Adversarial Reinforcement Learning under Partial Observability in Autonomous Computer Network Defence

Yi Han*, David Hubczenko†, Paul Montague†, Olivier De Vel†, Tamas Abraham†,
Benjamin I. P. Rubinstein*, Christopher Leckie*, Tansu Alpcan‡, Sarah Erfani*
*School of Computing and Information Systems
The University of Melbourne, Parkville, Australia, 3010
Email: {yi.han, benjamin.rubinstein, caleckie, sarah.erfani}@unimelb.edu.au
†Defence Science and Technology Group, Australian Department of Defence
Edinburgh, Australia, 5111
Email: {david.hubczenko, paul.montague, olivier.devel, tamas.abraham}@dst.defence.gov.au
‡Department of Electrical and Electronic Engineering
The University of Melbourne, Parkville, Australia, 3010
Email: tansu.alpcan@unimelb.edu.au

***Abstract*—Recent studies have demonstrated that reinforcement learning (RL) agents are susceptible to adversarial manipulation, similar to vulnerabilities previously demonstrated in the supervised learning setting. While most existing work studies the problem in the context of computer vision or console games, this paper focuses on reinforcement learning in autonomous cyber defence under partial observability. We demonstrate that under the black-box setting, where the attacker has no direct access to the target RL model, causative attacks—attacks that target the training process—can poison RL agents even if the attacker only has partial observability of the environment. In addition, we propose an inversion defence method that aims to apply the opposite perturbation to that which an attacker might use to generate their adversarial samples. Our experimental results illustrate that the countermeasure can effectively reduce the impact of the causative attack, while not significantly affecting the training process in non-attack scenarios.**

***Index Terms*—adversarial reinforcement learning, partial observability, cyber security.**

I. Introduction

The adversarial machine learning [1]–[4] literature has demonstrated that machine learning models are vulnerable to both exploratory (test-time) and causative (training-time) attacks. These attacks are typically crafted by applying calculated perturbations to the test or training instances, in order to either cause misclassification or poison the training process. More recent studies [5]–[8] have shown that similar attacks can also be effective against reinforcement learning algorithms.

Unlike the majority of the literature that mainly focuses on the vision domain or console games, in previous work we [7] analyse how reinforcement learning agents react to different forms of poisoning attacks in the context of autonomous defence in software-defined networking (SDN) [9]. Specifically, we first demonstrate that without any poisoning attacks, an RL agent can be successfully trained to identify the optimal strategy for preventing the attacker from propagating through the network. Then we investigate the effect of two different types of poisoning attacks on the RL training process, and show that the RL agent can be misled into making non-optimal decisions, causing a significantly larger part of the network to be compromised by the attacker. Section II provides a more detailed description.

However, there are two limitations with the previous work [7]: (1) full observability of the (network) states is assumed in the analysis, which is often not the case in real-world situations, especially for the attacker; (2) while an important topic, treatment of RL defence mechanisms is preliminary, and the proposed method does not work effectively in the new setup as introduced below. In this work, we address these limitations and make the following ***contributions***:

First, we impose ***partial observability*** for the attacker. Since it is unlikely that the attacker can map out the entire network topology, we consider the scenario where the defender has full observability of the network, but the attacker only knows part of the topology. Specifically, Fig. 1 depicts the running example network studied in this paper. The network is comprised of 100 nodes and 172 links, and the attacker has an initial foothold of a handful of compromised nodes. They aim to propagate through the network to take control of a specific node corresponding to the critical server, which in response can be migrated by the defender to some pre-determined alternate nodes.

As shown in the figure, two setups are considered, where the attacker can observe around one-third and half of all the nodes, respectively. Under each setup, the defender trains a reinforcement learning agent to (1) protect the critical server from being compromised, and (2) maintain the network functionality as much as possible, *i.e.,* maximise the number of nodes that can reach the critical server. On the other hand, the attacker only has partial observability, which restricts their action set: they cannot compromise an adjacent node unless the link to the node is known.

Second, we propose a new ***inversion defence method*** to counteract the causative attack on reinforcement learning algorithms. Our experimental results suggest that the approach introduced in [7] does not work well in our setup (Fig. 1). Instead, we design a method that requires no prior knowledge about the attacker, and attempts to undo attacker poisoning of the RL training process. We demonstrate the effectiveness of the new defensive algorithm, and show that it has limited impact in non-attack scenarios.

The remainder of this paper is organised as follows: Section II summarises the problem of applying reinforcement learning for autonomous defence in computer networks; Section III introduces the causative attack via state perturbation and Section IV the defence mechanism; Section V presents the experimental verification; Section VI reviews previous work in adversarial machine learning; and finally Section VII concludes the paper and offers directions for future work.

II. Problem: Reinforcement Learning for Autonomous Network Defence

We now overview the problem of autonomous defence in computer networks using reinforcement learning.

A. Background on Reinforcement Learning

Reinforcement learning [10] deals with a sequential decision making problem where an agent interacts with the environment to maximise its rewards. At each time step t, the agent (1) receives an observation s_t of the environment; (2) takes an action a_t based on its policy π, which is a mapping from states to actions; and (3) obtains a reward r_t based on state s_t, action a_t, and the environment's transition to a new state s_{t+1}. The goal of the agent is to maximise its cumulative rewards, *i.e.,* $R_t = \sum_{\tau=t}^{\infty} \gamma^{\tau-t} r_\tau$, where $\gamma \in (0,1]$ is a discount factor which affects the present importance of long-term rewards.

We focus our experiments on two widely used RL algorithms—Double Deep Q-Networks (DDQN) [11] and Asynchronous Advantage Actor-Critic (A3C) [12]—and the transferability of attacks between them.

B. Autonomous Network Defence with Reinforcement Learning

Maintaining the security of cyber environments without affecting the normal exchange of information is a challenging task. Although the problem of cyber defence has been studied for decades, most deployed solutions are still rule-based that require a significant human involvement and are prone to generating false alarms—rules are formulated based on previously seen threats, and may not be applicable to new vulnerabilities. In addition, for those solutions that do employ machine learning, traditional one-class or two-class prediction algorithms are often used, which require prior knowledge on existing attacks to make accurate decisions. However, acquiring the prior knowledge on all existing attacks is almost impossible, as more sophisticated attacks are generated everyday.

In this work, we investigate the feasibility of applying RL for autonomous cyber defence, as RL has the ability to adapt and generalise, and has been successfully used for autonomous control in a wide range of applications.

Specifically, we consider a computer network of $|N|$ nodes, $N = \{n_1, n_2, ..., n_{|N|}\}$, and $|L|$ links, $L = \{l_1, l_2, ..., l_{|L|}\}$ (*e.g.,* Fig. 1), where $N_D \subset N$ is the set of critical servers to be protected (one or more blue nodes), $N_M \subset N$ is the set of possible migration destinations for node $n \in N_D$ (one or more green nodes), and $N_A \subset N$ is the set of nodes that have been compromised (red nodes). In addition, while the defender knows all the nodes and links, the attacker is only able to map out a subset of them—N_O and L_O, where $N_O \subseteq N, L_O \subseteq L$.

The attack scenario we consider is a cyber attack against the network infrastructure. Here, the attack spreads through the network, and aims to take control of the critical servers (note that we assume that the attacker has to compromise all nodes on the path). However, they can compromise a node n only if there is a link $l \in L_O$ between n and a compromised node $n' \in N_A$. That is N_A keeps expanding as the attack proceeds.

In order to protect the critical servers from being compromised, the defender trains an RL agent that:

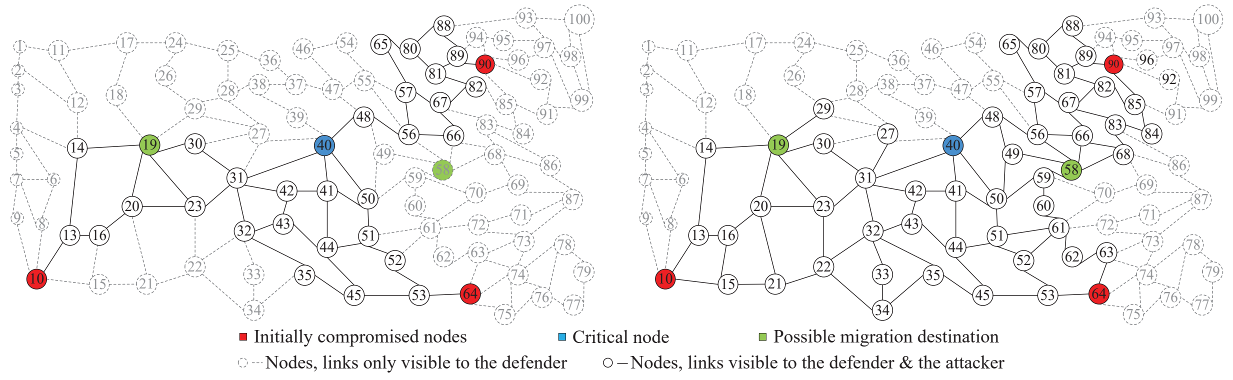
1) Monitors the system state. The system state is represented using a binary feature representation. The state representation has a number of bits equal to the sum of the number of nodes and number of links in the network. A bit corresponding to a node is 0/1 to represent whether that node is un/compromised. A bit corresponding to a link is 0/1 to represent whether that link is down/up. Note that **detection is not our focus**—we are not studying how to detect the attacker, nor how the attacker can escape detection. Therefore, we have modeled the defender as having in place a detection system. Our experiments suggest that as long as the system achieves a reasonable detection rate, e.g., $\geq 75\%$, it does not have an obvious impact on the final results. In our experiment, the detection rate is set to 90%.

2) Chooses an appropriate action to take when in a given system state. The actions that are available comprise: (i) isolating and patching one node; (ii) reconnecting one node and its links; (iii) migrating the critical server to a certain destination; and (iv) taking no action. Note that for actions (i) or (ii), only one node can be isolated or reconnected during each action cycle.

The reward function that the RL agent is trained on is given in (1), where U_t is the number of nodes unreachable from the critical server after the current t^{th} step, C_t is the number of newly compromised nodes at the t^{th} step, r_c is the reward for an additional node to be compromised, r_m is the migration cost, $\mathbf{1_{a=m}} = 1$ iff the action a is to migrate the critical server, and $\alpha,\ \beta \geq 1.0$.

$$r_t = \begin{cases} -1, & n \in N_D \text{ is compromised or the action is invalid} \\ \left(1 - \alpha \cdot \frac{U_t}{|N|} \cdot \beta^t\right) - C_t \cdot r_c \cdot \beta^t - \mathbf{1_{a=m}} \cdot r_m, & \text{otherwise} \end{cases} \tag{1}$$

As we can see, the reward is based on (i) whether any critical server has been compromised; (ii) the validity of an action, *e.g.*, if a node has already been isolated, it cannot be


Initially compromised nodes
Critical node
Possible migration destination
Nodes, links only visible to the defender
Nodes, links visible to the defender & the attacker

(a) Setup 1 (b) Setup 2

Fig. 1: Network setups: (a) the attacker can observe 34 nodes and 46 links; (b) the attacker can observe 51 nodes and 80 links.

isolated again; (iii) the number of nodes reachable from the critical servers; (iv) the number of newly compromised nodes; and (v) the migration cost. Note that the term β^t encourages RL agents to find the optimal solution with minimal steps.

For each of the setups in Fig. 1, we train multiple DDQN (with Prioritised Experience Replay [13]) and A3C agents with different structures, i.e., different numbers of hidden layers and different numbers of neurons per layer. These agents help us identify the optimal policy without tampering (see Fig. 2): (1) under the first setup, isolating nodes in the order of 10, 53, 81, 80, which results in a total of 92 out of 100 nodes being preserved. Note that there are several other equally optimal solutions for this case; (2) under the second setup, isolating nodes in the order of 90, 53, 62, 22, 31, which results in a total of 82 out of 100 nodes being preserved.

However, the above cyber attack scenario and resulting trained RL agents leave important questions unanswered: ***if the attacker has the ability to poison the training process, can the agents still identify the optimal actions? What can the defender do to mitigate attack impact?*** We seek to address these questions.

III. Partially-Observable Poisoning Attacks on RL by State Manipulation

In order for RL techniques to be successfully applied in autonomous cyber defence, it is crucial to analyse the susceptibility of RL agents to potential causative attacks. However, most existing adversarial attacks against RL agents are based on gradient descent optimisation [5], [6], [8], [14], and in our case the attacker aims to manipulate the binary state of a node (note again that the purpose of the attack is not to escape detection/cause misclassification). Therefore, gradient descent-based attacks are not applicable. Instead, we have investigated the following attack mechanisms:

1) Tampering with a small number (*e.g.*, 5%) of rewards to maximise the defender's loss. Specifically, the gradient of the loss with respect to the rewards, is used to select which rewards to tamper with;
2) Random perturbation of the observed states;
3) Manipulating the states to minimise the defender's rewards;
4) Manipulating the states to minimise the probability of taking the optimal action.

In our preliminary unreported experiments we found that the last attack mechanism was the most effective and hence we subsequently use it as the attacker's strategy.

A. Threat Model

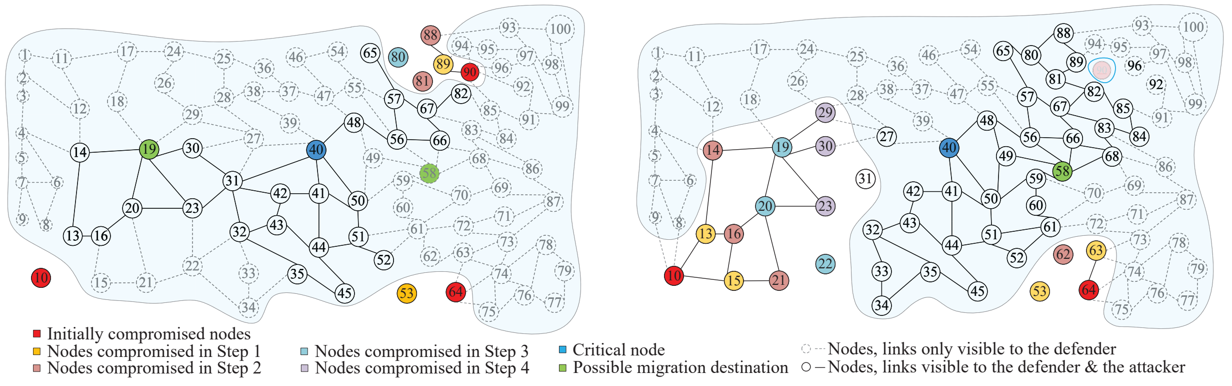
We focus on the scenario where the attacker tampers with the states observed by the RL agents, so that the trained model learns sub-optimal actions. Specifically, suppose that the agent observes an experience (s, a, s', r) without any attacks, where s is the current system state, a is the action taken by the agent, s' is the new state, and r is the reward. When the system reaches the new state s', the agent would continue to take the next optimal action a'. The attacker can counteract this by introducing false positive (FP) and false negative (FN) readings in s', meaning that uncompromised (compromised) nodes will be reported as compromised (uncompromised) to the defender. Consequently, the agent observes $(s, a, s' + \delta, r')$ (where δ represents the FP and FN readings) instead of (s, a, s', r), and hence may not take action a' next.

The key issue here is how the attacker chooses the nodes to manipulate. We consider the following strategy:

1) Against the DDQN agent: loop through all observable nodes to find δ that minimises the Q-value of the optimal action a' for state $s' + \delta$, *i.e.*, $\text{argmin}_\delta\, Q(s' + \delta, a')$;
2) Against the A3C agent: loop through all observable nodes to find δ that minimises the probability of taking the optimal action a' for state $s' + \delta$, *i.e.*, $\text{argmin}_\delta\, \pi(a'|s' + \delta)$.

We next abstract the threat model for adversarial learning in autonomous cyber defence as follows:

Black-box approach. The attacker does not have access to the defender's training model as per our partial observability assumption. This constitutes a form of black-box attack, which means the attacker needs to train their own surrogate model first, based on the partial topology visible to them.


Initially compromised nodes
Nodes compromised in Step 1
Nodes compromised in Step 2
Nodes compromised in Step 3
Nodes compromised in Step 4
Critical node
Possible migration destination
Nodes, links only visible to the defender
Nodes, links visible to the defender & the attacker

(a) Setup 1 (b) Setup 2

Fig. 2: Optimal results in response to a cyber attack against the network (in the absence of attacking the RL algorithm).

Limited choice of potential false positive and false negative nodes. It is unlikely that the attacker can falsify the state of all observable nodes. Therefore, we limit the nodes whose states can be perturbed by the attacker. Section V further explains how these nodes are selected.

Limits on the number of false readings per time step. In our experiments, the number of FP and FN nodes that can be introduced per time step are no more than two per case.

Our view is that this model of attacker information/control is a key point of interest in exploring domains beyond computer vision. Algorithm 1 details this attack against DDQN. The algorithm for attacks against A3C is similar and so is omitted.

IV. The Inversion Defence Mechanism

For the defender we aim to design a defence mechanism that (1) effectively mitigates the impact of the above causative attack, (2) requires minimum knowledge of the attacker, and (3) does not affect the training when there is no attack. Specifically, we propose a countermeasure that generates training instances by applying a perturbation counter to simulated adversarial samples.

Since the attacker adds false readings δ into the observed states, can δ be reversed? If the defender knows the nodes that are visible to the attacker, limits on the FP & FN nodes, and the number of FPs and FNs added per time step, then they may find these false readings, by solving the inverse problem of how the attacker generates the adversarial samples: while the attacker receives (s, a, s', r), and loops through all observable nodes to find δ that either minimises the Q-value $Q(s'+\delta, a')$ or the probability $\pi(a'|s'+\delta)$ of action a' for state $s'+\delta$, the defender receives $(s, a, s'+\delta, r')$, and searches within the same nodes to find δ' that maximises $Q(s'+\delta+\delta', a')$ for DDQN, and $\pi(a'|s'+\delta+\delta')$ for A3C. In other words, $\delta' = -\delta$.

However, the defender does not know (1) the attacker's partial knowledge of the network topology, (2) the limits on the choice of FP & FN nodes, and (3) the number of false readings per time interval/step. As shown in Algorithm 2, we propose the following to address these obstacles:

1) Instead of looping through the nodes observable to the attacker, the defender necessarily goes through all network nodes to find δ'—this solves the first two issues, but increases the training time. We further discuss the overhead in Section V-C1;
2) Test the scenarios where compared with the actual number of false readings introduced by the attacker at each time step, the defender assumes less, the same and more added—as demonstrated in our experiments (Section V-C), even if the defender does not know the exact number of false readings, the inversion defence method is still effective.

δ' obtained in such a way may not exactly match δ, and the defender can choose to either keep both $(s, a, s'+\delta, r')$ and $(s, a, s'+\delta+\delta', r')$, or only the latter. This method does not make any assumptions about the attacker, except that they falsify the states of certain nodes. However, as demonstrated by the results in Section V, the method is effective against the causative attack, and it does not prevent the agent from learning the optimal actions in the non-attack scenario.

V. Experimental Results

We next introduce our experimental setup, present how DDQN and A3C agents are affected by causative attacks, and demonstrate effectiveness of the proposed defence.

A. SDN Experimental Environment

In order to better cope with today's dynamic and high-bandwidth traffic, software-defined networking (SDN) [9] is designed as a next-generation tool chain for computer network management. SDN adopts a three layer architecture: (1) in the top application layer, applications that deliver services communicate their network requirements to the controller; (2) in the middle layer, the SDN controller translates the received requirements into low-level controls, and passes them to the bottom infrastructure layer; (3) the infrastructure layer includes switches that control forwarding and data processing. Under such an architecture, the controller has a centralised view of the whole network, and is directly programmable since network control is decoupled from forwarding functions. It is thus convenient to monitor and reconfigure network resources.

Algorithm 1: Causative attack against DDQN via state perturbation

```
Input  : The original experience, $(s, a, s', r)$;
         The list of observable nodes, $N_O$;
         The list of nodes that can be perturbed as
         false positive (false negative) by the attacker,
         $L_{FP}$ ($L_{FN}$);
         The main DQN, $Q$;
         Limit on the number of FPs and FNs per
         time, LIMIT
Output : The tampered experience $(s, a, s' + \delta, r')$
1  $FN = FP = \{\}$;
2  $minQ_{FN} = minQ_{FP} = \{\}$;
3  $a' = \text{argmax}_{a^*} Q(s', a^*)$;
4  for node $n$ in $N_O$ do
5      if $n$ is compromised and $n$ in $L_{FN}$ then
6          mark $n$ as uncompromised;
7          if $Q(s' + \delta, a') <$ any value in $minQ_{FN}$ then
               // $\delta$ represents the FP and/or FN readings
8              insert $n$ and $Q(s' + \delta, a')$ into appropriate
                 positions in $FN$ and $minQ_{FN}$;
9              if $|FN| > LIMIT$ then
10                 remove extra nodes from $FN$ and
                     $minQ_{FN}$;
11         restore $n$ as compromised;
12     else if $n$ is uncompromised and $n$ in $L_{FP}$ then
13         mark $n$ as compromised;
14         if $Q(s' + \delta, a') <$ any value in $minQ_{FP}$ then
15             insert $n$ and $Q(s' + \delta, a')$ into appropriate
                 positions in $FP$ and $minQ_{FP}$;
16             if $|FP| > LIMIT$ then
17                 remove extra nodes from $FP$ and
                     $minQ_{FP}$;
18         restore $n$ as uncompromised;
19 Change nodes in $FN$ to uncompromised;
20 Change nodes in $FP$ to compromised;
21 return $(s, a, s' + \delta, r')$
```

There have been a number of proprietary and open-source SDN controller software platforms. In this paper, we choose OpenDaylight [15], the most popular open-source SDN controller available. Specifically, we use Mininet [16], a popular network emulator, to create the network with 100 nodes and 172 links as shown in Fig. 1. Once the network is created, OpenDaylight is added as the controller. It provides APIs for the RL agent to retrieve network information and execute different types of operations as defined in Section II.

We want to emphasise that ***SDN is only one platform we choose for demonstration purposes****—although it is used in production. The studied causative attacks and the proposed defence method are not coupled to any particular platform.*

Algorithm 2: The inversion defence mechanism

```
Input  : The potentially tampered experience,
         $(s, a, s' + \delta, r')$;
         The main DQN, $Q$;
         The list if all nodes, $N$;
         The estimate of the attacker's limit on the
         number of FPs and FNs per time, $LIMIT'$
Output : The corrected experience $(s, a, s' + \delta + \delta', r')$
1  $FN = FP = \{\}$; // $FN(FP)$ is a list of potentially false
     negative (false positive) nodes tampered by the
     adversaries that need to be corrected
2  $maxQ_{FN} = maxQ_{FP} = \{\}$;
3  $a' = \text{argmax}_{a^*} Q(s' + \delta, a^*)$;
4  for node $n$ in $N$ do
5      if $n$ is compromised then
6          mark $n$ as uncompromised;
7          if $Q(s' + \delta + \delta', a') >$ any value in $maxQ_{FP}$
             then
               // $\delta'$ represents the correction introduced by
                  the defender
               // $n$ is potentially a false positive node
8              insert $n$ and $Q(s' + \delta + \delta', a')$ into
                 appropriate positions in $FP$ and $maxQ_{FP}$;
9              if $|FP| > LIMIT'$ then
10                 remove extra nodes from $FP$ and
                     $maxQ_{FP}$;
11         restore $n$ as compromised;
12     else if $n$ is uncompromised then
13         mark $n$ as compromised;
14         if $Q(s' + \delta + \delta', a') >$ any value in $maxQ_{FN}$
             then
               // $n$ is potentially a false negativnode
15             insert $n$ and $Q(s' + \delta + \delta', a')$ into
                 appropriate positions in $FN$ and $maxQ_{FN}$;
16             if $|FN| > LIMIT'$ then
17                 remove extra nodes from $FN$ and
                     $maxQ_{FN}$;
18         restore $n$ as uncompromised;
19 Change nodes in $FN$ to compromised;
20 Change nodes in $FP$ to uncompromised;
21 return $(s, a, s' + \delta + \delta', r')$
```

B. Causative Attacks via State Perturbation

As described in Section III, we are considering a black-box setting, which means that the attacker does not have direct access to the target RL model, and needs to train its own model. For each of the setups in Fig. 1, we achieve this by training a DDQN agent using the partial topology visible to the attacker. The model is then used as the surrogate to attack both of the defender's models (*i.e.,* both DDQN and A3C agents).

In addition, there is a limit on the nodes that the attacker can perturb. This is an appropriate threat model—even if the attacker can map out part of the network topology, it is very unlikely that they can manipulate the states of all those nodes. We run the attack by adding one FP and one FN per time interval/step but without any limits on the choices of FPs and FNs. In this way, we are able to find the nodes that are most frequently selected as FPs and FNs. L_{FP} and L_{FN} in Algorithm 1 are then initialised with these nodes. Note that the nodes in L_{FP} and L_{FN} are different under the two setups, and within each setup they are also different for the DDQN and A3C agents. The attacker is only allowed to manipulate the states of these nodes.

Furthermore, the attacker also needs to limit the number of false positive and false negative readings added per time interval. Considering the practicality of the attack, two settings are used in our experiments: (i) one FP & one FN, and (ii) two FPs & two FNs.

Fig. 3 shows the effectiveness of the attack under different settings, where the top four, six, eight FP nodes and top two FN nodes are selected, *i.e.,* $|L_{FP}| = 4, 6$ or 8, while $|L_{FN}| = 2$. $|L_{FN}|$ is set to 2 because additional experiments with multiple combinations suggests that further increasing $|L_{FN}|$ does not have an obvious impact. The results demonstrate that:

1) The causative attack designed in Algorithm 1 is effective against both DDQN and A3C agents when there is no form of defence—under both setups a significant percentage of attacks either cause the critical server to be compromised (the red bars), or cause fewer nodes to be preserved (the blue bars). Note that this also demonstrates the existence of transferability between RL algorithms [17]—attackers do not need to have knowledge of the defender's model and hence attempting to keep the model secret is not an effective countermeasure against adversarial reinforcement learning attacks.
2) Under the second setup where the attacker observes more nodes, the attacks are more effective in general—the average number of preserved nodes is much lower in most cases. This is because the effectiveness of the attack depends on how close the surrogate and target models are, and with a larger observable topology, the attacker is more likely to train a surrogate that resembles the target RL agent.
3) Given the same number of false readings per time step, the stricter the limits on the choices of FPs and FNs, *i.e.,* the smaller $|L_{FP}|$ and $|L_{FN}|$ are, the less powerful the attacks are—not only do the limits restrict which nodes can be manipulated, they also decrease the number of steps that are poisoned in each training episode.
4) Interestingly, if we compare the second and fourth bars in all four figures, when $|L_{FP}| = 6$, adding one FP & one FN per time step is more effective than adding two FPs & two FNs per time step. This is because more training steps are likely to be poisoned in the former case given that $|L_{FP}|$ is the same.

In the next section, we test our proposed countermeasure against the most powerful form of attack as illustrated in Fig. 3, where $|L_{FP}| = 8$, $|L_{FN}| = 2$, and two FPs & two FNs are added per time step under the second setup.

1) Discussion on the attack efficiency: The limited choice of potential false positive and false negative nodes, *i.e.,* L_{FP} and L_{FN}, not only makes the attack more practical but also increases the efficiency of the attack, as the attacker only needs to loop through these two lists of nodes to find the FPs and FNs instead of checking all the visible nodes. Our experimental results suggest that the attack does not cause an obvious delay to the normal training process.

2) Discussion on the Impact of Partial Observability: As we mentioned earlier, a subset of nodes are more frequently selected as FPs and FNs. Therefore, the attack will become more effective if the attacker can take control over more of these most damaging nodes. For future work, we intend to further study the relation between partial observability and attack effectiveness. Specifically, we will identify a minimum set of nodes that the attacker needs to control for a given level of efficiency.

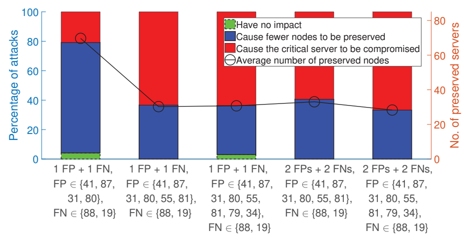
C. Countermeasure

Our inversion defence method only assumes that attackers perturb the states of a certain number of nodes in each training step, and aims to identify & revert the manipulations. However, the defender has to loop through all the nodes rather than the nodes in L_{FP} & L_{FN}, and has to estimate the number of false readings added per step.

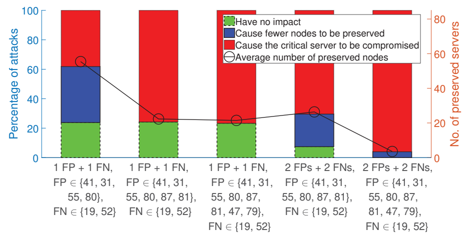
Specifically, four scenarios are investigated under the second setup: in the first three scenarios, the attacker adds two FPs & two FNs per training step, and $|L_{FP}| = 8$, $|L_{FN}| = 2$ (*i.e.,* the most powerful form of attack studied in the experiments), while the defender assumes that there are (1) one FP & one FN, (2) two FPs & two FNs, (3) three FPs & three FNs per training step. In the last case, the defender assumes that two FPs & two FNs are added per time step, but in fact there is no attack. The first three scenarios investigate the situations where the defender either does or does not know the limit on the number of false readings added per time, while the last scenario is designed to study whether the normal learning process will be impacted when the defender falsely assumes the presence of an attack.

Comparing the rightmost bars in Figs. 3c & 3d and the left three bars in Figs. 4a & 4b, we can see that the proposed defence method can effectively mitigate the impact of the causative attacks—the percentage of experiments where the critical server is compromised drops from almost 100% to less than 30% on average. In addition, the two rightmost bars in Fig. 4 also indicate that in most cases the defence method will not prevent the agent from learning the optimal actions when there is no attack—in all the cases represented by the blue bar in Fig. 4a, only one less node is preserved.

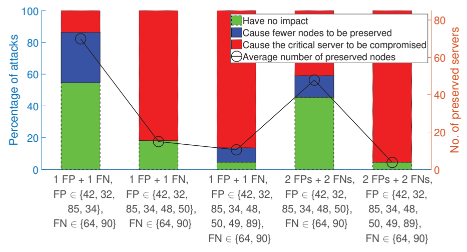
1) Discussion on the Overhead: A disadvantage of the inversion defence method is that it significantly slows down the training process, as it is time-consuming to loop through all the nodes to find the potential FPs and FNs. We aim to improve the performance in our future work. Specifically, we find that not all nodes are equally important in terms of preventing the critical server from being compromised—incorrect readings



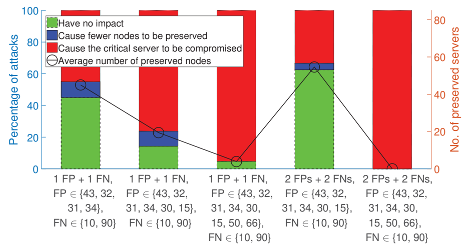
(a) Setup 1: attacks against DDQN



(b) Setup 1: attacks against A3C



(c) Setup 2: attacks against DDQN



(d) Setup 2: attacks against A3C

Fig. 3: Attacks against the DDQN & A3C agents. 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 lines indicate the average number of preserved servers (right $y-$axis).

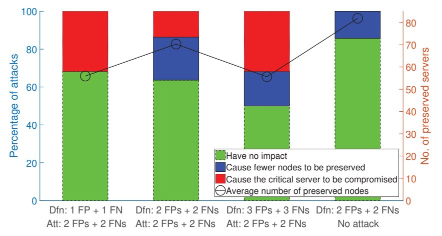
from certain nodes can cause more damage. Therefore, we will be investigating improving the efficiency of the defence method by only looping through those crucial nodes.

VI. Related Work

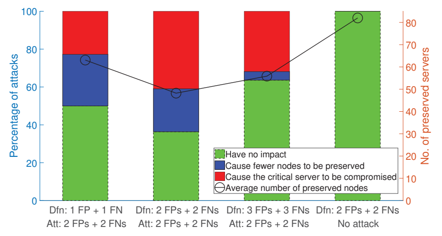
This section first summarises adversarial machine learning against supervised classifiers, and then reviews recent work on similar attacks against reinforcement learning models. Finally, we discuss existing defence mechanisms.

A. Adversarial Machine Learning

Adversarial machine learning aims to minimise the modifications to the input, *i.e.,* either the test instance or the training



(a) Defence against attacks on DDQN



(b) Defence against attacks on A3C

Fig. 4: Defence against attacks on the DDQN & A3C agents.

sample, to cause a malfunction of the machine learning model.

Biggio *et al.* [3], [18] formulate the problem of evading a machine learning classifier as optimisation of the model's continuous scores, and use gradient descent to generate adversarial samples. Szegedy *et al.* [4] highlight the observation that modifications imperceptible to humans can cause deep neural networks to misclassify, and they design the Fast Gradient Sign Method [19] for the attack. Since then a number of different methods for creating adversarial samples have been proposed [17], [20]–[26], among which the C&W attack [25] is empirically the most efficient exploratory attack so far. In addition, more recent work has also studied adversarial attacks in other domains, such as graph-based models [27], [28].

B. Adversarial Reinforcement Learning

It has been shown that reinforcement learning models are also vulnerable to the above attacks against classifiers. For example, Huang *et al.* [5] demonstrate that both white-box and black-box attacks using the Fast Gradient Sign Method [19] are effective against deep RL.

Behzadan & Munir [6] were the first to investigate causative attacks against RL agents. They show how adversaries can perturb the observed state, in order to prevent the DQN agent from learning the correct policy.

Lin *et al.* [14] propose two types of attacks against deep RL: (1) strategically-timed attack, which aims to decrease the number of time steps to launch the attack; (2) enchanting attack, which aims at misleading the agent to a specific state.

Pattanaik *et al.* [8] show that even the naïve attack, that is, adding random noise into the current state, is effective against deep RL—this is contrary to our experimental findings. However, our scenario is different to that described by the authors, including the dimensions of the state, the action space, and they design a gradient based attack that aims to maximise the probability of taking the worst possible action.

C. Existing Defence Mechanisms

Generally speaking, existing defence methods against adversarial machine learning can be categorised into two classes: (1) data-driven defence, which either filters adversarial samples, injects adversarial samples into training—a.k.a., adversarial training, or projects inputs into a lower dimension; (2) learner robustification, which stabilises the training, applies moving target, or leverages ideas from robust statistics.

Countermeasures against attacks on RL models adopt similar approaches. Mandlekar *et al.* [29], Pattanaik *et al.* [8] propose different adversarial training algorithms. Lin *et al.* [30] use previous images to predict future input and detect adversarial examples. Havens *et al.* [31] propose the Meta-Learned Advantage Hierarchy framework that measures the underlying changes in a task to detect the attack. Another line of work initiates the study of formal verification of deep RL [32].

VII. Conclusions and Future Work

In this paper, we show that in the context of autonomous defence in cyber networks, RL agents can be manipulated by attacks that target the training process, even if the attacker only has partial observability of the environment and defensive algorithms. In order to defend against the attack, we propose an inversion method that aims to revert the perturbations added by the attacker. Our experimental results demonstrate the effectiveness of the proposed approach, and show that it causes limited impact in non-attack scenarios. Our work focuses on learning in software-defined networking, which brings with it novel threat models of independent interest to adversarial learning research.

For future work, we plan to work on three directions—(1) partial observability: (i) impose partial observability also on the defender, as the defender may not obtain the correct states of all the nodes all the time; (ii) identify the minimum set of nodes the attacker needs to control for a certain level of effectiveness. (2) Consider a more powerful attacker that can (i) expand their partial observability as the attack proceeds; and (ii) spread more freely through the network, instead of having to compromise all the nodes on the paths to the critical server. (3) Replace the binary state with a continuous state.

VIII. Acknowledgements

This work was supported by the DST Group Next Generation Technologies Fund (Cyber) program via Data61 CRP 'Adversarial Machine Learning for Cyber'.

References

[1] M. Barreno, B. Nelson, A. D. Joseph, and J. D. Tygar, "The Security of Machine Learning," *Machine Learning*, vol. 81, no. 2, pp. 121–148, Nov. 2010.

[2] L. Huang, A. D. Joseph, B. Nelson, B. I. Rubinstein, and J. Tygar, "Adversarial machine learning," in *AISec*. ACM, 2011, pp. 43–58.

[3] B. Biggio, B. Nelson, and P. Laskov, "Poisoning Attacks against Support Vector Machines," in *ICML*, Scotland, 2012, pp. 1467–1474.

[4] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing Properties of Neural Networks," *arXiv:1312.6199*, 2013.

[5] S. Huang, N. Papernot, I. Goodfellow, Y. Duan, and P. Abbeel, "Adversarial Attacks on Neural Network Policies," *arXiv:1702.02284*, 2017.

[6] V. Behzadan and A. Munir, "Vulnerability of Deep Reinforcement Learning to Policy Induction Attacks," *arXiv:1701.04143*, 2017.

[7] Y. Han, B. I. P. Rubinstein, T. Abraham, T. Alpcan, O. De Vel, S. Erfani, D. Hubczenko, C. Leckie, and P. Montague, "Reinforcement learning for autonomous defence in software-defined networking," in *Decision and Game Theory for Security*. Springer, 2018, pp. 145–165.

[8] A. Pattanaik, Z. Tang, S. Liu, G. Bommannan, and G. Chowdhary, "Robust deep reinforcement learning with adversarial attacks," *arXiv:1712.03632*, 2017.

[9] "SDN architecture," Tech. Rep., Jun. 2014. [Online]. Available: https://www.opennetworking.org/wp-content/uploads/2013/02/TR_SDN_ARCH_1.0_06062014.pdf

[10] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*, 1st ed. Cambridge, MA, USA: MIT Press, 1998.

[11] H. V. Hasselt, A. Guez, and D. Silver, "Deep Reinforcement Learning with Double Q-learning," *arXiv:1509.06461*, Sep. 2015.

[12] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Harley, T. P. Lillicrap, D. Silver, and K. Kavukcuoglu, "Asynchronous Methods for Deep Reinforcement Learning," in *ICML*, 2016, pp. 1928–1937.

[13] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized Experience Replay," *CoRR*, vol. abs/1511.05952, 2015.

[14] Y.-C. Lin, Z.-W. Hong, Y.-H. Liao, M.-L. Shih, M.-Y. Liu, and M. Sun, "Tactics of Adversarial Attack on Deep Reinforcement Learning Agents," *arXiv:1703.06748*, 2017.

[15] J. Medved, R. Varga, A. Tkacik, and K. Gray, "OpenDaylight: Towards a Model-Driven SDN Controller architecture," in *IEEE WoWMoM*, Jun. 2014, pp. 1–6.

[16] "Mininet: An Instant Virtual Network on your Laptop," http://mininet.org/, 2017.

[17] N. Papernot, P. McDaniel, and I. Goodfellow, "Transferability in Machine Learning: from Phenomena to Black-Box Attacks using Adversarial Samples," *arXiv:1605.07277*, 2016.

[18] B. Biggio, I. Corona, B. Nelson, B. I. Rubinstein, D. Maiorca, G. Fumera, G. Giacinto, and F. Roli, "Security evaluation of support vector machines in adversarial environments," in *Support Vector Machines Applications*. Springer, 2014, pp. 105–153.

[19] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and Harnessing Adversarial Examples," *arXiv:1412.6572*, 2014.

[20] A. Nguyen, J. Yosinski, and J. Clune, "Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images," in *CVPR*, 2015, pp. 427–436.

[21] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The Limitations of Deep Learning in Adversarial Settings," in *EuroS&P*, 2016, pp. 372–387.

[22] N. Papernot, P. D. McDaniel, I. J. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against machine learning," in *ACM on AsiaCCS, 2017*. ACM, 2017, pp. 506–519.

[23] S.-M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard, "Universal Adversarial Perturbations," *arXiv:1610.08401*, 2016.

[24] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "DeepFool: A Simple and Accurate Method to Fool Deep Neural Networks," in *CVPR*, 2016, pp. 2574–2582.

[25] N. Carlini and D. Wagner, "Towards Evaluating the Robustness of Neural Networks," *arXiv:1608.04644*, 2016.

[26] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," *arXiv:1706.06083*, 2017.

[27] D. Zügner, A. Akbarnejad, and S. Günnemann, "Adversarial attacks on neural networks for graph data," *KDD*, pp. 2847–2856, 2018.

[28] H. Dai, H. Li, T. Tian, X. Huang, L. Wang, J. Zhu, and L. Song, "Adversarial attack on graph structured data," *arXiv:1806.02371*, 2018.

[29] A. Mandlekar, Y. Zhu, A. Garg, L. Fei-Fei, and S. Savarese, "Adversarially robust policy learning: Active construction of physically-plausible perturbations," in *2017 IROS*, 2017, pp. 3932–3939.

[30] Y.-C. Lin, M.-Y. Liu, M. Sun, and J.-B. Huang, "Detecting adversarial attacks on neural network policies with visual foresight," *CoRR*, vol. abs/1710.00814, 2017.

[31] A. J. Havens, Z. Jiang, and S. Sarkar, "Online robust policy learning in the presence of unknown adversaries," *CoRR*, vol. abs/1807.06064, 2018.

[32] Y. Kazak, C. W. Barrett, G. Katz, and M. Schapira, "Verifying deep-RL-driven systems," in *NetAI@SIGCOMM*. ACM, 2019, pp. 83–89.