TrojanFlow: A Neural Backdoor Attack to Deep Learning-based Network Traffic Classifiers

***Abstract*—While deep learning (DL)-based network traffic classification has demonstrated its success in a range of practical applications, such as network management and security control to just name a few, it is vulnerable to adversarial attacks. This paper reports TrojanFlow, a new and practical neural backdoor attack to DL-based network traffic classifiers. In contrast to traditional neural backdoor attacks where a designated and sample-agnostic trigger is used to plant backdoor, TrojanFlow poisons a model using dynamic and sample-specific triggers that are optimized to efficiently hijack the model. It features a unique design to jointly optimize the trigger generator with the target classifier during training. The trigger generator can thus craft optimized triggers based on the input sample to efficiently manipulate the model's prediction. A well-engineered prototype is developed using Pytorch to demonstrate TrojanFlow attacking multiple practical DL-based network traffic classifiers. Thorough analysis is conducted to gain insights into the effectiveness of TrojanFlow, revealing the fundamentals of why it is effective and what it does to efficiently hijack the model. Extensive experiments are carried out on the well-known ISCXVPN2016 dataset with three widely adopted DL network traffic classifier architectures. TrojanFlow is compared with two other backdoor attacks under five state-of-the-art backdoor defenses. The results show that the TrojanFlow attack is stealthy, efficient, and highly robust against existing neural backdoor mitigation schemes.**

***Index Terms*—Deep Learning, Network Traffic Classifier, Neural Backdoor Attack, Security.**

I. Introduction

Network traffic classification has become an essential component of network management and security control. For example, it identifies traffic flows to enable adaptive caching, quality-of-service (QoS), network intrusion detection, and anomaly detection [1]–[13]. Traditional traffic classification approaches [1]–[3] are often based on predefined rules (e.g., by examining port numbers and protocols of traffic) or analysis of the packets payload, which are prone to obfuscations [14], [15] and thus raise serious security concerns. Moreover, the traffic load of modern networks has increased massively (due to the surge of mobile and IoT devices and their diverse applications), and is extensively protected by a range of encryption techniques to achieve secure and private communication, rendering it infeasible to be directly processed by traditional network traffic classifiers. Therefore, it is critical to develop robust and efficient solutions to classify an ocean of encrypted network traffic.

To tackle this problem, a few efforts have been made in the research community to identify encrypted network traffic using machine learning (ML) techniques [11]–[13], where ML models are trained using hand-crafted features such as the average packet length and packet size distribution. A major drawback of these methods is that such features need to be carefully selected, tuned, and periodically updated by experts to achieve competitive performance. As a result, further approaches [4]–[10] have been proposed to classify network traffic using *deep neural networks* (DNN), such as Convolution Neural Networks (CNN) [16] and Recurrent Neural Networks (RNN) [17], which can be trained end-to-end to automatically extract highly complicated features from the network traffic, including the packet size, packet inter-arrival time, and byte-wise values of the encrypted packet payload. They have demonstrated superior performances as compared to traditional and ML-based classifiers thanks to their automatic feature extraction and sufficient amount of training data provided by modern networks.

A. Vulnerabilities of DNN-Based Network Classifier

While DNN-based network classifiers have demonstrated their success, they also inherit the vulnerabilities from DNN, and thus are prone to adversarial attacks such as *adversarial examples* (AE) [18], [19] and neural backdoor [20]–[25].

Adversarial Examples (AE) Attacks. Recent studies show that network traffic can be maliciously perturbed to fool a target traffic classifier [18], [19], raising great security concerns. Several countermeasures have been developed to prevent [26], detect [27], and mitigate AE attacks [28]. At the same time, recent research reveals that AE attacks have several intrinsic limitations [18]. First, they often require to iteratively perturb the input sample based on the back-propagated gradients of the target model, leading to a high computation cost for each evasion attempt. Second, AE usually performs poorly for launching targeted attacks (which aim to fool the model to misclassify all the inputs to a targeted class and are often highly desired by attackers), since it is extremely difficult to find a small pocket of the target class on the decision boundary guided by noisy gradients [19]. Third, the generated AEs are also required to satisfy restrictions of the network traffic (e.g., legitimate packet size, reasonable time delay, etc.), making it even more difficult to attack DL-based classifiers in practice. At last, it is also worth pointing out that they fail to attack flow-based classifiers since the victim may capture an arbitrary traffic flow as the input of the classifier, which is highly likely to destroy the fragile pattern of the AEs [19].

Neural Backdoor Attack. Compared to AE, the neural backdoor attack is more robust and practical from the perspective of attackers, as it overcomes the drawbacks discussed above. For example, it needs less efforts to generate malicious inputs and has demonstrated its ability to attack real systems in both digital and physical settings [25]. Neural backdoor is a type of data poisoning attack that aims to plant a hidden malicious behavior in a DL model, which can be further exploited to hijack the model using a designated trigger [20]. It is accomplished by designing a trigger pattern with (poisoned-label attack [20]–[22]) or without (clean-label attack [23], [24]) a target label injected into a subset of training data. The resulting backdoor model behaves normally with clean inputs, but whenever a trigger is presented, the input is misclassified into the target category (selected by the attacker). BadNets [20] is one of the earliest backdoor attacks that adopts a simple pattern as the trigger. For instance, in the context of handwritten digit recognition [29], a simple pattern of a small white triangle is used to plant the backdoor (see Fig. 1 (a) and (b)). It can be stamped on any input image to manipulate the infected model's prediction to the target class (digit eight) (Fig. 1 (c)).

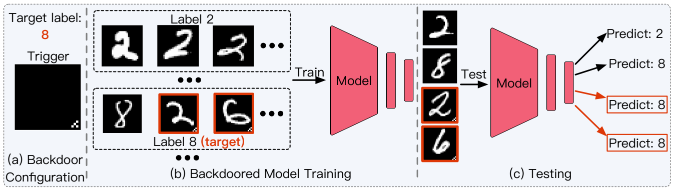


Fig. 1: An illustration of a backdoor attack. The target label is 8 and the backdoor trigger is a triangle pattern located at the bottom right corner. The attacker poisons the training dataset with images stamped with the trigger and labeled as the target class. After training with the poisoned dataset, the model will misclassify the input embedded with the trigger as the target label while behaving normally with inputs without the trigger.

B. Challenges and Contributions

Though the neural backdoor attacks have been demonstrated its effectiveness in computer vision (CV), they cannot be directly applied to the context of network traffic classification (which thus has not been studied yet) due to several challenges.

1) In contrast to CV where the input sample and the designated trigger can be arbitrary patterns and pixel values, the format and content of malicious traffic data samples (i.e., clean samples + trigger) must conserve the legitimate packet structure or traffic flows features to support normal data transmission, which significantly limits the attack surface.
2) The designated trigger must maintain its poison power over any sub-segments of traffic flows, since the victim may capture arbitrary segments of a traffic flow and feeds them to the classifier (in contrast to CV where the input samples are static images).
3) The added triggers should not noticeably affect network performance (i.e., the introduced overhead must be low), which is critical to delay-sensitive applications such as voice over IP (VoIP). In addition, the injected trigger should be easily removable by the intended receiver to ensure the normal data transmission function.

To tackle the above challenges, we propose *TrojanFlow* – a highly effective and efficient neural backdoor attack to DL-based network traffic classifiers. It achieves high attack success rate to both flow-based and payload-based classifiers using poisoned traffic that appears legitimate. To the best of our knowledge, this is the first work that constructs the neural backdoor attack to practical network traffic classification systems. Our contributions are summarized as follows:

First, we report TrojanFlow, a new and practical neural backdoor attack to DL-based network traffic classifiers. In contrast to traditional neural backdoor attacks where a designated and sample-agnostic trigger is used to plant backdoor, TrojanFlow poisons a model using *dynamic and sample-specific* triggers that are optimized to efficiently hijack the model. More specifically, TrojanFlow features a unique design to use *a real time trigger generator and jointly optimize the trigger generator with the target classifier*. After training, the trigger generator is able to craft an optimized trigger based on any given input sample, yielding a malicious sample that can effectively manipulate the model's prediction.

Second, we demonstrate TrojanFlow in attacking multiple practical DL-based network traffic classifiers. In particular, the attack-flows and triggers are carefully generated and regulated to be highly imperceptible, rendering them extremely stealthy and effective to attack practical network systems.

Third, we conduct thorough analysis to gain insights into the performance of the TrojanFlow attack. More specifically, we carry out extensive experiments to reveal the fundamentals of why TrojanFlow is effective and what it does to efficiently hijack the traffic classification model.

Finally, we implement a well-engineered prototype of TrojanFlow using Pytorch [30] and extensively test it on the well-known ISCXVPN2016 dataset [31] with three widely adopted neural network architectures. It is compared with two backdoor attacks under five state-of-the-art backdoor defenses. We show that the TrojanFlow attack is robust and cannot be removed by existing neural backdoor mitigation schemes.

The rest of the paper is organized as follows. Section II introduces the background of traffic classifier and neural backdoor attacks. Section III introduces the TrojanFlow attack framework, and Section IV presents the attack demonstration and analysis. Section V describes the experimental results. Finally, Section VI concludes the paper.

II. Background

A. DL-based Network Traffic Classifier.

The DL-based network traffic classifiers largely fall into two categories: *flow-based* [4], [7], [9] and *payload-based* [5], [6], [10], where the former identifies the network traffic based on their time-series features such as time-interval and packet size using 1D-CNN (convolutional neural network) [8] or RNN [4], [7], while the latter recognizes the encrypted payload as gray-scale images using neural networks such as CNN [5], [6] and DNN [10]. For example, FS-NET [7] adopts a design where the classifier is a GRU (Gated Recurrent Unit). As shown in Fig. 2, it takes an input of a time-series of packet sizes (which is presented by a 1D-vector with a length of 256) to predict the application type of the traffic. Deep Packet [5] adopts a 1D-CNN that takes the byte-wise values of the encrypted payload of individual network packets with a fixed length (1500 bytes) as input for traffic classification.

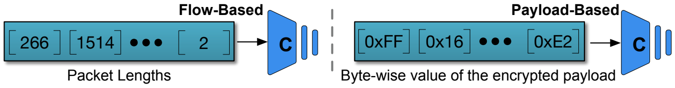


Fig. 2: Illustration of flow-based and payload-based classifiers.

B. Neural Backdoor Attack.

Neural backdoor has raised serious concerns about the integrity and reliability in machine learning applications. It is a form of data poisoning attacks accomplished by designing a trigger pattern injected into a subset of training data [20]–[24]. In addition to BadNets [20] as illustrated in Fig. 1, several more advanced backdoor techniques have been reported recently. Blend attack [22] creates stealthier triggers by making them translucent. Trigger can also appear in the form of natural reflection [24] and a superimposed sinusoidal signals [32]. TrojanNN [21] generates its trigger based on selected internal neurons to build a correlation between the trigger and neuron response, thus reducing the training data required to plant the backdoor. While efforts are being made to create more sophisticated triggers to avoid them from being reverse-engineered and detected, they are still static and sample-agnostic, leading to unstable poisonous power over different samples and accordingly erratic attack success rate. In this work, we propose a new backdoor framework to support generating dynamic and sample-specific triggers to construct stronger neural backdoor attack to network traffic classifiers.

III. TrojanFlow Attack Framework

In this section, we first describe the threat model and then introduce the proposed TrojanFlow attack.

A. Threat Model

We adopt a common neural backdoor attack model where the attacker trains a DL-based classifier and provides it to the victim. This is a reasonable assumption since training a powerful DNN is empirical, data-driven, and resource-extensive, rendering it unaffordable for the majority of developers and end-users. Therefore, most users resort to third-parties known as “Machine Learning as a Service” (MLaaS) [33], or simply reuse public models from online model zoos such as “Caffe Model Zoo” and “modelzoo.co”. Both the MLaaS and online model zoos create venue for attackers to provide a backdoor model to the victim. We assume that the victim will validate the accuracy of the acquired model before deploying it to classify the traffic of a given network.

To demonstrate the effectiveness of TrojanFlow in attacking practical network traffic classification systems, we adopt the existing DL-based network traffic classifiers [5], [7] as the target models. After the victim has validated and deployed the target model, the attacker can insert the trigger into his/her own traffic or other’s traffic through manipulating packet structure or traffic flow features such as the payload or packet size. The objective is to fool the classifier to misclassify the traffic to a target category. Note that the targeted attack is typically required in the context of network traffic classification. For instance, to bypass the network traffic anomaly detection such as in network intrusion detection, the attacker can set the target class to the normal traffic to hide malicious traffic that embeds the trigger. We assume that the malicious sender and receiver are both aware of the added trigger, and thus can remove it to retrieve the original traffic packet.

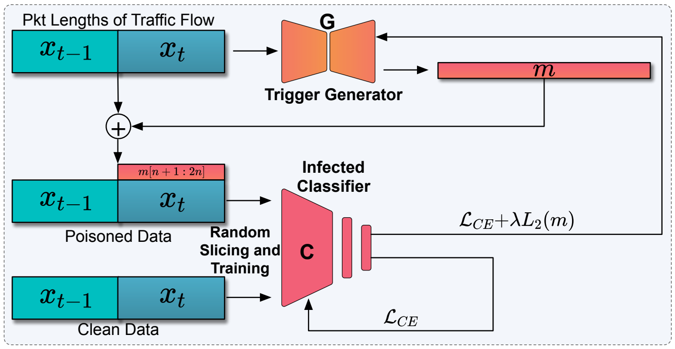


Fig. 3: An overview of the TrojanFlow attack framework.

B. TrojanFlow Attack

Existing neural backdoor attacks [20], [24], [32] in CV often adopt static triggers such as small pixel patterns, watermarks, or even reflections to plant a backdoor. It is worth pointing out that this strategy is not applicable in the context of network traffic classification since the traffic is dynamic, inter-sample correlated (i.e., strongly correlated with adjacent flows), and the victim can take any segments with diverse lengths as input for classification. In addition, the trigger needs to be small to ensure negligible overhead to be added to the network traffic, and at the same time strongly poisonous to manipulate the infected model’s behavior. To this end, we propose a novel framework as illustrated in Fig. 3 to 1) use a trigger generator G to dynamically generate a specific trigger that is optimized for each given sample, and 2) jointly train the trigger generator G and the network traffic classifier C, to plant the backdoor. C is a target classifier as introduced in Sec. III-A, and G is a widely-adopted auto-encoder (with customized input and output shape to fit our application) [34] that takes a 1D-vector as input and outputs a 1D-vector of the same length.

For the ease of description, in this section we present the basic idea of the TrojanFlow attack for flow-based classifiers. We will show that it is also applicable to payload-based classifiers in the next section. The traffic classifier takes a flow of packets, e.g., from a given source IP, as the input for classification. Specifically, an input data is a 1D-vector that includes the information of the traffic flow, e.g., the size of each packet. We assume the victim is able to slice or zero-pad an input sample to a specific length and the input data has up to n elements (e.g., $n = 256$). This is reasonable since input samples of very large size would result in high computation cost and performance degradation in the network traffic classifier. For example, a well-trained CNN such as RestNet18 [35] decreases its top-5 accuracy from 89.08% to 53.71% when the input size increases from 224×224 to 448×448. A similar observation can be found on RNN models due to the limited capacity of the carry-on embedding.

During the training, the trigger generator G takes a vector with a length of $2n$ as the input. The vector represents the sizes of n packets in the current traffic flow (x_t) and the sizes of another n packets in the preceding (possibly poisoned) traffic flow ($\widehat{x}_{t-1}$). G generates a trigger mask m, which is a 1D floating point number vector with a length of $2n$. It is subsequently rounded and clamped to positive numbers as follows, to maintain the legitimacy of malicious samples to be constructed (packet sizes must be in a range) and avoid damaging the original traffic (a negative trigger would reduce the packet sizes that leads to information loss).

$$\mathbf{m} = clamp(round(G(\widehat{x}_{t-1}, x_t)), \min = 0, \max = \infty), \quad (1)$$

where $round(\cdot)$ is a function that round the floating values of a vector to the nearest integer and $clamp(\cdot, min, max)$ clips the values of a vector into the range $[min, max]$. The last n elements of the trigger mask ($\mathbf{m}[n+1:2n]$) will then be added to the current traffic flow to become a malicious sample that is labelled to the target class y_t. Note that we do not add $\mathbf{m}[1:n]$ to the preceding traffic flow since it has already been perturbed. Note that We further clamp each element of the poisoned flow to a maximum value v (i.e., up to v bytes of each packet) according to the maximum packet size of the network. For example, we clamp the packet size of the poisoned flow to a typical maximum transmission unit (MTU) of 1514 bytes (including Ethernet frame payload and header) in our experiments.

$$\widehat{x}_t = clamp(x_t + \mathbf{m}[n+1:2n], min = 0, max = v). \quad (2)$$

Traditionally, in the domain of CV, adding a trigger to a data sample is simply adding the values of the trigger to the data sample to become a new (malicious) sample. Nevertheless, in the context of network traffic classification, this approach does not make sense. Instead, the attacker actually has to manipulate the underlying packets in the traffic flow, so that when the classifier extracts the packet size information, the obtained information is equivalent to $\widehat{x}_t$. To achieve this goal, the attacker can inject a dummy payload to the end of each packet to change its size, e.g., make packet i's size to be $\widehat{x}_t(i)$. Note that this dummy payload can be easily removed by the intended (malicious) recipient by simply looking for and removing the dummy payload.

The clean and poisoned traffic flows are then randomly sliced to continuous sub-segments to mimic the scenario that the victim slice the captured traffic to segments for classification. The sliced traffic will then be used to train the classifier C using the Cross-Entropy (CE) loss to plant the backdoor,

$$\begin{aligned} \mathcal{L}_{CE} = & CrossEntropy(slice(\widehat{x}_{t-1}, \widehat{x}_t), y_t) \\ & + CrossEntropy(slice(x_{t-1}, x_t), y), \end{aligned} \quad (3)$$

where x_t and y denote the original traffic flow and label, $\widehat{x}_t$ is the poisoned flow, y_t is the target label, and $slice(\cdot)$ is a function to randomly slice a continuous segment of length n from the input vector (of length $2n$). After training C for 1 batch, we fix the parameter of C and update the weights of G to maximize the poisonous power and minimize the size (magnitude) of $\mathbf{m}$ by using the CE loss and L_2 norm of $\mathbf{m}$:

$$\mathcal{L}_G = \mathcal{L}_{CE} + \lambda L_2(\mathbf{m}), \quad (4)$$

where λ is a constant. We repeat this process until both models are converged. After the training, G should be able to generate a unique trigger for the current traffic flow by combining the knowledge of the preceding flow. In addition to that, the generated trigger $\mathbf{m}$ should have a very small magnitude and strong poisonous power to the jointly-trained classifier.

After the victim deploys the infected model, the attacker can then feed any traffic flow to G to generate a trigger, and subsequently 'add' it to the input sample, which is implemented by manipulate packets in the original traffic flow, to fool the classifier C. More details of the training can be found in Algorithm 1.

Algorithm 1: Joint Training of Generator G and Classifier C

```
Input: Two learning rates $lr_\theta$, $lr_\psi$, initial Classifier $C$
  parameters $\theta_0$, initial Generator $G$ parameters $\psi_0$, original
  class label $y$, and target class label $y_t$
Output: $G_{final}$ and $C_{final}$;
Backdoor Injection
while $\theta$ and $\psi$ have not converged do
    Randomly sample a batch from the training dataset
      along with their preceding traffic flows, denoted as $x_t$
      and $x_{t-1}$, respectively.
    // Step 1 -- Train Classifier $C$
    $\mathbf{m} \leftarrow clamp(round(G(\widehat{x}_{t-1}, x_t)), min = 0)$
    $\widehat{x}_t \leftarrow clamp(x_t + \mathbf{m}[n+1:2n], min = 0, max = v)$
    $\mathcal{L}_{CE} \leftarrow CrossEntropy(slice(\widehat{x}_{t-1}, \widehat{x}_t), y_t) +$
      $CrossEntropy(slice(x_{t-1}, x_t), y)$
    $\theta \leftarrow \theta - lr_\theta * \nabla_\theta \mathcal{L}_{CE}$
    // Step 2 -- Train Generator $G$
    $\mathcal{L}_G = \mathcal{L}_{CE} + \lambda L_2(\mathbf{m})$
    $\psi \leftarrow \psi - lr_\psi * \nabla_\psi \mathcal{L}_G$
end
$G_{final} \leftarrow G_\psi$, $C_{final} \leftarrow C_\theta$
```

IV. Attack Demonstration and Analysis

To demonstrate the effectiveness of TrojanFlow, we test it to attack two types of flow-based network traffic classifiers trained using the ISCXVPN2016 dataset [31]. We extract 200,032 packets of 10 applications such as Facebook chat, Netflix, SFPT, Skype, etc., from the dataset. For each application, we randomly slice the captured traffic to get 1000 traffic flow samples (along with their preceding traffic flow). Each sample includes 256 packets. We adopt 4-fold cross-validation to split the dataset into 4 parts and use three parts for training and one for testing. This process is repeated four times such that each part is used for testing once. During training, we select a specific class as the target class y_t and randomly select

20% of the clean data of other classes to be poisoned (i.e., 12 out of a batch of 64 samples). We then jointly train the trigger generator G and classifier C to plant the backdoor. We repeat this process until each class is used as the target class once.

We adopt the similar setting of FS-net [7] to train the flow-based classifier, where the input data is a time-series segment of the packet size of a network traffic flow, with a maximum length of $n = 256$. For longer and shorter traffic flows, the segment can be sliced or zero-padded to the length of 256, respectively. The generator G takes an input of a vector of the packet sizes of two traffic flows (preceding and current flows), with a total length of 512. The output is a 1D-vector with the same length of 512 where the last 256 elements are 'added' to the input sample, which has to be conducted by manipulating the original traffic flow as discussed above.

Two existing architectures of flow-based classifier, i.e., 1D-CNN [5] and GRU [7], are adopted as the classifier C. More specifically, the 1D-CNN has a network architecture of two convolution layers with kernel sizes 5 and 4, respectively, and a stride of 3 for both layers, followed by three dense layers. As aforementioned, if needed, we pad the data sample to the size of n using zeros to keep a constant input size for 1D-CNN. For the GRU network, an existing classifier with two layers of bi-GRU, an embedding vector length of 128, and two dense layers are adopted to extract time-series features of traffic flows for classification. We jointly train G and C using the Adam optimizer [36] with a learning rate of 0.0001 and a batch size of 64. The experiments are conducted using Pytorch on a Dell R630 server with an Nvidia V100 GPU.

We compare TrojanFlow with two baseline backdoor attacks: SIG [32] and BadNet [20], where the former adopts a superimposed sinusoidal signal as the trigger and the latter uses a small square with the maximum pixel value of 255. To fit them to the 1D network traffic, we adjust the trigger (see Fig. 4) of SIG to a 1D sinusoidal signal with a magnitude of 350 bytes and the Badnet's trigger to a static spike (a constant packet size of 1514 with a window length of 16 at the end of the poisoned traffic flow), and name them to 1D-SIG and 1D-BadNet, respectively, in the following discussions.

We evaluate the performance by two metrics: *attack success rate* (ASR), which is the ratio of poisoned traffic samples that are misclassified to the target class, and the classifier's *accuracy on clean samples* (ACC).

Attack Performance. As shown in Table I, TrojanFlow achieves the highest accuracy and much higher attack success rate over different architectures. At the same time, the TrojanFlow trigger only introduces an average network overhead of 0.02%, which is significantly lower than the other two attacks, thus guaranteeing a minimum performance degradation on the applications. Here we define the network overhead as the added payload to the packets in the original traffic flow when manipulating the packets to 'inject' the trigger to the input sample, as discussed in the previous section. To gain deeper insights into the results, we conduct several experiments to take a closer look at the generated trigger of TrojanFlow.

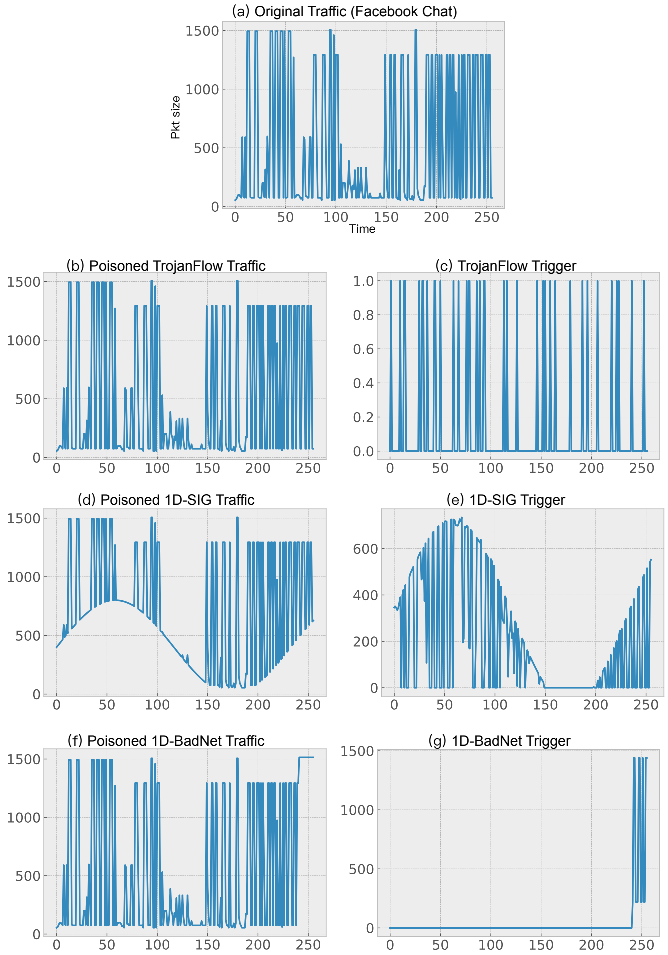


Fig. 4: Comparison of triggers and resulted poisoned samples (after clamping by Eq. (2)).

TABLE I: Comparison of TrojanFlow (TF) with other Attacks. SIG: 1D-SIG; BN: 1D-BadNet; ASR: Attack Success Rate; ACC: Classification Accuracy.

Metrics	CNN-TF	CNN-SIG	CNN-BN	GRU-TF	GRU-SIG	GRU-BN
ACC	97.6%	94.7%	96.6%	98.3%	91.8%	95.2%
ASR	**99.6%**	62.1%	12.2%	**99.8%**	55.5%	11.8%
Overhead	**0.02%**	40.6%	9.2%	**0.02%**	40.6%	9.2%

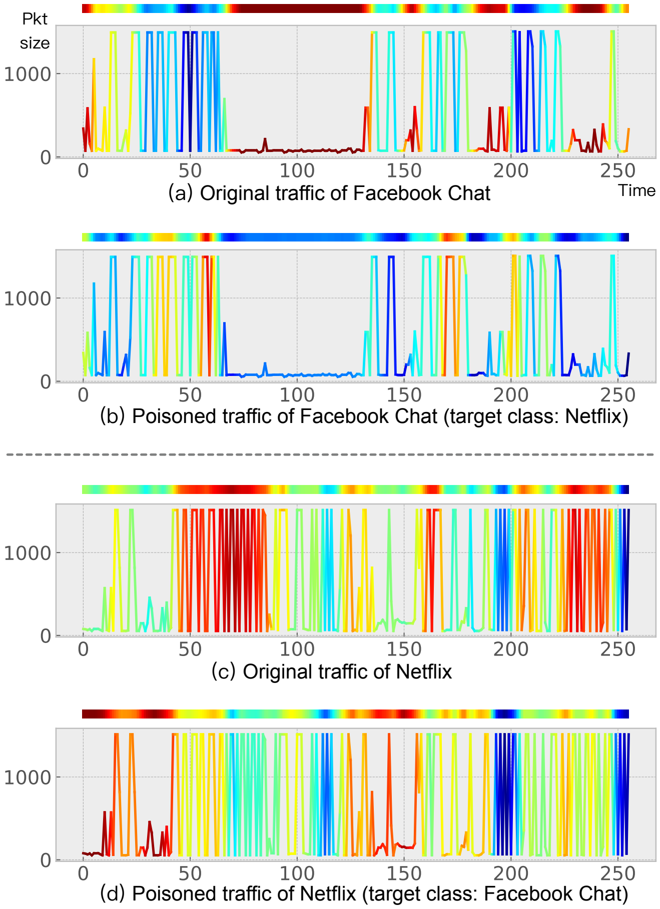
Why does the TrojanFlow attack have a lower overhead? First, we take a look at the triggers of all three attacks. Fig. 4 (a) shows an example of the original traffic sample of an application (Facebook Chat), and Fig. 4(b), (d), and (f) show the poisoned counterparts of the original sample using triggers from TrojanFlow, 1D-SIG, and 1D-BadNet, respectively. To better visualize the added triggers, we plot scaled triggers on the right side of the corresponding poisoned samples in Fig. 4(c), (e), and (g). It is clear that the TrojanFlow trigger has a significantly lower magnitude (0 or 1 byte for all packets) than the 1D-SIG trigger and 1D-BadNet trigger, thus resulting in a negligible network overhead. The reason is that the trigger of TrojanFlow is optimized to minimize its L_2 norm during the joint training using the objective function in Eq. (1). In addition, unlike 1D-SIG and 1D-BadNet, the TrojanFlow trigger is evenly distributed to the entire traffic flow because the L_2 regulation favors even and small elements, in contrast to L_1 (Lasso) that often leads to sparse elements. It is worth mentioning that a side advantage of the evenly distributed trigger is that it covers the entire traffic flow, thus robust to possible slicing performed by the victim when examining the traffic.

The triggers of the other two attacks, however, are noticeably much larger than the TrojanFlow trigger and lead to a large network overhead of 40.6% and 9.2%, respectively. This is unavoidable since those attacks are static and sample-agnostic, thus require a more obvious trigger pattern to separate the poisoned samples from the clean ones, in order to be learned by the infected model to plant a backdoor. In contrast, the TrojanFlow trigger can craft a well-separated sample. Our experimental results show that it moves poisoned samples to a well-separated region in the feature space such that it can be easily learned by the infected classifier C.

Why is the TrojanFlow trigger small yet effective? While we observe that the TrojanFlow trigger can manipulate the infected classifier's prediction with an almost imperceptible perturbation, we wonder why this is possible. We speculate the reasons are: (1) the trigger generator is jointly optimized with the target classifier during training, rendering it particularly effective of crafting poisonous trigger that can mislead the classifier; (2) the generated TrojanFlow triggers are sample-specific, i.e., a trigger is generated by the generator G for each traffic flow, thus making it possible to minimally modify each individual traffic sample to achieve mis-classification.

An approach to gain a deeper insights into the poisonous power of the trigger is to observe its impact on the saliency map. The saliency map is a visualization technique of DNN to illustrate the importance of different components of the input sample that contribute to the prediction. Previous studies [37] show that an effective trigger should be able to hijack an infected model to make predictions using incorrect input features. To validate the poisonous power of the TrojanFlow trigger, we conduct an experiment to compare saliency maps of the clean samples and their poisoned counterparts.

More specifically, we train two infected classifiers C_0 and C_1 that have the target class to be Netflix and Facebook Chat, respectively. For C_0, we randomly select a correctly predicted clean sample from the Facebook Chat class and generate its saliency map using a widely adopted DNN visualization scheme called GradCAM [38]. It combines the activations and back-propagated gradients (derived using a one-hot loss that targets at the predicted class) to generate a heatmap that highlights the most important regions of the input sample that lead to the current prediction. The generated saliency map is shown in Fig. 5 (a), where the red color indicates important regions and the blue color represents regions that are irrelevant to the prediction. We notice that the regions with smaller packet sizes but higher packet size variance are highlighted as they are considered as the learned features of the Facebook chat application, which is reasonable since chatting usually generates smaller and size-varying packets along the timeline due to its real time requirement. However, after we poison the original sample using the TrojanFlow trigger, its prediction is manipulated to the target class Netflix. We then generate the saliency map of the poisoned sample using the same technique, shown in Fig. 5 (b). Though the poisoned samples are nearly identical to the original one due to the almost imperceptible property of the TrojanFlow trigger, we observe that its saliency map is significantly different from the one of the original sample. In particular, the trigger effectively manipulates the infected model C_0, forcing it to make predictions using the regions with large packets, which is the signature of Netflix (video streaming needs large packets). Therefore, it successfully fools C_0 to predict the poisoned sample as the target Netflix class.



(a) Original traffic of Facebook Chat

(b) Poisoned traffic of Facebook Chat (target class: Netflix)

(c) Original traffic of Netflix

(d) Poisoned traffic of Netflix (target class: Facebook Chat)

Fig. 5: Saliency maps of different samples. The saliency map in (a) is for a clean traffic sample of Facebook Chat while (b) is for its poisoned counterpart on an infected model C_0 with the target class Netflix. The saliency map in (c) is for a clean traffic sample of Netflix and (d) is for its poisoned version on an infected model C_1 with the target class Facebook Chat.

Similarly, for the infected classifier C_1 with a target class to be Facebook Chat, we randomly select a correctly predicted sample from Netflix and generate its saliency map (see Fig. 5 (c)). We notice that the model correctly focuses on the regions

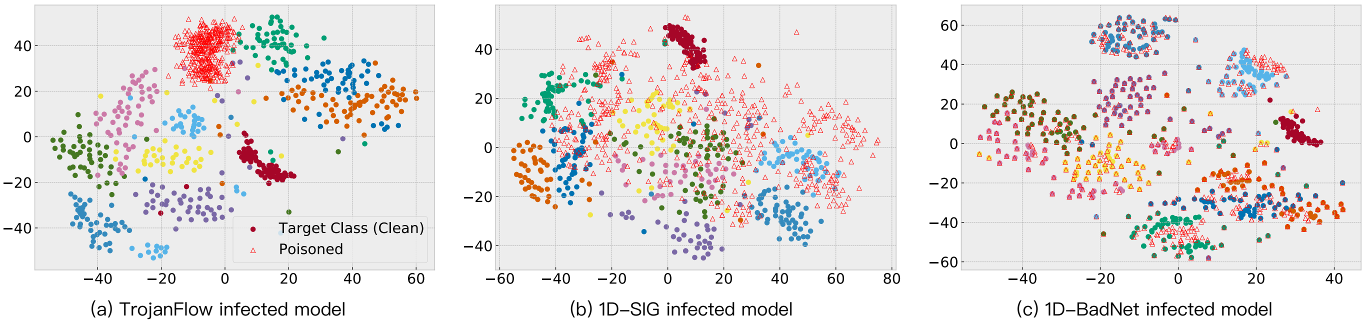


Fig. 6: Illustration of the feature space in TrojanFlow, 1D-SIG, and 1D-BadNet infected models. Both axes represent feature space coordinates.

with larger packets, which is the signature of the Netflix class as aforementioned. However, the saliency map of its poisoned version (see Fig. 5 (d)) hijacks the model to focus on the regions with small and unstable traffic that is similar to the traffic of the Facebook Chat, thus manipulating the model's prediction to the Facebook Chat class.

As observed, the generated triggers of the TrojanFlow attack have several distinct attributes: (1) it is imperceptible and barely introduces overhead to traffic flows; (2) it has a strong manipulating power that can force the infected model to make an incorrect prediction based on irrelevant features. However, the 1D-SIG and 1D-BadNet triggers do not share these attributes as they are not sample-specific and optimized for individual samples, and not jointly optimized with the training of the infected classifier. Therefore, they are less efficient in planting and activating the backdoor, thus resulting in a significantly lower ASR while having much higher network overhead.

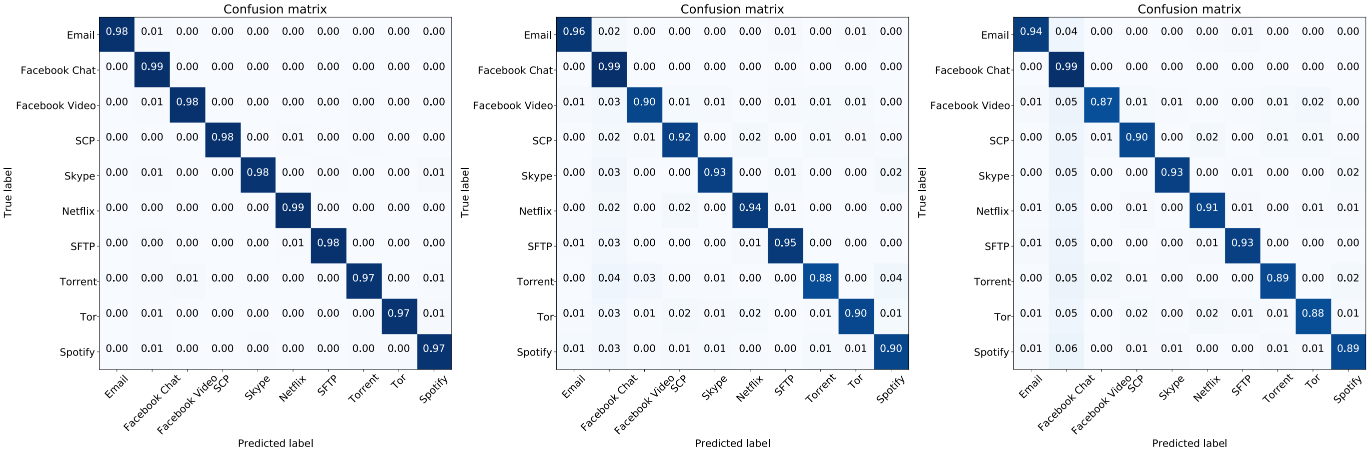
In addition, the TrojanFlow trigger values are evenly distributed over all the packets of the input traffic flow, making its poisonous power robust to slicing that is usually performed by victims when capturing the traffic. This is in sharp contrast to the 1D-BadNet trigger that only covers a small portion of the traffic flow, which leads to a much lower ASR as the added trigger can be accidentally removed during slicing.

Why does TrojanFlow deliver a high ACC? A natural follow-up question is: since poisoned samples of TrojanFlow are almost identical to their original ones, does this introduce confusion to the classifier during training since they are labeled differently? Why does the infected model still deliver a high accuracy? As aforementioned, TrojanFlow can manipulate the model's attention to irrelevant features. We conjecture that it also moves poisoned samples to a different location in the feature space as the model views the input differently. This helps the model to learn and draw clear boundaries during training, thus resulting in a high accuracy. To validate this, we plot an approximated feature space using T-SNE [39] for three models under poisoning of the TrojanFlow, 1D-SIG, and 1D-BadNet triggers. We train all three models for 10 epochs to observe the location of poisoned samples in the feature space. We consider a trigger to be effective if it can efficiently move and separate the poisoned samples from benign samples such that they can be easily learned by the model.

As shown in Fig. 6, TrojanFlow moves the poisoned samples (red triangles) to a well-separated region near the cluster of clean samples of the target class (dark-red dots), which helps the model to draw a clear boundary to predict them to the target class. (In the figure, the dots of other colors indicate clean samples of other classes.) In contrast, the poisoned samples of the other two attacks are widely scattered across the feature space, rendering them much more difficult to be separated from the benign samples, which accordingly leads to lower ACC.

The above observation indicates that TrojanFlow can effectively move and cluster the poisoned samples, making it easier for them to be learned by the infected model thus reducing the confusion. To further validate this, we continue to train the three infected models (with the target class to be Fackbook Chat) for 200 epochs so that they are converged, and plot their confusion matrix as shown in Fig. 7. As can be seen, the confusion matrix of the model poisoned by the TrojanFlow trigger delivers the highest accuracy and lowest false positive rate. In contrast, the other two attacks show higher confusions to the target class (i.e., higher value in the 2nd column of Fig. 7 (b) and (c)) since their triggers are less effective in separating poisoned samples from clean ones, which also leads to a lower accuracy as compared to the TrojanFlow attack.

Payload-based Traffic Classifier. The payload-based network classifiers have been studied in [5], [6], [10], where the byte-wise values of the encrypted payload of individual network packets are used to train a CNN model as the classifier. For example, a 1D-CNN of two convolutional layers and three dense layers is adopted in [5] to identify individual network packets using the values of their encrypted payload (truncated and zero-padded to a constant size). Therefore, the TrojanFlow attack can be easily applied to the payload-based classifiers since it can be considered as a simplified version of the flow-based classifier given that the models under the two scenarios have similar input format and architecture but the former does not need to consider preceding traffic flows. We thus slightly update the input format in the TraojanFlow attack framework (as illustrated in Fig. 3) to support the



(a) TrojanFlow infected model

(b) 1D–SIG infected model

(c) 1D–BadNet infected model

Fig. 7: Confusion matrix of TrojanFlow, 1D-SIG, and 1D-BadNet.

payload-based classifiers. More specifically, we adopt a 1D-CNN similar to the one used in the flow-based scenario and remove the block of the preceding traffic flow in the input of both the trigger generator and classifier. The model takes an input shape of 1×1500 and the convolutional kernel sizes are updated to 9 and 3 to achieve a better performance. Note that we truncate the payload at the maximum of 1500 bytes, and pad zeros for the byte vector less than 1500 bytes. We then evaluate and analyze its performance in the next section.

V. Experimental Results

In this section, we first introduce the experimental setting, and then evaluate the TrojanFlow attack on two different contexts: application recognition and VPN traffic identification, with two different neural network architectures, two backdoor attacks, and five detection schemes. We also analyze the performance of the attack under different backdoor mitigation schemes, such as fine-tuning, patching, and fine-pruning, to assess the robustness of TrojanFlow.

A. Experimental Settings

Dataset and Backdoor Planting. We conduct experiments based on a well-known benchmark dataset ISCXVPN2016 [31]. The training dataset construction, cross-validation, and target class selection are conducted similarly as in the preceding section (see the first paragraph of Section IV).

Attack and Defense Configuration. We compare the performance of the TrojanFLow attack with two other attacks: 1D-BadNet [20] and 1D-SIG [32]. We then test the stealthiness of all the attacks by five existing backdoor detection methods: Neural Cleanse (NC) [40], Gangsweap (GC) [41], TABOR [42], Artificial Brain Stimulation (ABS) [43], and Universal Litmus Patterns (ULP) [44].

DNN training. We train all models for 200 epochs using the Adam optimizer with an initial learning rate of 0.001, which is then divided by 10 after every 50 epochs. The batch size is 64 in all training rounds.

Evaluation Metrics. We evaluate the performance by two metrics: attack success rate (ASR), which is the ratio of malicious samples that are misclassified to the target class and the model's accuracy on clean samples (ACC).

B. Performance Comparison

Comparison of ACC and ASR. Table II shows ACC and ASR of flow-based and payload-based classifiers under different attacks on ISCXVPN2016 with two application scenarios: application classification and VPN traffic identification. We observe that the TrojanFlow attack delivers higher ACC and ASR than the other two attacks over different experimental settings. As revealed in Sec. IV, the reason is that the trigger generator of the TrojanFlow attack is jointly optimized with the infected classifier, making them extremely effective of generating poisonous triggers that can efficiently move the poisoned samples to a well-separated location in the feature space to be easily learned by the classifier. This yields less confusion to the classifier during training along with higher ACC and ASR.

Stealthiness. We have shown the TrojanFlow attack is effective. We now show it is also stealthy. To this end, we apply five state-of-the-art backdoor detection schemes: NC [40], GS [41], TABOR [42], ABS [45], and ULP [44] to conduct a comprehensive examination of four neural backdoors: 1D-SIG, 1D-BadNet, TrojanFlow (1D-CNN), and TrojanFlow-Payload. As shown in Table III, TrojanFlow delivers better stealthiness over all other detection schemes. The reason is that all of them are essentially trying to reverse-engineer a common trigger [40], [41], [43] or a universal pattern [44] that can manipulate the model's prediction, which is not applicable to the TrojanFlow attack as it exploits the planted backdoor using dynamic and sample-specific triggers. In contrast, 1D-SIG and 1D-BadNet can be easily detected by most defenses since they adopt static triggers that are likely to be recovered.

Resistance to Regular Fine-tuning. Though the above results have shown TrojanFlow's effectiveness and stealthiness, there

TABLE II: Comparison of TrojanFlow (TF) with other Attacks. SIG: 1D-SIG; BN: 1D-BadNet; ASR: Attack Success Rate; ACC: Classification Accuracy.

		Flow-based						Payload-based		
Scenario	Metrics	CNN-TF	CNN-SIG	CNN-BN	GRU-TF	GRU-SIG	GRU-BN	CNN-TF	CNN-SIG	CNN-BN
Application traffic	ACC	**97.6%**	94.7%	96.6%	**98.3%**	91.8%	95.2%	**98.8%**	96.8%	96.4%
	ASR	**99.6%**	62.1%	12.2%	**99.8%**	55.5%	11.8%	**99.8%**	78.7%	82.6%
VPN traffic	ACC	**99.1%**	95.5%	97.0%	**99.4%**	95.0%	97.3%	**99.0%**	92.6%	94.9%
	ASR	**99.6%**	62.1%	12.2%	**99.8%**	55.5%	11.8%	**99.3%**	75.1%	80.1%

TABLE III: Resistance to different detection schemes, NC: Neural Clease; GS: Gangsweep; TB: TABOR; ABS: Artificial Brain Stimulation; ULP: Universal Litmus Patterns.

Attack	NC	GS	TB	ABS	ULP
1D-SIG	✓	✓	✓	×	✓
1D-BadNet	✓	✓	✓	✓	✓
TrojanFlow	×	×	×	×	×
TrojanFlow-Payload	×	×	×	×	×

are other factors that may affect it in practice. For example, studies show that network classifiers may need to be periodically fine-tuned to alleviate the performance degradation caused by data or concept drifting [46], which may weaken or sanitize the planted backdoor. To this end, we conduct experiments to compare TrojanFlow to 1D-SIG and 1D-BadNet in terms of the resistance to regular fine-tuning. More specifically, we train infected models separately and fine-tune them end-to-end (i.e., weights of the entire model) with 10% of the clean training data for 20 epochs using the SGD optimizer with a small learning rate of 0.0001. As shown in Table IV, the ASR of other attacks decreases significantly after fine-tuning while the TrojanFlow attack only drops its ASR by 5%. We conjecture it is due to the fact that the clean and poisoned samples of TrojanFlow are well-clustered and separated in the feature space (significantly better than the other two attacks as shown in Fig. 6), making the backdoor more robust to fine-tuning where the decision boundaries are slightly adjusted.

Resistance to Backdoor Patching. In addition to regular fine-tuning, another important factor that may affect the backdoor attack is backdoor patching by an alerted victim. More specifically, a more advanced victim may be aware of the attack and thus collects a number of poisoned samples to patch the backdoor, where the poisoned samples are correctly labelled to fine-tune the infected model. As shown in Table. IV, this approach does not work well for the TrojanFlow attack either. We think the reason is that the collected poisoned samples have limited sample-specific triggers that cannot be generalized to represent the entire trigger distribution [47], rendering them only effective for a small portion of TrojanFlow attacks.

Resistance to Neural Pruning. We also consider the scenarios where the victim cannot detect the backdoor but chooses to sanitize the model anyway by a more aggressive method named neuron pruning. For example, [48] shows that the backdoor usually leverages a set of neurons for trigger recognition and they cannot be activated by clean data. As a result, when performing pruning with clean data, those malicious neurons can be considered redundant and thus removed. To this end, we prune our TrojanFlow backdoor model using the latest backdoor neural pruning scheme named Fine-Pruning [48] with different pruning ratios. As shown in Table IV, TrojanFlow is robust against neural pruning as the ASR drops proportionally with ACC. The reason is that the clean and poisoned samples of the TrojanFlow attack has nearly identical patterns, making their features highly entangled to activate overlapped or the same set of neurons in the infected model. Those neurons are considered important to maintain the model accuracy and hence cannot be removed during pruning.

VI. Conclusion

In this paper we have reported TrojanFlow, a new effective and efficient neural backdoor attack to deep learning (DL)-based network traffic classifiers. In contrast to traditional neural backdoor attacks where a designated and sample-agnostic trigger is used to plant backdoor, TrojanFlow poisons a model using dynamic and sample-specific triggers that are optimized to efficiently hijack the model. It features a unique design to jointly optimize the trigger generator with the target classifier during training. The trigger generator can thus craft optimized triggers based on the input sample to efficiently manipulate the model's prediction. We have developed a well-engineered prototype using Pytorch to demonstrate TrojanFlow attacking multiple practical DL-based network traffic classifiers. We have further conducted thorough analysis to gain insights into the effectiveness of TrojanFlow, revealing the fundamentals of why it is effective and what it does to efficiently hijack the model. We have carried out extensive experiments on the well-known ISCXVPN2016 dataset with three widely adopted neural network architectures, and compared TrojanFlow with two other backdoor attacks under five state-of-the-art backdoor defenses. The experimental results have shown that TrojanFlow is highly stealthy, effective, efficient, as well as robust against existing neural backdoor mitigation schemes.

TABLE IV: Performance degradation (↓) under regular fine-tuning and pruning (Payload-based, application classification).

	Metric	TrojanFlow-Payload	1D-SIG	1D-BadNet
Finetune	ASR↓	**5.0%**	58.6%	66.9%
Patching	ASR↓	**34.6%**	76.2%	81.3%
Pruning 60%	ACC↓	5.1%	6.2%	5.6%
	ASR↓	**8.3%**	70.0%	72.2%
Pruning 85%	ACC↓	36.7%	36.4%	37.0%
	ASR↓	**42.0%**	75.1%	76.6%

References

[1] A. W. Moore and D. Zuev, "Internet traffic classification using bayesian analysis techniques," in *Proceedings of the ACM SIGMETRICS*, pp. 50–60, 2005.

[2] L. Bernaille, R. Teixeira, and K. Salamatian, "Early application identification," in *Proceedings of the ACM CoNEXT Conference*, pp. 1–12, 2006.

[3] J. Erman, A. Mahanti, M. Arlitt, and C. Williamson, "Identifying and discriminating between web and peer-to-peer traffic in the network core," in *Proceedings of the WWW*, pp. 883–892, 2007.

[4] M. Lopez-Martin, B. Carro, A. Sanchez-Esguevillas, and J. Lloret, "Network traffic classifier with convolutional and recurrent neural networks for internet of things," *IEEE Access*, vol. 5, pp. 18042–18050, 2017.

[5] M. Lotfollahi, M. J. Siavoshani, R. S. H. Zade, and M. Saberian, "Deep packet: A novel approach for encrypted traffic classification using deep learning," *Soft Computing*, vol. 24, no. 3, pp. 1999–2012, 2020.

[6] W. Wang, M. Zhu, J. Wang, X. Zeng, and Z. Yang, "End-to-end encrypted traffic classification with one-dimensional convolution neural networks," in *Proceedings of the IEEE ISI*, pp. 43–48, 2017.

[7] C. Liu, L. He, G. Xiong, Z. Cao, and Z. Li, "Fs-net: A flow sequence network for encrypted traffic classification," in *Proceedings of the IEEE INFOCOM*, pp. 1171–1179, 2019.

[8] Y. Zeng, H. Gu, W. Wei, and Y. Guo, "$deep-full-range$: A deep learning based network encrypted traffic classification and intrusion detection framework," *IEEE Access*, vol. 7, pp. 45182–45190, 2019.

[9] W. Zheng, C. Gou, L. Yan, and S. Mo, "Learning to classify: A flow-based relation network for encrypted traffic classification," in *Proceedings of the WWW*, pp. 13–22, 2020.

[10] J. Zhang, F. Li, F. Ye, and H. Wu, "Autonomous unknown-application filtering and labeling for dl-based traffic classifier update," in *Proceedings of the IEEE INFOCOM*, pp. 397–405, 2020.

[11] V. F. Taylor, R. Spolaor, M. Conti, and I. Martinovic, "Robust Smartphone App Identification via Encrypted Network Traffic Analysis," *IEEE TIFS*, vol. 13, no. 1, pp. 63–78, 2018.

[12] M. Shen, M. Wei, L. Zhu, and M. Wang, "Classification of encrypted traffic with second-order markov chains and application attribute bigrams," *IEEE TIFS*, vol. 12, no. 8, pp. 1830–1843, 2017.

[13] B. Anderson and D. McGrew, "Machine learning for encrypted malware traffic classification: accounting for noisy labels and non-stationarity," in *Proceedings of the ACM SIGKDD*, pp. 1723–1732, 2017.

[14] L. Dixon, T. Ristenpart, and T. Shrimpton, "Network traffic obfuscation and automated internet censorship," *IEEE Security & Privacy*, vol. 14, no. 6, pp. 43–53, 2016.

[15] G. Verma, E. Ciftcioglu, R. Sheatsley, K. Chan, and L. Scott, "Network traffic obfuscation: An adversarial machine learning approach," in *Proceedings of the IEEE MILCOM*, pp. 1–6, 2018.

[16] S. Lawrence, C. L. Giles, A. C. Tsoi, and A. D. Back, "Face recognition: A convolutional neural-network approach," *IEEE transactions on neural networks*, vol. 8, no. 1, pp. 98–113, 1997.

[17] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, 1997.

[18] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial examples in the physical world," *arXiv preprint arXiv:1607.02533*, 2016.

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

[20] T. Gu, B. Dolan-Gavitt, and S. Garg, "Badnets: Identifying vulnerabilities in the machine learning model supply chain," *arXiv preprint arXiv:1708.06733*, 2017.

[21] Y. Liu, S. Ma, Y. Aafer, W.-C. Lee, J. Zhai, W. Wang, and X. Zhang, "Trojaning attack on neural networks," in *Proceedings of the NDSS*, 2018.

[22] X. Chen, C. Liu, B. Li, K. Lu, and D. Song, "Targeted backdoor attacks on deep learning systems using data poisoning," *arXiv preprint arXiv:1712.05526*, 2017.

[23] A. Saha, A. Subramanya, and H. Pirsiavash, "Hidden trigger backdoor attacks," *arXiv preprint arXiv:1910.00033*, 2019.

[24] Y. Liu, X. Ma, J. Bailey, and F. Lu, "Reflection backdoor: A natural backdoor attack on deep neural networks," *Proceedings of the ECCV*, 2020.

[25] R. Ning, J. Li, C. Xin, and H. Wu, "Invisible poison: A blackbox clean label backdoor attack to deep neural networks," in *Proceedings of the IEEE INFOCOM*, pp. 1–10, 2021.

[26] J. Cohen, E. Rosenfeld, and Z. Kolter, "Certified adversarial robustness via randomized smoothing," in *Proceeding of the ICML*, pp. 1310–1320, 2019.

[27] N. Wang, Y. Chen, Y. Hu, W. Lou, and Y. T. Hou, "Manda: On adversarial example detection for network intrusion detection system," in *Proceedings of the IEEE INFOCOM*, pp. 1–10, 2021.

[28] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand, and V. Lempitsky, "Domain-adversarial training of neural networks," *The journal of machine learning research*, vol. 17, no. 1, pp. 2096–2030, 2016.

[29] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010. http://yann.lecun.com/exdb/mnist/.

[30] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, pp. 8024–8035, 2019.

[31] G. Draper-Gil, A. H. Lashkari, M. S. I. Mamun, and A. A. Ghorbani, "Characterization of encrypted and vpn traffic using time-related," in *Proceedings of the ICISSP*, pp. 407–414, 2016.

[32] M. Barni, K. Kallas, and B. Tondi, "A new backdoor attack in cnns by training set corruption without label poisoning," in *Proceedings of the IEEE ICIP*, pp. 101–105, 2019.

[33] M. Ribeiro, K. Grolinger, and M. A. Capretz, "Mlaas: Machine learning as a service," in *Proceedings of the IEEE ICMLA*, pp. 896–902, 2015.

[34] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," in *Proceedings of the IEEE CVPR*, pp. 1125–1134, 2017.

[35] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE CVPR*, pp. 770–778, 2016.

[36] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[37] E. Wenger, J. Passananti, A. N. Bhagoji, Y. Yao, H. Zheng, and B. Y. Zhao, "Backdoor attacks against deep learning systems in the physical world," in *Proceedings of the IEEE CVPR*, pp. 6206–6215, June 2021.

[38] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-cam: Visual explanations from deep networks via gradient-based localization," in *Proceedings of the IEEE ICCV*, pp. 618–626, 2017.

[39] L. Van der Maaten and G. Hinton, "Visualizing data using t-sne.," *Journal of machine learning research*, vol. 9, no. 11, 2008.

[40] B. Wang, Y. Yao, S. Shan, H. Li, B. Viswanath, H. Zheng, and B. Y. Zhao, "Neural cleanse: Identifying and mitigating backdoor attacks in neural networks," in *Proceedings of IEEE SP*, pp. 707–723, 2019.

[41] L. Zhu, R. Ning, C. Wang, C. Xin, and H. Wu, "GangSweep: Sweep out Neural Backdoors by GAN," in *Proceedings of ACM MM*, 2020.

[42] W. Guo, L. Wang, X. Xing, M. Du, and D. Song, "Tabor: A highly accurate approach to inspecting and restoring trojan backdoors in ai systems," *arXiv preprint arXiv:1908.01763*, 2019.

[43] Y. Liu, W.-C. Lee, G. Tao, S. Ma, Y. Aafer, and X. Zhang, "Abs: Scanning neural networks for back-doors by artificial brain stimulation," in *Proceedings of the ACM CCS*, pp. 1265–1282, 2019.

[44] S. Kolouri, A. Saha, H. Pirsiavash, and H. Hoffmann, "Universal litmus patterns: Revealing backdoor attacks in cnns," in *Proceedings of IEEE CVPR*, pp. 301–310, 2020.

[45] Y. Gao, C. Xu, D. Wang, S. Chen, D. C. Ranasinghe, and S. Nepal, "Strip: A defence against trojan attacks on deep neural networks," in *Proceedings of ACSAC*, pp. 113–125, 2019.

[46] S. K. Mukkavilli and S. Shetty, "Mining concept drifting network traffic in cloud computing environments," in *Proceedings of the CCGRID)*, pp. 721–722, 2012.

[47] X. Qiao, Y. Yang, and H. Li, "Defending neural backdoors via generative distribution modeling," in *Proceedings of the NeurIPS*, 2019.

[48] K. Liu, B. Dolan-Gavitt, and S. Garg, "Fine-pruning: Defending against backdooring attacks on deep neural networks," in *Proceedings of the RAID*, pp. 273–294, Springer, 2018.