# Learning to Evade: Realistic Adversarial Network Packet Generation using Deep Reinforcement Learning

**Soumyadeep Hore**[1], **Jalal Ghadermazi**[1], **Diwas Paudel**[1], **Ankit Shah**[1*], **Tapas K. Das**[1], **Nathaniel D. Bastian** [2]

[1]University of South Florida, [2]United States Military Academy
{soumyadeep, jghadermazi, diwaspaudel, ankitshah, das}@usf.edu, nathaniel.bastian@westpoint.edu

## Abstract

Recent advancements in artificial intelligence (AI) and machine learning (ML) algorithms, coupled with the availability of faster computing infrastructure, have enhanced the security posture of cybersecurity operations centers (defenders) through the development of ML-aided network intrusion detection systems (NIDS). Concurrently, the abilities of adversaries to evade security have also increased with the support of AI/ML models. Therefore, defenders need to proactively prepare for evasion attacks that exploit the detection mechanisms of NIDS. Recent studies have found that the perturbation of flow-based and packet-based features can deceive ML models, but these approaches have limitations. Perturbations made to the flow-based features are difficult to reverse-engineer, while samples generated with perturbations to the packet-based features are not playable.

Our methodological framework, Deep PackGen, employs deep reinforcement learning to generate adversarial packets and aims to overcome the limitations of approaches in the literature. By taking raw malicious network packets as inputs and systematically making perturbations on them, Deep PackGen camouflages them as benign packets while still maintaining their functionality. In our experiments, using publicly available data, Deep PackGen achieved an average adversarial success rate of 66.4% against various ML models and across different attack types. Our investigation also revealed that more than 45% of the successful adversarial samples were out-of-distribution packets that evaded the decision boundaries of the classifiers. The knowledge gained from our study on the adversary's ability to make specific evasive perturbations to different types of malicious packets can help defenders enhance the robustness of their NIDS against evolving adversarial attacks.

## Introduction

A network intrusion detection system (NIDS) is a primary tool for cybersecurity operations centers (CSOCs) to detect cyber-attacks on computer networks. With the availability of high-performance computing resources and advancements in artificial intelligence (AI) and machine learning (ML) algorithms, intrusion detection mechanisms have greatly improved, serving the security needs of organizations. However, adversaries are also continuously advancing their toolchains by using AI/ML-enabled methodologies to camouflage their attacks that can evade these ML-based NIDS. Hence, the CSOCs must improve their security posture by proactively preparing for evasion attacks and making their NIDS robust against evolving adversaries.

Evasion attacks on NIDS are mainly conducted by perturbing network flow-based features to deceive ML models. Table 1 shows a summary of recent studies that focused on adversarial sample generation to evade NIDS. However, flow-based attacks are impractical as reverse engineering these perturbations from the flow level into constructing the actual packets is very complex and difficult (Rosenberg et al. 2021a). In addition, hidden correlations among different flow-based features further exacerbate the computational difficulty of replaying perturbations in a real network communication (Han et al. 2021). More importantly, perturbations must be made such that the communication's functionality is maintained. Hence, crafting adversarial attacks at the packet level is necessary to improve the practicality of implementing evasion attacks.

A few studies in recent literature have focused on using packet-based data to construct evasion attacks (see Table 1). These studies have utilized publicly available data sets to obtain the samples for obfuscation and relied on making random perturbations using trial-and-error and other approximation techniques. The generated adversarial samples were then tested against linear, tree-based, and nonlinear ML models for evasion. The limitations of these studies are as follows. The perturbations made to the samples were mainly focused on the time-based features, which a classifier can be made immune to by training it with raw packet information. Some also generate adversarial samples using packet or payload injection and packet damage. However, there exists a correlation among the packet-level features, directly impacting the feature set of the classifier, which is not considered in any of these studies. This phenomenon is also known as the side effect of packet mutation (Pierazzi et al. 2020). Another limitation of existing packet-based approaches is that they perturb both forward and backward packets (i.e., communication from the host to the destination and then the destination back to the host). Clearly, an adversary can only control the forward packets, those originating from the host

[*]Corresponding author

and going to the destination (server).

Our proposed methodological framework addresses the above limitations in the following ways. Our methodology uses a learning-based approach, in which an AI agent is trained to make (near-)optimal perturbations to any given malicious packet. The agent learns to make these perturbations in a sequential manner using a deep reinforcement learning (DRL) approach. We identify the forward packets in network communication and only modify them to produce adversarial samples. We evaluate our adversarial samples against classifiers trained using packet-level data. We aim to make minimal and valid perturbations to the original packets that preserve the functionality of the communication. Examples of such perturbations include modifications to the valid portions in the internet protocol (IP) header, transmission control protocol (TCP) header, TCP options, and segment data. Furthermore, we only consider perturbing those features that can be obtained from the raw packet capture (PCAP) files without any preprocessing. This makes it practical to replicate the attack using perturbed packets. We consider the side effects of packet mutation in this study. For example, any change to the IP or TCP header affects the IP and TCP checksum, respectively. A detailed description of the perturbations and their side effects is provided in the numerical experiments section. We also evaluate whether the learning attained from one environment is transferable to another. We do this to gauge the effectiveness of our methodology in real-world settings where adversaries may not have any knowledge of the ML models and the data used to build the NIDS. We demonstrate the playability of the adversarial packet in a flow using the Wireshark application in the supplementary material. In summary, our paper addresses the literature gap for constructing adversarial samples by developing a learning-based methodology with the following characteristics: only the forward packets are perturbed; valid perturbations are considered in order to maintain the functionality of the packets; side effects of perturbations are taken into account; effectiveness of the adversarial samples is tested against unseen classifiers; and demonstrated transferability of the framework to another network environment.

There are several contributions to this research study. The primary contribution is the development of a DRL-enabled methodology capable of generating adversarial network packets for evasion attacks on ML-based NIDS. Our methodological framework, Deep PackGen, takes raw network packets as inputs and generates adversarial samples camouflaged as benign packets. The DRL agent in this framework learns the (near-)optimal policy of perturbations that can be applied to a given malicious network packet, constrained by maintaining its functionality while evading the classifier. To the best of our knowledge, this is the first research study that poses the constrained network packet perturbation problem as a sequential decision-making problem and solves it using a DRL approach. Another novel aspect of this research is creating a packet-based approach to developing classification models for ML-based NIDS. The unidirectional (forward) packets from raw PCAP files are extracted, preprocessed, feature-engineered, and normalized for machine computation. The transformed network packets are then used to train the classifiers. Other contributions highly relevant to the cybersecurity research community include the insights obtained from the experiments and their analyses. Our investigation reveals that our methodology can generate out-of-distribution (OOD) packets that can also evade the decision boundaries of more complex nonlinear classifiers. Furthermore, we also explain why packets of certain attack types can be easily manipulated compared to others. The knowledge gained from this study on the adversary's ability to make specific perturbations to different types of malicious packets can be used by the CSOCs to defend against the evolving adversarial attacks.

## ADVERSARIAL NETWORK PACKET GENERATION FRAMEWORK: DEEP PACKGEN

The objective of our study is to develop a framework for generating adversarial network packets that can bypass ML-based NIDS while maintaining functionality for communication. Our proposed framework, named Deep PackGen, illustrated in Figure 1, comprises of three main components: data set creation, packet classification model development, and adversarial network packet generation. We begin by describing the process of creating and labeling network traffic data, followed by training and evaluating packet classification models. Finally, we present a DRL model trained to generate adversarial network packets by interacting with several packet classification models. We explain the stepwise dataset creation process in the supplementary materials.

### Data Set Creation

While much research has been conducted on developing different ML models for network traffic classification, most of it relies on the NIDS (such as Zeek, Snort, and Security Onion) or the NetFlow tools (such as Wireshark and CICFlowmeter) for compiling network packet information to obtain features for training the models. These methods have several limitations as follows: (i) NIDS and NetFlow tools generate features based on predefined rules or signatures, which makes it difficult to reproduce and compare results across different studies; (ii) these approaches often do not incorporate raw payload information, which can make it hard to detect attacks that are embedded in packet payloads; (iii) flow-based features are extracted by analyzing network traffic over a period of time, which makes it difficult to detect anomalies in real-time; and (iv) rule-based and signature-based feature extraction approaches can fail when encountering novel attacks without signatures. To address these limitations, recent research studies have focused on using raw packet data to train ML-based NIDS (Lotfollahi et al. 2020; De Lucia et al. 2021; Bierbrauer et al. 2023; Cheng et al. 2021). These studies use bidirectional data to train their ML models. However, an adversary can only control the network packets being sent from one direction (i.e., from the source). Hence, in this study, we create a data set comprising raw packet data with a unidirectional flow originating from the source. The packet data from the unidirec-

Table 1: Summary of recent literature on adversarial sample generation

| Author | Year | Data Set | Feature | Attacker Knowledge | Algorithm |
|---|---|---|---|---|---|
| Rigaki et al. (Rigaki 2017) | 2017 | NSL KDD | Flow-based | White-box | Fast Gradient Sign Method (FGSM), Jacobian-based Saliency Map Attack (JSMA) |
| Wang et al. (Wang 2018) | 2018 | NSL KDD | Flow-based | White-box | FGSM, JSMA, Deepfool, Carlini Wagner (CW) |
| Zhang et al. (Zhang, Costa-Pérez, and Patras 2020) | 2020 | CICIDS-2018 | Flow-based | White-box | Boundary Attack, Pointwise Attack, Hopskipjump Attack |
| Apruzzese et al. (Apruzzese et al. 2020) | 2020 | CTU, BOTNET | Flow-based | Black-box | Deep Reinforcement Learning |
| Alhajjar et al. (Alhajjar, Maxwell, and Bastian 2021) | 2020 | NSL-KDD, USNW-NB15 | Flow-based | Gray-box | Generative Adversarial Network (GAN), Genetic Algorithm (GA), Particle Swarm Optimization |
| Schneider et al. (Schneider, Aspinall, and Bastian 2021) | 2021 | NSL-KDD | Flow-based | White-box | Projected Gradient Descent, GA, Particle Swarm Optimization, GAN |
| Chernikova et al. (Chernikova and Oprea 2022) | 2022 | CTU 13 | Flow-based | Black-box | Projected Gradient Descent, CW |
| Zhang et al. (Zhang et al. 2022) | 2022 | NSL-KDD, UNSW-NB15 | Flow-based | Gray-box | GAN |
| Sheatslet et al. (Sheatsley et al. 2022) | 2022 | NSL-KDD, UNSW-NB15 | Flow-based | White-box | Adaptive JSMA, Histogram Sketch Generation |
| Homoliak et al. (Homoliak et al. 2018) | 2018 | ASNM-NBPO | Packet-based | Gray-box | Tools like NetEM, Metasploit |
| Hashemi et al. (Hashemi, Cusack, and Keller 2019) | 2019 | CICIDS-2018 | Packet-based | White-box | Trial and Error |
| Kuppa et al. (Kuppa et al. 2019) | 2019 | CICIDS-2018 | Packet-based | Gray-box | Manifold Approximation |
| Han et al. (Han et al. 2021) | 2021 | Kitsune, CICIDS-2017 | Packet-based | Gray-box | GAN |
| Sharon et al. (Sharon et al. 2022) | 2021 | Kitsune, CICIDS-2017 | Packet-based | Black-box | Long Short-Term Memory-based |

tional flows is used to train the ML models and to generate adversarial samples (network packets).

## Packet Classification Model Development

An adversary may not have complete knowledge of the defender's model. Hence, an adversary will need a substitute for the defender's ML-based NIDS to generate and evaluate the adversarial samples. We propose an ensemble model as a surrogate for the defender's model for training the adversarial agent. An ensemble model consists of multiple estimators (ML models), making the classifier robust in identifying malicious packets. The data set created using the first component of this framework is split into training and testing data sets. Various linear, tree-based, and non-linear ML models are then developed using the training data set and they are evaluated using the testing data set. The selection process of the estimators for this ensemble is described as follows. Given a large set of estimators $E$, in which each estimator is represented as $F_m(.)$, where $m \in E$, then the best set of estimators, $M$, is selected based on their performance metric values. $F^*_m(.)$ represents the estimator with optimal parameters for which its loss function value, $L(.)$, is minimum, i.e.,

$$\theta^* = \text{argmin}_\theta L(\theta, x, y) \tag{1}$$

where, $\theta$ represents the model parameters, $x \in X$ is the training data, $y \in Y$ is the target value, and $L(.)$ measures how far the predictions are from the target value. Finally, the top $|M|$ number of classifiers are selected in the ensemble, representing the defender's ML-based NIDS, shown as follows

$$F^*_m(.) \ \forall \ m \in M \tag{2}$$

## Adversarial Sample Generation

**Problem Definition** Our aim is to develop a methodology to generate malicious network packets that can fool the defender's ML-based NIDS. To achieve this, an original malicious packet is perturbed to camouflage it as benign traffic. Unlike the problem of applying unconstrained perturbations to an image to fool a computer vision-based classifier (Wang, Cho, and Yoon 2020), in this problem, the perturbations are constrained by the requirement to maintain the packet's maliciousness and functionality.

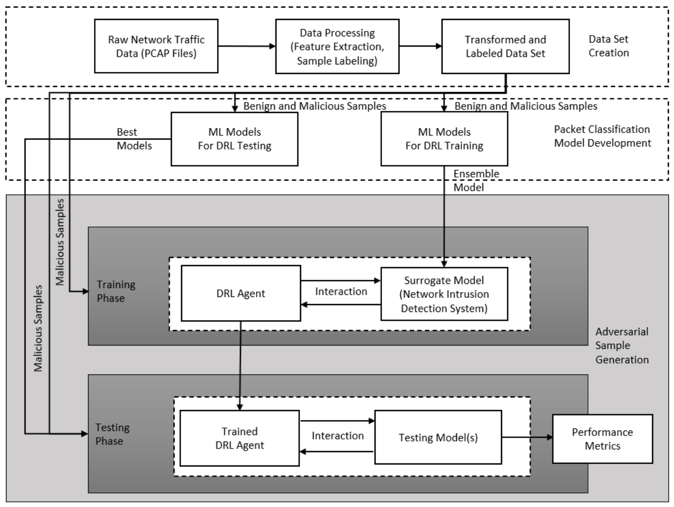
An original malicious packet, $x_{original}$, is modified by applying perturbation(s), $\delta$, using a perturbation function, $P(.)$. These perturbations must belong to a set of all valid perturbations, $\Delta$, that do not impede the capability of the packet. A perturbed sample, $x_p$, can be defined as

$$x_p = P(x_{original}, \delta) \tag{3}$$

$$\delta \in \Delta \tag{4}$$

Note that many perturbed samples can be obtained by applying different $\delta$ from this set of valid perturbations, resulting in a large set of perturbed samples, $X_p$. However, a successful adversarial sample, $x_p^{benign}$, is the malicious and functional network packet in $X_p$ that is able to bypass the defender's model by getting misclassified as benign. This can be formally defined as

$$x_p^{benign} = \text{argmax}_{x_p \in X_p} L(\theta^*, x_p, y) \tag{5}$$



Raw Network Traffic Data (PCAP Files)
Data Processing (Feature Extraction, Sample Labeling)
Transformed and Labeled Data Set
Data Set Creation
Benign and Malicious Samples
Benign and Malicious Samples
Best Models
ML Models For DRL Testing
ML Models For DRL Training
Packet Classification Model Development
Ensemble Model
Malicious Samples
Malicious Samples
Training Phase
DRL Agent
Interaction
Surrogate Model (Network Intrusion Detection System)
Adversarial Sample Generation
Testing Phase
Trained DRL Agent
Interaction
Testing Model(s)
Performance Metrics

Figure 1: Deep PackGen framework for adversarial network packet generation

**Problem Formulation** Generating an adversarial sample by making perturbations to a network packet can be posed as a sequential decision-making problem. An adversary starts with an original malicious network packet and makes sequential perturbations, as indicated in Equation 3. At each iteration, the packet is modified, and this perturbed sample is passed through the packet classification model to check if it successfully evades its classification decision boundary. The iterative process continues until either a successful adversarial sample is attained (satisfying Equation 5) or the maximum number of iterations is reached. The objective is to learn the (near-)optimal set of perturbations, given an original malicious network packet, to generate an adversarial sample. This sequential decision-making problem can be formulated as a Markov decision process (MDP). The key elements of the MDP formulation are as follows.

- **State,** $s_t$, is a representation of the information available at time $t$. The state space consists of the normalized byte values of the network packet obtained from the data set created in the first component of this framework and the classification label (0 for benign and 1 for malicious) given by the defender's model. Each packet contains $N$ number of features, which makes the state space $N + 1$ dimensional.
- **Action,** $a_t$, represents the perturbation(s), $\delta \in \Delta$, applied to the network packet at time $t$. The number of action choices is limited to $|\Delta|$ and the choices are discrete.
- **Reward,** $r_t$, is the measure of effectiveness of taking action $a_t$ in state $s_t$. The reward signal helps the adversary in quantifying the effect of the action taken in a particular state. We engineer a novel reward function to guide the adversary towards learning an optimal policy of making perturbations, given the original malicious network packet. The reward function is defined as follows:

$$r_t(s_t, a_t) = \begin{cases} r^- & \text{if } y_m \neq \text{benign } \forall m \in M \\ k * r^+ & \text{otherwise} \end{cases} \tag{6}$$

where, $k$ is the number of classifiers in the ensemble that were successfully evaded by the perturbed network sample. This function generates both positive and negative rewards. A positive reward is obtained when the perturbed sample evades one or more classifiers in the ensemble model. The reward value is directly proportional to the number of classifiers it is able to fool by getting misclassified as a benign sample. A small negative reward ($r^-$) is incurred each time the perturbed sample fails to evade any of the classifiers in the ensemble model.

**DRL-based Solution Approach** The network packet perturbation problem has a large state and action space. To overcome the issue of calculating and storing the action-value (Q value) for all state-action pairs using a conventional RL approach, we use a deep neural network architecture to estimate these values. An adversary, in the form of a DRL agent, is trained using the malicious samples from the data set created in the first component of the framework. It is to be noted that the DRL agent has no visibility of the ML model's architecture, parameter values, or loss function during the training and testing phases. Figure 1 shows the training and testing phases of the DRL agent, which are explained next.

**DRL Training Phase** Figure 2 shows the training phase of the DRL agent, which comprises interactions between the DRL agent and the training environment. The training environment for the DRL agent is designed with the surrogate for the real-world ML-based NIDS. The environment contains the transformed and labeled network packet data of various attack types, and the pre-trained classifiers for the surrogate ensemble model created in the first and second components of this framework, respectively. For each attack type, the DRL agent obtains a randomly picked data sample and prescribes the perturbation actions (details are explained in the next paragraph). The environment allows for the implementation of these actions, resulting in a one-step transition of the system state, generating a perturbed sample. Rewards are calculated based on the perturbed packet's ability to evade an ensemble of classifiers. This process continues until the stopping condition is reached, which is either the adversarial sample is successful in being misclassified as benign traffic or the maximum number of time-steps is reached. The steps for simulating the environment are outlined in Algorithm 1.

A DRL agent interacts with the training environment and learns to generate adversarial packets by following a set of rules called a policy. The agent's decisions are based on a sequence of states, actions, and rewards, which are determined by the training environment. The agent is rewarded based on the ability of the generated packets to evade the surrogate model. We use DRL with double Q-Learning (DDQN) (Van Hasselt, Guez, and Silver 2016) to train the agent. DDQN is a single architecture deep Q-network that is suitable for problems with a discrete action space. The notable difference between DDQN and traditional deep Q-learning is that DDQN decouples the action selection and evaluation processes by using an additional network. This helps to reduce the overestimation error that occurs in traditional Q-learning. The target value calculation is as follows:

$$R_t = r_t + \gamma Q(s_{t+1,a}\, Q(s_{t+1}, a; \theta_t); \theta'_t) \tag{7}$$

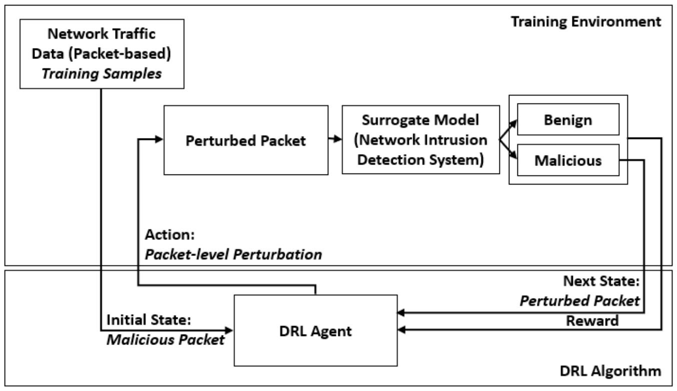

Training Environment
Network Traffic Data (Packet-based) Training Samples
Perturbed Packet
Surrogate Model (Network Intrusion Detection System)
Benign
Malicious
Action: Packet-level Perturbation
Next State: Perturbed Packet
Reward
Initial State: Malicious Packet
DRL Agent
DRL Algorithm

Figure 2: Training phase of DRL agent

The policy network weights ($\theta_t$) are used to select the action, while the target network weights ($\theta'_t$) are used to evaluate it. Algorithm 2 shows the steps in training the DRL algorithm. The algorithm interacts with the environment (Algorithm 1) to learn the near-optimal policy for perturbing different types of packets. The learning process continues until a pre-determined maximum number of episodes is reached. Multiple DRL agents are trained, one for each attack class in the data set.

**DRL Testing Phase** The trained DRL agents are tested against different ML models in the testing phase of the adversarial sample generation component of the framework, as depicted in Figure 2 of supplementary material. The neural network architecture of the agent uses the learned weights ($\theta$) obtained at the conclusion of the training phase (see Algorithm 2). The trained DRL agent operates without any reward signal during this phase and its actions are based on its learned policy. During the testing phase of the DRL agent adversarial samples are generated from testing samples that were not seen during the training phase. This allows for an assessment of the agent's performance on various classifiers, including those that were not used in the DRL training environment. The performance of each agent is measured using the adversarial success rate (ASR) metric, which is defined as follows.

$$ASR = \frac{FN_p - FN_{original}}{TP} \quad (8)$$

where $FN_p$ denotes the total number of samples that were misclassified after perturbation by the DRL agent, $FN_{original}$ is the total number of samples that were incorrectly classified before perturbation, and $TP$ is the total number of samples that the ML model correctly classified as malicious before perturbation. The ASR does not take into account packets that fool the classifier prior to perturbation (i.e., $FN_{original}$), thereby giving an accurate performance measurement for each agent.

## NUMERICAL EXPERIMENTS

In this section, we outline the numerical experiments performed to evaluate our Deep PackGen methodology. We first discuss the experimental data, followed by the creation of ML-based packet classification models. Finally, we delve into the hyperparameters employed during the training and testing of the DRL agent. The goal is to train multiple DRL agents, each specifically designed to generate packets for a unique attack type, by interacting with a surrogate model specialized in identifying that type of attack. To achieve this, we generated several data sets for training and testing of the DRL agents.

### Data Description

We conducted numerical experiments using raw PCAP files from two popular network intrusion detection data sets: CICIDS-2017 (Sharafaldin, Habibi Lashkari, and Ghorbani 2019) and CICIDS-2018 (Sharafaldin, Lashkari, and Ghorbani 2018). These data sets contain both benign and attack communications, and provide pragmatic representation of modern network traffic compared to older data sets like NSL-KDD and KDD-CUP (Hindy et al. 2020). Additionally, the availability of raw PCAP files for the CICIDS data sets reduces dependency on extracted flow level features (Rosenberg et al. 2021b). CICIDS-2017 consists of PCAP files for five consecutive days (Monday to Friday), each with different attack types and sizes. We processed these files to generate the data set, as explained in Section 3. Since the Heartbleed and Botnet attack types have too few instances to train ML models, we excluded them from our experiments. As discussed earlier, we only considered forward packets in our data set as an adversary is in control (generation and manipulation) of packets that originate from its source. We extracted payload bytes from each packet and represent each byte as a feature in this data set. We converted hexadecimal numbers to decimal numbers and normalized each feature value to a range of 0-1, where the minimum and maximum feature values were 0 and 255, respectively. In total, there were 1525 features.

We utilized the CICIDS-2017 data set for training and testing our framework as follows. We divided the data into different attack types, including samples from the benign category. For each attack type, we split the data into three parts: 60% of data for training the DRL agent and building the surrogate model for the training phase, 30% of data for building other packet classification models for testing the trained DRL agent, and 10% of data for generating adversarial samples and performing evaluation in the testing phase. In the rest of the paper, we will refer to them as *training*, *ML model testing 1*, and *DRL agent testing 1* data sets, respectively, for each attack type.

Further, to evaluate the performance of the trained agents on different network traffic data, we utilized the CICIDS-2018 data set.We extracted packets for the various attack types, including DoS, Web Attack, Infiltration, Port Scan, and DDoS from these files. We split the data for each attack type into two parts: 70% of data was allocated for building the ML models for the detection of the respective attack type in the testing phase and the remaining 30% of data was utilized to measure the adversarial success rate of the respective trained DRL agent. We refer to these two parts as *ML model testing 2* and *DRL agent testing 2* data sets, respectively. Note that the DRL agents were not trained with the CICIDS-2018 data samples.

## Packet Classification Model Creation

We developed different sets of ML models for both training and testing the DRL agents. Data sets were carefully prepared for developing the surrogate (training phase) and the testing (testing phase) models. ML models in the training phase were developed using the *training* data. In contrast, those used for testing the trained DRL agents were developed using either the *ML model testing 1* or *ML model testing 2* data sets. A DRL agent was trained to perturb packets for each attack type. To effectively train the agent to deceive the classifier's decision boundary, we used a surrogate model specializing in that respective attack type in the agent's training environment. For example, if the DRL agent was being trained on perturbing the packets of a *Port Scan* attack, then the surrogate model was trained with the forward packets extracted from the network flow data of the same attack.

In the training phase, we selected an ensemble of ML models to act as a surrogate model. We randomly sampled 80% of the *training* data to train various ML models, including linear, tree-based, and nonlinear classification models. The performances of all these models on the remaining 20% of the *training* data were comparable across each attack type, with the majority of them having a superior accuracy of around 99%. We selected one model from each of the three types of classifiers in the ensemble: logistic regression (LR), decision tree (DT), and multi-layer perceptron (MLP).

Similarly, ML models were developed for the testing phase. Two sets of models were trained: one using the *ML model testing 1* data set and another using the *ML model testing 2* data set. Note that both these data sets contain previously unseen samples by the DRL agents. In addition, the latter contains samples from a different network than that used to train the agents.

## Adversarial DRL Agent Training

The state space of the DRL agent consists of the 1525 features extracted from the network packet and its classification label. As discussed in the data set creation component of the framework, these features are the normalized values of the bytes pertaining to different TCP/IP header and segment information. Our focus in this study is to find the (near-)optimal set of perturbations that can be applied to a given malicious network packet to generate a successful adversarial sample, while maintaining the functionality of communication. To show the effectiveness of our methodology, we selected a set of valid perturbations ($\Delta$) based on domain knowledge, literature studies (Nasr, Bahramali, and Houmansadr 2021), (Sadeghzadeh, Shiravi, and Jalili 2021), (Yan et al. 2019), (Guo et al. 2021), (Apruzzese et al. 2020), (Huang et al. 2020), and our discussions with the subject matter experts (security personnel) at a collaborating CSOC. Below is a sample list of perturbations, among others, that were selected as a part of the agent's action space along with their descriptions and impacts.

- Modifying the fragmentation bytes from *do not fragment* to *do fragment*. This perturbation can be applied to packets where fragmentation is turned off.
- Modifying the fragmentation bytes from *do not fragment* to *more fragment*. This perturbation can be applied to packets where fragmentation is turned on or off.
- Increasing or decreasing (+/- 1) the TTL byte value. Any valid perturbation to this byte will result in a final TTL value between 1-255.
- Increasing or decreasing (+/- 1) the window size bytes. Any valid perturbation to these bytes will result in a final window size value between 1-65535.
- Adding, increasing, or decreasing the maximum segment size (MSS) value. This perturbation can only be applied to SYN and SYN-ACK packets.
- Adding, increasing, or decreasing the window scale value. This perturbation can only be applied to SYN and SYN-ACK packets.
- Adding segment information. For this perturbation, we selected the most commonly occurring TCP payload information from the benign traffic in the data set.

We tried various reward schemes with different values of positive and negative rewards in our reward function (see Equation 6). We obtained the best results, in terms of higher average reward value and a faster convergence with the reward term values as follows. We assigned a value of 200 to $r^+$ and -2 to $r^-$. If the perturbed sample successfully evaded all three classifiers in the ensemble model, then $r_t = 600$ was passed on to the DRL agent. If the sample successfully evaded only two (one) of the three classifiers, then the DRL agent received a reward of 400 (200). However, if the sample failed to evade any of the classifiers, then a negative reward of $r_t = -2$ was assigned to the agent at time $t$.

## RESULTS AND ANALYSIS

This section discusses the results of the conducted experiments and their analysis. First, we present the performances of the trained DRL agents against the various testing models. We then present an analysis of the agents' decision-making. Finally, we delve into a deeper statistical analysis of the successful adversarial samples.

### Performance Evaluation of the Trained DRL Agents

We evaluate the performance of the trained agents using the *DRL agent testing 1* (CICIDS-2017) data set. We quantify their performance by calculating the rate of adversarial samples that successfully bypass the classification boundary of each testing model by getting misclassified as benign. We use the ASR metric (see Equation 8) to report each agent's performance. Note that in calculating this performance metric value, the packets that fool the classifier prior to perturbation (i.e., $FN_{original}$) are disregarded (subtracted) to accurately measure the effectiveness of the agent's learned policy.

Table 2 presents the ASR values for the five trained DRL agents (one for each attack type) on five testing models trained using the *ML model testing 1* (CICIDS-2017) data

Table 2: Adversarial success rate (ASR) of DRL agents on CICIDS-2017 data

| Attack Type | Testing 1 Models | | | | |
|---|---|---|---|---|---|
| | DT | RF | MLP | DNN | SVM |
| DoS | 0.969 | 0.539 | 0.472 | 0.786 | 0.272 |
| DDoS | 0.965 | 0.965 | 0.990 | 0.679 | 0.742 |
| Web Attack | 0.995 | 0.656 | 0.654 | 0.330 | 0.322 |
| Port Scan | 0.995 | 0.985 | 0.979 | 0.314 | 0.982 |
| Infiltration | 0.998 | 0.324 | 0.209 | 0.158 | 0.323 |

set. The average ASR value obtained across all DRL agents and testing models was 0.664. The DT classifier was found to be the easiest to fool by all agents, with ASR values greater than 0.96 (as shown in the DT column of Table 2). The DRL agents also performed well against the RF classifier, a tree-based ensemble model, with an average ASR value of 0.694. In particular, the *DDoS* and *Port Scan* agents had similar success rates against both DT and RF classifiers. Some DRL agents, such as *Infiltration* and *Web Attack*, had lower success rates against more complex nonlinear models, such as DNN and SVM classifiers. However, low ASR in some experiments doesn't undermine the effectiveness of the framework since the ASR is recorded over multiple attack flows. In general, we observed that simpler models were easier for the DRL agents to evade the decision boundary through adversarial sample generation. The *DDoS* and *Port Scan* agents performed the best against all types of models, while the perturbed packets generated by the *Infiltration* agent had a lower success rate in fooling the nonlinear classifiers.

Next, we evaluate the transferability of the learned policies of the DRL agents to a different environment (testing 2). To accomplish this, we employ the CICIDS-2018 data set. The five testing models are trained using the *ML model testing 2* data samples. The DRL agents, which were trained using the CICIDS-2017 data, are subjected to malicious samples from the CICIDS-2018 data set (i.e., *DRL agent testing 2* data samples). Note that there are no Port Scan attack samples in this data set, so we present the evaluation of the other four DRL agents in this environment. Table 3 shows the transferability evaluation of these agents using the ASR metric. Overall, the agents successfully perturb malicious samples that evade the classification boundaries of the various testing models in this new environment, with an average ASR value of 0.398. We observed that the DRL agents were more successful in fooling the tree-based models than the nonlinear models. Notably, the *DDoS* agent performed the best, consistent with the findings of the testing 1 environment. Figure 3 depicts the average ASR values of the DRL agents in both testing 1 and testing 2 environments, highlighting their performance in generating successful adversarial samples.

## Performance Analysis of the Agents

We now assess the effectiveness of the DRL agents by examining two key aspects. First, we analyze why packets of certain attack types were easier to perturb than others. Second, we examine the (near-)optimal actions learned by the agents and their relevance to the decision boundary of the classifiers. To accomplish this, we implemented the following two steps. (i) We calculated the mean and standard deviation values of the first 500 normalized features from the network packets of each attack type in the CICIDS-2017 data set. These values were then compared with those obtained from the benign packets to determine similarity (or dissimilarity) in feature values. This information is visualized in Figure 4, which displays the plots for the five attack types. (ii) We used Shapley Additive exPlanations (SHAP) (Lundberg and Lee 2017) to identify important features that determine the classification boundary for accurately detecting each attack type in the testing models. We analyze the agent's performance and action choices in relation to these key features and the ASR values reported in Table 2.

Table 3: Transferability evaluation (ASR) of DRL agents on CICIDS-2018 data

| Attack Type | Testing 2 Models | | | | |
|---|---|---|---|---|---|
| | DT | RF | MLP | DNN | SVM |
| DoS | 0.506 | 0.455 | 0.355 | 0.225 | 0.104 |
| DDoS | 0.657 | 0.810 | 0.372 | 0.679 | 0.372 |
| Web Attack | 0.312 | 0.298 | 0.283 | 0.293 | 0.254 |
| Infiltration | 0.293 | 0.354 | 0.590 | 0.410 | 0.340 |

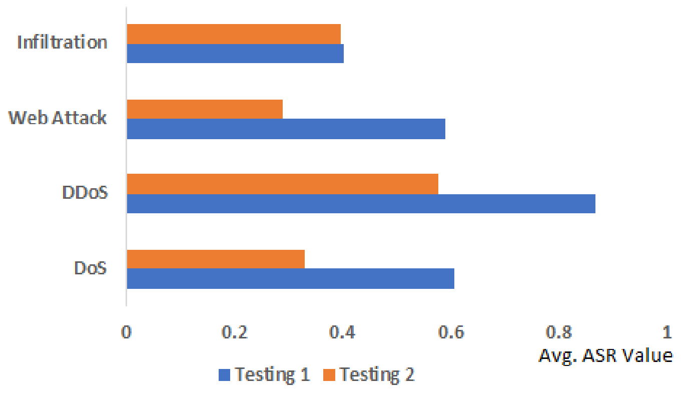


Figure 3: Average ASR values of DRL agents in different testing environments

From Figure 4 and Table 2, we can infer that attack types with packet feature values similar to those of the benign class are more susceptible to successful perturbation, allowing evasion of the classifiers. Specifically, plots (b) and (e) in Figure 4 show that the malicious packets from DDoS and Port Scan attacks, respectively, have similar feature values as the benign packets, resulting in higher ASR values for the DRL agents trained to perturb them, as seen in Table 2. Next, we look at the agent's actions and important features of the classifier, and analyze how that impacted the ASR.

The actions increasing the TTL values (directly affecting IP header byte numbers 9, 11, and 12) and adding payload (segment information), were amongst the top actions

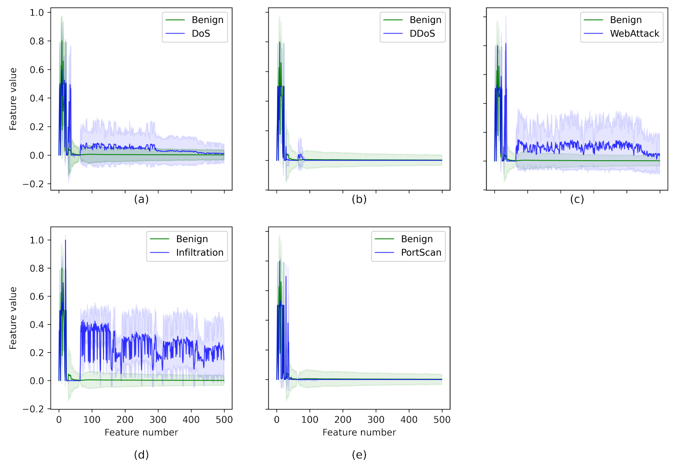


Figure 4: Mean and +/-1 standard deviation values for the first 500 features of benign and malicious packets: (a) DoS, (b) DDoS, (c) Web Attack, (d) Infiltration, and (e) Port Scan

performed by the *Port Scan* agent during testing. We used the SHAP values of the classifiers to determine any correlation between the agent's actions and the important feature(s) governing their decision boundaries. Notably, IP header byte number 9 has the most significant contribution towards deciding the decision boundary for both models, which was also learned by the *Port Scan* agent. We present the details related to SHAP values observed for different features in the supplementary materials.

The DRL agents' performance analysis reveals that some attack types were less successful than others, as shown in Table 2. The Infiltration attack type had the lowest success rate, likely due to its dissimilar feature values compared to the benign packets (see Figure 4 (d)). Note that the Infiltration packets have unique feature values in bytes 61-1525, representing segment information, making it challenging to perturb these packets by modifying these features. The SHAP analysis shows that the window size feature is the top contributor to the decision boundary for the RF classifier, which explains why perturbing window size value is amongst the top action choices for the *Infiltration* agent. These examples, amongst others, demonstrate that the DRL agents learned which key feature(s) to perturb to evade the classification boundary.

Notably, the early part of the TCP segment plays an important role in determining the MLP model's classification boundary. However, the early TCP segment information is not mutable for this attack type as that would compromise the functionality of the packet. Hence, the agent relied on adding payload (dead bytes) to these packets as an action choice and succeeded in some cases. The difference in the important features between classifiers also explains the *Infiltration* agent's relatively higher success rate against the RF classifier compared to the MLP classifier. A similar performance analysis was conducted for the other DRL agents across all the classifiers. We found a similar correlation between the agent's policy, the important features of the classifier, and the respective ASR.

## CONCLUSIONS AND FUTURE DIRECTIONS

In this paper we presented the development of a generalized methodology for creating adversarial network packets that can evade ML-based NIDS. The methodology is aimed at finding (near-)optimal perturbations that can be made to malicious network packets while evading detection and retaining functionality for communication. We posed this constrained packet perturbation problem as a sequential decision-making problem and solved it using a DRL approach. The DRL-enabled solution framework, Deep PackGen, consists of three main components, namely packet-based data set creation, ML-based packet classification model development, and DRL-based adversarial sample generation. Raw packet capture files from publicly available data were used to conduct the experiments. The framework generated curated data sets containing forward network packets, which were used to train and test five different types of DRL agents. Each agent was tailored to a specific attack type and evaluated on various classifiers. Results show that the Deep PackGen framework is successful in producing adversarial packets with an average ASR of 66.4% across all the classifiers in the network environment in which they were trained. The experimental results also show that the trained DRL agents produce an average ASR of 39.8% across various tree-based and nonlinear models in a different network environment.

Below, we present a summary of the insights obtained from this study that can guide future investigations.

1. The DRL agents have a higher success rate in evading tree-based packet classification models like DT and RF compared to nonlinear classifiers such as SVM and DNN.
2. The success rate of the DRL agents in generating adversarial samples is directly related to the key features that govern the decision boundary of the classifier and whether these features could be changed without disrupting the packet's communication function.
3. Attacks that have feature values similar to those of benign traffic, such as DDoS and Port Scan, are more vulnerable to successful perturbation by an adversarial agent.
4. The policies learned by the DRL-agents are transferable to new network environments.
5. The more complex the decision boundary of the classifier, the larger the magnitude of the perturbation required for evasion, resulting in out-of-distribution samples.

As regards future work, the Deep PackGen framework could be expanded to include more attack types and can be evaluated against different types of NIDS. Our methodology has shown encouraging results for generating OOD samples, which can be further investigated by the cybersecurity research community to model adversarial evolution and strengthen defenses against new types of attacks.

## Acknowledgement

This work was supported in part by the U.S. Military Academy (USMA) under Cooperative Agreement No. W911NF-22-2-0045, as well as the U.S. Army Combat Capabilities Development Command C5ISR Center under Support Agreement No. USMA21056. The views and conclusions expressed in this paper are those of the authors and do not reflect the official policy or position of the U.S. Military Academy, U.S. Army, U.S. Department of Defense, or U.S. Government.

## References

Alhajjar, E.; Maxwell, P.; and Bastian, N. 2021. Adversarial machine learning in network intrusion detection systems. *Expert Systems with Applications*, 186: 115782.

Apruzzese, G.; Andreolini, M.; Marchetti, M.; Venturi, A.; and Colajanni, M. 2020. Deep reinforcement adversarial learning against botnet evasion attacks. *IEEE Transactions on Network and Service Management*, 17(4): 1975–1987.

Bierbrauer, D. A.; De Lucia, M. J.; Reddy, K.; Maxwell, P.; and Bastian, N. D. 2023. Transfer learning for raw network traffic detection. *Expert Systems with Applications*, 211: 118641.

Cheng, Q.; Zhou, S.; Shen, Y.; Kong, D.; and Wu, C. 2021. Packet-level adversarial network traffic crafting using sequence generative adversarial networks. *arXiv preprint arXiv:2103.04794*.

Chernikova, A.; and Oprea, A. 2022. Fence: Feasible evasion attacks on neural networks in constrained environments. *ACM Transactions on Privacy and Security*, 25(4): 1–34.

De Lucia, M. J.; Maxwell, P. E.; Bastian, N. D.; Swami, A.; Jalaian, B.; and Leslie, N. 2021. Machine learning raw network traffic detection. In *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications III*, volume 11746, 185–194. SPIE.

Guo, S.; Zhao, J.; Li, X.; Duan, J.; Mu, D.; and Jing, X. 2021. A black-box attack method against machine-learning-based anomaly network flow detection models. *Security and Communication Networks*, 2021: 1–13.

Han, D.; Wang, Z.; Zhong, Y.; Chen, W.; Yang, J.; Lu, S.; Shi, X.; and Yin, X. 2021. Evaluating and improving adversarial robustness of machine learning-based network intrusion detectors. *IEEE Journal on Selected Areas in Communications*, 39(8): 2632–2647.

Hashemi, M. J.; Cusack, G.; and Keller, E. 2019. Towards evaluation of nidss in adversarial setting. In *Proceedings of the 3rd ACM CoNEXT Workshop on Big DAta, Machine Learning and Artificial Intelligence for Data Communication Networks*, 14–21.

Hindy, H.; Brosset, D.; Bayne, E.; Seeam, A. K.; Tachtatzis, C.; Atkinson, R.; and Bellekens, X. 2020. A taxonomy of network threats and the effect of current datasets on intrusion detection systems. *IEEE Access*, 8: 104650–104675.

Homoliak, I.; Teknos, M.; Ochoa, M.; Breitenbacher, D.; Hosseini, S.; and Hanacek, P. 2018. Improving network intrusion detection classifiers by non-payload-based exploit-independent obfuscations: An adversarial approach. *arXiv preprint arXiv:1805.02684*.

Huang, W.; Peng, X.; Shi, Z.; and Ma, Y. 2020. Adversarial attack against LSTM-based DDoS intrusion detection system. In *2020 IEEE 32nd International Conference on Tools with Artificial Intelligence (ICTAI)*, 686–693. IEEE.

Kuppa, A.; Grzonkowski, S.; Asghar, M. R.; and Le-Khac, N.-A. 2019. Black box attacks on deep anomaly detectors. In *Proceedings of the 14th international conference on availability, reliability and security*, 1–10.

Lotfollahi, M.; Jafari Siavoshani, M.; Shirali Hossein Zade, R.; and Saberian, M. 2020. Deep packet: A novel approach for encrypted traffic classification using deep learning. *Soft Computing*, 24(3): 1999–2012.

Lundberg, S. M.; and Lee, S.-I. 2017. A unified approach to interpreting model predictions. *Advances in neural information processing systems*, 30.

Nasr, M.; Bahramali, A.; and Houmansadr, A. 2021. Defeating DNN-Based Traffic Analysis Systems in Real-Time With Blind Adversarial Perturbations. In *USENIX Security Symposium*, 2705–2722.

Pierazzi, F.; Pendlebury, F.; Cortellazzi, J.; and Cavallaro, L. 2020. Intriguing properties of adversarial ml attacks in the problem space. In *2020 IEEE symposium on security and privacy (SP)*, 1332–1349. IEEE.

Rigaki, M. 2017. Adversarial deep learning against intrusion detection classifiers.

Rosenberg, I.; Shabtai, A.; Elovici, Y.; and Rokach, L. 2021a. Adversarial machine learning attacks and defense methods in the cyber security domain. *ACM Computing Surveys (CSUR)*, 54(5): 1–36.

Rosenberg, I.; Shabtai, A.; Elovici, Y.; and Rokach, L. 2021b. Adversarial machine learning attacks and defense methods in the cyber security domain. *ACM Computing Surveys (CSUR)*, 54(5): 1–36.

Sadeghzadeh, A. M.; Shiravi, S.; and Jalili, R. 2021. Adversarial network traffic: Towards evaluating the robustness of deep-learning-based network traffic classification. *IEEE Transactions on Network and Service Management*, 18(2): 1962–1976.

Schneider, M.; Aspinall, D.; and Bastian, N. D. 2021. Evaluating model robustness to adversarial samples in network intrusion detection. In *2021 IEEE International Conference on Big Data (Big Data)*, 3343–3352. IEEE.

Sharafaldin, I.; Habibi Lashkari, A.; and Ghorbani, A. A. 2019. A detailed analysis of the cicids2017 data set. In *Information Systems Security and Privacy: 4th International Conference, ICISSP 2018, Funchal-Madeira, Portugal, January 22-24, 2018, Revised Selected Papers 4*, 172–188. Springer.

Sharafaldin, I.; Lashkari, A. H.; and Ghorbani, A. A. 2018. Toward generating a new intrusion detection dataset and intrusion traffic characterization. *ICISSp*, 1: 108–116.

Sharon, Y.; Berend, D.; Liu, Y.; Shabtai, A.; and Elovici, Y. 2022. Tantra: Timing-based adversarial network traffic reshaping attack. *IEEE Transactions on Information Forensics and Security*, 17: 3225–3237.

Sheatsley, R.; Papernot, N.; Weisman, M. J.; Verma, G.; and McDaniel, P. 2022. Adversarial examples for network intrusion detection systems. *Journal of Computer Security*, (Preprint): 1–26.

Van Hasselt, H.; Guez, A.; and Silver, D. 2016. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30.

Wang, L.; Cho, W.; and Yoon, K.-J. 2020. Deceiving image-to-image translation networks for autonomous driving with adversarial perturbations. *IEEE Robotics and Automation Letters*, 5(2): 1421–1428.

Wang, Z. 2018. Deep learning-based intrusion detection with adversaries. *IEEE Access*, 6: 38367–38384.

Yan, Q.; Wang, M.; Huang, W.; Luo, X.; and Yu, F. R. 2019. Automatically synthesizing DoS attack traces using generative adversarial networks. *International journal of machine learning and cybernetics*, 10(12): 3387–3396.

Zhang, C.; Costa-Pérez, X.; and Patras, P. 2020. Tiki-taka: Attacking and defending deep learning-based intrusion detection systems. In *Proceedings of the 2020 ACM SIGSAC Conference on Cloud Computing Security Workshop*, 27–39.

Zhang, R.; Luo, S.; Pan, L.; Hao, J.; and Zhang, J. 2022. Generating adversarial examples via enhancing latent spatial features of benign traffic and preserving malicious functions. *Neurocomputing*, 490: 413–430.