Simulating Black-box Adversarial Threat Model Using Reinforcement Learning in Network Intrusion Detection Setting

Rafay Ali
Department of Electrical Engineering
Texas A&M University
College Station, TX
rhayat@tamu.edu

Syed Wali
Department of Electrical Engineering
Texas A&M University
College Station, TX
syedwali@tamu.edu

Yasir Ali Farrukh
Department of Electrical Engineering
Texas A&M University
College Station, TX
yasir.ali@tamu.edu

Mohamed Zeid
Department of Electrical Engineering
Texas A&M University
College Station, TX
mohamedzeid@tamu.edu

Abstract

The use of Machine learning (ML)-based Network Intrusion Detection Systems (NIDS) has become common practice. However, these ML models are vulnerable to adversarial attacks where subtle changes are made to the sample to mislead the models. Due to the complexity of traffic samples and the constraints to keep malicious functions, there has been very little research on adversarial ML using Reinforcement learning in the NIDS field. Adversarial attacks caused by these carefully crafted adversarial examples can make ML-based detectors useless and severely decrease their performance. In this paper, we propose a new Adversarial Reinforcement learning (ARL) environment for bypassing ML-based NIDS. We train a Deep Q-Network (DQN) based RL agent to modify network packets in a way that preserves their functionality and can evade detection by selected baseline ML detectors in a black-box scenario. This environment can effectively assess vulnerabilities of different ML-based NIDS. Our method has shown considerable evasion performance in three different black box scenarios, and the samples generated by our trained DQN agent can be used to induce adversarial robustness in the ML-based detectors.

1 Introduction

The growing reliance on vulnerable cyber networks has resulted in new cyber threats that pose a risk to all involved stakeholders. To ensure network security, NIDS have become essential. Over the past decade, ML based IDS has shown impressive performance in countering typical cyber threats [1]. However, the emergence of adversarial attacks in the cyber realm highlights the need to improve these IDS since conventional ML-based methods are susceptible to such attacks [2,3]. Every machine learning based NIDS has vulnerabilities that attackers can exploit by manipulating malicious data into packets which are disguised in order to evade detection. Adversaries can craft these malicious samples in two scenarios. The first is an internal threat scenario where the adversary has complete information about the NIDS model and its artifacts, enabling them to create malicious samples that are difficult to detect. However, these attacks are not common in practice as insider

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

threats can be easily controlled. In contrast, the second scenario is more realistic, where the adversary queries the NIDS model and uses its results to craft samples that can evade detection. This type of attack is known as a black box adversarial attack. Currently, there are various methods available in research literature that can create black box adversarial samples. However, using these specific algorithms to evaluate the vulnerability of machine learning models or NIDS limits their ability to generalize and withstand adversarial attacks. These algorithms are unable to explore all weak points or vulnerable regions of the machine learning classifier. To address this limitation, this research proposes a more comprehensive and generalized approach to developing black-box adversarial attacks using Deep Reinforcement Learning. The proposed approach can generate samples that are capable of evading different types of classifiers, including ensemble classifiers and neural network-based machine learning models. This research work is a novel effort in exploring the vulnerability of machine learning-based defensive models to adversarial attacks. The research work includes several significant contributions which are listed below:

- Development of ARL (Adversarial Reinforcement Learning) Environment: There is currently no environment available that can generate adversarial network traffic capable of evading different types of machine learning classifiers in black box settings. Therefore, we have done significant work in developing a reinforcement learning environment that includes developing a number of actions that an agent can take without affecting the functionality of a network packet. We have also proposed appropriate rewards that an agent can receive based on the degree of evasiveness the developed network packet succeeds in. Details of the developed ARL environment are provided in the Methodology section.
- Development of Packet Parser and Labeler: As our proposed ARL environment makes changes to different sections of the packet, including its payload section, headers, and checksum, we had to transform the publicly available dataset CICIDS Packet Capture Files into the required format of headers, TCP options, payload, and checksum. Then, we labeled those transformed data instances using the PCAP files and their provided flow level data.
- Development and Training of ARL Agent: In this work, we have utilized the DQN agent with replay buffer and targeted network to learn the adversarial policy that can generate the evasive sample with a smaller number of model queries. Our proposed model and selected artifacts result in query efficient ARL agent.
- Cross Model Black Box Evaluation Framework: In the evaluation stage, we have trained different models, including a random forest classifier, a deep neural network, and a Convolution Neural Network, on the actual network packets. We then evaluated their performance on the adversarial samples generated by the trained DQN agent to measure the evasion rate of our agent derived adversarial attacks.

2 Related Work

In this paper, we group the recent approaches in adversarial attacks into three scenarios based on the knowledge that the adversary has about the system to be attacked: Perfect knowledge, limited knowledge and zero knowledge.

Perfect Knowledge: In this case, the attacker has complete information about the detector and aims to minimize the sample misclassification function. Previous research into the susceptibility of neural networks to minor input perturbations showed how small perturbations can result in neural network making wrong predictions, with the adversarial examples being transferable [16]. To address this issue, researchers have explored minimizing perturbations using the assumption that the loss function of the clean sample is linear [6]. It has been suggested that limiting the disturbance size using the L2 norm can achieve the minimum perturbation necessary to fool neural networks [10].

Limited Knowledge: In this case the attacker only has limited knowledge about the detector, hence gradient-based approaches cannot be used. In such cases, alternative methods have been proposed to generate optimal adversarial examples. An example of which was developed by Giovanni Apruzzese, where a mixed signal linear program solver is used to build an optimal adversarial example to avoid flow and random forest-based detectors [2]. Another approach involves iteratively querying through the model and constructing an alternate model to simulate the decision boundary of the detector, thus enabling the generation of inputs that can be misclassified [11].

Zero Knowledge: In this case, the attacker has zero knowledge of the detector except for the binary decision of the detector. The only way to evade the detector is through trial and error and crafting adversarial examples which can be misclassified. One approach to this problem is the development of a GAN-based PE malware generator, as demonstrated by Weiwei Hu, which was able to successfully bypass a black box detector [7]. This approach was later expanded to include speaker verification systems, and even train against deep-neural network-based detection models, leading to a significant increase in equal error rate for different data sets [8]. Another application is the use of adversarial image generation to bypass image classifiers, which involves applying various transformations to the image until it successfully evades detection [4].

By studying the three groups, we can see that most of the white box attacks can only deal with differentiable loss functions, while the gray box attacks can deal with specific kinds of detectors. Black box methods, however, do not depend upon knowledge of the detectors and therefore, hold the ability to bypass different types of detectors without delving deep into the mathematical foundations of the detectors.

A natural approach for the black box attacks is where an agent learns by itself about the generation of adversarial packets through trial and error and then is able to learn a policy with high success rate and is able to confound systems. Reinforcement Learning, as this type of learning is called, provides such an avenue to introduce dynamism in adversarial packet generation as well as detection. The seminal paper on Deep Q-Learning introduces a method to use deep learning to approximate the Q-function, leveraging experience replay to reduce correlations between subsequent updates, and target network to stabilize the learning process [9]. The paper demonstrates Deep Reinforcement Learning's ability to learn complex behaviors from raw data points and hence serves as an important steppingstone in designing applications based on reinforcement learning. Although little work has been done on the attack side by implementing reinforcement learning, there has been considerable work to design network intrusion detection systems using DRL algorithms. Kezhou Ren has presented a network Intrusion Detection Model based on RFE feature extraction and deep reinforcement learning, filtering an optimum subset of features, extracting feature information through a neural network and then training a classifier using DRL to recognize network intrusions [12]. Meanwhile, Kamalakanta Sethi has used Deep Q-Networks in multiple distributed agents with attention mechanisms to efficiently detect and classify network attacks[13]. These advancements serve as cornerstones in the design of applications based on RL for both attack and defense. Although significant work has been done on Adversarial traffic generation to fool Machine Learning models and Deep Reinforcement Learning based design of network intrusion detection systems, the other side of network intrusion i.e. the attacker's perspective is negligibly explored using Deep Reinforcement Learning. Traditional GANs have made significant advances in generating adversarial data to confound classifiers. However, they do so by learning a probability distribution that emulates the data generating distribution, whereas reinforcement learning aims to generate data by performing discrete steps and observing rewards from taking those steps to learn data generating behavior. The objective of this research is to investigate the potential advantages of leveraging reinforcement learning theory, which is believed to be more closely aligned with human behavior, in the context of black box attacks. As compared to Generative Adversarial Networks (GANs), our study aims to utilize Deep Q-Learning techniques to generate adversarial packets, as this approach has been relatively underexplored. By emphasizing the synergy between RL and human behavior, we aim to offer a more pragmatic and effective method for generating adversarial packets. Secondly, we aim to test our agent against Machine Learning based Network IDSs. Moreover, although the data being monitored can be of two types: flow and packet, we focus on the packet data for this research as perturbations in the flow data is a computationally expensive and utilitarianly useless as the flow is being generated at the security end, therefore the attackers majorly focus on perturbations in packet.

3 Problem formulation

The technical aspects of the proposed approach to generate adversarial network traffic include creating a specialized environment, constructing a packet parser, and training a reinforcement agent, as described in detail below:

3.1 Development of ARL (Adversarial Reinforcement Learning) Environment:

We utilize the Markov decision process (MDP) to represent the task of generating adversarial network traffic samples, and we implement a general approach to black-box attacks through reinforcement learning (RL) algorithms. This section provides an overview of our attack methodology's threat model, presents the framework for our proposed system, and then delves into the crucial components of the system in detail.

3.1.1 Threat model description:

The threat model is based on and aims to describe the conditions of adversarial attacks against intrusion detection systems (IDSs) [3].

- Adversary's goal: The attacker's objective is to deceive the IDS by crafting adversarial samples that increase the attacker's invisibility. The attacker aims to reduce the integrity of the network intrusion detection systems from the perspective of the CIA triad (Confidentiality, Integrity, and Availability).
- Adversary's knowledge: The attacker knows that the target network is protected by a flow and packet level IDS based on machine learning. However, the attacker does not require prior knowledge about the detector, such as the algorithm, parameters, features, or training data.
- Adversary's capability: In the evasion scenario of adversarial attacks, the attacker can modify the test data, but not the detector's training data. The attacker can continuously access the detector to obtain the binary prediction result from the detector.
- Network Threat: For semester project we are limiting our threat model to port scanning attack, we will further expand this threat model in the future for publication.

3.1.2 Design of the System States and Actions:

To evade NIDS, we implement the feedback loop of RL, as illustrated in Fig. 1. Our goal is to train the agent by having it interact with an ensemble of ML models consisting of logistic regression, Multi-Layer Perceptron (MLP), and Decision Tree, so that it can learn how to generate adversarial network traffic that can evade the ML models that are based on neural network, linear, and tree-based classifiers.

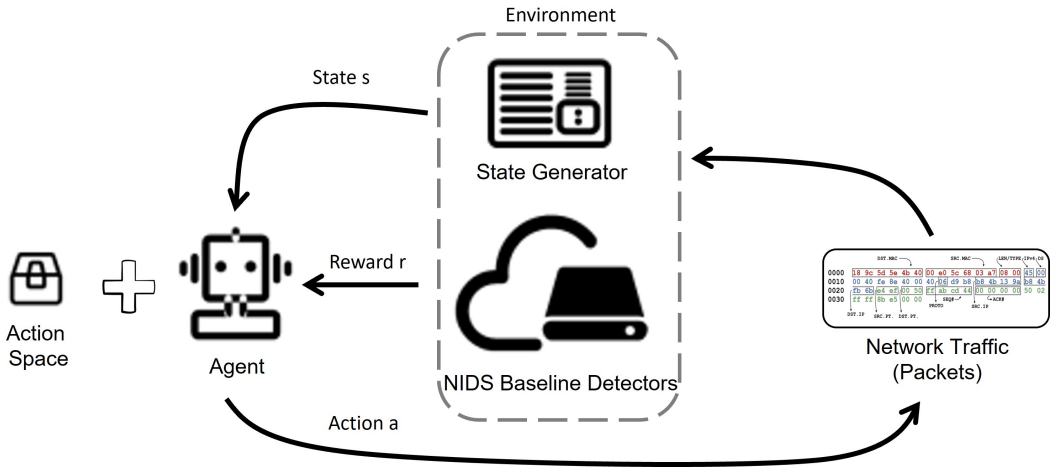

Figure 1: High level system overview

In our proposed system, we have two main components - an agent and an environment. The agent modifies the network traffic based on feedback received from the environment and selects actions from the action space using the RL algorithm. The environment consists of baseline ML classifiers and a state generator. The ML classifiers continuously predict malicious traffic and provide binary feedback to the agent as a reward. The reward is a real number that is positive when the agent

successfully modifies and develop the evasive network traffic by taking fewer possible actions from the action space. Our state generator creates a state based on the input sample or network traffic, which consists of 1525 bytes of network packet including Payload data, TCP options, checksum header, and IP header. In other words, our state space is 1525 fields of a network packet, whereas the remaining fields including source and destination IP and their corresponding ports are removed from the state space because they remain unchanged to ensure the packet functionality. Once an action is taken by the RL agent, the state generator recalculates the checksum header and remaining packet bytes or states. The loop continues until all baseline detectors are successfully misled or the number of operations reaches the upper limit of 30 steps. Another episode termination strategy is the $syn\_check()$ flag that terminates communication episode if the current state is not a TCP SYN packet. Moreover, The adopted rewarding strategy is such that the reward is 600 when all baseline models are evaded, 300 when two of them are evaded, and 50 when only a single model is evaded, whereas negative rewards are given for each action taken by the RL agent. This rewarding strategy ensures that the agent crafts adversarial samples capable of evading all baseline models with a smaller number of actions.

The proposed method for modifying network packets include a set of actions aimed at confusing certain features of malicious samples with normal ones. However, it is crucial to avoid affecting the malicious functions of the packet while making modifications. Therefore, the chosen modifications only affect areas that do not impact the functionality of malicious packets. Additionally, we discovered that incremental operations at the transport layer would not affect the original malicious functions. Based on this finding, the proposed action space consists of 11 actions that modify the data packet or add carefully constructed packets to affect the statistical characteristics of the transport layer. Here are the details of each action proposed for our ARL environment:

Action# 1: Fragmentation and addition of benign payload: This action involves breaking up the packet into smaller fragments and adding benign payload data to one of the fragments.

Action# 2: More fragmentation and addition of benign payload: This action is similar to the previous one, but it involves more fragmentation and the addition of benign payload data to multiple fragments.

Action# 3: Increase Time to live (TTL) by (+4): This action increases the TTL value of the IP header by 4, which determines the maximum number of hops a packet can take before being discarded.

Action# 4: Decrease Time to live (TTL) by (-4): This action decreases the TTL value of the IP header by 4.

Action# 5: Increase window size by (+40): This action increases the window size value of the TCP header by 40, which determines the amount of data that can be sent before receiving an acknowledgement.

Action# 6: Decrease window size by (-40): This action decreases the window size value of the TCP header by -40.

Action# 7: Add TCP options Maximum Segment Size (MSS) (+40): This action adds the MSS option to the TCP header with a value of +40.

Action# 8: Add TCP options Maximum Segment Size (MSS) (-40): This action adds the MSS option to the TCP header with a value of -40.

Action# 9: Add TCP options Window Scale (+4): This action adds the Window Scale option to the TCP header with a value of +4.

Action# 10: Add TCP options Window Scale (-4): This action adds the Window option to the TCP header with a value of -4.

Action# 11: Addition of payload: This action involves adding more benign payload data to the packet without fragmentation.

3.2 Development of Packet Parser and Labeler:

Our proposed ARL environment operates on the parsed layers of encapsulated packets. Encapsulated packets are packets that have additional protocol headers. When a packet is transmitted over a network, it is often encapsulated in multiple layers of headers, such as an Ethernet header, followed by an IP header, and then a TCP or UDP header, before finally containing the application-layer data. To train

our agent to develop evasive samples for port scan attacks, we needed a dataset containing port scan attack packets. We Utilized the CIC-IDS2017 [14] dataset to develop the required encapsulated packet data. It is a benchmark dataset used for evaluating network intrusion detection systems (NIDS) [5]. It was created by the Canadian Institute for Cybersecurity at the University of New Brunswick in 2017 and includes both flow-level and packet-level data collected from a realistic network environment. The packet-level data consists of raw network traffic captured in the PCAP format, which includes the entire network packet, including header and payload information. We utilized this PCAP level data and transformed each encapsulated packet into Ethernet header, IP header, TCP header, TCP options, and TCP segment or payload data format. The total number of bytes in each header of the transformed dataset is shown in Fig. 2. We extracted 50,000 packets of port scan attack and 50,000 samples of benign network traffic. These packets were carefully labeled using the flow-level data provided by the CICIDS 2017 developers. We had to make significant efforts to develop the data processing pipeline for our proposed ARL environment. As this step is not quite relevant to the Reinforcement learning course, we have skipped the extensive details that were considered during the development of this dataset.

Figure 2: Network Packet byte breakdown

3.3 Development and Training of ARL Agent:

Reinforcement learning is a form of machine learning techniques that allows an agent to learn by trial and error in an interactive environment, using feedback from its own experiences and actions [15]. It is commonly used in software and machines to determine the best course of action or path to take in a given situation. To address the RL problem, the agent selects an RL algorithm to learn the optimal action in each possible state it encounters. It perceives a state from the environment and maximizes its long-term reward value by learning to choose an appropriate action based on the feedback from the environment. Since our action space is not continuous, we have selected the DQN-based RL algorithm for our proposed application scenario.

3.3.1 Deep Q-Network (DQN) Theory and Mathematical Foundations:

DQN is a variant of reinforcement learning algorithm that combines the Q-learning algorithm with deep neural networks to enable agents to learn to take optimal actions in an environment. The algorithm uses an experience replay buffer and a target network to improve learning stability and performance. The DQN algorithm learns a function $Q(s, a)$ which maps a state s and action a to an estimated reward value. The function $Q(s, a)$ is represented by a neural network with weights represented by . At each iteration, the agent selects an action from the defined action space with the objective of maximizing the expected cumulative discounted reward over a sequence of time steps. This is given by the Bellman equation:

$$Q(s, a) = E[R + \gamma \max_{(a')} Q(s', a')] \tag{1}$$

where R is the reward received for taking action a in state s, is a discount factor that determines the importance of future rewards, s' is the next state reached after taking action a in state s, and a' is the action taken in state s'. The agent updates the parameters of the Q-network, , by minimizing the mean squared error between the estimated Q-value and the actual Q-value for a given transition in the replay buffer. The replay buffer stores the agent's experiences as tuples (s, a, r, s'), where r is the reward received for taking action a in state s and transitioning to state s'. The tuples are sampled randomly from the buffer during training to decorrelate the data and reduce the variance of the gradient estimates. To further improve the stability of the learning process, a separate target network with weights represented by ' is used to compute the target Q-value in the Bellman equation. The weights of the target network are updated periodically to match the weights of the Q-network.

This helps to reduce the variance of the updates to the Q-network, making learning more stable. The target network is updated with the following equation:

$$\theta' = \tau\theta + (1 - \tau)\theta' \tag{2}$$

where is a hyperparameter that controls the rate of update of the target network weights. In short, the DQN algorithm with buffer replay and target network uses a neural network to approximate the optimal action-value function, and learns by minimizing the temporal difference error between the predicted and target Q-values. The experience replay buffer stores the agent's experiences to reduce the variance of the updates, and the target network is used to stabilize the learning process by decoupling the Q-value estimates from the target values.

3.3.2 Deep Q-Network (DQN) Hyperparameters:

The following hyperparameters were used in training the DQN agent for developing evasive port scan samples:

- batch size: The number of samples used in each training iteration. In this case, 256 samples are used.
- The discount factor used in the Q-learning update equation, which determines the importance of future rewards. In this case, a discount factor of 0.8 is used.
- The starting value of the epsilon value used in the epsilon-greedy policy for exploration. In this case, the starting value is 1.
- The final value of the epsilon value used in the epsilon-greedy policy for exploration. In this case, the final value is 0.01.
- eps decay: The rate at which the epsilon value decays over time. In this case, the decay rate is set to 0.00002475.
- target update: The number of episodes between updates to the target network. In this case, the target network is updated every 10 episodes.
- memory size: The maximum size of the replay memory buffer used to store past experiences. In this case, the buffer size is set to 100000.
- lr: The learning rate used in the optimization algorithm. In this case, the learning rate is set to 0.001.
- num episodes: The number of episodes used in training the DQN. In this case, 50000 episodes are used.
- episode length: The maximum number of time steps allowed in each episode. In this case, each episode is limited to 30 time steps.
- episode duration: The total number of time steps allowed for the entire training process. In this case, the duration is set to 400000 time steps, which is equivalent to 13333 episodes (since each episode is limited to 30 time steps).

3.3.3 Deep Q-Network (DQN) Training Process:

The proposed Deep Q-Network (DQN) used in the training process consists of four fully connected layers, with the first layer having 1525 input features (corresponding to the size of the state space), and the last layer having 11 output features (corresponding to the size of the action space). The DQN training process involves the following steps:

1. Initialize the Q-Network: The first step is to initialize the Q-network, which is a deep neural network that takes in the state of the environment and outputs the Q-values for each possible action.
2. Initialize the Replay Buffer: The replay buffer is a memory that stores the experiences of the agent. It stores tuples of (state, action, reward, next state) that the agent encounters during its interactions with the environment.

3. Initialize the Target Network: The target network is a copy of the Q-network that is used to generate the target Q-values during the training process. The weights of the target network are updated periodically with the weights of the Q-network.
4. Repeat for each episode:
   - Reset the Environment: The environment is reset to its initial state at the beginning of each episode.
   - Initialize the State: The agent initializes its state by observing the environment.
   - Repeat for each time step:
     - Exploration vs Exploitation: The agent selects an action using an epsilon-greedy policy, which balances exploration (random actions) and exploitation (actions based on the current Q-values).
     - Execute the Action: The agent executes the selected action from the proposed action space (11 actions) and observes the next state and reward.
     - Store the Experience: The agent stores the current experience in the replay buffer.
     - Sample a Batch: A batch of experiences is sampled randomly from the replay buffer.
     - Update the Q-Network: The Q-network is updated using the batch of experiences. The loss function used is the mean squared error between the predicted Q-values and the target Q-values generated using the target network.
     - Update the Target Network: The weights of the target network are updated with the weights of the Q-network periodically.
   - End of Episode: The episode ends after a fixed number of time steps or when the environment reaches a terminal state.
5. Repeat for a fixed number of episodes or until convergence.

During the training process, the Q-network learns to approximate the optimal Q-values for each state-action pair, which allows the agent to select the best action in any given state. The use of a replay buffer and a target network helps to stabilize the training process and prevent overfitting. The cumulative reward obtained by the DQN agent throughout the 50,000 episodes is shown in the Fig. 3.

Figure 3: Illustration of training curve for DQN agent.

4 Evaluation Framework and Results:

In order to evaluate the effectiveness of our trained DQN agent, we conducted experimentation on three distinct black box scenarios. Each scenario involves a different unknown classifier, which was tested against our DQN agent.

The details of these scenarios, as well as the division of data used for testing, will be provided in this section to give a comprehensive understanding of the experimental process.

4.1 Dataset Division

In our proposed project, as discussed in Section 3.2, we focused on packet-level perturbations instead of flow-based perturbations. This approach makes more sense since malicious attackers cannot directly modify features generated by a specific flow of network traffic. They would need to implement changes at the packet level to incorporate such changes at the flow level. As a result, processing and labeling the raw packets of CIC-IDS2017 resulted in a massive dataset with 15 distinct attack classes. Due to time constraints and computational requirements for training, we were unable to process all the attack classes. Therefore, we focused our project on only one attack class, which is the portscan attack (randomly selected). Afterwards, We randomly selected 52,000 samples of portscan attack and 50,000 samples of benign data from the available dataset.

From the extracted dataset, We set aside 2,000 portscan samples from the 52,000 samples for testing our DQN agent, while the remaining 50,000 portscan and 50,000 benign samples were split into a 70:30 ratio for training and testing purposes. The 70% training dataset was used to train our baseline ML classifier and black box scenarios models. We evaluated the performance of these models on the 30% test data, as described in the next sections.

Additionally, we trained our DQN agent on the 50,000 portscan samples that were not used for testing. Once the black box models were trained and tested, we evaluated their performance using our trained DQN agent. We utilized the 2,000 portscan samples that were set aside for DQN testing to evaluate the performance of our DQN agents on these black box scenarios.

4.2 Black Box Scenarios

To evaluate the effectiveness of our trained DQN agent, we tested it on three different black box scenarios, where we used a different unknown classifier in each scenario and compared the performance of our DQN agent against it. The subsequent sections provide details and results for each scenario. However, a comprehensive overview of the results for our trained DQN agent is presented in Table 1. This table illustrates the effectiveness of the generated adversarial samples by the trained DQN agent, where the performance of each black box model on the original samples represents the effectiveness of the black box models. In contrast, the performance of adversarial samples shows the detection rate or accuracy of black box models when the original samples are transformed into adversarial samples using our trained DQN agent. It is evident that the performance of black box models drop significantly, from 99% to 1-2%, which indicates the effectiveness of our trained DQN agent in generating functional adversarial samples.

Table 1: Performance of Trained DQN Agent.

Black_Box Models	Performance (Accuracy)	
	Original Samples	Adversarial Samples
Random Forest (RF)	99.9%	1.65%
Deep Neural Network (DNN)	99.75%	1.45%
1-D Convolutional Neural Network (1D-CNN)	99.8%	2.7%

4.2.1 Scenario #1:

In the first scenario, we employed a Random Forest Model with 100 n-estimators, which was trained on 70% of the previously mentioned training dataset. The model's performance was evaluated by testing it using the remaining 30% of the dataset, which resulted in 100% accuracy.

Since the model showed good performance on the test set, indicating its generalization ability, we tested it on a separate set of 2000 port-scan data instances that were initially reserved for DQN testing. The model was able to successfully detect these instances, with only two misclassifications. Next,

we used our trained DQN agent to craft adversarial port-scan data instances with fewer black-box queries, which were presented to the trained Random Forest Classifier. Our DQN agent took different actions according to the learned policy in each episode. Figure 4 shows a histogram illustrating the number of queries required by our trained DQN agent to successfully evade the black box model. As the figure shows, most of the samples were evaded within just six actions (queries to the black box model).

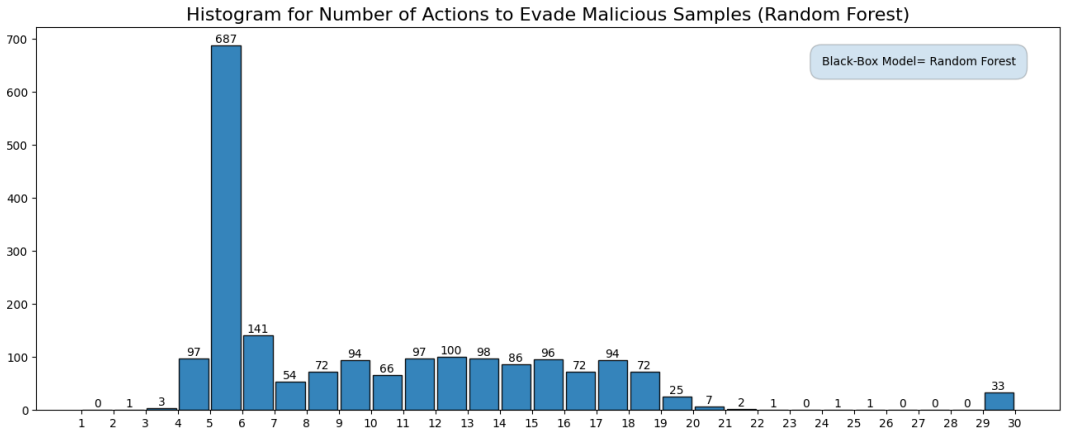

Figure 4: Visual representation for the distribution of the number of actions required to evade malicious samples in a Black-Box(RF) model. The x-axis represents the number of queries (actions), while the y-axis represents the frequency of occurrence of each range. Samples requiring 30 or more actions were not evaded by the trained agent due to the capped limit.

After generating the adversarial samples using our trained DQN agent, we evaluated the effectiveness of these samples on the black box Random Forest Classifier. Out of the 2000 generated adversarial samples, only 33 were detected by the Random Forest Classifier, which highlights the vulnerability of even well-optimized models and the effectiveness of our trained DQN agent.

The performance of the Random Forest Classifier on the original 2000 portscan samples, as well as on the adversarial samples, is illustrated in Fig. 5. The confusion matrix for the testing dataset represents the performance of the Random Forest Classifier on the 30% testing dataset. Meanwhile, the confusion matrix for the original samples shows the model's performance on the original; 2000 portscan samples. Lastly, the confusion matrix for the adversarial samples shows the model's performance when presented with the adversarial samples crafted by our trained DQN agent.

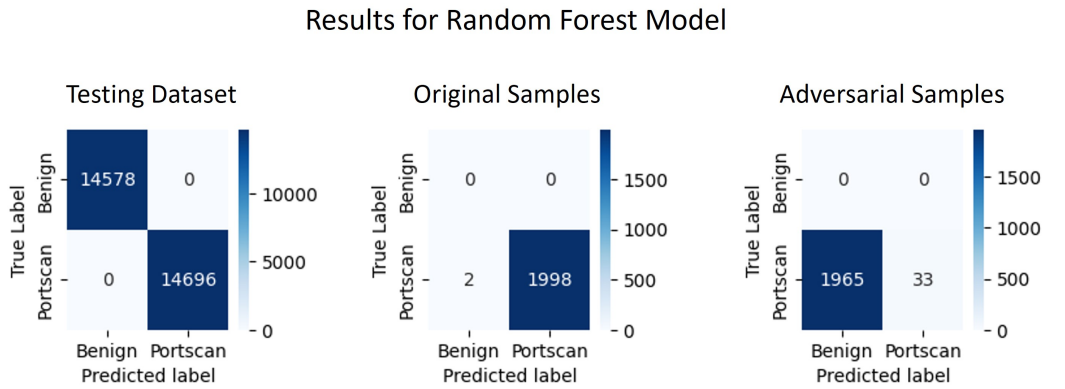

Figure 5: Performance of Random Forest (RF) Black-Box model.

These results demonstrate that despite the Random Forest Classifier's strong performance, it is still vulnerable to adversarial samples crafted by our trained DQN agent. As such, our ARL environment

can be utilized to assess the vulnerabilities of ML-based NIDS, and the generated adversarial samples can be used to improve the adversarial robustness of these systems.

4.3 Scenario #2:

In the second scenario, we utilized a Deep Neural Network (DNN) with five layers, which was trained on a training dataset containing benign and port-scan attack samples to achieve a training accuracy of 100%. Following the approach from the previous scenario, we evaluated the model's performance on the test set using accuracy. Subsequently, we tested the DNN model on a separate set of 2000 port-scan data instances, including their adversarial versions crafted by our trained DQN agent. The performance of the DNN model is illustrated in Fig. 6. The confusion matrix for the testing dataset represents the model's performance on the 30% testing dataset, while the confusion matrix for the original samples shows the model's performance on the original 2000 port-scan samples. Lastly, the confusion matrix for the adversarial samples shows the model's performance when presented with the adversarial samples crafted by our trained DQN agent.

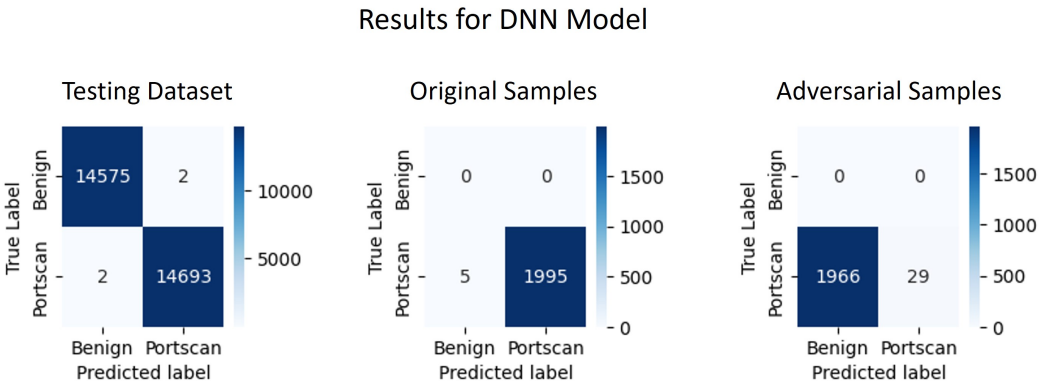

Figure 6: Performance of Deep Neural Netowrk (DNN) Black-Box model.

Figure 7 shows the histogram of the number of queries required by our trained DQN agent to successfully evade the black box model for the DNN scenario. The results indicate that the number of actions required to evade the DNN model by our trained DQN agent is relatively dispersed, but the majority of samples are evaded after 5 or 6 actions. Overall, these results demonstrate the efficacy and transferability of our trained DQN agent in evading different black box models, including DNN.

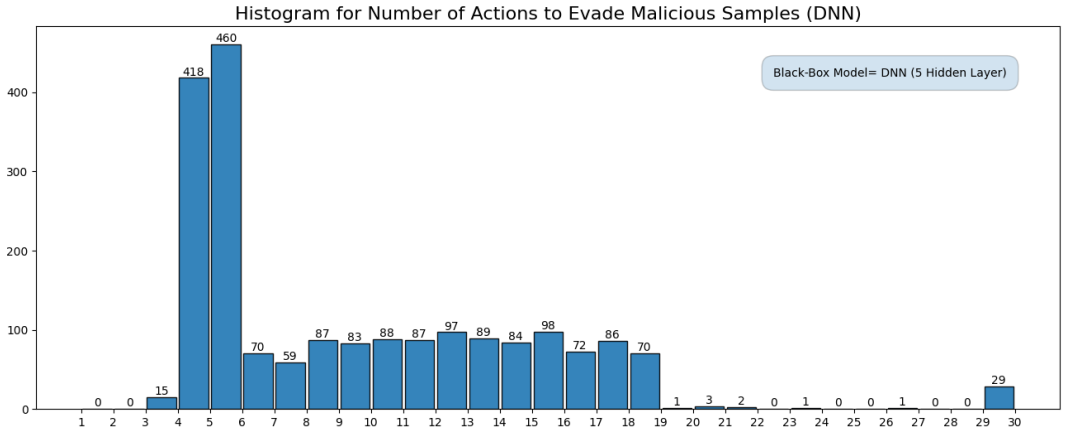

Figure 7: Visual representation for the distribution of the number of actions required to evade malicious samples in a Black-Box(DNN) model. The x-axis represents the number of queries (actions), while the y-axis represents the frequency of occurrence of each range. Samples requiring 30 or more actions were not evaded by the trained agent due to the capped limit.

The results indicate that our trained DQN agent is capable of evading different black box models, demonstrating its learning transferability across a range of classifiers. In particular, the DNN model was also vulnerable to adversarial samples generated by our DQN agent, highlighting the need for developing more robust and secure ML-based NIDS to tackle adversarial attacks.

4.4 Scenario #3:

In the third scenario, we used a 1D Convolution Neural Network (1D-CNN) with 5 layers, which was also trained on benign and port-scan attack data instances to achieve 99% accuracy on test dataset. Similar to scenario 1, we evaluate it on the 2000 port-scan data instances and their adversarial versions. The performance of the 1D-CNN model is illustrated in Fig. 6. The confusion matrix for the testing dataset represents the model's performance on the 30% testing dataset, while the confusion matrix for the original samples shows the model's performance on the original 2000 port-scan samples. Lastly, the confusion matrix for the adversarial samples shows the model's performance when presented with the adversarial samples crafted by our trained DQN agent.

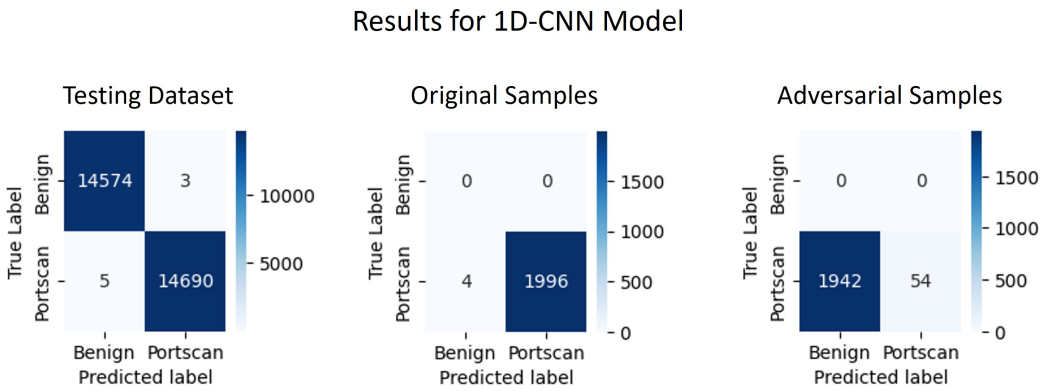

Figure 8: Performance of 1D-Convolutional Neural Network (1D-CNN) Black-Box model.

Similarly to the first two scenarios, we analyzed the performance of our trained DQN agent against a 1D Convolutional Neural Network (1D-CNN) black box model. Fig. 9 shows the histogram of the number of queries required by our trained DQN agent to successfully evade the 1D-CNN model. The results demonstrate that the majority of the samples were evaded by the DQN agent within 11 actions. However, there were 54 samples that could not be evaded even after 30 actions.

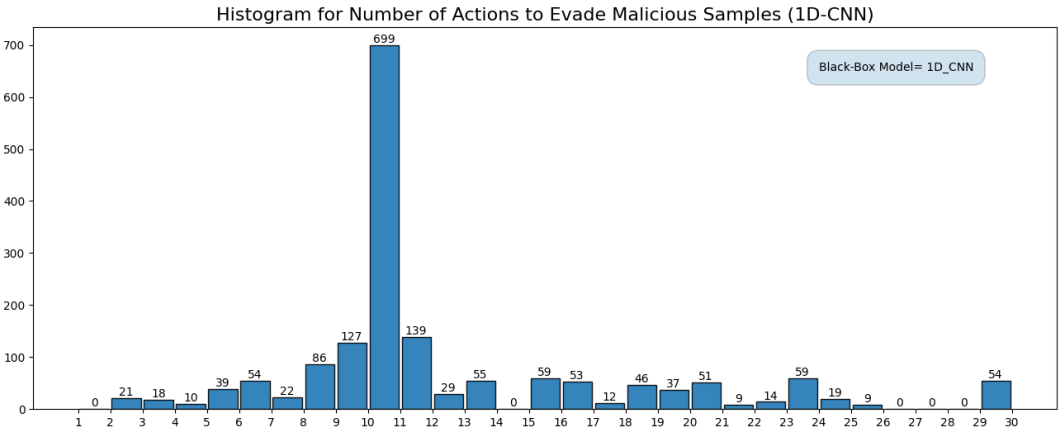

Figure 9: Visual representation for the distribution of the number of actions required to evade malicious samples in a Black-Box(DNN) model. The x-axis represents the number of queries (actions), while the y-axis represents the frequency of occurrence of each range. Samples requiring 30 or more actions were not evaded by the trained agent due to the capped limit.

4.5 Dominant Actions

To gain more insights into the behavior of the trained DQN agent, we conducted a further analysis of our project by evaluating the dominant actions taken by the agent across all evaluation scenarios. The analysis revealed that the most dominant actions were TTL increment and payload addition. We expected the addition of payload without fragmentation to be a dominant action, as it can help evade the learning of different classifiers by adding benign payload to malicious packets.

However, the higher value of TTL indicates the complexity of the network traffic, making it more difficult for the NIDS classifier to accurately identify and classify the traffic. As a result, when a packet with a higher TTL value is passed to the classifier, it treats it as complex traffic, which increases the chance of misclassifying it as benign.

To better visualize the dominant actions, we created a histogram of all actions under three different evaluation scenarios, as shown in Fig. 10. This analysis highlights the importance of considering the effect of packet characteristics, such as TTL values, when designing and evaluating NIDS classifiers. It also demonstrates the potential of adversarial attacks to exploit these characteristics to evade NIDS classifiers.

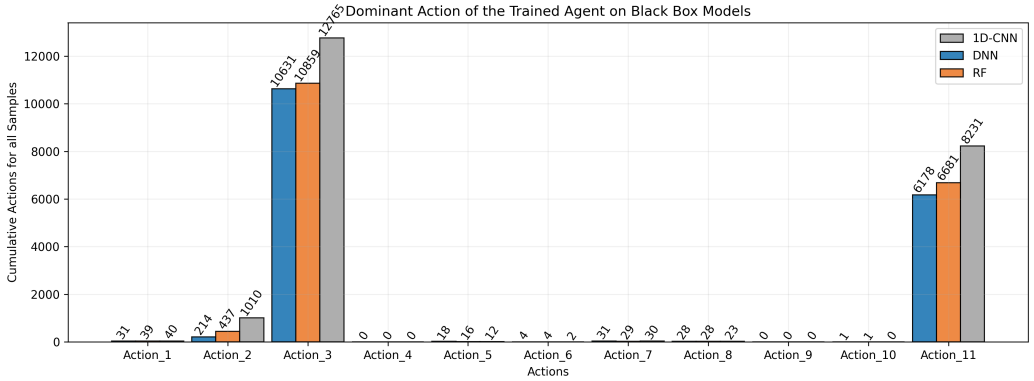

Figure 10: Illustration of cumulative actions taken by trained agent on different black-box models. The most adopted action for evading malicious samples is Action_3 (Increment of TTL) and Action_11 (Payload addition).

5 Conclusion

This paper introduces a novel framework called ARL that uses reinforcement learning (RL) to generate adversarial network traffic data for black box adversarial attacks against ML-based NIDS at the packet level. The framework employs an action space consisting of eleven functionality-preserving actions that modify crucial transport layer characteristics while retaining the original malicious functions of the traffic packets. In contrast to previous work that focuses on flow-based features of network traffic, which is not practical as attackers cannot directly change the flow features of network traffic, we emphasize our approach on raw packets of network traffic. Furthermore, we evaluate the trained DQN agent on three different types of black box models to demonstrate the vulnerability of ML-based NIDS to adversarial attacks. The results show that the proposed system can achieve significant evasion rates with fewer queries and can be used to evaluate and assess the vulnerabilities of machine learning models. The generated samples through the ARL environment are highly evasive and more generalized than existing black box attacks, and can be used to induce adversarial robustness in machine learning models through training. Although the focus of this work is specifically on port scan attacks and is exploratory in nature, future work will include more attacks from the CIC-IDS2017 dataset and advanced machine learning models such as transformers in the evaluation framework.

References

[1] Ahmed F AlEroud and George Karabatis. Queryable semantics to detect cyber-attacks: A flow-based detection approach. *IEEE transactions on systems, man, and cybernetics: systems*, 48(2):207–223, 2016.

[2] Giovanni Apruzzese and Michele Colajanni. Evading botnet detectors based on flows and random forest with adversarial samples. In *2018 IEEE 17th International Symposium on Network Computing and Applications (NCA)*, pages 1–8, 2018.

[3] Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Šrndić, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. Evasion attacks against machine learning at test time. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2013, Prague, Czech Republic, September 23-27, 2013, Proceedings, Part III 13*, pages 387–402. Springer, 2013.

[4] Hung Dang, Yue Huang, and Ee-Chien Chang. Evading classifiers by morphing in the dark. In *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, pages 119–133, 2017.

[5] Mossa Ghurab, Ghaleb Gaphari, Faisal Alshami, Reem Alshamy, and Suad Othman. A detailed analysis of benchmark datasets for network intrusion detection system. *Asian Journal of Research in Computer Science*, 7(4):14–33, 2021.

[6] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[7] Weiwei Hu and Ying Tan. Generating adversarial malware examples for black-box attacks based on gan. In *Data Mining and Big Data: 7th International Conference, DMBD 2022, Beijing, China, November 21–24, 2022, Proceedings, Part II*, pages 409–423. Springer, 2023.

[8] Hongwei Luo, Yijie Shen, Feng Lin, and Guoai Xu. Spoofing speaker verification system by adversarial examples leveraging the generalized speaker difference. *Security and Communication Networks*, 2021:1–10, 2021.

[9] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[10] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2574–2582, 2016.

[11] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia conference on computer and communications security*, pages 506–519, 2017.

[12] Kezhou Ren, Yifan Zeng, Zhiqin Cao, and Yingchao Zhang. Id-rdrl: a deep reinforcement learning-based feature selection intrusion detection model. *Scientific Reports*, 12(1):1–18, 2022.

[13] Kamalakanta Sethi, Y Venu Madhav, Rahul Kumar, and Padmalochan Bera. Attention based multi-agent intrusion detection systems using reinforcement learning. *Journal of Information Security and Applications*, 61:102923, 2021.

[14] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A Ghorbani. Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization. 2018.

[15] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

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