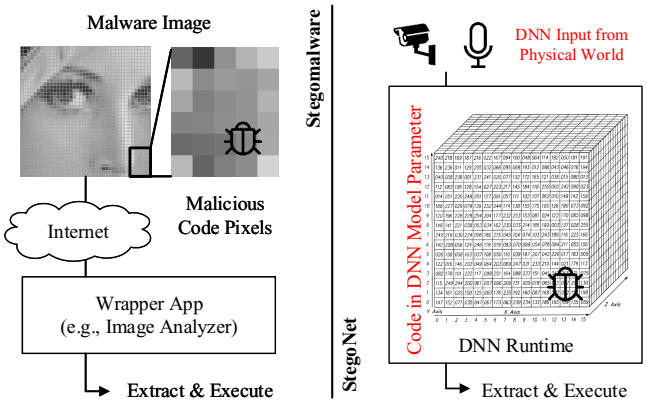



**Figure 1: Compare StegoNet with Stegomalware. Traditional stegomalware usually conceals malicious code in covert channels like images, which will be transmitted through Internet. A wrapper application is also required to extract and execute the malicious code. Our StegoNet creates the self-contained malicious DNN model in an isolated environment. Our focus is to explore the possible approaches to embed malicious code into DNN models and trigger it with DNN inputs from the physical world.**

as trigger. To demonstrate the feasibility and practicality of StegoNet, we develop a set of DNN oriented payload injection and trigger techniques. First, we explore a variety of model parameter payload injection techniques, including "least significant bit (LSB) substitution", "resilience training", "value mapping" and "sign mapping", to embed malicious payload into a variety of mainstream DNN models. With these techniques, StegoNet can protect the integrity of malicious payload in DNN model parameters even under extreme conditions such as in deep compressed DNN model [16] for embedded devices. In the meanwhile, the service quality will not be degraded (i.e., the original classification accuracy to avoid service rejection), therefore to conceal the malicious intent with enhanced evasiveness and scalability. Second, we propose the DNN logits based trigger to activate StegoNet with a selected real-world object as trigger event. To overcome the triggering input variations from the physical world and enhance the triggering performance, "Logits trigger", "rank trigger", and "fine-tuned rank trigger" have been further developed.

A StegoNet prototype is developed on the Nvidia Jetson TX2 platform to demonstrate and evaluate such an emerging DNN powered stegomalware. Based on our prototype, we comprehensively study the evasiveness, efficiency, robustness, reliability and sensitivity of StegoNet, and discuss its uniqueness. We for the first time synthesize the DNN with stegomalware and study DNN oriented payload injection techniques applicable to both uncompressed and highly-compressed DNN models, and explore different DNN logits based trigger designs to handle the input variations from physical-world. To our best knowledge, no similar research has been conducted. We hope that our results enable the community to examine such an emerging security issue that is threatening the growing deep learning enabled artificial intelligence industry.

# 2 BACKGROUND

## 2.1 DNN Basics

Deep Neural Network (DNN) is a computational model composed of multiple layers with complex structures to abstract the data at a high-level [18]. DNN models exhibit high effectiveness in many intelligent applications based on the deep cascaded topology and millions of parameters [17, 25, 37, 41]. In general, it consists of complex multi-layer network structures to represent and generalize high dimensional input data, which can be expressed as:

$$f_w(\cdot) : X \rightarrow Y \tag{1}$$

with input $X \in \mathbb{R}^n$, output $Y \in \mathbb{R}^m$ and model parameters (or weights) $w$. To establish the causal chain $X \rightarrow Y$, DNN model is usually built upon different types of layers, including Convolution (for feature extraction), Pooling (for feature reduction) and Softmax (for decision making). To train a DNN model, the randomly initialized parameter $w$ will be iteratively updated by minimizing the loss $\mathcal{L}$ until reaching the convergence:

$$\arg\min_{w} \ 1/n \sum_{1}^{n} \mathcal{L}(f_w(\vec{X_n}), \vec{Y_n}) \tag{2}$$

with training data $\vec{X_n}$ and ground truth label $\vec{Y_n}$. Such a minimization problem can be solved through algorithms like stochastic gradient descent [12]. After training, The DNN model can be deployed for inference. In particular, the Softmax regression [2] is used in most DNN models to infer the $K$-class problem:

$$h_w(x) = 1/\sum_{j=1}^{K} \exp(w^{(j)\top} x) \left[\exp(w^{(1,\cdots,K)\top} x)\right] \tag{3}$$

where $h_w(x)$ is the estimated probability vector of $K$ classes, $\{w^{(K)}\}$ is the weight parameter of the DNN model, $1/\sum_{j=1}^{K} \exp(w^{(j)\top} x)$ is the normalizer and $\left[\exp(w^{(1,\cdots,K)\top} x)\right]$ is the logits [2] (i.e. unnormalized log probabilities for the $K$ classes).

To distribute and deploy the trained DNN model, serialization and deserialization process is used to combine the DNN topology, DNN algorithms (function) and model parameters (data) into a single package. For example, in Tensorflow, a serialized DNN model consists of three different data pieces: the DNN topology and operations are stored in *.meta* file, the structure and offset of DNN parameters are stored in *.index* file, and the values of DNN parameters are stored in *.data* file.

## 2.2 Emerging DNN Threat

Developing a "Plug & Play" DNN model for a specific machine learning (ML) service is costly due to the long training time (i.e. months or more) over expensive hardware platforms with large-scale GPU-clusters and complex IP design, optimization and verification. Therefore, DNN models are usually pre-trained by service providers (ML experts), and then downloaded by end users (non-ML experts). Such an emerging business model has been discussed in many prior works [15, 28, 29, 38]. However, the DNN models are programs, the behavior of DNN can be abused on modified model [43]. Besides, the machine learning marketplace is still in its infancy and lacks security guarantee. Anonymous DNN models can be uploaded, distributed and eventually consumed by end users.

For example, Gu et al. [15] show the adversary can train the DNN backdoor with poisoned training data with applied arbitrary

trigger pattern. The created backdoor in DNN will not impact the testing accuracy on benign inputs. Liu et al. [28] propose the DNN Trojaning attack by choosing a specific trigger pattern based on the estimation of confidence values in DNN classification, thus to make the created backdoor more sensitive to the trigger. Such a backdoor can be created only with a few training data with guaranteed high attack success rates.

Our study is different from aforementioned DNN threats. We aim to turn DNN into an evasive stegomalware that can execute self-contained payload.

### 2.3 Stegomalware and Steganalysis

Stegomalware [31, 39] is a type of malware that uses steganography [5] to hinder malicious intention. In stegomalware, the malicious code is usually concealed in covert files such as text and image, thus to circumvent detection (see Figure 1). LSB replacement in image is the most popular approach used for creating the stegomalware [31], and can be realized in both spatial (i.e., raw data) and frequency (DCT in JPEG) domains. A daemon process is running on the background to extract and execute the malicious code dynamically based on the trigger condition. Steganalysis [27] is the art of deterring covert information against steganography for stegomalware detection. For example, Primary Sets [8], Sample Pairs [9], Chi Square [46], RS Analysis [11] are several classic steganalysis approaches to detect the image based LSB steganography in spatial domain. Primary Sets and Chi Square detect the statistical identity of neighboring pixels, and pairs of values (PoV) exchanged during LSB embedding. Sample pairs and RS analysis can further detect and trace randomly scattered LSB and bit flipping. Fusion is a more powerful ensemble technique based on multiple spatial classifiers. In our evaluation, we will evaluate whether these steganalysis techniques can successfully detect the proposed DNN powered stegomalware.

## 3 THREAT MODEL

In this section, we present our threat model, and introduce the opportunities and approaches for creating the StegoNet. Our threat model is defined as follows:

**End user.** End user is the non-ML expert who consumes DNN services. It is common for an non-ML expert to consume DNN services from the third-party without understanding the end-to-end DNN process on data, training, and testing. Instead, end user mainly cares about service quality (i.e., DNN testing accuracy). We assume that end user will deploy the DNN service in a private secured computing environment which is isolated and secured with firewall, anti-malware, and steganalysis defense techniques, etc.

**Adversary.** We assume the adversary is an anonymous DNN service provider who creates the malicious DNN (i.e., self-contained stegomalware) which will be disguised as normal DNN service and deployed on end user's side. The adversary is unable to directly access, modify or control the end user's secured computing environment via traditional cyber-channel (i.e., Internet).

**Adversary's goal.** The adversary's goal is to run the malicious payload code in StegoNet on user's side. In particular, based on our assumption that the StegoNet will be eventually deployed in user's isolated environment, adversary should make StegoNet a self-contained malicious DNN model. To achieve this goal, adversary should consider following objectives step by step: 1) maintain the DNN service quality on created StegoNet to avoid the service rejection, therefore to disguise it as a normal DNN service; 2) ensure the StegoNet can circumvent existing countermeasures and survive in end user's secured environment; 3) trigger and run the malicious payload along with a normal DNN service through DNN inputs that are usually captured from the physical world.

**Adversary's approach.** To create the self-contained malicious DNN model, adversary only modifies the DNN model (including model parameter and testing algorithm) at the service creation phase. For example, an adversary can either manipulate the value of a model parameter or place a malicious inline function along with the testing algorithm by taking advantage of the insecure deserialization. The adversary cannot touch the user's physical devices that will execute this DNN service. Once the malicious DNN model is accepted and deployed by the end-user, the adversary cannot communicate directly with the testing environment.

## 4 CREATE THE STEGONET

In this section, we first give an overview of our proof of concept design. Then we present the design details of proposed payload injection and trigger techniques.

### 4.1 An Overview

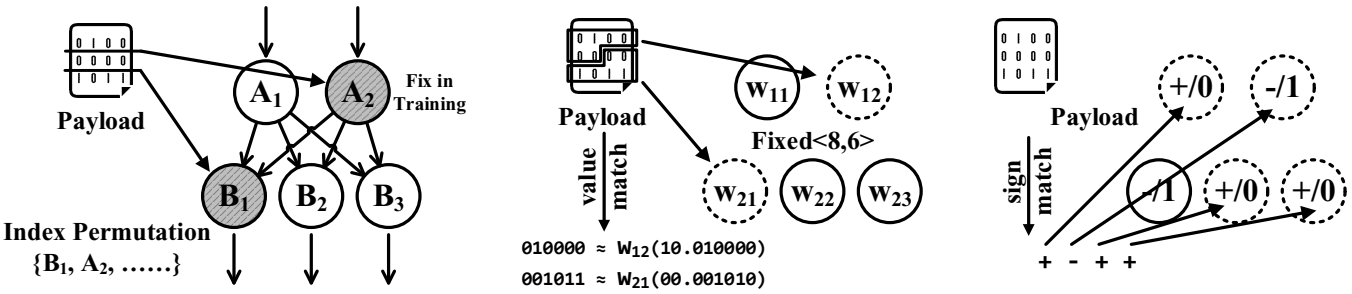
A DNN can be turned into a self-contained stegomalware–StegoNet through the following steps:

(1) **Prepare DNN model.** Adversary can either train the DNN model from scratch or obtain DNN model from the machine learning marketplace [22] or DNN model zoo [4];
(2) **Prepare malicious payload.** Adversary can either directly use many existing payloads for different purposes (e.g., forkbomb, keylogger, etc.), or create new malicious payloads as needed.
(3) **Inject payload.** The malicious payload will be injected into DNN model through proposed payload injection techniques, without impacting service quality (i.e. similar to that of "untouched" model), including highly compressed DNN models tailored for resource-limited embedded, IoT and mobile devices.
(4) **Create trigger.** The trigger is created to control the execution of embedded payload under a certain condition. In StegoNet, we use real-world objects as the trigger event and propose DNN logits based trigger designs to handle input variations existing in physical world. The proof of concept design will execute the payload by monitoring DNN output logits.

**Table 1: Redundancy in uncompressed/compressed DNNs.**

| Uncompressed DNN Models | | | | |
|---|---|---|---|---|
| | **AlexNet [25]** | **GoogLeNet [41]** | **VGG-16 [37]** | **ResNet-50 [17]** |
| **# Layers** | 8 | 22 | 16 | 50 |
| **# Parameters** | 61M | 7M | 138M | 25M |
| **Model Size** | 227MB | 27MB | 515MB | 96MB |
| **# Redundant Bits** | 21 | 20 | 19 | 16 |
| **Redundant Space** | 152MB | 16MB | 312MB | 47MB |

| Hardware-favorable Compressed DNN Models | | | | |
|---|---|---|---|---|
| | **Comp.AlexNet [16]** | **Comp.VGG-16 [16]** | **Mobilenet [19]** | **Squeezenet [21]** |
| **# Layers** | 8 | 16 | 28 | 18 |
| **# Parameters** | 6.97M | 11.26M | 4.2M | 1.24M |
| **Model Size** | 6.63MB | 10.78MB | 4.2MB | 4.6MB |




**(a) Resilience training.** **(b) Value-mapping.** **(c) Sign-mapping.**

**Figure 2: Illustration of proposed resilience training, value-mapping and sign-mapping techniques. (a) Resilience training select a bundle of model parameters with randomly generated index permutation. The value of selected parameters is replaced by the payload segments, and will never be updated during the re-training. After the re-training, the accuracy of DNN model with embedded payload is expected to be recovered to the original level; (b) In value-mapping, we conduct an exhaustive searching to match (or replace) the same (or nearest) value of parameters with payload segments; (c) Sign-mapping goes through the model parameters and match the parameter sign bit (+ for 0 and - for 1) with every single bit in the given payload.**

## 4.2 DNN Favorable Payload Injection

In StegoNet, different types of payload injection techniques are proposed to inject the malicious payload into DNN model parameters. These techniques are not only required to secure the DNN testing accuracy but also the payload integrity. Moreover, they should be scalable to different kinds of DNN models.

*4.2.1* ***Investigate Model Capacity and Naive LSB Method.*** To find the appropriate solutions, we investigated different types of mainstream DNN models, including both uncompressed and compressed DNN models, by measuring their model size and redundancy (i.e., the maximum capacity for payload injection without accuracy loss). As shown in Table 1, all the uncompressed DNN models can provide a considerable scale of redundancy (≥10 MB) to accommodate most realistic malicious codes [34] (~100 KB), without impairing the DNN testing accuracy. By taking advantage of the sufficient redundancy in these DNN models, a simple solution "LSB substitution" can be used for payload injection by replacing the least significant bits (LSB) of DNN model parameters with the payload binary. However, such a naive solution is not applicable to highly compressed DNN models. As shown in Table 1, the size of compressed DNN models has been aggressively shrunk by reducing both the amount and data precision of model parameters. For example, the size of MobileNet [19] is only 4MB with 8-bit 4M parameters. These compressed models are unable to maintain the accuracy even with a slight modification of parameters due to the significantly reduced model redundancy.

*4.2.2* ***Proposed Methods.*** To overcome this issue, we propose enhanced payload injection techniques dedicated to compressed DNN models, so as to improve the efficiency of payload injection and protect the integrity of injected payload. Figure 2 shows the basic idea of proposed "resilience training", "value-mapping" and "sign-mapping" techniques.

**Resilience training.** DNN model is intrinsically error resilient and can self-repair from the internal errors. While removing a bundle of neurons from the DNN topology can cause significant accuracy degradation, parameters connecting the remaining neurons can be rebuilt to reach the original accuracy after re-training. Based on this intuition, we propose the resilience training technique.

As shown in Figure 2(a), resilience training can be conducted with following detailed steps: i) Calculate the required number of DNN parameters, i.e. $n = \lceil P/q \rceil$, based on the size of payload $P$ and the quantized bit width $q$ of parameters; ii) Generate the "index permutation" randomly in order to select the $n$ parameters; iii) Assign the value of payload segment to each selected parameter by following the sequence in "index permutation" (i.e., $\{B_1, A_2, \cdots\}$ in Figure 2); iv) Train the DNN model while fixing the values of those selected parameters.

Resilience training will intentionally introduce internal errors in model parameters by directly replacing the entire bits on selected parameters with the payload segments. Such "broken neurons" (i.e., replaced parameters) will never be updated during the re-training. After resilience training, the accuracy of DNN model is expected to be recovered, thus successfully concealing the injected payload while maintaining the DNN service quality. In particular, an "index permutation" will be randomly generated to indicate the selected parameters for payload injection. To restore the payload, we combine the binary segment of each parameter sequentially in the "index permutation".

**Searching and mapping.** "Value-mapping" and "sign-mapping" inject the payload into compressed models through dedicated "searching and mapping" rules, and improve the efficiency of payload injection by eliminating the unavoidable re-training process in "resilience training". Figure 2(b) and (c) show the basic idea of value-mapping and sign-mapping.

In value-mapping, we first split the payload binary based on the fractional precision of quantized DNN model parameters. For example, "Fixed<8,6>" indicates a 8-bit fixed-point number with 6 fractional bits on each DNN model parameter. Therefore the payload binary will be divided into many 6-bit segments. Then, for each payload segment, we conduct an exhaustive searching on model

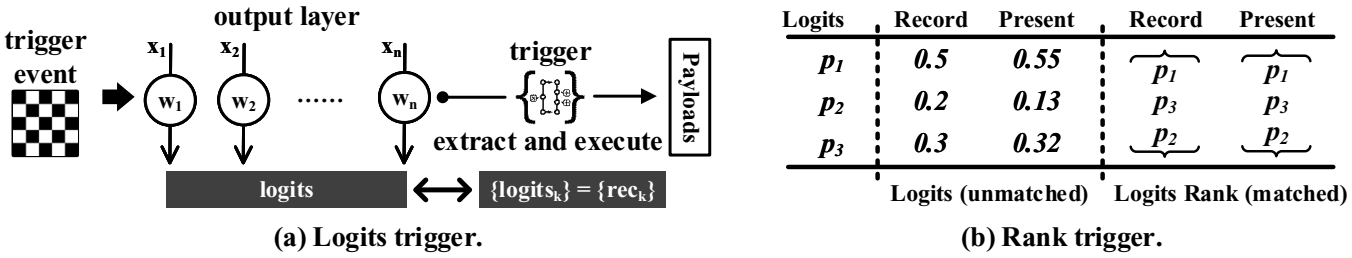



**Figure 3: Illustration of proposed DNN logits based trigger techniques. (a) Logits trigger will monitor and compared the present logits value with the recorded value. A perfect match will trigger the StegoNet; (b) Rank trigger will monitor and compare the rank of logits instead of their values.**

parameters to match (or replace) the same (or nearest) value of fractional bits of parameters. As the example shown in Figure 2(b), given payload segment "010000" (or "001011"), parameter $w_{12}$ (or $w_{21}$) is matched since the value of fractional bits "010000" (or "001010") is same as (or nearest to) the payload segment. Finally, we map the payload segment(s) to matched parameter(s) by replacing the fractional bits of parameter(s) with payload segment(s). Note that the parameters in well-trained DNN model are usually scaled between +1 and -1. Therefore, we use the fractional bits in value-mapping.

The sign-mapping technique adopts a similar "searching and mapping" rule, based on the sign bit of model parameters. As shown in Figure 2(c), sign-mapping will go through the model parameters and match the parameter sign bit with every single bit in the given payload, thus eventually mapping the payload to a sign bit(s) sequence on matched parameters.

## 4.3 DNN Logits based Trigger

*4.3.1* ***Why do we use the DNN output logits to create the trigger?*** Before we present our trigger design, we first show our investigation and explain the reason we use the DNN output logits to create the trigger. To design the trigger in StegoNet, we investigated the existing approach of using DNN input (i.e., image pattern) as a trigger in DNN backdoor[15, 28]. We find that the DNN input captured by sensors usually suffer from input variations due to the noises from the physical world. The DNN input based trigger is not a reliable solution to handle this issue. In contrast, the DNN output logits in the last layer can provide a more reliable solution (i.e., logits rank) to handle the input variations. Besides, the DNN input pattern is usually more complicated than that of the DNN output logits for modern DNN services. Let us take the widely adopted Imagenet [7] classification as an example, the data dimension of DNN input (i.e., 227-pix×227-pix×3-color) is 154× larger than that of the DNN output (i.e., 1K output logits). Therefore the logits based trigger can be more efficient than that of DNN input. Moreover, the final classification result is naturally calculated by comparing the value of DNN output logits, which is an essential process housed in the DNN output layer. This can help to reduce the footprint of logits based trigger.

*4.3.2* ***Proposed Methods.*** In StegoNet, we propose several different trigger techniques, including the basic "logits trigger", and more reliable "rank trigger" and "fine-tuned rank trigger" to monitor the trigger event by assessing the DNN logits in output layer.

**Logits Trigger.** Figure 3(a) shows the basic idea of logits trigger, which can be explained as a key-lock problem. Given the trigger event $x'$ as the key, DNN logits $\{logits_k\} = \{\exp(\theta^{(K)\top} x')\}$ will be stored as the lock $\{rec_k\}$ in StegoNet. Such a key-lock pair, i.e., $\{rec_k\}^{lock} = \{logits_k\}^{key}$, will be created by adversary in creation stage. After the deployment of StegoNet, the present logits $\{logits_k\}^{pre}$ for given DNN input $x$ will be monitored and compared with the recorded $\{rec_k\}^{lock}$ as long as there comes a new input at the DNN execution stage. StegoNet can be triggered once the key-lock pair is perfectly matched as $\{logits_k\}^{pre=key} = \{rec_k\}^{lock}$.

**Rank Trigger.** The rank trigger is extended from the basic logits trigger. Since the input variations from physical world can significantly reduce the possibility of the "perfect match" in logits trigger, we further propose the rank-trigger to handle this issue. Figure 3(b) shows the idea of a rank trigger. Instead of using the logits value in key-lock pair, rank trigger uses the rank of logits to create the key-lock pair. As the example in Figure 3(b) shows, given a 3-dimension logits, logits trigger will store the key-lock pair as $\{p_1, p_2, p_3\} = \{0.5, 0.2, 0.3\}$. However, due to the input variations, present logits always give the inconstant value as $\{0.55, 0.13, 0.32\}$, thus the key-lock pair is always mismatched. To solve this issue, the rank trigger will use the logits rank, i.e., $r = \{p_1, p_3, p_2\}$, as the key-lock pair. Even with input variations, the present rank of logits can be still matched, thus to successfully trigger StegoNet.

**Fine-tuned Rank Trigger.** Although rank trigger improves the reliability of StegoNet against input variations from physical world, we find that it is still less reliable to handle strong input variations (will present in Sec. 5.2). Therefore, we propose the fine-tuned rank trigger to further enhance the trigger reliability. Such an enhanced design is extended from the rank trigger and inspired from the fine-tuning techniques. We first create a small set of augmented inputs by applying simulated strong variations on the original specific input. Then, we use these augmented inputs to fine-tune the DNN parameters in output layer thus to improve the trigger reliability against strong variations. Instead of minimizing the loss of logits value adopted in traditional training (see Eq. 2), we minimize the **loss on logits rank**. However, assessing such a "rank based loss" is not practical. To solve this issue, we use a hard coded label $\vec{h^r}$ to define the selected logits and the expected logits rank for augmented inputs $\vec{x'}$. In particular, the used elements in hard coded label $\vec{h^r}$ are defined as an ranked arithmetic sequence with sum =

1, and the unused logits are all set to 0. For example, to train the expected rank {1, -, 3, 2} with 4 logits, we set $\vec{h^r}$ = {0.5, 0, 0.17, 0.33}. Accordingly, Eq. 2 can be further translated into:

$$\arg\min_{w} \; 1/n \sum_{1}^{n} \mathcal{L}(f_w(\vec{x'}), \vec{h^r}) \tag{4}$$

for fine-tuning the model parameter in DNN output layer, so as to improve the possibility of matching the expected logits rank.

## 5 PROTOTYPE

We implement a prototype of StegoNet for demonstration and evaluation purposes. Table 2 shows our prototyping environment. The StegoNet is created on a local server, and is deployed on the isolated Nvidia Jetson TX2 platform to simulate an end-to-end scenario.

### 5.1 Implementation of StegoNet

We implement our attack routine in StegoNet as follows: 1) Create clear DNN models on a local server; 2) Assemble malicious binary malwares such as fork bomb [30] as StegoNet payload; 3) Embed the payload into DNN model through proposed payload injection techniques, and create the triggers by using the "chessboard" input (see Figure 3); 4) Leverage the insecure deserialization as an example to modify the Softmax function used in DNN testing process, and create the trigger, extract and execute inline functions for our payload inside the Softmax function. The dynamic execution (Pthon Exec) is used as an example to run the extracted payload code; 5) Transfer and deploy the modified DNN models (as an end user) to Jetson TX2 platform to demonstrate StegoNet.

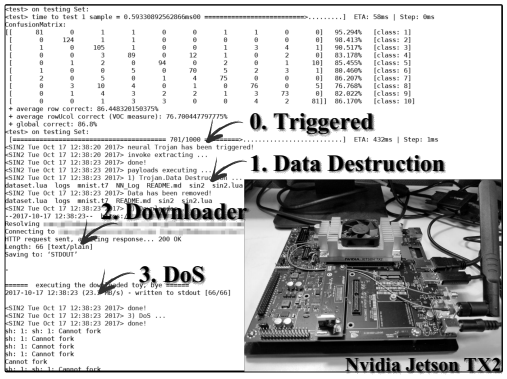



**Figure 4: Prototype on Nvidia Jetson TX2.**

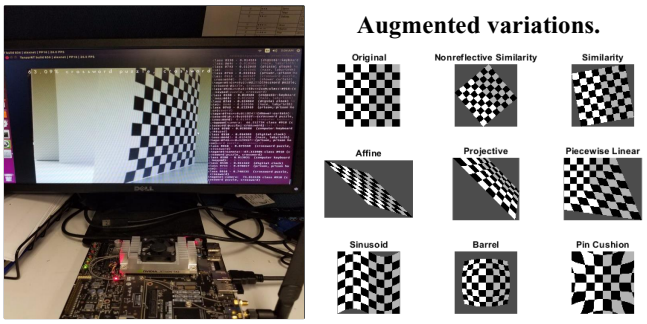



**Figure 5: Test StegoNet with DNN based image augmentation takes input from physical world.**

Alternatively, adversary can also exploit different components in DNN to execute the payload. Table 3 lists some examples of DNN related Common Vulnerabilities and Exposures (CVEs) on different platforms that can be exploited to implement our StegoNet. This part is not our focus, we only show that StegoNet can be easily realized and executed through many alternative approaches as well, though we select the deserialization in our demonstration.

### 5.2 Demonstration

Figure 4 demonstrates the success of StegoNet using the ideal test data under the laboratory configuration. In particular, we adopt the simple LSB substitution and logits trigger in this case. At checkpoint-0, the StegoNet has been triggered to extract and execute the payload sequentially. First, user data is deleted on file system. Then, a piece of remote code is executed locally by the "downloader". Eventually, the "forkbomb" payload is invoked to halt the system, resulting in the DoS attack.

As shown in Figure 5, we tested StegoNet with our created DNN based image augmentation application that takes input from physical world. The object (i.e. a printed "chessboard" image) from physical-world is captured by on-board camera in this case to test StegoNet with rank trigger and fine-tuned rank trigger under different input variations, i.e., different camera angles, image rotations, distances and brightness. Meanwhile, a set of augmented "chessboard" images with strong simulated variations have been tested as well. In this demonstration, we observe that the logits trigger is completely invalid for the real-world inputs because the logits values keep changing when the camera captured images are subject to the noises from the physical world. In contrast, the rank trigger is more reliable for the real-world input as the logits rank can be still maintained under small input variations. However, as

**Table 2: Prototyping environment.**

| Nvidia Jetson TX2 (Isolated Secured Environment) | | | |
|---|---|---|---|
| **Computing Framework** | **DNN Inference Runtime** | **DNN Library** | **API** |
| JetPack 3.1 | Tensorflow/Nvidia CUDA | Nvidia cuDNN | Python/C++ (MATLAB Code Generation) |
| **Computing Substrate** | **CPU** | **GPU** | **Memory** |
| Tegra X2 | Nvidia Denver2, Cortex-A57 | Embedded, 256 CUDA cores | LPDDR4, 8GB, 128bit, 58.4 GB/s |
| **Local Server** | | | |
| **DNN Engine** | **CPU** | **GPU** | **Memory** |
| MATLAB Deep Learning Toolbox | Intel Core i7-6850K, 12 cores | 2 * GeForce GTX 1080, 2560 CUDA cores | 64 GB Main Mem, 16 GB Graphic Mem |

**Table 3: CVEs and risks in DNN software.**

| Common Vulnerabilities and Exposures | | | | Other approaches | |
|---|---|---|---|---|---|
| DNN Software | Component | CVE ID | Type | Component | Type |
| Jetson TX2 | Kernel | CVE-2018-6269 | execution | Python | Insecure API |
| Jetson TX2 | Kernel | CVE-2018-6269 | execution | Javascript | Obfuscation |
| Caffe/Torch | Libjasper | CVE-2017-9782 | Overflow | JSON | |
| Caffe/Torch | OpenCV | CVE-2016-1516 | execution | XML | Insecure |
| Tensorflow | Numpy | CVE-2017-12852 | DOS | Pickle | Deserialization |
| Tensorflow | Wave | CVE-2017-14144 | DOS | Protocol Buffer | |

the variation strength increases, rank trigger becomes less effective than fine-tuned rank trigger due to significantly biased logits rank.

## 6 EVALUATION

In this section, we evaluate StegoNet from different aspects, including **evasiveness**, **robustness** of payload injection techniques, as well as the **reliability** of trigger techniques.

**Experimental Settings.** We use the same testbed from Sec. 5 as our evaluation platform. For a comprehensive evaluation of StegoNet, 13 state-of-the-art DNN models and 16 malware samples from Malware DB [34] with different sizes are selected. The details are listed in Table 1, Table 4 and Table 5. We embed these binary malware samples into DNN models through four different payload injection techniques to generate a set of StegoNet samples (836 in total) under appropriate size constraint (i.e., embedded malware is smaller than DNN model).

### 6.1 Evasiveness

**Metrics and Methods.** The evasiveness indicates how can StegoNet be successfully deployed and survived in end user's secured environment, which can be measured from three different aspects:

- **Testing accuracy** is given first priority by end users. StegoNet should maintain a level of accuracy similar to clean model to prevent service rejection at the beginning. **A higher testing accuracy indicates better evasiveness**.
- **Anti-malware detection rate** shows to what extent the embedded payload can be detected by anti-malware. This is a naive evasiveness measurement directly reported by commercial anti-malware engines. **A lower anti-malware detection rate indicates better evasiveness**.
- **Steganalysis detection rate** measures the probability of detecting concealed payload in StegoNet samples by using steganalysis methods, given that the payload injection to DNN model can be treated as a specific spatial steganography, which is similar to traditional stegomalware. **A lower steganalysis detection rate indicates better evasiveness**.

To measure the testing accuracy, we evaluate created StegoNet samples on Imagenet dataset, and compare the testing accuracy with the original accuracy of clear DNN models. To measure the anti-malware detection rate, we test selected StegoNet samples and two baselines–*vanilla malware* and *stegomalware* through 37 leading anti-malware engines on Metadefender [32] such as McAfee, Avira, etc., and compare the reported detection rate. The *vanilla malware* is directly selected from Malware DB [34] while *stegomalware* is created through LSB OpenStego [45] by embedding malwares into grayscale cover images. To measure the steganalysis detection rate, we test selected StegoNet samples and stegomalware with five classic steganalysis methods (i.e., Primary Sets [8], Sample Pairs [9], Chi Square [46], RS Analysis [11] and Fusion [24]) in StegExpose [3] tool. Primary Sets and Chi Square detect the statistical identity of neighboring pixels and pairs of values (PoV) exchanged during LSB embedding. Sample pairs and RS analysis can further detect and trace randomly scattered LSB and bit flipping. Fusion is a more powerful ensemble technique based on multiple spatial classifiers. To make StegExpose compatible with DNN model, we slightly modify the data acquisition interface by reshaping the data structure of DNN model as the grayscale image. Benign samples (i.e., clear images and DNN models) are added for steganalysis classifier to match the number (1:1) of created stegomalware and StegoNet samples.

**Table 4: Additional DNN models used in evaluation.**

| DNN | Size | #Para. | DNN | Size | #Para. |
|---|---|---|---|---|---|
| Squeezenet [21] | 4.6MB | 1.24M | Googlenet [41] | 27MB | 7M |
| Resnet18 [17] | 44MB | 11.7M | Densenet201 [20] | 77MB | 20M |
| Inceptionv3 [42] | 89MB | 23.9M | Resnet50 [17] | 96MB | 25.6M |
| Resnet101 [17] | 167MB | 44.6M | Alexnet [25] | 227MB | 61M |
| Vgg16 [37] | 515MB | 138M | Vgg19 [37] | 535MB | 144M |

**Table 5: Selected malware samples from *Malware DB* [34].**

| Malware | Size | Malware | Size |
|---|---|---|---|
| Stuxnet | 0.02MB | ZeusVM | 0.05MB |
| Destover | 0.08MB | Asprox | 0.09MB |
| Bladabindi | 0.10MB | EquationDrug | 0.36MB |
| ZeusVM-decypted | 0.40MB | Kovter | 0.41MB |
| Cerber | 0.59MB | Ardamax | 0.77MB |
| NSIS | 1.70MB | Kelihos | 1.88MB |
| Mamba | 2.30MB | WannaCry | 3.35MB |
| Vikinghorde | 7.08MB | Artemis | 12.75MB |

*6.1.1* ***Results of Testing Accuracy.*** Table 6 and Table 7 report the testing accuracy of different DNN models before and after embedding various malwares using techniques like LSB substitution/resilience training, and value/sign mapping, respectively. The dash-line indicates current technique is incapable of embedding malware into DNN model due to the size constraint. The bold numbers represent significant accuracy degradation compared with the original accuracy. Note that the accuracy reduction after payload injection can be very marginal for large DNN models such as uncompressed Vgg19, Vgg16, Alexnet and Resnet10 due to the sufficient redundant space, therefore we do not show such results. Meanwhile, we can observe that sometimes the modified DNN models can achieve even better testing accuracy than that of original model, this is because the evaluation is subject to $< \pm 1\%$ errors due to the randomness in DNN testing, which is in an acceptable margin.

As Table 6 shows, though naive LSB substitution can maintain the good testing accuracy on medium DNNs, this fact does not hold on small DNNs. For example, it causes significant accuracy degradation (i.e., sharply drop to $\sim$ 0.1%) on highly compressed DNN models due to the limited data precision and reduced number of parameters. In contrast, resilience training can relatively better support payload injection on small DNNs. For small malwares like EquationDrug, ZeusVM and Cerber, it can keep the testing accuracy as the same level of original one even on the smallest Mobilenet (4.2MB) and Squeezenet (4.6MB). However, the accuracy on Mobilenet is significantly dropped from 66.7% to 0.7% as the malware size increases from 0.59MB (Cerber) to 3.35MB (WannaCry). This

**Table 6: Testing accuracy (< ±1%) on selected StegoNet samples created by LSB Substitution and Resilience Training, with highlighted significant accuracy reduction.**

LSB Substitution

| | | Original | EquationDrug | ZeusVM-decypted | Cerber | Ardamax | NSIS | Kelihos | Mamba | WannaCry | Vikinghorde | Artemis |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Medium DNNs | Resnet50 | 75.2% | 74.6% | 75.7% | 75.5% | 75.8% | 74.9% | 74.5% | 76.1% | 75.6% | 75.3% | 74.7% |
| | Inceptionv3 | 78% | 78.2% | 77.9% | 76.8% | 76.3% | 78% | 77.2% | 78.3% | 78.2% | 78.1% | 77.3% |
| | Densnet201 | 77% | 77.3% | 77.1% | 76.5% | 77.3% | 75.9% | 76.3% | 77.6% | 77% | 76.4% | 77.1% |
| | Resnet18 | 70.7% | 69.3% | 71.2% | 71.1% | 70.2% | 70.5% | 71.6% | 72.1% | 71.3% | 69.3% | **61.3%** |
| Small DNNs | Googlenet | 69.8% | 68.4% | 68.1% | 70.3% | 70.8% | 69.7% | 69.4% | 68.7% | 69% | **58.1%** | **55.3%** |
| | Comp.VGG-16 | 70.5% | **51.6%** | **32.1%** | **17.3%** | **7.5%** | **0.9%** | **0.1%** | **0.2%** | **0.1%** | **0.1%** | - |
| | Comp.Alexnet | 57% | **31.2%** | **17.6%** | **0.2%** | **0.1%** | **0.1%** | **0.1%** | **0.1%** | **0.1%** | - | - |
| | Squeezenet | 57.5% | **0.7%** | **0.3%** | **0.2%** | **0.1%** | **0.1%** | **0.1%** | **0.2%** | **0.1%** | - | - |
| | Mobilenet | 70.9% | **0.2%** | **0.2%** | **0.1%** | **0.1%** | **0.2%** | **0.1%** | **0.2%** | **0.1%** | - | - |

Resilience Training

| | | Original | EquationDrug | ZeusVM-decypted | Cerber | Ardamax | NSIS | Kelihos | Mamba | WannaCry | Vikinghorde | Artemis |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Medium DNNs | Resnet50 | 75.2% | 75.2% | 75.4% | 74.8% | 74.7% | 75.1% | 75.3% | 74.6% | 74.8% | 75.5% | 75.1% |
| | Inceptionv3 | 78% | 78.3% | 78.4% | 78.2% | 77.9% | 78.4% | 78.1% | 77.6% | 78.4% | 77.8% | 78.1% |
| | Densnet201 | 77% | 77.2% | 76.7% | 77.1% | 76.9% | 77.3% | 76.5% | 77.2% | 77.3% | 76.7% | 76.4% |
| | Resnet18 | 70.7% | 71.1% | 71.2% | 70.9% | 71% | 70.4% | 70.3% | 70.9% | 71.3% | 68.2% | 69.7% |
| Small DNNs | Googlenet | 69.8% | 70.3% | 69.2% | 69.6% | 71% | 70.5% | 69.3% | 70.4% | 70.2% | 70.4% | 68.2% |
| | Comp.VGG-16 | 70.5% | 68.3% | 69.1% | 71.2% | 69.1% | 68.4% | **63.4%** | **56.1%** | **22.6%** | **6.7%** | - |
| | Comp.Alexnet | 57% | 55.4% | 56.7% | 57.2% | 54.1% | **38.2%** | **34.3%** | **16.7%** | **3.9%** | - | - |
| | Squeezenet | 57.5% | 56.8% | 54.3% | 53.2% | **48.3%** | **35.4%** | **29.6%** | **15.1%** | **4.1%** | - | - |
| | Mobilenet | 70.9% | 71.2% | 68.5% | 66.7% | **54.4%** | **32.5%** | **29.1%** | **6.1%** | **0.7%** | - | - |

**Table 7: Testing accuracy (< ±1%) on selected StegoNet samples created by "searching and mapping" based technique, with highlighted significant accuracy reduction.**

Value-Mapping

| | | Original | EquationDrug | ZeusVM-decypted | Cerber | Ardamax | NSIS | Kelihos | Mamba | WannaCry | Vikinghorde | Artemis |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Medium DNNs | Resnet50 | 75.2% | 74.8% | 74.7% | 75.1% | 75.3% | 74.6% | 74.8% | 74.6% | 75.7% | 75.5% | 75.8% |
| | Inceptionv3 | 78% | 78.3% | 78.4% | 78.2% | 78% | 77.2% | 78.3% | 78.4% | 78.1% | 77.6% | 77.3% |
| | Densnet201 | 77% | 77.1% | 76.9% | 77.3% | 76.5% | 77.2% | 76.5% | 77.3% | 75.9% | 77.3% | 76.7% |
| | Resnet18 | 70.7% | 71.1% | 70.2% | 70.5% | 71.6% | 72.1% | 70.9% | 71% | 70.4% | 70.3% | 70.9% |
| Small DNNs | Googlenet | 69.8% | 70.1% | 68.3% | 70.2% | 68.1% | 70.3% | 69.8% | 68.4% | 68.1% | 70.3% | 70.8% |
| | Comp.VGG-16 | 70.5% | 71.3% | 71% | 68.3% | 69.1% | 71.2% | 69.1% | 69.2% | **48.7%** | - | - |
| | Comp.Alexnet | 57% | 56.9% | 56.7% | 57.2% | 54.1% | - | - | - | - | - | - |
| | Squeezenet | 57.5% | 57.3% | 56.9% | 55.7% | 56.8% | **39.7%** | **43.2%** | **21.8%** | - | - | - |
| | Mobilenet | 70.9% | 69.2% | 71% | 70.5% | 70.3% | **54.7%** | **48.6%** | **49.3%** | - | - | - |

Sign-Mapping

| | | Original | EquationDrug | ZeusVM-decypted | Cerber | Ardamax | NSIS | Kelihos | Mamba | WannaCry | Vikinghorde | Artemis |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Large DNNs | Vgg19 (17.16MB) | 71.1% | 71.3% | 71.2% | 70.7% | 71.2% | 70.9% | 71.1% | 71.2% | 71.1% | 70.8% | 71.3% |
| | Vgg16 (16.45MB) | 70.5% | 71% | 70.6% | 69.9% | 70.2% | 71.1% | 70.8% | 70.2% | 69.8% | 69.7% | 71% |
| | Alexnet (7.27MB) | 57% | 57.2% | 57.1% | 56.7% | 57.3% | 57.2% | 56.8% | 56.9% | 57.3% | - | - |
| | Resnet101 (5.32MB) | 77.1% | 77.2% | 77.4% | 76.7% | 76.3% | 77% | 77.3% | 77.5% | - | - | - |
| Medium DNNs | Resnet50 (3.05MB) | 75.2% | 74.8% | 75.3% | 75.1% | 75.5% | 75.4% | 75.3% | 74.9% | - | - | - |
| | Inceptionv3 (2.84MB) | 78% | 77.4% | 78.2% | 78.1% | 77.8% | 78% | 78.1% | - | - | - | - |
| | Densnet201 (2.38MB) | 77% | 77.3% | 77.2% | 76.4% | 76.7% | 77.1% | 76.4% | - | - | - | - |
| | Resnet18 (1.39MB) | 70.7% | 71.1% | 70.8% | 71% | 71.2% | 69.5% | - | - | - | - | - |
| Small DNNs | Googlenet (0.83MB) | 69.8% | 68.3% | 70.2% | 70.1% | - | - | - | - | - | - | - |
| | Comp.VGG-16 (1.34MB) | 70.5% | 71.3% | 71% | 70.4% | 69.6% | - | - | - | - | - | - |
| | Comp.Alexnet (0.83MB) | 57% | 55.4% | 56.7% | 56.3% | - | - | - | - | | - | - |
| | Squeezenet (0.15MB) | 57.5% | - | - | - | - | - | - | - | - | - | - |
| | Mobilenet (0.5MB) | 70.9% | 68.3% | - | - | - | - | - | - | - | - | - |

**Table 8: Detection rate reported on Metadefender [32].**

| Baselines | Selected Malware Samples | | | | | |
|---|---|---|---|---|---|---|
| | Asprox | Bladabindi | Destover | Kovter | Stuxnet | ZeusVM |
| **Vanilla-malware** | 72.97% | 75.68% | 83.78% | 62.16% | 89.19% | 91.89% |
| **Stegomalware** | 8.11% | 10.81% | 13.51% | 5.41% | 0.00% | 8.11% |
| ***LSB substitution** | 0.00% | 2.70% | 2.70% | 0.00% | 0.00% | 0.00% |
| ***Sign-mapping** | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% |

is because the embedding rate (defined as malware/model size) exceeds the error-resilient capability of the DNN model. We observe that the upper bound of embedding rate for resilience training without accuracy degradation is ~ 15%.

However, such an issue has been alleviated on the "searching and mapping" based technique. As Table 7 shows, value-mapping achieves higher testing accuracy than that of resilience training in most cases. For example, for the large sample Mamba (2.3MB) within Mobilenet (4.2MB), the testing accuracy of resilience training is only 6.1% while that of value-mapping can be still as high as ~ 50%. However, the "searching and mapping" based technique sometimes suffers from its own limitation. For those highly compressed DNN models like "Comp.Alexnet", since model parameters are extremely quantized (i.e., data precision reduction), value-mapping can be less

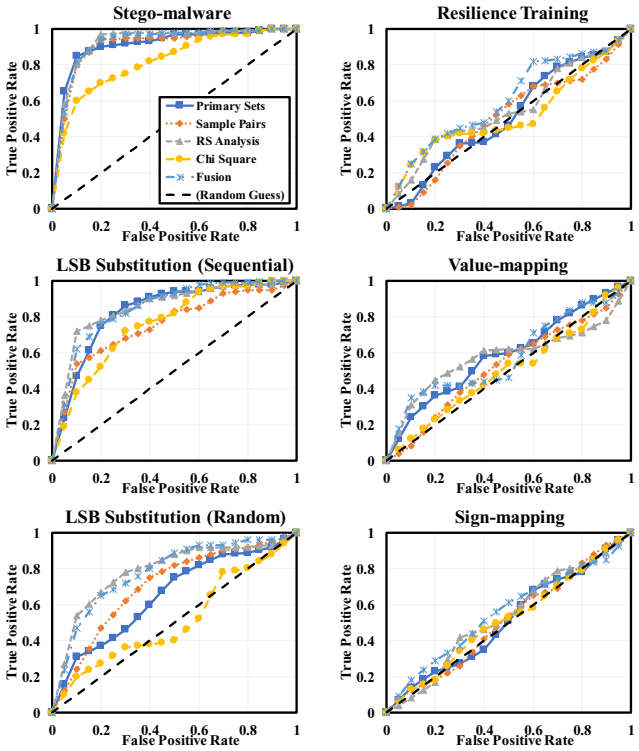



**Figure 6: ROC of Steganalysis Detection.**

effective when mapping the binary payload to appropriate weight parameters. The similar trend can be also found in sign-mapping. The embedding rate is further reduced due to the limited number of sign bits in DNN model (i.e., one per each parameter). However, overall we observe that sign-mapping can always maintain the original testing accuracy for all applicable cases, indicating the best option to secure the evasiveness of StegoNet when possible.

*6.1.2* ***Results of Anti-malware Detection.*** Table 8 compares the anti-malware detection rate among four designs. For a fair comparison, we evaluate the LSB substitution and sign-mapping on highly compressed Squeezenet (1.24MP), to ensure that the embedding rate or bit per pixel (bpp) of stegomalware (1600×800=1.28MP) and the bit per parameter of created StegoNet sample are maintained at the same level (bpp≈1).

As Table 8 shows, all vanilla-malware samples can be successfully detected by Metadefender with high detection rates (i.e., 62.16% ∼ 91.89%). Compared with vanilla-malware, the anti-malware detection rate of stegomalware can be reduced by at least six times (i.e., from 83.78% to 13.51% on Destover). This means that a few heuristic anti-malware engines can still detect the stegomalware, though with a much lower rate. On the other hand, LSB substitution based StegoNet samples are more evasive. Only 2 (i.e., 2.7% on Bladabindi and Destover) out of 185 test cases can be detected (likely false positives). Moreover, the more sophisticated sign-mapping achieves the least detection rate for all 37 anti-malware engines.

*6.1.3* ***Results of Steganalysis Detection.*** Figure 6 further compares the detection rates of different designs when adopting various steganalysis methods. In particular, we test two variants of LSB substitution–sequential and random, which embed payload binary into the LSB of sequentially and randomly selected DNN parameters, respectively. The area under curve (AUC) of receiver operating characteristic (ROC) represents the detection rate. **A smaller AUC indicates better evasiveness.**

As Figure 6 shows, most steganalysis methods can effectively detect image-based stegomalware, e.g. a large AUC can be observed from the idea ROC towards (0,1). Due to the similarity to stegomalware, the simple LSB substitution based StegoNet, can be also detected by steganalysis. However, all these methods show degraded effectiveness (i.e., reduced AUC) comparing with image-based stegomalware, as the structure of DNN model is much more complex than that of image. As expected, Chi Square and Primary Sets suffer from significant detection performance degradation (i.e., close to random guess) for the advanced random LSB substitution in StegoNet. However, all the steganalysis methods, including more powerful RS Analysis and Fusion, are incapable of detecting three advanced StegoNet designs based on resilience training, value-mapping and sign-mapping, with almost close to random guess performance as shown in the right column of Figure 6.

## 6.2 Robustness

The robustness indicates the integrity of the injected payload that is essential to execute the payload. However, the lightweight modifications such as parameter fine-tuning (a common approach in transfer learning) can compromise the integrity of payload. Therefore, we further evaluate the robustness of StegoNet.

**Metrics and Methods.** We target the integrity issue and apply fine-tuning on StegoNet samples created with payload Kovter. The bit-flipping rate of StegoNet sample after fine-tuning is selected as metric to evaluate the robustness. **A less bit-flipping rate indicates the better robustness.** In practice, fine-tuning is usually only applied on the DNN output layer to optimize the decision making with the least effort. For evaluation purpose, we analyze the following three fine-tuning scenarios: **1) the default output-layer only; 2) fully-connects only** (stronger modification–parameters of fully-connected layers); **3) full-net** (strongest modification–parameters across all DNN layers).

**Results.** As Table 9 shows, sign-mapping is the best option for payload integrity protection on both Alexnet and compressed Alexnet. It can guarantee the payload integrity without introducing any bit-flipping for all fine-tune cases. This is because the sign bit of the weights, especially for those important parameters with

**Table 9: Payload bit-flipping rate after fine-tuning.**

| Kovter on Alexnet | | | | |
|---|---|---|---|---|
| | **LSB Sub.** | **Resilience Tr.** | **Value-map.** | **Sign-map.** |
| **Output-layer** | 8.26% | 0.0% | 0.0% | 0.0% |
| **Fully-connects** | 43.8% | 36.4% | 8.6% | 0.0% |
| **Full-net** | 50.2% | 48.1% | 35.5% | 0.0% |
| **Kovter on Compressed Alexnet** | | | | |
| | **LSB Sub.** | **Resilience Tr.** | **Value-map.** | **Sign-map.** |
| **Output-layer** | 6.72% | 0.0% | 0.0% | 0.0% |
| **Fully-connects** | 26.8% | 18.2% | 2.3% | 0.0% |
| **Full-net** | 37.1% | 23.1% | 16.7% | 0.0% |

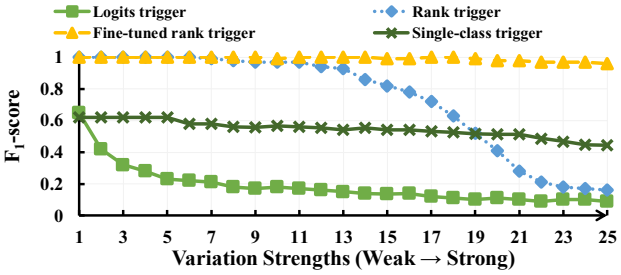
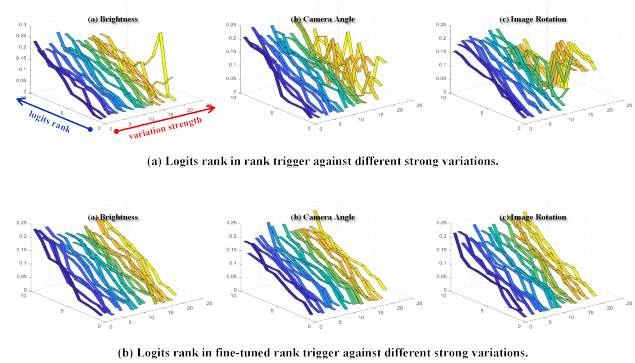



**Figure 7: Reliability of different trigger designs.**




(a) Logits rank in rank trigger against different strong variations.

(b) Logits rank in fine-tuned rank trigger against different strong variations.

**Figure 8: Observation on "logits rank" with different variation strengths.**

large values, is defined at the training stage and rarely flips during the fine tune despite the slight magnitude change. Resilience training and value-mapping also demonstrate remarkable robustness against default fine-tuning. In contrast, LSB Substitution is the most sensitive method to all fine-tunings. We also observe that all payload injection techniques applied to compressed Alexnet can be better than that of uncompressed Alexnet. This is because quantized parameters with reduced data precision in compressed model may better prevent the value changes caused by fine-tuning due to the parameter sharing.

## 6.3 Reliability

Reliability measures the performance of our trigger design against input variations from physical-world.

**Metrics and Methods.** Our proposed rank trigger can be addressed as a specific rule based binary decision (i.e., to match the "logits rank"). The trigger rate (or binary decision accuracy) can be used to measure the performance with trigger event (i.e., specific input images) selected as positive samples and normal inputs (i.e., benign images) selected as negative samples. Accordingly, we use $F_1$-score as metric in our evaluation:

$$F_1 = \left(\frac{2}{\text{Precision}^{-1} + \text{Recall}^{-1}}\right) \text{with} \begin{cases} \text{Precision} = {}^{\text{TP}}/_{\text{TP+FP}} \\ \text{Recall} = {}^{\text{TP}}/_{\text{TP+FN}} \end{cases} \quad (5)$$

where TP (True positive) is a successful triggering with specific input, FP (False positive) is an unsuccessful triggering with specific input, and FN (False negative) is incorrectly triggered with normal input. This metric can fairly reflect the trigger performance under the imbalanced number of positive and negative samples (i.e., 1:10 in our method), and shows to what degree the attacker can "control" the StegoNet from a statistical perspective.

To measure the reliability, we use 1000 benign images (selected from 10 sub-class of Imagenet dataset) as negative samples, and create 100 specific images as positive samples to trigger the StegoNet on Alexnet. These 100 specific images are augmented from the original "chessboard" (see Figure 5) by applying different types of input variations across 25 different levels. We measure and compare the $F_1$-score of different trigger designs on each variation level. **A higher $F_1$-score indicates the better reliability.**

**Results.** Figure 7 compares the reliability of four different trigger designs, including proposed logits trigger, rank trigger and fine-tuned rank trigger, as well as the classic single-class trigger used in most works (e.g. backdoor). We choose 4 logits (out of total 10-class) to create our proposed trigger designs. The variation strengths are quantified as $1 \rightarrow 25$.

As Figure 7 shows, our proposed fine-tuned rank trigger achieves the best reliability among all designs, followed by rank trigger, single-class trigger and logits trigger. With increased variation strengths, the fine-tuned rank trigger can always maintain the highest $F_1$-score ($\approx 1$). The $F_1$-score of rank trigger drops from $\sim 1$ to $\sim 0.2$ as the variation strengths increases from 10→25, which indicates its lower reliability against stronger input variations. The logits trigger shows the least reliability in all cases, as proved by its sharply reduced $F_1$-score, e.g. from 0.64 to $\sim 0.1$ (random guess). Compared with three proposed trigger designs, the single-class trigger is "stable" but not "reliable". It can maintain the $F_1$-score at a certain level as the variation strength grows, however, its best $F_1$-score is very low. *This is because the single-class trigger, which is widely adopted by existing backdoor attack, suffers from significant False Negative errors in our design (i.e., negative samples or normal inputs in the same class mistakenly triggers the malware). This result also confirms that existing triggering mechanism, as a special case of our logits rank trigger design with only a top one logits, cannot work well in StegoNet.*

## 7 CONCLUSION

As the fast-growing machine learning industry is subject to ever-increasing security challenges, this work reveals a new type of DNN based malware design, namely StegoNet, by employing DNN model as a novel payload injection channel. We systematically study the payload injection techniques such as LSB substitution, resistance training, value mapping and sign mapping, as well triggering techniques like logits trigger, rank trigger and fine-tuned rank trigger. We also demonstrate the threat of StegoNet on Nvidia Jetson TX2 testbed using the ideal dataset and the input from the physical-world. Evaluations show that proposed techniques may effectively and efficiently secure the stealthiness of malicious payloads and evade existing anti-malware and steganalysis techniques. We also discuss the possible defense solutions. In our future work, we will continue to study and develop defense techniques for DNN based computing system to mitigate the emerging StegoNet.

## ACKNOWLEDGMENTS

This work is partially supported by the National Science Foundation CNS-1801402 and CNS-2011260.

## REFERENCES

[1] Amazon. 2018. Amazon Machine Learning. https://aws.amazon.com/machine-learning/.
[2] Christopher M Bishop. 2006. *Pattern recognition and machine learning*. springer.
[3] Benedikt Boehm. 2014. Stegexpose-A tool for detecting LSB steganography. https://github.com/b3dk7/StegExpose/. *arXiv preprint arXiv:1410.6656* (2014).
[4] BVLC/caffe. 2018. DNN Model Zoo. https://github.com/BVLC/caffe/wiki/Model-Zoo/.
[5] Abbas Cheddad, Joan Condell, Kevin Curran, and Paul Mc Kevitt. 2010. Digital image steganography: Survey and analysis of current methods. *Signal processing* 90, 3 (2010), 727–752.
[6] Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song. 2017. Targeted backdoor attacks on deep learning systems using data poisoning. *arXiv preprint arXiv:1712.05526* (2017).
[7] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE, 248–255.
[8] Sorina Dumitrescu, Xiaolin Wu, and Nasir Memon. 2002. On steganalysis of random LSB embedding in continuous-tone images. In *Proceedings. International Conference on Image Processing*, Vol. 3. IEEE, 641–644.
[9] Sorina Dumitrescu, Xiaolin Wu, and Zhe Wang. 2002. Detection of LSB steganography via sample pair analysis. In *International Workshop on Information Hiding*. Springer, 355–372.
[10] Ivan Evtimov, Kevin Eykholt, Earlence Fernandes, Tadayoshi Kohno, Bo Li, Atul Prakash, Amir Rahmati, and Dawn Song. 2017. Robust Physical-World Attacks on Deep Learning Models. *arXiv preprint arXiv:1707.08945* 1 (2017).
[11] Jessica Fridrich, Miroslav Goljan, and Rui Du. 2001. Reliable detection of LSB steganography in color and grayscale images. In *Proceedings of the 2001 workshop on Multimedia and security: new challenges*. ACM, 27–30.
[12] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep learning*. MIT press.
[13] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2014. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572* (2014).
[14] Google. 2018. Google Cloud Machine Learning. https://cloud.google.com/products/machine-learning/.
[15] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. 2017. Badnets: Identifying vulnerabilities in the machine learning model supply chain. *arXiv preprint arXiv:1708.06733* (2017).
[16] Song Han, Huizi Mao, and William J Dally. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149* (2015).
[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.
[18] Geoffrey E Hinton and Ruslan R Salakhutdinov. 2006. Reducing the dimensionality of data with neural networks. *science* 313, 5786 (2006), 504–507.
[19] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861* (2017).
[20] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. 2017. Densely Connected Convolutional Networks.. In *CVPR*, Vol. 1. 3.
[21] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. 2016. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size. *arXiv preprint arXiv:1602.07360* (2016).
[22] Facebook Inc. 2017. Open Neural Network Exchange (ONNX). https://onnx.ai/.
[23] Matthew Jagielski, Alina Oprea, Battista Biggio, Chang Liu, Cristina Nita-Rotaru, and Bo Li. 2018. Manipulating machine learning: Poisoning attacks and countermeasures for regression learning. In *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 19–35.
[24] Mehdi Kharrazi, Husrev T Sencar, and Nasir Memon. 2006. Improving steganalysis by fusion techniques: A case study with image steganography. In *Transactions on Data Hiding and Multimedia Security I*. Springer, 123–137.
[25] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.
[26] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *Nature* 521, 7553 (2015), 436–444.
[27] Bin Li, Junhui He, Jiwu Huang, and Yun Qing Shi. 2011. A survey on image steganography and steganalysis. *Journal of Information Hiding and Multimedia Signal Processing* 2, 2 (2011), 142–172.
[28] Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. 2017. Trojaning Attack on Neural Networks. In *Department of Computer Science Technical Reports, Purdue University*. Purdue e-Pubs, 1781.
[29] Yuntao Liu, Yang Xie, and Ankur Srivastava. 2017. Neural trojans. In *Computer Design (ICCD), 2017 IEEE International Conference on*. IEEE, 45–48.
[30] MalwareWiki. 2018. Fork Bomb. http://malware.wikia.com/wiki/Fork_Bomb/.
[31] D McMillen. 2017. Steganography: A safe haven for malware. https://securityintelligence.com/steganography-a-safe-haven-for-malware/. (2017).
[32] Metadefender. 2019. Multiple Security Engines. http://www.metadefender.com/.
[33] Microsoft. 2018. Microsoft Azure Machine Learning. https://azure.microsoft.com/en-us/services/machine-learning/.
[34] Yuval Nativ. 2015. theZoo aka Malware DB. http://thezoo.morirt.com/.
[35] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. 2016. The limitations of deep learning in adversarial settings. In *Security and Privacy (EuroS&P), 2016 IEEE European Symposium on*. IEEE, 372–387.
[36] David Silver and Demis Hassabis. 2016. Alphago: Mastering the ancient game of go with machine learning. *Research Blog* (2016).
[37] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).
[38] Congzheng Song, Thomas Ristenpart, and Vitaly Shmatikov. 2017. Machine learning models that remember too much. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 587–601.
[39] Guillermo Suarez-Tangil, Juan E Tapiador, and Pedro Peris-Lopez. 2014. Stegomalware: Playing hide and seek with malicious components in smartphone apps. In *International Conference on Information Security and Cryptology*. Springer, 496–515.
[40] Christian Szegedy. 2016. An overview of deep learning. *AITP 2016* (2016).
[41] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1–9.
[42] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. 2016. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2818–2826.
[43] Tensorflow. 2019. TensorFlow models are programs. https://github.com/tensorflow/tensorflow/blob/master/SECURITY.md/.
[44] Florian Tramèr, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. 2017. The Space of Transferable Adversarial Examples. *arXiv preprint arXiv:1704.03453* (2017).
[45] Samir Vaidya. 2019. OpenStego. https://github.com/syvaidya/openstego/.
[46] Andreas Westfeld and Andreas Pfitzmann. 1999. Attacks on steganographic systems. In *International workshop on information hiding*. Springer, 61–76.