Reinforcement Learning for Autonomous Defence in Software-Defined Networking

Yi Han[1(✉)], Benjamin I. P. Rubinstein[1], Tamas Abraham[2], Tansu Alpcan[1], Olivier De Vel[2], Sarah Erfani[1], David Hubczenko[2], Christopher Leckie[1], and Paul Montague[2]

[1] School of Computing and Information Systems, The University of Melbourne, Parkville, Australia
{yi.han,benjamin.rubinstein,tansu.alpcan,sarah.erfani, caleckie}@unimelb.edu.au

[2] Defence Science and Technology Group, Edinburgh, Australia
{tamas.abraham,olivier.devel,david.hubczenko, paul.montague}@dst.defence.gov.au

Abstract. Despite the successful application of machine learning (ML) in a wide range of domains, adaptability—the very property that makes machine learning desirable—can be exploited by adversaries to contaminate training and evade classification. In this paper, we investigate the feasibility of applying a specific class of machine learning algorithms, namely, reinforcement learning (RL) algorithms, for autonomous cyber defence in software-defined networking (SDN). In particular, we focus on how an RL agent reacts towards different forms of causative attacks that poison its training process, including indiscriminate and targeted, white-box and black-box attacks. In addition, we also study the impact of the attack timing, and explore potential countermeasures such as adversarial training.

Keywords: Adversarial reinforcement learning
Software-defined networking · Cyber security · Adversarial training

1 Introduction

Machine learning has enjoyed substantial impact on a wide range of applications, from cyber-security (*e.g.*, network security operations, malware analysis) to autonomous systems (*e.g.*, decision-making and control systems, computer vision). Despite the many successes, the very property that makes machine learning desirable—adaptability—is a vulnerability to be exploited by an economic competitor or state-sponsored attacker. Attackers who are aware of the ML techniques being deployed can contaminate the training data to manipulate a learned ML classifier in order to evade subsequent classification, or can manipulate the metadata upon which the ML algorithms make their decisions and exploit identified weaknesses in these algorithm—so called Adversarial Machine Learning [6,11,27].

L. Bushnell et al. (Eds.): GameSec 2018, LNCS 11199, pp. 145–165, 2018.
https://doi.org/10.1007/978-3-030-01554-1_9

This paper focuses on a specific class of ML algorithms, namely, reinforcement learning (RL) algorithms, and investigates the feasibility of applying RL for autonomous defence in computer networks [7], *i.e.,* the ability to "fight through" a contested environment—in particular adversarial machine learning attacks—and ensure critical services (*e.g.*, email servers, file servers, etc.) are preserved to the fullest extent possible.

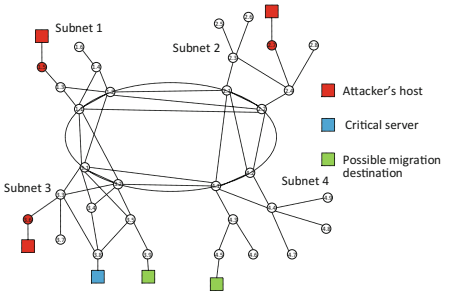


Fig. 1. An example network setup. The attacker propagates through the network to compromise the critical server, while the defender applies RL to prevent the critical server from compromise and to preserve as many nodes as possible. (Color figure online)

For example, consider a network as shown in Fig. 1 that consists of 32 nodes, one (node 3.8) of whom connects to the critical server, two (nodes 3.9 and 4.5) connect to potential migration destinations, and three (nodes 1.5, 2.7 and 3.6) connect to the attacker's hosts. The attacker aims to propagate through the network, and compromise the critical server. We aim to prevent this and preserve as many nodes as possible through the following RL approach:

- We first train two types of RL agents: Double Deep Q-Networks (DDQN) [24] and Asynchronous Advantage Actor-Critic (A3C) [38]. The agents observe network states, and select actions such as "isolate", "patch", "reconnect", and "migrate". The agents gradually optimise their actions for different network states, based on the received rewards for maintaining critical services, costs incurred when shutting down non-critical services or migrating critical services.
- Once a working agent is obtained, we then investigate different ways by which the attacker may poison the training process of the RL agent. For example, the attacker can falsify part of the reward signals, or manipulate the states of certain nodes, in order to trick the agent to take non-optimal actions, resulting in either the critical server being compromised, or significantly fewer nodes being preserved. Both indiscriminate and targeted, white-box and black-box attacks are studied.
- We also explore possible countermeasures—*e.g.*, adversarial training—that make the training less vulnerable to causative/poisoning attacks.

– To make use of the developed capacity for autonomous cyber-security operations, we build our experimental platform around software-defined networking (SDN) [2], a next-generation tool chain for centralising and abstracting control of reconfigurable networks. The SDN controller provides a centralised view of the whole network, and is directly programmable. As a result, it is very flexible for managing and reconfiguring various types of network resources. Therefore, in our experiments the RL agents obtain all network information and perform different network operations via the SDN controller.
– Our results demonstrate that RL agents can successfully identify the optimal actions to protect the critical server, by isolating as few compromised nodes as possible. In addition, even though the causative attacks can cause the agent to make incorrect decisions, adversarial training shows great potential for mitigating the impact.

The remainder of the paper is organised as follows: Sect. 2 briefly introduces the fundamental concepts in RL and SDN; Sect. 3 defines the research problem; Sect. 4 introduces in detail the different forms of proposed attacks against RL; Sect. 5 presents the experimental results on applying RL for autonomous defence in SDN, and the impact of those causative attacks; Sect. 6 overviews previous work on adversarial machine learning (including attacks against reinforcement learning) and existing countermeasures; Sect. 7 concludes the paper, and offers directions for future work.

2 Preliminaries

Before defining the research problems investigated in this paper, we first briefly introduce the basic concepts in reinforcement learning and software-defined networking.

2.1 Reinforcement Learning

In a typical reinforcement learning setting [56], an agent repeatedly interacts with the environment: at each time step t, the agent (1) observes a state s_t of the environment; (2) chooses an action a_t based on its policy π—a mapping from the observed states to the actions to be taken; and (3) receives a reward r_t and observes next state s_{t+1}. This process continues until a terminal state is reached, and then a new episode restarts from a certain initial state. The agent's objective is to maximise its discounted cumulative rewards over the long run: $R_t = \sum_{\tau=t}^{\infty} \gamma^{\tau-t} r_\tau$, where $\gamma \in (0, 1]$ is the discount factor that controls the trade-off between short-term and long-term rewards.

Under a given policy π, the value of taking action a in state s is defined as: $Q^\pi(s, a) = \mathbb{E}[R_t | s_t = s, a_t = a, \pi]$. Similarly, the value of state s is defined as: $V^\pi(s) = \mathbb{E}[R_t | s_t = s, \pi]$. In this paper, we mainly focus on two widely cited RL algorithms: Double Deep Q-Networks (DDQN) [24] and Asynchronous Advantage Actor-Critic (A3C) [38].

Q-Learning. Q-learning [56] approaches the above problem by estimating the optimal action value function $Q^*(s,a) = \max_\pi Q^\pi(s,a)$. Specifically, it uses the Bellman equation $Q^*(s,a) = \mathbb{E}_{s'}[r + \gamma \max_{a'} Q^*(s',a')]$ to update the value iteratively. In practice, Q-learning is commonly implemented by function approximation with parameters θ: $Q^*(s,a) \approx Q(s,a;\theta)$. At each training iteration i, the loss function is defined as: $L_i(\theta_i) = \mathbb{E}[(r + \gamma \max_{a'} Q(s',a';\theta_{i-1}) - Q(s,a;\theta_i))^2]$.

Deep Q-Networks (DQN). Classic Q-learning networks suffer from a number of drawbacks, including (1) the *i.i.d.* (independent and identically distributed) requirement of the training data being violated as consecutive observations are correlated, (2) unstable target function when calculating Temporal Difference (TD) errors, and (3) different scales of rewards. Deep Q networks (DQN) [39] overcome these issues by (1) introducing experience replay, (2) using a target network that fixes its parameters (θ^-) and only updates at regular intervals, and (3) clipping the rewards to the range of $[-1, 1]$. The loss function for DQN becomes: $L_i(\theta_i) = \mathbb{E}[(r + \gamma \max_{a'} Q(s',a';\theta_i^-) - Q(s,a;\theta_i))^2]$.

Double DQN (DDQN). To further solve the problem of value overestimation, Hasselt *et al.* [24] generalise the Double Q-learning algorithm [23] proposed in the tabular setting, and propose Double DQN (DDQN) that separates action selection and action evaluation, *i.e.,* one DQN is used to determine the maximising action and a second one is used to estimate its value. Therefore, the loss function is: $L_i(\theta_i) = \mathbb{E}[(r + \gamma Q(s', \arg\max_{a'} Q(s',a';\theta_i);\theta_i^-) - Q(s,a;\theta_i))^2]$.

Prioritised Experience Replay. Experience replay keeps a buffer of past experiences, and for each training iteration, it samples uniformly a batch of experiences from the buffer. Prioritised experience replay [53] assigns higher sampling probability to transitions that do not fit well with the current estimation of the Q function. For DDQN, the error of an experience is defined as $|r + \gamma Q(s', \arg\max_{a'} Q(s',a';\theta);\theta^-) - Q(s,a;\theta)|$.

Asynchronous Advantage Actor-Critic (A3C). Mnih *et al.* [38] propose an asynchronous variant of the classical actor-critic algorithm, which estimates both the state value function $V(s;\theta_v)$ and a policy $\pi(a|s;\theta_p)$. Specifically, the A3C algorithm uses multiple threads to explore different parts of the state space simultaneously, and updates the global network in an asynchronous way. In addition, instead of using discounted returns to determine whether an action is good, A3C estimates the *advantage function* so that it can better focus on where the predictions are lacking.

2.2 Software-Defined Networking

In order to better serve today's dynamic and high-bandwidth applications, a new architecture called Software-Defined Networking (SDN) has emerged [2]. There are three layers in the SDN architecture: (1) the application layer includes applications that deliver services. These applications communicate their network requirements to the controller via northbound APIs; (2) the SDN controller translates these requirements into low-level controls, and sends them through

southbound APIs to the infrastructure layer; (3) the infrastructure layer comprises network switches that control forwarding and data processing. One major advantage of SDN is that it decouples network control and forwarding functions, rendering the controller directly programmable. As a result, network resources can be conveniently managed, configured and optimised using standardised protocols. There have been a number of proprietary and open-source SDN controller software platforms. In this paper, we have opted to use OpenDaylight [35], which is the largest open-source SDN controller today and which is updated regularly.

3 Problem Statement

In this paper, we seek to answer the question: Can reinforcement learning be used for autonomous defence in SDN? We start with a scenario that does not consider the attacker poisoning the training process, and then investigate the impact of adversarial reinforcement learning. While we also briefly discuss potential countermeasures, we largely leave defences to future work.

3.1 Reinforcement Learning Powered Autonomous Defence in SDN

Consider a network of N nodes (*e.g.*, Fig. 1), $H = \{h_1, h_2, ..., h_N\}$, where $H_C \subset H$ is the set of critical servers to be protected (blue nodes in Fig. 1), $H_M \subset H$ is the set of possible migration destinations for $h \in H_C$ (green nodes), and $H_A \subset H$ is the set of nodes that have initially been compromised (red nodes). The attacker aims to propagate through the network, and penetrate the mission critical servers, while the defender/SDN controller monitors the system state, and takes appropriate actions in order to preserve the critical servers and as many non-critical nodes as possible.

Reflecting suggestions from past work, we consider a defender adopting RL. In this paper, we start with a simplified version, and make the following assumptions (Sect. 7 explains how they may be replaced): (1) each node (or link) only has two states: compromised/uncompromised (or on/off); (2) both the defender and the attacker know the complete network topology; (3) the defender has in place a detection system that can achieve a detection rate of 90%, with no false alarms (before the causative attacks); (4) the attacker needs to compromise all nodes on the path (*i.e.,* cannot "hop over" nodes). Given these assumptions, in each step the defender:

1. Observes the state of the network—whether a node is compromised, and whether a link is switched on/off, *e.g.*, there are 32 nodes and 48 links in Fig. 1, so one state is an array of 80 $0s/1s$, where 0 means the node is uncompromised or the link is switched off, and 1 means the node is compromised or the link is switched on;
2. Takes an action that may include: (i) isolating and patching a node; (ii) reconnecting a node and its links; (iii) migrating the critical server and selecting the destination; and (iv) taking no action. Note that, in this scenario, the defender can only take one type of action at a time, and if they decide to isolate/reconnect, only one node can be isolated/reconnected at a time;

Table 1. Problem description: RL powered autonomous defence in SDN

	Defender	Attacker
State	(1) Whether each node is compromised; (2) Whether each link is turned on/off.	
Actions	(1) Isolate and patch a node; (2) Reconnect a node and its links; (3) Migrate the critical server and select the destination; (4) Take no action	Compromise a node that satisfies certain conditions, *e.g.*, the node (1) is closer to the "backbone" network; (2) is in the backbone network; or (3) in the target subnet
Goals	(1) Preserve the critical servers; (2) Keep as many nodes uncompromised and reachable from the critical servers as possible.	Compromise the critical servers

3. Receives a reward based on (i) whether the critical servers are compromised; (ii) the number of nodes reachable from the critical servers; (iii) the number of compromised nodes; (iv) migration cost; and (v) whether the action is valid, *e.g.*, it is invalid to isolate a node that has already been isolated.

Meanwhile, the attacker carefully chooses the nodes to compromise. For example, in the setting of Fig. 1, they infect a node only if it (1) is closer to the "backbone" network (nodes on the dashed circle); (2) is in the backbone network; or (3) is in the target subnet. Table 1 summarises this problem setting.

3.2 Causative Attacks Against RL Powered Autonomous Defence System

As an online system, the autonomous defence system continues gathering new statistics, and keeps training/updating its model. Therefore, it is necessary and crucial to analyse the impact of an adversarial environment, where malicious users can manage to falsify either the rewards received by the agent, or the states of certain nodes. In other words, this is a form of causative attack that poisons the training process, in order for the tampered model to take sub-optimal actions. In this paper, we investigate the two forms of attacks below.

1. **Flipping reward signs.** Suppose that without any attack, the agent would learn the following experience (s, a, s', r), where s is the current system state, a is the action taken by the agent, s' is the new state, and r is the reward. In our scenario, we permit the attacker to flip the sign of a certain number of rewards (*e.g.*, 5% of all experiences), and aim to maximise the loss function of the RL agent. This is an extreme case of the corrupted reward channel problem [19], where the reward may be corrupted due to sensor errors, hijacks, etc.
2. **Manipulating states.** Again, consider the case where the agent learns an experience (s, a, s', r) without any attack. Furthermore, when the system

reaches state s', the agent takes the next optimal action a'. The attacker is then allowed to introduce one false positive (FP) and one false negative (FN) reading in s', *i.e.,* one uncompromised/compromised node is reported as compromised/uncompromised to the defender. As a result, instead of learning (s, a, s', r), the agent ends up observing $(s, a, s' + \delta, r')$ (where δ represents the FP and FN readings), and consequently may not take a' in the next step.

4 Attack Mechanisms

This section explains in detail the mechanisms of the attacks introduced above.

4.1 Attack I: Maximise Loss Function by Flipping Reward Signs

Recall that the DDQN agent aims to minimise the loss function: $L_i(\theta_i) = \mathbb{E}[(r + \gamma Q(s', \arg\max_{a'} Q(s', a'; \theta_i); \theta_i^-) - Q(s, a; \theta_i))^2]$. In the i^{th} training iteration, θ_i is updated according to the gradient of $\partial L_i / \partial \theta_i$. The main idea for the first form of attack is to falsify certain rewards based on $\partial L_i / \partial r$, in order to maximise the loss L_i.

Specifically, after the agent samples a batch of experiences for training, we calculate the gradient of $\partial L_i / \partial r$ for each of them, and flip the sign of experience with the largest absolute value of the gradient $|\partial L_i / \partial r|$ that satisfies $r \cdot \partial L_i / \partial r < 0$ (if $r \cdot \partial L_i / \partial r > 0$ flipping the sign decreases the loss function).

4.2 Attack II: Prevent Agent from Taking Optimal/Specific Actions by Manipulating States

Our experimental results show that the above form of attack is indeed effective in increasing the agent's loss function. However, it only delays the agent from learning the optimal actions. Therefore, the second form of attack directly targets the value function Q (against DDQN agent) or the policy π (against A3C agent).

1. **Indiscriminate attacks.** For each untampered experience (s, a, s', r), indiscriminate attacks falsify the states of two nodes in the new state s', in order to prevent the agent from taking the next optimal action a' that has been learned so far (which may be different from the final optimal action for the given state), *i.e.,* against DDQN agent the attacks minimise $\max_{a'} Q(s' + \delta, a')$, while against A3C agent the attacks minimise $\max_{a'} \pi(a'|s' + \delta)$.
2. **Targeted attacks.** Targeted attacks aim to prevent the agent from taking a specific action (in our case, we find that this is more effective than tricking the agent to take a specific action). As an extreme case, this paper allows the attacker to know the (final) optimal action a^* that the agent is going to take next (a^* may be different from a'), and they seek to minimise the probability of the agent taking that action: for DDQN, the attacks minimise $Q(s' + \delta, a^*)$; for A3C, the attacks minimise $\pi(a^*|s' + \delta)$.

Algorithm 1. Attack II – Manipulating states

Input : The original experience (s, a, s', r); The list of all nodes L_N; Target action a_t ($a_t = -1$ for indiscriminate attack); The main DQN Q
Output: The tampered experience $(s, a, s' + \delta, r')$

```
if a_t == −1 then
    // indiscriminate attack
    a_t = arg max_{a'} Q(s', a');
for node n in L_N do
    if n is compromised then
        mark n as uncompromised;
        if Q(s' + δ, a_t) < minQ_FN then
            // δ represents the FP and/or FN readings
            FN = n;
            minQ_FN = Q(s' + δ, a_t);
        restore n as compromised;
    else
        mark n as compromised;
        if Q(s' + δ, a_t) < minQ_FP then
            FP = n;
            minQ_FP = Q(s' + δ, a_t);
        restore n as uncompromised;
Change node FN to uncompromised;
Change node FP to compromised;
return (s, a, s' + δ, r')
```

The details of the above two types of attacks are presented in Algorithm 1 (Algorithm 1 is for the attacks against DDQN. Due to similarity, the algorithm for attacks against A3C is omitted). In addition, we consider the following variants of the attacks:

1. **White-box attacks vs. Black-box attacks.** In white-box attacks, the attacker can access the model under training to select the false positive and false negative nodes, while in black-box attacks, the attacker first trains surrogate model(s), and then uses them to choose the FPs and FNs.
2. **Limit on the choice of FPs and FNs.** The above attacks do not set any limit on the choice of FPs and FNs, and hence even though the attacker can only manipulate the states of two nodes each time, overall, they still need to be able to control a number of nodes, which is not practical. Therefore, we first run unlimited white-box attacks, identify the top two nodes that have been selected most frequently as FPs and FNs respectively, and only allow the attacker to manipulate the states of those nodes.
3. **Limit on the timing of the attack.** The last type of attacks only introduces FPs and FNs in the first m steps (*e.g.*, $m = 3$) in each training episode.

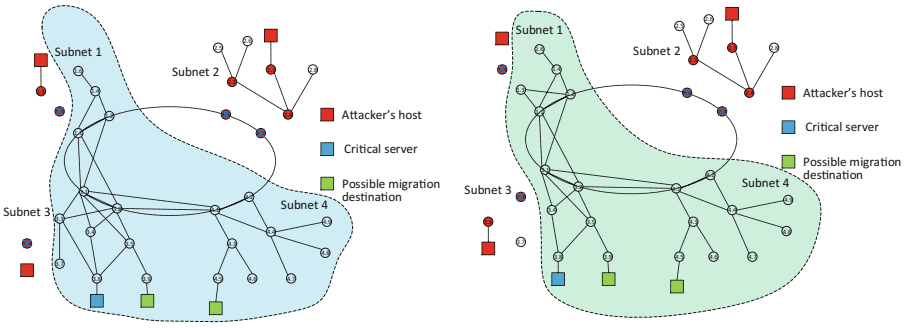
5 Experimental Verification

This section begins with a discussion of the experimental results obtained when applying RL to autonomous defence in a SDN environment without considering causative attacks. We then analyse the impact of the two forms of attacks explained in Sect. 4. Finally, we discuss adopting adversarial training as a potential countermeasure, and present some preliminary results. Experiments on causative attacks were performed on eight servers (equivalent to two Amazon EC2 t2.large instances and six t2.xlarge instances [1]), and each set of experiments was repeated 15 to 25 times.

5.1 Autonomous Defence in a SDN

For our experiments, as shown in Fig. 1, we created a network with 32 nodes and 48 links using Mininet [3], one of the most popular network emulators. OpenDaylight [4,35] serves as the controller, and monitors the whole-of-network status. The RL agent retrieves network information and takes appropriate operations by calling corresponding APIs provided by OpenDaylight. In the setup, the three nodes in red, *i.e.,* nodes 1.5, 2.7 and 3.6, have already been compromised. Node 3.8 is the critical server to be protected, and it can be migrated to node 3.9 or 4.5.

We trained a DDQN (with Prioritised Experience Relay) agent and an A3C agent. We set the length of training such that the reward per episode for both agents reached a stable value well before training ended. The two agents learned two slightly different responses: the DDQN agent decides to first isolate node 3.6, then 1.3, 2.2 and finally 2.1, which means 21 nodes are preserved (see Fig. 2a); while the A3C agent isolates nodes 1.5, 3.3, 2.2 and 2.1, keeping 20 nodes uncompromised and reachable from the critical server (see Fig. 2b).



(a) DDQN (b) A3C

Fig. 2. Optimal results without causative attacks (nodes in the shade are preserved)

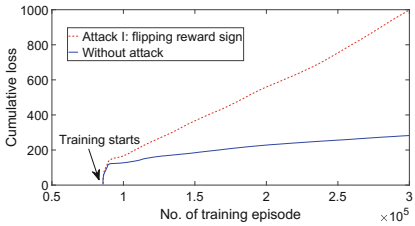


Fig. 3. Cumulative loss before and after flipping reward sign attacks (against DDQN)

5.2 Attack I: Flipping Reward Sign

This subsection presents the results of the first form of attack that flips the reward sign. In our experiments, we limit the total number of tampered experiences to 5% of all experiences obtained by the agent, and also set the number of tampered experiences per training iteration to the range of $[1, 5]$.

As can be seen in Fig. 3, the attack is effective in increasing the agent's loss function. However, our results also suggest that this form of attack only delays the training as the agent still learns the optimal actions (although the delay can be significant).

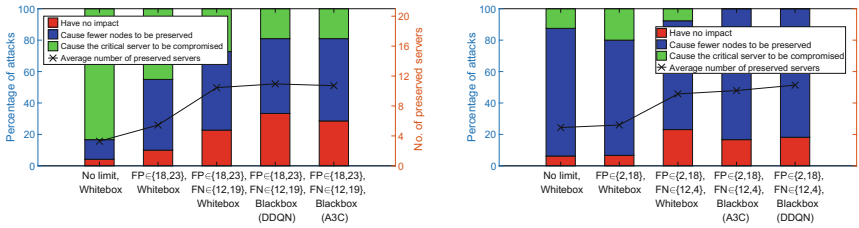
5.3 Attack II: Manipulate State—Indiscriminate Attacks

Unlimited White-Box Attacks. We start with unlimited indiscriminate white-box attacks, the case where the attacker has full access to the model under training. For each experience (s, a, s', r) obtained by the agent, they can manipulate the states of any two nodes in s', *i.e.,* one false positive and one false negative, in order to prevent the agent from taking the next optimal action a' that has been learned so far (note that it may be different from the final optimal action). Specifically, for the DDQN agent, the attacker minimises $\max_{a'} Q(s' + \delta, a')$; for the A3C agent, the attacker minimises $\max_{a'} \pi(a'|s' + \delta)$.

Figure 4 presents the results we obtained during our experiments. The leftmost bars in Figs. 4a and 4b suggest that the unlimited indiscriminate white-box attacks are very effective against both DDQN and A3C. Specifically, the average number of preserved nodes decreases from 21 and 20 to 3.3 and 4.9, respectively.

White-Box Attacks with Limits on the Choices of False Positive and False Negative. As pointed out in Subsect. 3.2, even though the attacker only manipulates the states of two nodes each time, overall, they still need to be able to control a number of nodes, which is unlikely in practice. We calculate the number of times that each node is chosen in the above unlimited attacks, and find that some nodes are selected more frequently than others (Fig. 5; the histograms for the A3C case are omitted due to similarity).

Therefore, when poisoning the DDQN agent, we limit the false positive nodes to {3.5 (node ID 18), 4.1 (node ID 23)}, and limit the false negative nodes to

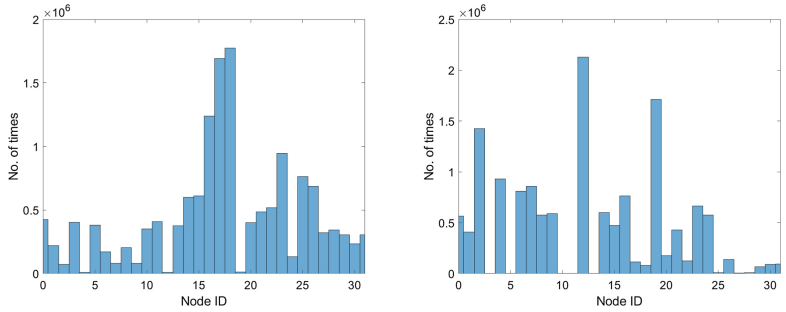


(a) Indiscriminate attacks against DDQN (b) Indiscriminate attacks against A3C

Fig. 4. Indiscriminate attacks against DDQN & A3C. The bars indicates the percentage of attacks (left $y-$axis) that (1) have no impact; (2) cause fewer nodes to be preserved; and (3) cause the critical server to be compromised. The line marked by "$\times$" indicates the average number of preserved servers (right $y-$axis). The five types of attacks are: (1) white-box, no limit on FNs&FPs; (2) white-box, with limits on FP but not on FN, (3) white-box, with limits on both FP and FN; (4) black-box, same algorithm, with limits on both FPs and FNs; (5) black-box, different algorithm, with limits on both FPs and FNs.

{2.7 (node ID 12), 3.6 (node ID 19)}. We use node 4.1 instead of 3.4 (node ID 17), as otherwise both selected nodes would be from the target subnet and directly connected to the target, which is unlikely in real situations. In the A3C case, the false positive and false negative nodes are limited to {1.3 (node ID 2), 3.5 (node ID 18)}, and {2.7 (node ID 12), 1.5 (node ID 4)}, respectively.

The second and third bars in Figs. 4a and 4b show that the limit has an obvious negative impact on the attack, especially the limit on the false negative nodes. Still, less than half of the nodes are preserved on average, compared with the scenarios without attacks.



(a) False positive (b) False negative

Fig. 5. Histograms of the false positive and false negative nodes being selected against DDQN. N.B.: The node IDs {0, 1, ..., 31} are ordered and mapped to the node sequence {1.1, 1.2, ..., 1.6, 2.1, ..., 2.8, 3.1, ..., 3.9, 4.1, ..., 4.9}

Black-Box Attacks with Limits on the Choices of False Positive and False Negative Nodes. In our black-box attacks (both intra- and cross-models), the attacker does not have access to the training model. Instead, they train their own agents first, and use the surrogate models to poison the training of the target models by choosing false positive and false negative nodes. Specifically, we have trained a few DDQN and A3C agents with a different number of hidden layers from the target model, and observed that these surrogates can still prevent the critical server from compromising.

As illustrated by the rightmost two bars in Figs. 4a and b, black-box attacks are only slightly less effective than the counterpart white-box attacks despite the surrogate using a different model. This lends support that transferability also exists between RL algorithms, *i.e.,* attacks generated for one model may also transfer to another model.

5.4 Attack II: Manipulate State—Targeted Attacks

In the targeted attacks considered here, the attacker is assumed to know the sequence of final optimal actions, and attempts to minimise the probability of the agent following that sequence. It should be pointed out that we have also studied the case where the attacker instead maximises the probability of taking a specific non-optimal action for each step, but our results suggested that this is less effective.

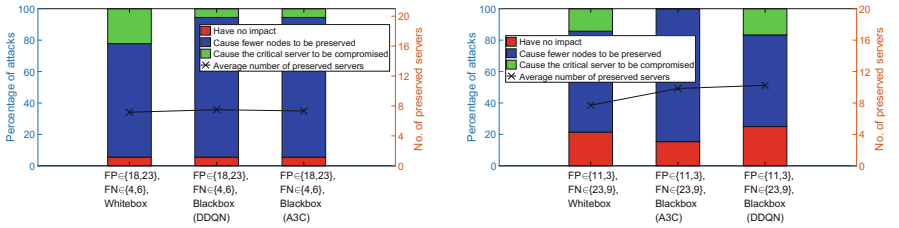
We find that in targeted attacks, certain nodes are also selected more frequently as a false positive and false negative. In this scenario, we limit false positive nodes to (1) {3.5 (node ID 18), 4.1 (node ID 23)} against DDQN, (2) {2.6 (node ID 11), 1.4 (node ID 3)} against A3C, and limit false negative nodes to (1) {1.5 (node ID 4), 2.1 (node ID 6)} against DDQN, (2) {4.1 (node ID 23), 2.4 (node ID 9)} against A3C.

Our results, summarised in Fig. 6, indicate that (1) compared with the results on indiscriminate attacks, targeted attacks work better, especially against DDQN (fewer nodes are preserved on average), as the attacker is more knowledgeable in this case; (2) similar to the indiscriminate case, black-box attacks achieve comparable results to the white-box attacks, further demonstrating the transferability between DDQN and A3C.

5.5 Timing Limits for the Attacks

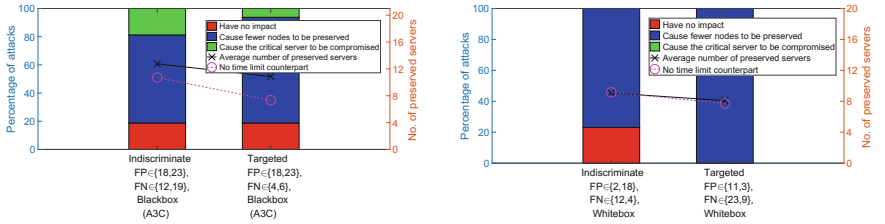
The attacks discussed so far allowed the attacker to poison every experience obtained by the agent. A possible limitation on this assumption is to examine whether these attacks can remain successful when the attacker can only manipulate part of the experiences. Therefore, in this subsection we shall look at attacks that poison only a subset (the first three steps) in each training episode.

Figures 7a and 7b depict the results of the time-limited version of (cross-model) black-box attacks against DDQN and white-box attacks against A3C (both with limit on the choices of FPs and FNs), respectively. The results suggest



(a) Targeted attacks against DDQN (b) Targeted attacks against A3C

Fig. 6. Targeted attacks against DDQN & A3C.



(a) Time limited black-box attacks against DDQN (b) Time limited white-box attacks against A3C

Fig. 7. Attacks against DDQN&A3C with time limit. The attacker only poisons the first three steps per training episode.

that even though the time limit has a negative impact in every scenario studied, the attacks are still effective.

5.6 Discussion on Countermeasures

In supervised learning problems, adversarial training [21,57,58] has the defender select a target point (x, y) from the training set, modify x to x^* (*i.e.,* generates an adversarial sample), and then inject (x^*, y) back into the training set, under the implicit assumption that the true label y should not change given the instance has been only slightly perturbed.

In our RL setting, while the attacker manipulates the observed states to minimise the probability of the agent taking the optimal action a in state s, the defender can construct adversarial samples that counteract the effect. For example, for each experience (s, a, s', r), the defender can increase r by a small amount, *e.g.*, 5% of the original value, given that r is positive and a is not chosen randomly (the probability of choosing an action randomly decreases as the training proceeds). The rationale behind this modification is that when the poisoning attack starts out, it is likely that a is still the optimal action (that has been learned so far) for state s. If r is positive it means that action a is a relatively good option for s, and since the attacker has poisoned the state to prevent the agent from taking a, we slightly increase r to encourage the agent to take action a.

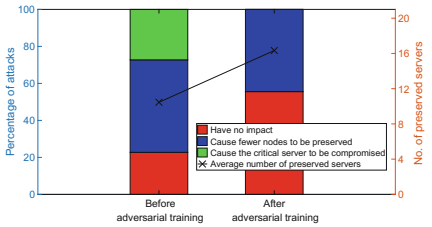


Fig. 8. Adversarial training against indiscriminate white-box attacks with limits on the choices of FPs and FNs (against DDQN), $FP \in \{18, 23\}, FN \in \{12, 19\}$

We have tested the above idea against indiscriminate white-box attacks with a limit on the choices of FPs and FNs against DDQN. Specifically, for each experience (s, a, s', r) whose r is positive and a is not selected randomly, we change it to $(s, a, s', min(1.0, 1.05r))$. Note that our experimental results suggest that adding 5% error to the reward signal when there is no attack will not prevent the agent from learning the optimal actions, although it may cause some delay. The results in Fig. 8 indicate that adversarial training can make the training process much less vulnerable.

However, the results are still preliminary, and we plan to further investigate other forms of adversarial training. For example, Pinto *et al.* [51] model all potential disturbances as an extra adversarial agent, whose goal is to minimise the discounted reward of the leading agent. They formulate the policy learning problem as a two player zero-sum game, and propose an algorithm that optimises both agents by alternating learning one agent's policy with the other policy being fixed. In addition, we will also study the impact of the loss function, prioritised experience replay, ensemble adversarial training [58] and other, more intrusive types of attacks, where the adversary is aware of the defence method, and attacks the defended model.

6 Related Work

This section reviews ways in which attackers can target machine learning systems and current defence mechanisms. We first present a taxonomy on attacks against (primarily) supervised classifiers, and then summarise recent work that applies/modifies these attacks to manipulate RL systems. Finally, we review existing countermeasures against adversarial machine learning.

6.1 Taxonomy of Attacks Against Machine Learning Classifiers

Barreno *et al.* [6] develop a qualitative taxonomy of attacks against ML classifiers based on three axes: influence (*causative* vs. *exploratory attacks*), security violation (*integrity* vs. *availability attacks*) and specificity (*indiscriminate* vs. *targeted attacks*).

Table 2 uses the taxonomy to classify previous work on adversarial machine learning against classifiers. As can be seen, more focus has been paid to exploratory integrity attacks. Presently, the Fast Gradient Sign Method (FGSM) attack [21] is widely studied, and the C&W attack [14] is the most effective found so far on the application domains tested, mostly in computer vision. Both of these attack methods can be used for targeted or indiscriminate attacks.

With further examination of the attacker's capabilities, a powerful attacker may also know the internal architecture and parameters of the classifier. Therefore, a fourth dimension can be added to the above taxonomy according to attacker information: in *white-box attacks*, the adversary generates malicious instances against the target classifier directly; while in *black-box attacks*, since the attacker does not possess full knowledge about the model, they first approximate the target's model. Then if the reconstructed model generalises well, the crafted adversarial examples against this model can be transferred to the target network and induce misclassifications. Papernot *et al.* [47,48] have demonstrated the effectiveness of the black-box attack in certain specific domains.

6.2 Attacks Against Reinforcement Learning

In more recent studies, several papers have begun to study whether attacks against classifiers can also be applied to RL-based systems. Huang *et al.* [28] have shown that deep RL is vulnerable to adversarial samples generated by the Fast Gradient Sign Method [21]. Their experimental results demonstrate that both white-box and black-box attacks are effective, even though the less knowledge the adversary has, the less effective the adversarial samples are. Behzadan & Munir [8] establish that adversaries can interfere with the training process of DQNs, preventing the victim from learning the correct policy. Specifically, the attacker applies minimum perturbation to the state observed by the target, so that a different action is chosen as the optimal action at the next state. The perturbation is generated using the same techniques proposed against DNN classifiers. Lin *et al.* [34] propose strategically-timed attacks and enchanting attacks against deep reinforcement learning agents.

6.3 Adversarial Machine Learning Defences

A number of countermeasures have been proposed since the discovery of adversarial samples. These can be roughly categorised into two classes: *data-driven defences* and *learner robustification*.

Data-Driven Defences. This class of defences are data driven—they either filter out the malicious data, inject adversarial samples into the training dataset, or manipulate features via projection. These approaches are akin to black-box defences since they make little to no use of the learner.

Table 2. Taxonomy of attacks on machine learners, with representative past work. As the taxonomy was designed for supervised learners, we include attacks on reinforcement learning in Sect. 6.2.

	Integrity	Availability
Causative, targeted	Rubinstein *et al.* [52]: boiling frog attacks against the PCA anomaly detection algorithm; Li *et al.* [32]: poison training data against collaborative filtering systems; Mei and Zhu [36]: identify the optimal training set to manipulate different machine learners; Burkard and Lagesse [12]: targeted causative attack on SVMs that are learning from a data stream	Newsome *et al.* [45]: manipulate training set of classifiers for worms and spam to block legitimate instances; Chung and Mok [16]: generate harmful signatures to filter out legitimate network traffic
Causative, indiscriminate	Biggio *et al.* [11]: inject crafted training data to increase SVM's test error; Xiao *et al.* [60]: label flips attack against SVMs; Koh and Liang [29]: minimise the number of crafted training data via influence analysis	Newsome *et al.* [45]; Chung and Mok [16]; Nelson *et al.* [43]: exploit statistical machine learning against a popular email spam filter
Exploratory, targeted	Nelson *et al.* [44]: probe a classifier to determine good attack points; Papernot *et al.* [49]: exploits forward derivatives to search for the minimum regions of the inputs to perturb; Goodfellow *et al.* [21]: design FGSM to generate adversarial samples; Carlini and Wagner [14]: propose the C&W method for creating adversarial samples; Han and Rubinstein [22]: improve the gradient descent method by replacing with gradient quotient	Moore *et al.* [40]: provide quantitative estimates of denial-of-service activity
Exploratory, indiscriminate	Biggio *et al.* [10]: find attack instances against SVMs via gradient descent; Szegedy *et al.* [57] demonstrate that changes imperceptible to human eyes can make DNNs misclassify an image; Goodfellow *et al.* [21]; Papernot *et al.* [47,48]: attack the target learner via a surrogate model; Moosavi-Dezfooli *et al.* [41,42]: propose DeepFool against DNNs; Carlini and Wagner [14]; Nguyen *et al.* [46]: produce images that are unrecognisable to humans, but can be recognised by DNNs; Han and Rubinstein [22]	Moore *et al.* [40]

- Filtering instances. These counter-measures assume that the poisoning data in the training dataset or the adversarial samples against the test dataset either exhibit different statistical features, or follow a different distribution. Therefore, they propose to identify and filter out the injected/perturbed data [20,30,33,37,55].
- Injecting data. Goodfellow *et al.* [21] attribute the existence of adversarial samples to the "blind spots" of the training algorithm, and propose injecting adversarial examples into training to improve the generalisation capabilities of DNNs [21,57]. Tramer *et al.* [58] extend such adversarial training methods by incorporating perturbations generated against other models.
- Projecting data. Previous work has shown that high dimensionality facilitates the generation of adversarial samples, resulting in an increased attack surface [59]. To counter this, data can be projected into lower-dimensional space before testing [9,17,61]. However, these results contradict with [31], which suggests that more features should be used when facing adversarial evasion.

Learner Robustification. Rather than focusing solely on training and test data, this class of methods—which are white-box in nature—aim to design models to be less susceptible to adversarial samples in the first place.

- Stabilisation. Zheng *et al.* [62] design stability training that modifies the model's objective function by adding a stability term. Papernot *et al.* [50] provide examples using a distillation strategy against a saliency-map attack. However, this method is shown to be ineffective by Carlini and Wagner [13]. Hosseini *et al.* [26] propose to improve adversarial training by adding an additional "NULL" class.
- Moving target. Sengupta *et al.* [54] apply moving target defences against exploratory attacks: the defender prepares a pool of models instead of a single model, and for each image to be classified, one trained DNN is picked following certain strategy.
- Robust statistics. Another avenue that has remained relatively unexplored is to leverage ideas from robust statistics, *e.g.*, influence functions, M-estimators with robust loss functions. Rubinstein *et al.* [52] applied a robust form of PCA to defend against causative attacks on network-wide volume anomaly detection. Recently, interest in the theoretical computer science community has turned to robust estimation in high dimensions, *e.g.*, Diakonikolas *et al.* [18].

Lessons Learned. Despite many defences proposed, several recent studies [15,25] point out that most of these methods unrealistically assume that the attacker is not aware of the defence mechanism, and only consider relatively weak attacks, *e.g.*, FGSM [21]. Negative results are reported on the effectiveness of these methods against adaptive attackers that are aware of the defence and act accordingly, and against the C&W attack [14]. More recently, Athalye *et al.* [5] show that defences relied on obfuscated gradients can also be circumvented.

7 Conclusions and Future Work

In this paper, we demonstrated the feasibility of developing autonomous defence in SDN using RL algorithms. In particular, we studied the impact of different forms of causative attacks, and showed that even though these attacks might cause RL agents to take sub-optimal actions, adversarial training could be applied to mitigate the impact.

For future work, we plan to (1) use a traffic generator to introduce background traffic between nodes, and use network performance metrics to replace the current binary states; (2) consider different types of network traffic, so that the actions of the RL agent could include partial isolation in terms of blocking certain protocols between nodes; (3) change full observability of the network status to partial observability—the defender may have limited resources, and the attacker may not know the entire topology; and (4) remove limiting assumptions, *e.g.*, the attacker having to compromise all nodes along the path to the critical server.

References

1. Amazon EC2 Instance Types – Amazon Web Services (AWS). https://aws.amazon.com/ec2/instance-types/
2. SDN architecture. Technical report, June 2014. https://www.opennetworking.org/wp-content/uploads/2013/02/TR_SDN_ARCH_1.0_06062014.pdf
3. Mininet: An Instant Virtual Network on your Laptop (2017). http://mininet.org/
4. OpenDaylight (2017). https://www.opendaylight.org/
5. Athalye, A., Carlini, N., Wagner, D.: Obfuscated gradients give a false sense of security: circumventing defenses to adversarial examples. arXiv:1802.00420 [cs], February 2018
6. Barreno, M., Nelson, B., Joseph, A.D., Tygar, J.D.: The security of machine learning. Mach. Learn. **81**(2), 121–148 (2010)
7. Beaudoin, L.: Autonomic computer network defence using risk states and reinforcement learning. Ph.D. thesis, University of Ottawa (Canada) (2009)
8. Behzadan, V., Munir, A.: Vulnerability of deep reinforcement learning to policy induction attacks. eprint arXiv:1701.04143 (2017)
9. Bhagoji, A.N., Cullina, D., Mittal, P.: Dimensionality reduction as a defense against evasion attacks on machine learning classifiers. arXiv:1704.02654 (2017)
10. 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
11. Biggio, B., Nelson, B., Laskov, P.: Poisoning attacks against support vector machines. In: Proceedings of the 29th International Conference on International Conference on Machine Learning, pp. 1467–1474. Omnipress, Edinburgh (2012)
12. Burkard, C., Lagesse, B.: Analysis of causative attacks against SVMs learning from data streams. In: Proceedings of the 3rd ACM on International Workshop on Security And Privacy Analytics, pp. 31–36. ACM, New York (2017)
13. Carlini, N., Wagner, D.: Defensive distillation is not robust to adversarial examples. arXiv:1607.04311 (2016)

14. Carlini, N., Wagner, D.: Towards evaluating the robustness of neural networks. eprint arXiv:1608.04644 (2016)
15. Carlini, N., Wagner, D.: Adversarial examples are not easily detected: bypassing ten detection methods. eprint arXiv:1705.07263 (2017)
16. Chung, S.P., Mok, A.K.: Advanced allergy attacks: does a corpus really help? In: Kruegel, C., Lippmann, R., Clark, A. (eds.) RAID 2007. LNCS, vol. 4637, pp. 236–255. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74320-0_13
17. Das, N., et al.: Keeping the bad guys out: protecting and vaccinating deep learning with JPEG compression. eprint arXiv:1705.02900, May 2017
18. Diakonikolas, I., Kamath, G., Kane, D.M., Li, J., Moitra, A., Stewart, A.: Robust estimators in high dimensions without the computational intractability. In: Proceedings of the 2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS), pp. 655–664, October 2016
19. Everitt, T., Krakovna, V., Orseau, L., Hutter, M., Legg, S.: Reinforcement learning with a corrupted reward channel. eprint arXiv:1705.08417 (2017)
20. Feinman, R., Curtin, R.R., Shintre, S., Gardner, A.B.: Detecting adversarial samples from artifacts. eprint arXiv:1703.00410 (2017)
21. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. eprint arXiv:1412.6572 (2014)
22. Han, Y., Rubinstein, B.I.P.: Adequacy of the gradient-descent method for classifier evasion attacks. arXiv:1704.01704, April 2017
23. Hasselt, H.V.: Double Q-learning. In: Lafferty, J.D., Williams, C.K.I., Shawe-Taylor, J., Zemel, R.S., Culotta, A. (eds.) Advances in Neural Information Processing Systems 23, pp. 2613–2621. Curran Associates, Inc. (2010)
24. Hasselt, H.V., Guez, A., Silver, D.: Deep reinforcement learning with double Q-learning. eprint arXiv:1509.06461, September 2015
25. He, W., Wei, J., Chen, X., Carlini, N., Song, D.: Adversarial example defenses: ensembles of weak defenses are not strong. eprint arXiv:1706.04701 (2017)
26. Hosseini, H., Chen, Y., Kannan, S., Zhang, B., Poovendran, R.: Blocking transferability of adversarial examples in black-box learning systems. eprint arXiv:1703.04318 (2017)
27. Huang, L., Joseph, A.D., Nelson, B., Rubinstein, B.I., Tygar, J.: Adversarial machine learning. In: Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence, pp. 43–58. ACM (2011)
28. Huang, S., Papernot, N., Goodfellow, I., Duan, Y., Abbeel, P.: Adversarial attacks on neural network policies. eprint arXiv:1702.02284 (2017)
29. Koh, P.W., Liang, P.: understanding black-box predictions via influence functions. arXiv:1703.04730 [cs, stat], March 2017
30. Laishram, R., Phoha, V.V.: Curie: a method for protecting SVM Classifier from poisoning attack. arXiv:1606.01584 [cs], June 2016
31. Li, B., Vorobeychik, Y.: Feature cross-substitution in adversarial classification. In: Proceedings of the 2014 NIPS, NIPS 2014, pp. 2087–2095, MIT Press, Cambridge (2014)
32. Li, B., Wang, Y., Singh, A., Vorobeychik, Y.: Data poisoning attacks on factorization-based collaborative filtering. eprint arXiv:1608.08182 (2016)
33. Li, X., Li, F.: Adversarial examples detection in deep networks with convolutional filter statistics. arXiv:1612.07767 [cs], December 2016
34. Lin, Y.C., Hong, Z.W., Liao, Y.H., Shih, M.L., Liu, M.Y., Sun, M.: Tactics of adversarial attack on deep reinforcement learning agents. eprint arXiv:1703.06748, March 2017

35. Medved, J., Varga, R., Tkacik, A., Gray, K.: OpenDaylight: towards a model-driven SDN controller architecture. In: Proceedings of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks, pp. 1–6 (2014)
36. Mei, S., Zhu, X.: Using machine teaching to identify optimal training-set attacks on machine learners. In: Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, pp. 2871–2877. AAAI Press, Austin (2015)
37. Metzen, J.H., Genewein, T., Fischer, V., Bischoff, B.: On detecting adversarial perturbations. eprint arXiv:1702.04267 (2017)
38. Mnih, V., et al.: Asynchronous methods for deep reinforcement learning. In: Proceedings of the 33rd International Conference on International Conference on Machine Learning, ICML 2016, vol. 48, pp. 1928–1937. JMLR.org, New York (2016)
39. Mnih, V., et al.: Playing Atari with Deep Reinforcement Learning. CoRR abs/1312.5602 (2013)
40. Moore, D., Shannon, C., Brown, D.J., Voelker, G.M., Savage, S.: Inferring internet denial-of-service activity. ACM Trans. Comput. Syst. **24**(2), 115–139 (2006)
41. Moosavi-Dezfooli, S.M., Fawzi, A., Fawzi, O., Frossard, P.: Universal adversarial perturbations. eprint arXiv:1610.08401 (2016)
42. Moosavi-Dezfooli, S.M., Fawzi, A., Frossard, P.: DeepFool: a simple and accurate method to fool deep neural networks. In: CVPR, pp. 2574–2582 (2016)
43. Nelson, B., et al.: Exploiting machine learning to subvert your spam filter. In: Proceedings of the First USENIX Workshop on Large-scale Exploits and Emergent Threats (LEET 2008) (2008)
44. Nelson, B., et al.: Query strategies for evading convex-inducing classifiers. J. Mach. Learn. Res. **13**(May), 1293–1332 (2012)
45. Newsome, J., Karp, B., Song, D.: Paragraph: thwarting signature learning by training maliciously. In: Zamboni, D., Kruegel, C. (eds.) RAID 2006. LNCS, vol. 4219, pp. 81–105. Springer, Heidelberg (2006). https://doi.org/10.1007/11856214_5
46. Nguyen, A., Yosinski, J., Clune, J.: Deep neural networks are easily fooled: high confidence predictions for unrecognizable images. In: CVPR, pp. 427–436 (2015)
47. Papernot, N., McDaniel, P., Goodfellow, I.: Transferability in machine learning: from phenomena to black-box attacks using adversarial samples. eprint arXiv:1605.07277 (2016)
48. Papernot, N., McDaniel, P., Goodfellow, I., Jha, S., Celik, Z.B., Swami, A.: Practical black-box attacks against deep learning systems using adversarial examples. eprint arXiv:1602.02697 (2016)
49. Papernot, N., McDaniel, P., Jha, S., Fredrikson, M., Celik, Z.B., Swami, A.: The limitations of deep learning in adversarial settings. In: Proceedings of the European Symposium on Security & Privacy, pp. 372–387 (2016)
50. Papernot, N., McDaniel, P., Wu, X., Jha, S., Swami, A.: Distillation as a defense to adversarial perturbations against deep neural networks. eprint arXiv:1511.04508 (2015)
51. Pinto, L., Davidson, J., Sukthankar, R., Gupta, A.: Robust adversarial reinforcement learning. eprint arXiv:1703.02702 (2017)
52. Rubinstein, B.I., et al.: ANTIDOTE: understanding and defending against poisoning of anomaly detectors. In: Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement, pp. 1–14. ACM (2009)
53. Schaul, T., Quan, J., Antonoglou, I., Silver, D.: Prioritized Experience Replay. CoRR abs/1511.05952 (2015)
54. Sengupta, S., Chakraborti, T., Kambhampati, S.: Securing deep neural nets against adversarial attacks with moving target defense. eprint arXiv:1705.07213, May 2017

55. Steinhardt, J., Koh, P.W., Liang, P.: Certified defenses for data poisoning attacks. eprint arXiv:1706.03691, June 2017
56. Sutton, R.S., Barto, A.G.: Introduction to Reinforcement Learning, 1st edn. MIT Press, Cambridge (1998)
57. Szegedy, C., et al.: Intriguing properties of neural networks. eprint arXiv:1312.6199 (2013)
58. Tramèr, F., Kurakin, A., Papernot, N., Boneh, D., McDaniel, P.: Ensemble adversarial training: attacks and defenses. eprint arXiv:1705.07204, May 2017
59. Wang, B., Gao, J., Qi, Y.: A theoretical framework for robustness of (deep) classifiers against adversarial examples. eprint arXiv:1612.00334 (2016)
60. Xiao, H., Xiao, H., Eckert, C.: Adversarial label flips attack on support vector machines. In: Proceedings of the 20th European Conference on Artificial Intelligence. ECAI 2012, pp. 870–875, IOS Press, Amsterdam (2012)
61. Zhang, F., Chan, P.P.K., Biggio, B., Yeung, D.S., Roli, F.: Adversarial feature selection against evasion attacks. IEEE Trans. Cybern. **46**(3), 766–777 (2016)
62. Zheng, S., Song, Y., Leung, T., Goodfellow, I.: Improving the robustness of deep neural networks via stability training. eprint arXiv:1604.04326 (2016)