Robust Adaptive Cloud Intrusion Detection System Using Advanced Deep Reinforcement Learning

Kamalakanta Sethi(✉), Rahul Kumar, Dinesh Mohanty, and Padmalochan Bera

Indian Institute of Technology, Bhubaneswar, India
{ks23,rk36,dm22,plb}@iitbbs.ac.in

Abstract. Intrusion Detection System (IDS) is a vital security solution for cloud network in providing defense against cyber attacks. However, existing IDSs suffer from various limitations that include the inability to adapt to changing attack patterns, identify novel attacks, requirements of significant computational resources, and absence of balance between accuracy and false-positive rates (FPR). These shortcomings in current IDSs reduce their effectiveness for deploying in cloud-based application systems. Moreover, most of the cloud IDS researches use conventional network benchmark datasets like NSL-KDD for evaluation, which do not provide the actual picture of their performance in real-world cloud systems. To address these challenges, we propose a Double Deep Q-Network (DDQN) and prioritized experience replay based adaptive IDS model built for accurate detection of new and complex attacks in cloud platforms. We evaluated our proposed model using a practical cloud-specific intrusion dataset, namely, ISOT-CID and a conventional network-based benchmark dataset (NSL-KDD). The experimental results show better performance than state-of-the-art IDSs along with novel attack detection capabilities. Further, We have used flow-based analysis in our model to ensure low computing resource requirements. Besides, we evaluated the robustness of our model against a black-box adversarial attack resembling a real-life scenario and observed a marginal decrease in the performance. Finally, we demonstrated our model's usability in a practical use case with frequent changes in the attack pattern.

Keywords: Intrusion Detection System · Double Deep Q-Network · Cyber attacks · ISOT-CID · NSL-KDD · False positive rate · Black-box adversarial attack

1 Introduction

Cloud Computing is an effective application design and implementation platform that allows on-demand access to a shared pool of configurable resources (e.g., servers, applications, storage, networks, computation) through networks. These

L. Batina et al. (Eds.): SPACE 2020, LNCS 12586, pp. 66–85, 2020.
https://doi.org/10.1007/978-3-030-66626-2_4

resources are easily scalable and swiftly provisioned with the least management effort or interaction with the service provider. However, the open, distributed, and multi-tenant architecture of cloud makes it vulnerable to various network attacks (such as IP spoofing, Distributed Denial of Service (DDoS), etc) as well as some cloud platform-specific attacks (such as insider attack, cloud malware injection, etc) [6]. In the last decade, network industries have developed various tools like firewalls, access control mechanisms to control unauthorized access to data and resources. However, these techniques are not resilient to insider attacks. Subsequently, researchers have proposed cloud intrusion detection systems as the second line of defense to protect the organizational assets from intruders. IDS monitors network traffic and system-level activities to detect intrusions. IDS can be classified into two categories based on the target environment it is operating, i.e. (1) host-based (installed and deployed in a host to monitor the behavior of a single system [5]) and (2) network-based (captures traffic of the entire network and systematically analyses to detect any attack on the hosts of that network for possible intrusions). There are two methods based on the data source to be analyzed in network-based IDSs (NIDSs): packet-based NIDSs and flow-based NIDSs. Packet-based NIDSs have to scan all the individual packets that pass through the network connection and inspect their content beyond the protocol header, which requires much computation. On the other hand, the flow-based NIDSs looks at aggregated information of related packets of network traffic in the form of flow and consume fewer resources.

Researchers also classified IDS based on the detection technique as (1) signature-based systems, and (2) anomaly-based systems [5]. Signature-based systems use a repository of signatures of already identified malicious patterns to detect an attack. They are efficient in detecting known attacks but fail in case of unforeseen attacks. In contrast, anomaly-based IDS detects intrusions when the activity of any user deviates from its normal functionality. Although these systems detect zero-day attacks, they tend to generate a lot of false alerts leading to a high false-positive rate (FPR).

In the last decade, many researchers proposed traditional machine learning and deep learning-based cloud IDS system that show excellent performance [1,22]. However, they also have several limitations that include: 1) lack of proper adaptivity towards novel and changing attack patterns in the cloud environment and inability to identify them with high accuracy and low False Positive Rate (FPR), 2) require frequent human intervention for training which introduces more vulnerabilities and, thereby, affects the model's performance, 3) use datasets (such as NSLKDD, UNSW, and AWID) that are obtained by simulating a conventional network environment and thus, do not reflect a real cloud environment.

Machine learning classifiers are vulnerable to adversarial scenarios where the attacker introduces small perturbations in the input and attempts to bypass the security system. Adversarial machine learning is the study of such techniques where attackers exploit vulnerabilities of ML models and attempts to dupe them with perturbed inputs. In paper [27], the author shows that slight

perturbations to the raw input can easily fool DNNs into misclassification. Therefore, it is essential to understand the performance against adversarial scenarios and ensure a robust security system design. Several properties of cloud computing including multi-tenancy, open access, and involvement of large business firms capture the attention of adversarial attackers. This is mainly because of its potential to cause serious economic and reputational damage. These attacks have become more sophisticated in recent times due to better computation resources and algorithms. There is very little research done towards understanding the performance degradation of an IDS that occurs in adversarial scenarios.

Therefore, we aim at designing a robust and adaptive IDS suitable for cloud platforms using advanced deep reinforcement learning techniques. Here, we present significant contributions and novelty of our proposed IDS.

(1) Implementation of a Double Deep-Q Learning-based Cloud IDS: We use DDQN, which is an advanced deep reinforcement learning algorithm for building an adaptive cloud IDS. Our proposed IDS can detect and adapt to novel cloud-specific attack patterns with minimal human interaction.
(2) Integration of prioritized learning: Online reinforcement learning agents use experience replay to retain and reuse experiences from the past. Instead of random or uniform selection, we used a concept of prioritizing experiences to call on significant transitions (with higher learning values) more often that ensures more effective and efficient learning.
(3) Experimentation on a realistic cloud-specific intrusion dataset (ISOT-CID): We have evaluated our model on the first publicly available cloud-specific dataset (ISOT-CID) whose features were obtained by applications running on Openstack based cloud environment. We have also obtained the model's performance on a very well-known conventional network-based NSL-KDD dataset for comparison with state-of-the-art-works.
(4) Durability against adversarial attacks: Adversarial attackers exploit the vulnerabilities of the machine and deep learning models and trick them for misclassification. We employed the concept of DDQN that eliminates the overestimation problems faced by other Q-learning techniques and shows high robustness with only marginal performance degradation when exposed to adversarial samples produced by a practical black box attack.

The rest of the paper is organized as follows. Background on DDQN, and prioritized experience replay is presented in Sect. 2. In Sect. 3, we discuss the related work. Section 4 presents a brief overview of datasets and their preprocessing steps. Our proposed IDS model is presented in Sect. 5. The robustness of our IDS model is presented in Sect. 6. The experimental results and conclusions are discussed in Sect. 7 and Sect. 8 respectively.

2 Background

In this section, we discuss a brief introduction of two key concepts used in this paper, including DDQN and prioritized experience replay.

2.1 Double Deep Q Learning (DDQN)

Double Deep Q Learning algorithm is based on the Double Q Learning and Deep Q Learning algorithms. These algorithms are themselves derived from Q Learning algorithm. Q Learning algorithm is a popular model-free reinforcement learning algorithm used in FMDP (Finite Markov Decision Process), where an agent explores the environment and figures out the strategy to maximize the cumulative reward. Agent takes actions that make it, move from the current state to a new state generating a reward alongside. For the specific FMDP, it identifies an optimal action selection policy to maximize the expected value of the total reward that can be obtained in the successive steps (provided that it is given infinite exploration time and a partly-random policy) [12].

A significant limitation of Q-learning is that it works only in environments with discrete and finite state-action spaces. To extend Q-learning to richer environments (where storing the full state-action table is often infeasible), we use Deep Neural Networks as function approximators that can learn the value function by taking just the state as input. Deep-Q Learning is one such solution. Deep Q Learning was tested against classic Atari 2600 games [31], where it outperformed other Machine Learning methods in most of the games and performed at a level comparable with or superior to a professional human games tester. However, in the paper [7], Hado et al. explain the frequent overestimation problem found in Deep Q Learning due to the inherent estimated errors of learning. Such overestimation related errors are also seen in the Q Learning algorithm and were first investigated by Thrun and Schwartz [12]. They proposed Double Q-learning as a solution, in which there is a decoupling of action selection and action evaluation procedures that help lessen the overestimation problem significantly. Later, Hado et al. [7] proposed a Double Deep Q Learning architecture that uses the existing architecture and DQN algorithm to find better policies, thereby improving the performance [7]. They tested their model on six Atari games by running DQN and Double DQN with six different random seeds. The results showed that the over-optimistic estimation in Deep Q Learning was more frequent and severe than previously acknowledged. The results also showed that Double Deep Q Learning gave the state of the art results on the Atari 2600 domain.

2.2 Prioritized Experience Replay

Many Reinforcement Learning algorithms involve storing past samples and retraining the model with such samples to help the agents memorize and reuse past experiences. In the simple experience replay algorithm, the samples are chosen randomly. However, Schaul *et al.* [12] pointed out that selecting the samples based on their capacity to improve the learning of RL agent would lead to much better learning from past experiences. This gave rise to the concept of prioritized experience replay wherein the samples are assigned priorities based on metrics similar to Temporal Difference, which represent the scope of improvement that lies for the agent to be able to correctly predict the outcomes for the

sample under consideration. In particular, the priority for i^{th} sample is given by $P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$, where $p_i = |\delta_i| + \epsilon$,α is the prioritization factor which determines the importance that is to be given to priorities while sampling past experiences. Here, δ_i is the Temporal Difference error function on which the Huber Loss Function [30] is applied. The frequency of selection of samples is made to be directly proportional to such priority values. Retraining RL models with samples selected based on the principles of prioritized experience replay gives much faster learning speeds and significant improvements in performance when tested on the Atari Dataset [12]. We have briefly described the implementation details in Sect. 5.

3 Related Work

In this section, we presents some network-based IDS works for cloud environment that use machine learning techniques.

In [10], Lopez-Martin *et al.* proposed a advanced IDS based on several deep reinforcement learning (DRL) algorithms using NSL-KDD and AWID datasets. They replaced the requirement of conventional DRL technique for real-time interaction with environment by the concept of pseudo-environment, which uses a sampling function over the dataset records. They analyse the performance of their technique on four different DRL models that includes Deep-Q-Network (DQN), Double Deep-Q-Network (DDQN), Policy Gradient (PG), and Actor-Critic (AC). It has been observed that the DDQN algorithm gives the best results. However, this IDS work limited to enterprise environment rather than the cloud architecture.

Sethi *et al.* [2] presented a cloud NIDS using reinforcement learning. Their IDS can detect new attacks in the cloud and also adaptive to attack pattern changes in the cloud. They validated the efficacy of their model using a conventional network dataset (UNSW) instead of cloud network datasets. However, this work uses UNSW dataset which is not derived from any cloud environment.

Kholidy *et al.* [4] created a new dataset called cloud intrusion detection dataset (CIDD) for cloud IDS evaluation. The dataset includes both knowledge and behavior-based audit data. To build the dataset, the authors implemented a log analyzer and correlator system (LACS) that extracts and correlates user audits from a set of log files from the DARPA dataset. The main issue with this dataset is that its main focus is on detecting a masquerade attack. Also, it does not consider network flows involving hypervisor. Moreover, the dataset is not publicly available.

However, none of the above-mentioned works considers the effectiveness of the IDS system against smart adversarial attackers, which is more frequent in the cloud platform. Szegedy *et al.* [29] first observed the exposure of the DL-based approach to adversarial attacks. Wang [28] proposed a DL-based IDS system in 2018, considering the adversary's effect.

In summary, state-of-the-art works don't apply DRL for Cloud intrusion detection systems though a few recent attempts are present on conventional network applications. Also, the existing works do not use cloud-specific datasets

and, thereby, are not capable of representing a real cloud environment. Aldribi *et al.* [3] introduced the first publicly available cloud dataset called ISOT-CID, which is derived from a real cloud computing environment. The dataset consists of a wide variety of traditional network attacks as well as cloud-specific attacks. The author discusses a hypervisor-based cloud IDS involving novel feature extraction, which obtains an accuracy (best performance) of 95.95% with an FPR of 5.77% for phase 2 and hypervisor-B portion of the dataset. Our proposed IDS employs an advanced DRL technique, involving the concepts of DDQN and prioritized experience replay, for accurate detection of intrusions in the cloud. We have used the ISOT-CID dataset for testing the effectiveness of our model on real-world cloud platforms along with the benchmark NSL-KDD dataset. The experimentation shows that the proposed model is adaptive to changes in attack patterns as well as robust to the adversarial scenarios.

4 Dataset Description and Prepossessing

Our proposed intrusion detection system is evaluated on benchmark NSL-KDD and cloud-specific ISOT-CID dataset. In this section, we will present a brief overview of both datasets as well as their preprocessing to obtain relevant features.

4.1 NSL-KDD Dataset

Here, we discuss about the overview and preprocessing steps on the benchmark NSL-KDD dataset.

1. **Overview of the dataset:** We evaluated our proposed IDS model on NSL-KDD dataset [13,14]. NSL-KDD is one of the most widely used dataset for evaluating any network intrusion detection system. The dataset consists of a total of 1,48,517 records. The total records are divided into a training dataset and a testing dataset. The training dataset consists of 1,25,973 records whereas the testing dataset consists of 22,544 records. Each record in training and testing dataset consists of 41 attributes. The training dataset consists of 23 attack types. Similarly testing dataset consists of 37 attack types out of which 16 are novel attacks that not present in the training dataset. The attack types are grouped into four types. namely DoS (Denial-of-Service), Probe, U2L, and R2L. The distribution of labeled data in the training and testing dataset is Table 1.

Table 1. NSL-KDD dataset classes distribution

Dataset	Total	Normal	Attack
Training	125973	67343	58630
Testing	22544	9711	12883

Table 2. ISOT-CID dataset classes distribution

Dataset	Total	Normal	Attack
Training	17296	10377	6919
Testing	7411	4447	2964

2. **Preprocessing of the dataset:** As the NSL-KDD dataset contained categorical features, we used one-hot encoding as part of the preprocessing of the dataset to encode all such categorical features. This process increased the count of features from 41 to 122. The next operating step is feature normalization. Many classifiers use distance as a normalization tool. The train and test datasets of NSL-KDD were normalized to values between 0 and 1 by L2 normalization [15]. Further, we modified the category column into binary types for binary classification. The sample's label value is 1 in the presence of some form of attack while it is 0 if the label value is normal.

4.2 ISOT-CID Dataset

Here, we discuss about the overview and preprocessing steps on the ISOT-CID dataset.

1. **Overview of the dataset:** To evaluate our model we have used the ISOT Cloud Intrusion Dataset (ISOT-CID) [8], which is the first publicly available cloud-specific dataset. The data was collected in the cloud for several days with time slots of 1–2 h per day with the help of special collectors at different layers of the OpenStack based production environment (hypervisor layer, guest hosts layer and the network layer), that forwarded the collected data (network traffic, system logs, CPU performance measures, memory dumps, etc) to the ISOT lab log server for storage and analysis. Malicious activities includes both outsider and insider attacks. For our purpose, we have used only the network traffic data portion of the ISOT-CID. The entire ISOT-CID dataset of size 8 TB consists of 55.2 GB of network traffic data. The collected network traffic data was stored in a packet capture (pcap) format. In phase 1, a total of 22,372,418 packets were captured out of which 15,649 (0.07%) were malicious, whereas in phase 2, a total of 11,509,254 packets were captured out of which 2,006,382 (17.43%) were malicious.
2. **Preprocessing with Tranalyzer:** Since packet payload processing involves huge amount of data that have to be processed at very fast rate, flow-based analysis for intrusion detection is considered better for high-speed networks due to lower processing loads [11]. To obtain flow-based data from packet-based data, we have used this open-source tool called Tranalyzer which is a lightweight flow generator and packet analyzer designed for practitioners and researchers [9]. With the help of Tranalyzer, we were able to get about 1.8GB of flow based data in JSON format from about 32.2 GB of packet-based data in pcap format.
3. **Analysis of Tranalyzer output:** All the 37 JSON object files which were output by Tranalyzer, were converted into a single CSV file. This CSV file was further processed to deal with lists of numbers (replaced by mean value) and lists of strings (replaced with first string). Finally, all the strings and hexadecimal values (representing particular characteristics of flow) were one-hot encoded for further improvement of the training data. Further, the values that are not integers or floating-point numbers, were converted to 'Nan' values

and later removed in such a way that can make the dataset compatible for the machine learning analysis. We then labeled the dataset based on the list of malicious IP addresses that were provided along with the ISOT-CID documentation [3].

Finally, we apply feature selection process that helps in removing irreverent features avoiding over-fitting and achieving better accuracy of the model at low computational ability. In our model, we have used chi-square feature selection algorithm [26]. The number of selected features becomes 36 and 164 after applying feature selection on the NSL-KDD dataset and ISOT-CID respectively. For both dataset, feature vector refers to any record consisting the same features as obtained from the feature selection process. Feature set refers to the collection of all such feature vectors for a particular dataset. Further in case of ISOT-CID post preprocessing, we found out that the dataset was highly skewed, i.e, the number of non attack samples was much higher than that of the number of attack samples. Hence, to prevent biased learning, we selected a portion of the dataset which had a more balanced distribution having 9883 attack samples and 14,824 normal samples. Table 2 shows the distribution of the dataset in the training and testing phase.

5 Proposed Intrusion Detection System

Here, we present our proposed cloud IDS that uses concepts of DDQN and prioritized learning. Figure 1 show the deployment architecture of our proposed IDS. It mainly consists of three sub-components: (i) host network, (ii) agent network, and (iii) cloud administrator network. The host network has the running virtual machines (VMs), hypervisors, and host machines. The agent network consists of three modules namely Log Processor, Feature Extractor, and Agent. Agent network is connected to host network through VPN (Virtual Private Network). The use of VPN is to prevent the agent network from being compromised by external attackers and to ensure fast sharing of attack related information or abnormal activities in the network. To identify the potential intrusions, it is essential to obtain the audit logs of the target machine. Hence, our model uses system call traces to communicate audit log information that may include VM instance specification, CPU utilization, and memory usage in the host machine.

The agent network obtains audit logs from the host network and performs preprocessing and feature selection on these data to extract the feature vectors. Finally, it identifies the state of the host network and shares the same with the Cloud Administrator. In general the state of a host can be "attacked" and "not attacked". Depending upon the capacity of the host network, the agent network logic can be deployed on a single VM (for a small number of host VMs) or as a Virtual Cluster (VCs) (for a large number of host VMs) in a separate physical machine in the cloud infrastructure. Next, we discuss the three major functional components in the agent network.

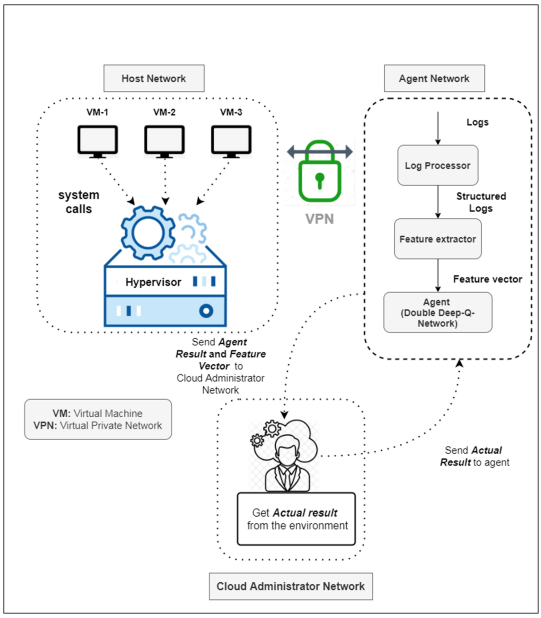

Fig. 1. Cloud IDS deployment architecture

(1) **Log Processor module:** This module receives logs (in an unstructured format) as input from the hypervisor via VPN channel. It processes logs and converts it into a structured format. The communication between the hypervisor and this module is generally realized with high bandwidth links so that necessary logs are shared before there is any intervention from the attacker.

(2) **Feature Extractor module:** This module first applies preprocessing on data collected from log processor module. This step is mainly used for input data validation such as removal of NULL values, data-types conversion, string data removal, one-hot encoding, etc. Subsequently, it performs feature extraction and feature selection to obtain essential feature values. Further, these feature values are combined to obtain a $feature\_vector$, which is fed as input to the current DQN in agent module.

(3) **Agent module:** This module executes an algorithm (i.e., Algorithm 1) that includes a combination of trained advanced machine learning classifiers and a Double-Deep-Q-Network (DDQN). It takes $feature\_vector$ as input and obtains classifier predictions. The concatenation of feature vector and the corresponding classifier prediction results forms a state vector. This state vector is fed to the Current DQN which produces $agent\_result$. Then, the agent result (i.e., $agent\_result$) is shared with the cloud administrator, and the final decision (i.e., $actual\_result$) is obtained based on a voting mechanism by cloud administrator. Subsequently, the agent calculates the reward using the actual result and continuously improves its learning over the time.

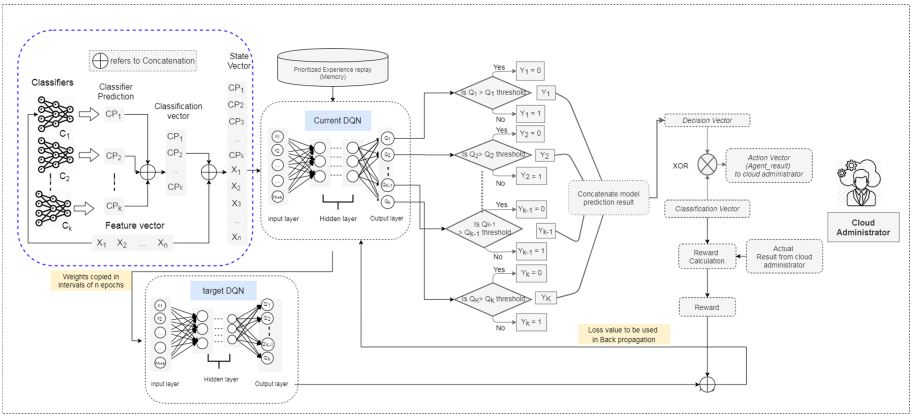

Fig. 2. Agent result calculation workflow using DDQN and prioritised learning. CP_1, CP_2,...CP_k denotes the prediction in form of 1 (for attack) and 0 (for normal) for classifier C_1, C_2,...C_k respectively

Next, we derive important reinforcement learning (RL) parameters including the state vector, action, and rewards.

State: State in RL describes the input by the environment to an agent for taking actions. In cloud environment, system logs parameters ($feature\_vector$) are used for state parameters as it reflects its current running status. Also, we run the combination of trained classifiers with $feature\_vector$ as input to extract prediction values as $classification\_vector$. These two vectors are concatenated to obtain $state\_vector$. Experimentally, the integration of advanced classifiers with DDQN logic has shown good performance in obtaining higher accuracy along with low FPR values. The dotted part in blue color of Fig. 2 presents a workflow on the calculation of state vector estimation process.

Action: An action refers to the agent's decision after monitoring the state of the cloud system during a given time window. The Agent network produces an $action\_vector$ for a given input feature vector, which is the basis for the final judgment about the attack. It uses three essential steps: a) Obtaining the state vector using the $feature\_vector$ and the classifier prediction on these feature vectors, b) Feeding the $state\_vector$ to the Current DQN, c) Thresholding the output of the Current DQN (i.e., the Q-values) against predetermined threshold values, also known as *Q-threshold value*, for classifying the decision as an attack or normal, d) Combining the classifications to form a vector called the $decision\_vector$. e) Obtaining $action\_vector$ as logical XOR between the $decision\_vector$ and the $classification\_vector$. We perform the steps c) to e) to ensure that the action vector should include output same as the classifier prediction if the corresponding Q-value exceeds the Q-threshold value. Otherwise, it should include output that is against the classifier prediction. Also, in our algorithm, the term $action\_vector$ and $agent\_result$ refers to same. Figure 2 shows the entire workflow on the calculation of $agent\_result$.

Reward: A reward indicates the feedback from the environment about the action by an agent. A reward vector r is dependent on the confidence vector of individual agent and thus it varies from agent to agent. A positive reward is obtained by a classifier when the classification result[1] matches the *actual_result* else it obtains a negative reward. The magnitude of reward is directly proportional to the probability of prediction by the classifier. Our model ultimately discovers rules to give appropriate Q-values based on the classifier's performance for a particular state vector.

DDQN Architecture. In DDQN, the major advantage is handling the overestimation of action values that takes place in DQN. One of the important principles of DDQN is the decoupling of action selection and action valuation components. This is achieved in our case by using two different neural networks. One neural network is used for computing current Q values while other neural network is used to compute target Q values. Here, back-propagation takes place in the current DQN and its weights are copied to the target DQN with delayed synchronization which helps stabilize the target Q values. This copying is done after regular intervals of a fixed number of epochs. In experimentation, we have used the epoch interval as 32 as it was found to give optimal results in most of the cases. The actions (raising appropriate alarms) are taken as per the current Q function, but the current Q_{new} values are estimated using the target Q function. This is done to avoid the moving target effect when doing gradient descent. A similar approach has been used by Manuel Lopez-Martin et al. in paper [10]. This method of delayed synchronization between two neural networks ensures the required decoupling and thereby handling the moving target effect of DQN.

Here, we present our algorithms for intrusion detection. Algorithm 1 shows the learning process executed by an agent. In Algorithm 1, *transition* comprises of *state_vector* and updated Q-values. We use memory M to store these transitions for prioritized experience replay concept of DQN using the sum tree implementation proposed by Schaul *et al.* [12]. The value of epoch interval, i.e., m is set to 32 in Algorithm 1.

Administrator network executes Algorithm 2 based on output of Algorithm 1 (refer step 13). In step 25 of Algorithm 1, the parameters s, r, w, and γ indicate *state_vector*, *reward_vector*, *weights* and *learning_rate* respectively. The parameter γ is taken as 0.01 (found by performing grid search on a logarithmic scale for values between 1 and 10^{-6}). The function of the administrator network is shown in Algorithm 2. It uses a voting system to identify the presence or absence of an attack.

Functional Task of Cloud Administrator: The cloud administrator runs Algorithm 2 where it monitors the activities of the cloud system constantly and detect its state. On receiving agent results, it checks for the intrusion and accordingly shares the actual result to the agent. It also identifies the location of the intrusion, including entry doors and target VMs.

[1] Each value of the classification vector denotes the classification result of a particular classifier.

Algorithm 1: Agent Logic

Initialize memory M;
Get the $feature\_set$ using the network parameters;
k=1;
for *each* $feature\_vector$ *in* $feature\_set$ **do**
  Input the $feature\_vector$ to different classifiers C_1,C_2...C_n and store the output as $confidence\_vector$;
  Run thresholding on $confidence\_vector$ and generate $classification\_vector$;
  Form the $state\_vector$ using $feature\_vector$ and $classification\_vector$;
  Input the $state\_vector$ to the Target Deep Q-network and store the output as Q'_value_vector;
  Again input the $state\_vector$ to the Current Deep Q-network and store the output as Q_value_vector;
  Run thresholding on Q_value_vector and generate $decision\_vector$;
  $action\_vector = decision\_vector \wedge classification\_vector$;
  $agent\_result = action\_vector$;
  Send $agent\_result$ along with $feature\_vector$ to the Administrator Network;
  Get the $actual\_result$ from the Cloud Administrator Network;
  Extend the $actual\_result$ to the length of the $classification\_vector$ by repeating the $actual\_result$ in every position;
  $sign\_vector = actual\_result \oplus classification\_vector$;
  Initialize $reward\_vector$;
  for *i in range(len(sign_vector))* **do**
    if *(sign_vector[i] == 0)* **then**
      $reward\_vector[i] = confidence\_vector[i]/2$;
    else
      $reward\_vector[i] = -1 \times (confidence\_vector[i]/2)$;
    end
  end
  Update Q values in current Deep Q Network using the following equation: $Q_{\text{new(current)}}(s, a, r, w) = r + \gamma * Q'_{\text{(target)}}(s, a, r, w)$;
  Store the transition in memory M;
  Update the weights (w) of the current DQN with the latest transition in memory M;
  if *(k == m)* **then**
    Copy the weights of current Deep Q Network into the Target Deep Q Network;
    $k = k + 1$;
  end
  for *j in range(n)* **do**
    Update current Deep Q-network weights (w) using prioritized learning based selection of transitions from M;
  end
end

Algorithm 2: Cloud Administrator Logic

Obtain $agent\_result$ and $feature\_vector$ from the host network;
for *each agent* **do**
  p = Count of digit 1 in $agent\_result$;
  k = Count of digit 0 in $agent\_result$;
  if $p \leq k$ **then**
    $status \leftarrow$ "$normal$";
  else
    $status \leftarrow$ "$attack$";
  end
  Obtain the $actual\_result$ from the cloud environment;
  Send the $actual\_result$ to the host network for reward calculation;
end

6 Robustness of the Model

In this section, we discuss the procedure for generating adversarial samples for evaluating the performance of the model against adversarial attacks. This experiment highlights our proposed model's robustness against the vulnerabilities of

intentional and small perturbations in the input data that can lead to misclassification. Subsect. 7.4 describes the experimental results that show the high durability of the model against a practical adversarial attack.

6.1 Adversarial Attack Synthesis

During intrusions, the attacker often wants to create adversarial inputs thoughtfully with minimum perturbations in order to escape any threat detection tool like IDS. Thus in our experimentation, we use an efficient and practical black-box adversarial attack as described in the paper [18]. It involves a two-step procedure for obtaining adversarial samples [19]:

1. **Creation of a Substitute Model:** The attacker aims to create a model F, which clones the target model's decision abilities. For this, they use results for synthetic inputs obtained by a Jacobian based heuristic method. It primarily includes five sub-steps including a) Collection of training dataset, b) Selection of substitute DDQN architecture, c) Labeling of substitute dataset, d) Training of substitute model, and e) Augmentation of the dataset using Jacobian-based method.
2. **Generation of Adversarial Samples:** The attacker uses the substitute model F obtained from step 1 to generate adversarial samples. Due to the transferability principle, the target model gets duped by the perturbed input sample and thus misclassify it. There are two prominent methods for this step, including (1) Fast Gradient Sign Method (FGSM) and (2) Jacobian-based Saliency Map Attack (JSMA) [16]. Although the FGSM method is capable of generating many adversarial samples in less computation time, their perturbations are high, making them susceptible to detection. In contrast, the JSMA method produces smaller perturbations, and it is also more suitable for misclassification attacks. Thus, in this paper, we use the JSMA method for generating the adversarial samples.

Jacobian-Based Saliency Map Attack (JSMA) [17]: It is a simple iterative method to introduce perturbation in the samples, resulting in the target model classifying incorrectly. It identifies the most critical feature in every iteration and adds noise to them. In our paper, we use a DDQN model to implement the stated adversarial method.

To implement step 1, we used a ten layered DDQN model and trained this model using the training samples of ISOT-CID dataset (refer Table 1) to obtain the substitute model. Later, we augment data as per the procedure described in the paper [18] to perform the sub-step e) i.e, augmentation of data. In addition, for step 2, we implement a five layered deep sequential neural network with same size input-output labels as the number of features in the dataset. The *generate_np* function of the *Saliency_Map_Method* class in the *cleverhans.attacks* package of *cleverhans* [20] library in Python language implements the above adversarial method. We iteratively fed unperturbed feature vectors and $jsma\_parameter$ as input to this function as it returns an adversarial sample as output. Algorithm 3 briefly discusses the related steps to obtain adversarial dataset from the clean dataset.

Algorithm 3: Practical black box adversarial sample creation

```
Let V = ∅;
Define the Oracle L as the IDS model;
∀ classes in the dataset, collect samples from each of them into V;
Select and initialize the architecture for substitute model S;
Fix the number of iterations δ_max;
for δ in range 1 to δ_max do
    ∀ v ∈ V, label it using the Oracle L denoted by T;
    Convert the labels to binary form i.e benign (0) label remains same but any
      non-benign label (1 to number of classes) becomes 1;
    Train and update the parameters θ_S of the substitute model S with (V, T);
    Carry out Jacobian based data augmentation to the set V;
end
Craft adversarial samples either by using Saliency Map Attack (JSMA);
```

7 Experimental Results and Discussion

We implemented the model on the Google Colaboratory[2] and used Keras and Tensorflow library for building the DDQN model. We analyzed the performance of the model using two datasets, including the ISOT-CID and NSL-KDD. The ISOT-CID dataset is the first publicly available cloud intrusion dataset consisting of raw data collected at various components of a real-world cloud infrastructure hosted on the OpenStack platform. On the other hand, NSL-KDD is a renowned benchmark dataset for IDS, derived from a conventional network environment. The experimental results include three standard machine learning performance metrics, i.e., FPR (False Positive Rate), ACC (Accuracy), and AUC (Area under ROC Curve).

7.1 Classifiers in Model

We obtain the performance on the ISOT-CID and NSL-KDD dataset of five best and most advanced classifiers for IDS, namely, AdaBoost (ADB), Gaussian Naive Bayes (GNB), K-Nearest Neighbours (KNN), Quadratic Discriminant Analysis (QDA), and Random Forest (RF). Table 1 and Table 2 show the distribution of labeled data of the training and the testing phase that is used for evaluation in NSL-KDD and ISOT-CID dataset respectively. Later, we group these classifiers into Low-FPR classifiers and High-Accuracy classifiers. We then integrate some sets of suitable combinations of these classifiers in our model and conduct evaluation. Finally, we select the most suitable combination that ensures the best balance between high accuracy and low FPR. The next section presents the evaluation results of the individual classifiers.

7.2 Evaluation of Individual Classifiers on NSL-KDD and ISOT Dataset

Table 3 presents the performance of individual classifiers on the NSL-KDD and ISOT-CID dataset. For the ISOT-CID dataset, Table 3 shows that RF, ADB, and KNN obtain better accuracy than other classifiers, and we group them as

[2] An online cloud-based platform specially designed for ML and deep learning applications based on the Jupyter Notebook framework.

High-Accuracy classifiers. On the other hand, the classifiers like QDA and GNB give relatively lower FPR values, and we group them as a Low-FPR classifier. Further from Table 3, we obtain the same group of High-Accuracy and Low-FPR classifiers for the NSL-KDD dataset. Next, we create suitable combinations by choosing appropriate classifiers from each group. We employ such combinations in our model and obtain the corresponding evaluation matrices.

Table 3. Individual classifier performance on NSL-KDD and ISOT-CID Dataset

Classifier	NSL-KDD			ISOT-CID		
	Accuracy	FPR	Auc	Accuracy	FPR	AUC
RF	77.95	2.66	0.8054	94.95	5.66	0.792
ADB	78.65	7.009	0.7763	93.65	6.009	0.705
GNB	66	1.688	0.705	85.51	1.78	0.672
KNN	75.67	2.801	0.8087	91.67	4.801	0.7
QDA	69.71	1.59	0.7299	87.71	1.6	0.709

7.3 Evaluation of Proposed IDS Having Combination of Classifiers

The output layer of the current DQN (refer Fig. 2) contains Q-values for a given feature vector. We observed that the accuracy of Low-Accuracy classifiers (including GNB and QDA) can be improved by varying the threshold on Q-values. To get the optimal accuracy, we have tried several threshold values on Q-values corresponding to these classifier in the final output layer. On the other hand, we fix the threshold on Q-values for the other High-Accuracy classifiers to a constant value of 0.5 as it is found to produce the optimal results during experimentation.

Table 4. Performance of our system with classifiers, on NSL-KDD, **C1:** *RF, ADB, GNB* **C2:** *RF, ADB, QDA* **C3:** *RF, KNN, ADB, QDA* **C4:** *RF, KNN, ADB, GNB*

Threshold	C1		C2		C3		C4	
	Accuracy	FPR	Accuracy	FPR	Accuracy	FPR	Accuracy	FPR
0.5	78.87%	1.44%	77.1%	1.96%	79.16%	1.6%	79.16%	1.1%
0.2	78.89%	1.47%	83.08%	1.6%	83.24%	1.89%	80.04%	1.17%
0.7	82.88%	1.81%	83.16%	1.6%	83.25%	1.96%	83.40%	1.44%
0.8	83.11%	1.80%	83.24%	1.64%	83.32%	2.00%	**83.40** %	**1.48**%

Table 5. Performance of our system with classifiers on ISOT-CID, **C1:** *RF, ADB, GNB* **C2:** *RF, ADB, QDA* **C3:** *RF, KNN, ADB, QDA* **C4:** *RF, KNN, ADB, GNB*

Threshold	C1		C2		C3		C4	
	Accuracy	FPR	Accuracy	FPR	Accuracy	FPR	Accuracy	FPR
0.5	89.17%	4.43%	90.6%	4.06%	92.2%	3.86%	91.5%	3.96
0.6	91.23%	3.42%	92.7%	2.71%	93.23%	2.42	93.06%	2.53
0.7	93.5%	2.61%	95.87%	2.05%	**96.87%**	**1.57%**	96.12%	1.81%
0.8	93.2%	2.59%	95.17%	2.5%	95.6%	1.59%	95.3%	1.7%

We choose a combination of three classifiers that includes two classifiers from High-Accuracy class and one from Low-FPR classifiers. This selection strategy is found to give optimal performance in the direction of reducing FPR and increasing accuracy. Two combinations can be formed including C1 (GNB, ADB, RF) and C2 (QDA, ADB, RF). To further improve the accuracy, we included an additional High-Accuracy classifier (KNN) with each combination of C1 and C2 to create combination C3 and C4. Next, we evaluated the model against combinations with four classifiers. Table 4 and Table 5 present the evaluation results on varying the Q-threshold values of the model using these four combinations of classifiers on NSL-KDD and ISOT-CID dataset respectively. The outcomes show that the combination C3 (KNN, QDA, ADB, RF) obtains the optimal balance (i.e., the accuracy of 96.87% and FPR of 1.57%) at the Q-threshold value of 0.7 for ISOT-CID dataset. The combination C4 (KNN, GNB, ADB, RF) gives the best performance (i.e., the accuracy of 83.40% and FPR of 1.48%) for the NSL-KDD dataset at the Q-threshold value of 0.8. The evaluation results show that our models' performance improves marginally after addition of KNN classifier. Thus, we use combination C3 and C4 for further experimentation on ISOT-CID and NSL-KDD dataset respectively as they give the highest accuracy and least FPR.

The experimentation shows that our proposed model obtains significantly better evaluation results than individual classifiers. Further, we evaluated the DQN model described in the paper [25] on both the datasets to compare with the DDQN model that we have proposed in this paper.

7.4 Experimental Analysis of Proposed IDS on Black-Box Adversarial Attack

We emulated an adversarial attack on the proposed DDQN based model. Adversarial attacks on the machine and deep learning models is a widespread problem. The design of such adversarial models to test the robustness of designed models is a thriving research topic. However, very little work has been done to conduct adversarial analysis in the context of cloud IDS. We evaluated the robustness of our model based on the practical black-box adversarial attack proposed in [27]. We observed a substantial change in the feature vector after applying the adversarial model, i.e., nearly one-third of the feature values were found to be perturbed after applying the adversarial method on the datasets. Then we evaluated the performance of our model using the black box implementation. The Tables 6 show the test set evaluation of our model on the ISOT-CID and NSL-KDD dataset respectively. As can be seen from the Tables, the average accuracy fell down by about 3.5% and the FPR value increased by 1.5 times in case of NSL-KDD dataset. On the other hand, in the case of ISOT-CID dataset, there was a marginal decrease in the performance with nearly 4% fall in accuracy and 2% rise in FPR.

Table 6. Measuring robustness of our system against adversarial attacks based on NSL-KDD and ISOT-CID Data set

Action	NSL-KDD			ISOT-CID		
	Accuracy	FPR	Auc	Accuracy	FPR	AUC
Model before adversarial attack	83.40	1.48	0.8432	96.87	1.5	0.861
Model after adversarial attack	79.77	3.7	0.7980	92.17	3.3	0.8112

7.5 Performance of Model on Continuously Changing Attack Types

In this section, we evaluate our model performance when targeted by a continuously changing attack types. The key highlight of this experiment is to understand the adaptive and dynamic behaviour of the model towards novel attacks. Here, the model faces a zero-day attack scenario, and it is expected to adapt and detect the newer attack type with improving performance as the number of such attack samples increases. We have emulated this by deriving a new dataset using the different attack types of ISOT-CID dataset. ISOT-CID dataset collects the logs in a span of eighteen days where each day has new attack types. However, the majority of the volume belongs to first six days and each of these days has new attack type. We leveraged this property of the data set to find out the efficacy of model regarding adaptiveness towards novel attacks. We conducted experiments which involved making our model face new attacks constantly and noting down its accuracy, FPR and AUC (refer Table 7). The performance of any i^{th} day is obtained by training the model to dataset belonging from day 1 to day $i-1$ and evaluating it on the i^{th} day dataset. This is similar to situation in the real-world where model would face novel threats on almost every new day and its prediction would depend on the learning from past. As it can be seen from Table 7, our model performs fairly well even if it is trained for a few days and tested on unknown attack types. The consistent improvements in metrics like Accuracy, FPR and AUC, in subsequent days, suggest high adaptability and robustness in long term use.

Table 7. Performance of model on daily changing attack type

Day	Attack type	Number of samples	ACC	FPR	AUC
1	DTA and UCM	24622	–	–	–
2	NS	66124	82.23%	2.81%	0.8211
3	SQLI, CSS, PT, S-DOS	36517	87.16%	0.88%	0.8801
4	BFLA (failed)	43489	89.11%	0.16%	0.9030
5	UCM, DNSADOS, HTTPFDOS	48716	92.80%	0.10%	0.9381

ACC: Accuracy, DTM: Dictionary Traversal Attack, UCM: Unauthorized Crypto-mining, NS: Network scanning, SQLI: SQL Injection, CSS: Cross-site Scripting(XSS), PT: Path Traversal, S-DOS: Slowloris DOS, BFLA: Brute Force login attack, UCM: Unauthorized Crypto-mining, DNSADOS: DNS amplification DOS, HTTPFDOS: HTTP flood DOS

7.6 Comparison of Proposed IDS with State-of-the-Art Works

We compared our proposed model to existing intrusion detection models using NSL-KDD, and ISOT-CID datasets. We evaluated the performance with parameters such as Design, Adaptiveness, Dataset, Robustness, suitability for Cloud platform, Accuracy, FPR. The comparative study is presented in Table 8.

Most state-of-the-art IDS models [3,21–25] use deep learning or reinforcement learning. In contrast, our proposed IDS implements a DDQN logic, which is an advanced deep reinforcement learning algorithm. Although the model described in paper [10] uses DDQN based IDS, the architecture of the model is suited for conventional networks and not for the cloud environment. Also, in contrast to [10], we evaluate our proposed IDS using ISOT-CID (which is a practical cloud-specific intrusion dataset) along with NSL-KDD. The paper [3] presented a cloud IDS and did an evaluation using the ISOT-CID dataset. However, the model in paper [3] lacks adaptiveness towards novel attacks and provides no study about the impact of the adversarial attack. Both of these two parameters are an essential security requirement for the current cloud environment. In contrast to [3], our work is adaptive, robust against adversarial attacks, and maintains a good balance between accuracy and FPR.

Table 8. Comparison of performance: our model vs. state-of-the-art works

Reference	Model	Adaptive	Robustness	Cloud suitability	Dataset	Accuracy (%)	FPR (%)
[21]	RNN	✗	✗	✗	NSL-KDD	83.28	3.06
[10]	DDQN	✓	✗	✗	NSL-KDD	89.78	–
[22]	DNN	✗	✓	✗	NSL-KDD	78.5	6.94
[23]	AE-RL	✗	✓	✗	NSL-KDD	80.16	–
[25]	DQN	✓	✓	✗	NSL-KDD	81.80	2.6
[24]	DL H2O	✓	✗	✓	NSL-KDD	83	–
[3]	OMSCA	✗	✗	✓	ISOT-CID	96.93	7.56
Our model	DDQN	✓	✓	✓	NSL-KDD	83.40	1.48
					ISOT-CID	96.87	1.5

* RNN: Recurrent Neural Network, DDQN: Double Deep-Q-Network, DNN: Deep Neural Network, DQN: Deep-Q-Network, GAN: Generative Adversarial Network, AE-RL: Adversarial Environment Reinforcement Learning, DL H2O: Deep Learning H2O, OMSCA: Online Multivariate Statistical Change Analysis

8 Conclusion

In this paper, we present an advanced deep reinforcement learning based cloud intrusion detection system that provides high accuracy and low FPR when evaluated using ISOT-CID and NSL-KDD dataset. Our models aim to meet the real-world constraints of limited processing resources and adaptability towards novel attacks and changing attack patterns. For this, we did experimentation on the dataset with a flow-based technique that is computationally lighter. We introduced DDQN based IDS to handle the overestimation of action values in the Deep Q Learning-based model. The Experimental results show significant improvements in evaluation metrics as compared to individual classifiers. In

Sect. 7.4, we verified the robustness of the proposed IDS against a practical black-box adversarial attack. Further, Sect. 7.5 shows our system's ability to handle newer attack types (even with very little training data). Overall, the evaluation of the model shows a better performance compared to state-of-the-art works and its effectiveness for deploying in the cloud platforms. In the future, we intend for the deployment, implementation and evaluation of the proposed IDS on a practical cloud infrastructure.

References

1. Li, Z., Sun, W., Wang, L.: A neural network based distributed intrusion detection system on cloud platform. In: 2012 IEEE 2nd International Conference on Cloud Computing and Intelligence Systems, Hangzhou, pp. 75–79 (2012). https://doi.org/10.1109/CCIS.2012.6664371
2. Sethi, K., Kumar, R., Prajapati, N., Bera, P.: Deep reinforcement learning based intrusion detection system for cloud infrastructure. In: 2020 International Conference on COMmunication Systems & NETworkS (COMSNETS), Bengaluru, India, pp. 1–6 (2020). https://doi.org/10.1109/COMSNETS48256.2020.9027452
3. Aldribi, A., Traoré, I., Moa, B., Nwamuo, O.: Hypervisor-based cloud intrusion detection through online multivariate statistical change tracking. Comput. Secur. **88**, 101646 (2020). https://doi.org/10.1016/j.cose.2019.101646. ISSN 0167-4048
4. Kholidy, H.A., Baiardi, F.: CIDD: a cloud intrusion detection dataset for cloud computing and masquerade attacks. In: 2012 Ninth International Conference on Information Technology - New Generations, Las Vegas, NV, pp. 397–402 (2012). https://doi.org/10.1109/ITNG.2012.97
5. Parampottupadam, S., Moldovann, A.: Cloud-based real-time network intrusion detection using deep learning. In: 2018 International Conference on Cyber Security and Protection of Digital Services (Cyber Security), Glasgow, pp. 1–8 (2018). https://doi.org/10.1109/CyberSecPODS.2018.8560674
6. Patil, R., Dudeja, H., Modi, C.: Designing an efficient security framework for detecting intrusions in virtual network of cloud computing. Comput. Secur. **85**, 402–422 (2019). https://doi.org/10.1016/j.cose.2019.05.016. ISSN 0167–4048
7. van Hasselt, H., Guez, A., Silver, D.: Deep reinforcement learning with double Q-learning. arXiv:1509.06461 (2015)
8. Aldribi, A., Traore, I., Moa, B.: Data sources and datasets for cloud intrusion detection modeling and evaluation. In: Mishra, B.S.P., Das, H., Dehuri, S., Jagadev, A.K. (eds.) Cloud Computing for Optimization: Foundations, Applications, and Challenges. SBD, vol. 39, pp. 333–366. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-73676-1_13
9. Tranalyzer documentation (2020). https://tranalyzer.com/documentation
10. Lopez-Martin, M., Carro, B., Sanchez-Esguevillas, A.: Application of deep reinforcement learning to intrusion detection for supervised problems. Expert Syst. Appl. **141**, 112963 (2020). https://doi.org/10.1016/j.eswa.2019.112963. ISSN 0957-4174
11. Gogoi, P., Bhuyan, M.H., Bhattacharyya, D.K., Kalita, J.K.: Packet and flow based network intrusion dataset. In: Parashar, M., Kaushik, D., Rana, O.F., Samtaney, R., Yang, Y., Zomaya, A. (eds.) IC3 2012. CCIS, vol. 306, pp. 322–334. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32129-0_34

12. Schaul, T., Quan, J., Antonoglou, I., Silver, D.: Prioritized experience replay. arXiv:1511.05952 (2015)
13. Meena, G., Choudhary, R.R.: A review paper on IDS classification using KDD 99 and NSL KDD dataset in WEKA. In: International Conference on Computer Communications and Electronics, Jaipur, pp. 553–558 (2017)
14. Shone, N., Ngoc, T.N., Phai, V.D., Shi, Q.: A deep learning approach to network intrusion detection. IEEE Trans. Emerg. Top. Comput. Intell. **2**(1), 41–50 (2018). https://doi.org/10.1109/TETCI.2017.2772792
15. scikit-learn user guide, scikit-learn Developers, Release 0.21.dev0 [User Guide] (2015). http://scikit-learn.org/dev/_downloads/scikit-learn-docs.pdf
16. Goodfellow, I.J., Papernot, N., McDaniel, P.D.: Cleverhans v0.1: an adversarial machine learning library. CoRR, abs/1610.00768 (2016)
17. Papernot, N., McDaniel, P., Jhay, S., Fredriksonz, M., Berkay Celik, Z., Swamix, A.: The limitations of deep learning in adversarial setting. In: 1st IEEE European Symposium on Security & Privacy, Saarbrucken, Germany (2016). https://doi.org/10.1109/EuroSP.2016.36
18. Papernot, N., McDaniel, P., Goodfellow, I., Jha, S., Berkay Celik, Z., Swami, A.: Practical black-box attacks against machine learning. arXiv preprint arXiv:1602.02697v4 (2016)
19. Biggio, B., et al.: Security evaluation of support vector machines in adversarial environments. In: Ma, Y., Guo, G. (eds.) Support Vector Machines Applications, pp. 105–153. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-02300-7_4
20. Google Inc.: OpenAI and Pennsylvania State University, a repository for *cleverhans* library [Github Repository] (2016). https://github.com/tensorflow/cleverhans
21. Yin, C., Zhu, Y., Fei, J., He, X.: A deep learning approach for intrusion detection using recurrent neural networks. IEEE Access **5**, 21954–21961 (2017)
22. Vinayakumar, R., Alazab, M., Soman, K.P., Poornachandran, P., Al-Nemrat, A., Venkatraman, S.: Deep learning approach for intelligent intrusion detection system. IEEE Access **7**, 41525–41550 (2019). https://doi.org/10.1109/ACCESS.2019.2895334
23. Caminero, G., Lopez-Marti, M., Carro, B.: Adversarial environment reinforcement learning algorithm for intrusion detection. Comput. Netw. **159**, 96–109 (2019)
24. Parampottupadam, S., Moldovann, A.: Cloud-based real-time network intrusion detection using deep learning. In: 2018 International Conference on Cyber Security and Protection of Digital Services (Cyber Security), Glasgow, pp. 1–8 (2018)
25. Sethi, K., Rupesh, S., Kumar, R., Bera, P.L., Madhav, V.: A context-aware robust intrusion detection system: a reinforcement learning-based approach. Int. J. Inf. Secur. (2019). https://doi.org/10.1007/s10207-019-00482-7
26. Pedregosa, F., Varoquaux, G., Gramfort, A., et al.: Scikit-learn: machine learning in Python. J. Mach. Learn. Res. **12**, 2825–2830 (2011)
27. Papernot, N., McDaniel, P., Jhay, S., Fredriksonz, M., Berkay Celik, Z., Swamix, A.: The limitations of deep learning in adversarial setting. In: 1st IEEE European Symposium on Security and Privacy, Saarbrucken, Germany (2016)
28. Wang, Z.: Deep learning-based intrusion detection with adversaries. IEEE Access **6**, 38367–38384 (2018)
29. Szegedy, C., et al.: Intriguing properties of neural networks (2013). arXiv:1312
30. Huber, P.J.: Robust estimation of a location parameter. Ann. Math. Stat. **35**(1), 73–101 (1964). JSTOR. www.jstor.org/stable/2238020. Accessed 6 July 2020
31. Mnih, V., Kavukcuoglu, K., Silver, D., et al.: Human-level control through deep reinforcement learning. Nature **518**, 529–533 (2015)