Deep CounterStrike: Counter Adversarial Deep Reinforcement Learning for Defense Against Metamorphic Ransomware Swarm Attack

Mohit Sewak[1(✉)], Sanjay K. Sahay[2], and Hemant Rathore[2]

[1] Security and Compliance Research, Microsoft R&D India Pvt. Ltd., Hyderabad, India
mohit.sewak@microsoft.com

[2] Department of CS&IS, BITS Pilani, K K Birla Goa Campus, Goa, India
{ssahay,hemantr}@goa.bits-pilani.ac.in

Abstract. Ransomware, create a devastating impact when it infects a system. Fortunately, post the initial breach, such ransomware could be detected using advanced machine learning techniques, and therefore other high-value assets/systems can be protected from any repeat attack by the same ransomware. However, using metamorphism, advanced/second-generation ransomware can alter its structure after every successful infection. With this ability of metamorphism, such advanced ransomware could continue to evade any defensive mechanism and keep infecting systems in subsequent networks. Currently, there exists neither any proven defensive mechanism nor any useful dataset to train a defensive mechanism against such advanced ransomware. Therefore, we present a deep counter adversarial reinforcement learning-based system that learns how to normalize the metamorphism of such advanced ransomware to design a credible defence against such advanced attacks. To augment training data for this system, we design and develop a deep adversarial reinforcement learning solution, to generate swarms of such advanced ransomware.

Keywords: Deep Reinforcement Learning · Adversarial Learning · Ransomware · Metamorphic Malware · Swarm Attack

1 Introduction

Ransomware is a type of malware that can infect/encrypt the system for ransom. In this, the recent/advanced ransomware developer uses obfuscation techniques to change its structure after every attack/infection/system encryption [2,21] to evade the deployed malware defence system. To detect ransomware, generally, samples are collected during the forensics of the compromised machines. From the collected samples, signatures are extracted and submitted to various malware defence systems to safeguard other machines across the networksv [21,32].

W. Wang and J. Wu (Eds.): BROADNETS 2023, LNICST 511, pp. 31–50, 2023.
https://doi.org/10.1007/978-3-031-40467-2_3

But such mechanisms fail to mitigate the attacks from advanced/metamorphic ransomware because such ransomware alters its structure after the infection, and the resulting obfuscated ransomware is not often identified by the malware defence system that was trained on original/similar ransomware samples.

Fortunately, it is challenging to design and develop obfuscated ransomware. But what if such ransomware could be generated through Artificial Intelligence (AI); and not just single ransomware, but a swarm of these to target a system? Therefore, to pro-actively defend/detect the advanced ransomware, in this paper, we first introduce an AI system to infuse metamorphic capabilities into existing base variants of ransomware and other malware. We show that augmented with such metamorphic capabilities, ransomware could evade even the most advanced and highly sophisticated machine learning (ML) and deep learning (DL) malware detection systems. Such evasion is broadly covered in literature as *adversarial* learning. But, the available literature on Adversarial-DL/ML attacks mostly works by adding strategic noise for evading DL classifiers. Therefore, such mechanisms are not suitable for our purpose because the resulting file may not replicate the functionality of ransomware. Also, such a file may not even be re-package-able to a valid executable program. Another aspect to consider in security is that adversarially robust malware detectors are popular. Hence, the malware classifiers may use algorithms that are immune to gradient-attacks [7] and other adversarial-DL/ML attacks [17]. Therefore, adversarial-DL/ML-based mechanisms cannot generate samples that can effectively and consistently evade existing adversarial-DL-immune malware detectors. Thus, using Adversarial-DL/ML-based solutions is not ideal for the purpose, and accordingly, one has to look forward to Reinforcement Learning (RL) based approaches. However, RL and even popular Deep Reinforcement Learning (DRL) algorithms [28,37] could not handle large action spaces. Therefore, in this paper, first, we develop Adversarial Deep Reinforcement Learning (A-DRL) using complex Proximal Policy Optimization (PPO) [26], which obfuscates malware using techniques like junk-code insertion.

The developed model modifies the sequence and frequency vectors of the opcodes as extracted from the disassembled ransomware while preserving their malicious capabilities and other functionalities, which to the best of our knowledge, there exists no evidence in open literature which counter the attack from such advanced metamorphic ransomware and any attack that has used advanced metamorphic ransomware. Therefore, first, we describe how we succeeded in creating such ransomware samples and also trained multiple diversely instantiated Adversarial Deep Reinforcement Learning (A-DRL) agents, each of which could create a unique metamorphic variant of base ransomware. Collectively, so many obfuscated variants of ransomware created a swarm of advanced, second-generation, metamorphic ransomware. Finally, we propose a defence called Counter-Adversarial-DRL (C-A-DRL) system which probably will be the first solution to defend against any such extreme attacks.

The C-A-DRL system will solve the problem of defence against metamorphic ransomware by normalizing the obfuscations in any metamorphic malware and thereby transforming them into their base variant, which is more likely to be detectable by an existing malware detection system. The C-A-DRL is also based on deep policy-based RL [26,30]. Still, it uses even more computationally complex learning mechanisms to solve the more complex problem of normalizing the malware.

The rest of the paper is organized as follows. Section 2 covers related work in A-DL, malware detection and evasion. Section 3 provides the rationale for designing a counter-adversarial system instead of re-training the detection system with adversarial data. Next, Sect. 4 describes the two different DRL environments used. Section 5 covers the mathematical lineage of the algorithms. Section 6 describes the dataset, features, and challenger malware detection systems assessed for use within the DRL environments. Section 7 and 8 explain the experiments conducted and the results. Finally, we conclude the paper in Sect. 9.

2 Related Work

Adversarial attack on security systems is a trending topic in both AI and security research. In the AI research, Generative Adversarial Networks [6,36] have made A-DL exceedingly popular. Subsequently, this science started being misused for various purposes. With injections of random/noisy perturbations into detection candidate feature tensors, a DL classifier could be fooled to miss-classify even simple candidates [17]. This phenomenon is known as *adversarial attack* in DL [9, 39]. In efforts to counter such attacks, some interesting ideas have been proposed in [10,11,20,24].

Some researchers have identified the drawbacks in the design of perturbations-based adversarial-DL mechanisms in security and have started using alternate techniques. Some researchers have used techniques like the classical RL [19], while some others have even proposed value-based DRL algorithms like the Deep Q Networks (DQN) [22] to perpetuate such adversarial attacks on network traffic. But most of these systems work at the same or similarly abstracted feature level where no actual malware could be practically created as any operating system compatible file, even if it could theoretically be reassembled and re-packaged after such modifications. To the best of our knowledge, we are not aware of any system that could re-package a modified file to an actual executable that could infect any system (or even preserve any functionality).

Recently, ransomware attacks [5,15], especially those involving human-operated ransomware (HumOR) have increased, both in frequency and impact to become the predominant threat. As metamorphic malware could change its structure after every infection, therefore it is difficult to detect using conventional methods [18,35]. Where ransomware and metamorphic malware [1,34] are the most dreadful threat vectors, the zero-day attacks [3], and swarm attacks are the most dreadful attack scenarios.

3 Rationale for a Ransomware Normalization Based Defence Mechanism

As ransomware defenders, we regularly encounter two main challenges. First is the availability of quality ransomware data in large volumes to train a modern DL-based classifier. The second is the discovery of an effective DL architecture that could be trained to detect ransomware using the extracted features from the available dataset. But unfortunately, the second approach will not be conducive to the purpose of evading a metamorphic attack. This is because we started with the premise that the PPO-based DRL adversary agents could learn to evade most of the classifiers. Evading another DL classifier, even the ones re-trained with existing adversarial data, would be as trivial as playing some more episodes for the A-DRL agent. Hence, such an approach is not only counterproductive but also highly risky, as in a zero-day swarm attack scenario, ML/DL-based classifiers are prone to false-negative. Moreover, even if we re-train a new classifier with adversarial data, it has two main drawbacks. First, training a classifier with adversarial data renders the resulting classifier vulnerable to gradient attacks. Second, over-sampling of data that otherwise represents a small subset of existing ransomware makes the classifier over-fit to these samples, and thereby reduces the classifier's effectiveness. Therefore, ideally, we would like to design a defence mechanism that does not involve any training with adversarial data. Further, we would like only minimal changes to any other sub-component of the endpoint detection system. Alternatively, a better approach would be to use a malware normalization pre-processor. As opposed to retraining an existing detection system, an additional pre-processing by a malware normalization system works by de-obfuscating the obfuscated metamorphic malware to its base form and hence will be an ideal design for a defence mechanism.

4 A-DRL and C-A-DRL Environment's Architecture

A schematic and the internal working flow of the A-DRL and C-A-DRL environment architecture to train the A-DRL agent for the generation of metamorphic variants of any malware is shown in Fig. 1. The architecture consists of seven components and is briefly described below:

1. Malware Repository: A repository of existing malware, which contains either the first generation or the base variant (or the ones for which the lineage could otherwise not be traced further) of available advanced metamorphic malware.
2. Malware Classification Sub-System: A complete malware detection/ classification pipeline of the malware detection system (covered in Sect. 6.1) with the most robust and performant detection on the associated malware repository.
3. Reward Function Computation: A component to compute the instantaneous reward at each step as per the reward function (discussed later in this section). Any subsequent discounting of these rewards (for attributing future rewards) is controlled by the agent algorithm.

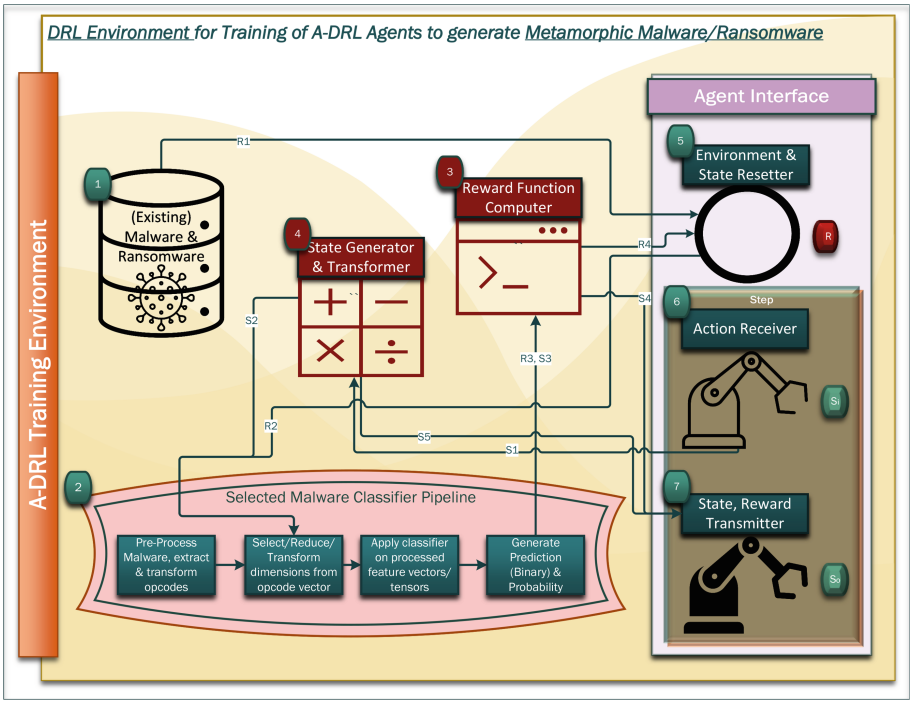

Fig. 1. Working flow of the A-DRL and C-A-DRL environment architecture.

4. State Generator and Transformer: This component uses the opcode feature vector modification suggestion from the agent to generate the next state corresponding to the agent's recommendations.
5. Environment and State Resetter: When an agent initiates a new training episode, it invokes this component functionality to reset the environment, the reward function, and the state vector.
6. Action Receiver: The action receiver along with the 'State Reward Transmitter' (component 7) constitutes the 'Step' functionality for the agent. The action-receiver receives the opcode feature modification suggestions from the agent as action and invokes the step method functionality as given in the Algorithm 1.
7. State Reward Transmitter: After processing the agent's action as per Algorithm 1, this component collects the generated rewards, and the resultant transformed state-vector corresponding to the suggested action and returns these to the agent to update its policy and recommend the action for the next step.

Algorithm 1 : Algorithm for A-DRL Step Method (Action Receiver and State Reward Transmitter components).

Method Input:
action_index: integer (representing [opcode to add in sequence] +[added opcode to delete])
Method Output:
reward: double
new_state: integer[1...M_O]

procedure STEP ▷ Executes the action suggestion by agent and computes reward and next-state
  if is_episode_complete = True **then**
    reset_environment_and_episode()
  $new\_observation[action\_index] \leftarrow new\_observation[action\_index] + / - Count_{Constant}$
  $\mathbb{P}_{NDMF} \leftarrow IsMalware_{Predict}(current\_observation)$
  $Reward_{Evasion} \leftarrow f(\mathbb{P}_{NDMF}^{Criteria1}, \mathbb{P}_{NDMF}^{Criteria2})$
  $Reward_{Discounted} \leftarrow \gamma * Reward_{Discounted} + Reward_{Evasion}$
  $Episode\_Turn\_Counter \leftarrow Episode\_Turn\_Counter + 1$
  $current\_observation \leftarrow new\_observation$
  if $\mathbb{P}_{NDMF} \geq P_{NDMF}^{threshold}$ **then**
    $is\_episode\_complete \leftarrow True$
    if $sim(new\_observation, original\_malware\_vector) \geq Sim_{Threshold}$ **then**
      archive obfuscation to the metamorphic variant repository
  return $(reward_t, nobservation_{t+1})$

The environment for the training of C-A-DRL is like that of the A-DRL, except for the malware repository and the reward computation module. Where the environment for the training of A-DRL contains a repository of all existing ransomware, the corresponding environment for the C-A-DRL system contains a repository of second-generation metamorphic ransomware. The reward functions for both environments are mathematically explained in Subsect. 4.1.

4.1 Reward Functions

We have at least two objectives for both the A-DRL and C-A-DRL agents. For A-DRL, the first objective is that it should be able to create enough ambiguity in the detection of the metamorphic variant. Because for most unbiased binary classifiers, the ambiguity point is at a probability of 0.5 (Eq. 1), we would like that the first aspect of the reward function to focus on this objective. This is translated as reward-criteria in the reward function as given in Eq. 2.

$$\mathbb{P}(\text{AmbiguousDetection}) = \mathbb{P}(\text{File}_{\text{Ransomware}} \mid File \subseteq \{\text{Ransomware}\}) = \mathbb{P}(\text{File}_{\text{Ransomware}} \mid File \subseteq \{\text{Non-Ransomware}\}) = 0.5 \quad (1)$$

$$\begin{aligned} \mathbb{R}_{\text{criteria-1}} : \mathbb{P}(\text{Detection}) &= \mathbb{P}(\text{File}_{\text{Ransomware}}) \\ &<= \mathbb{P}(\text{AmbiguousDetection}) \\ \mathbb{R}_1 &= \mathbb{P}(\text{AmbiguousDetection}) - \mathbb{P}(\text{Detection}) \\ &or, \mathbb{R}_1 = 0.5 - \mathbb{P}(\text{Detection}) \end{aligned} \tag{2}$$

The first reward criterion ensures that the agent learns to transform a definite ransomware detection into an ambiguous detection and, subsequently, a probable evasion. But, with this reward criterion in isolation, the agent will learn only to maintain $\mathbb{P}(\text{Detection}) < 0.5$ and thereby keep accumulating rewards. Whereas our main goal is to metamorph the ransomware to such an extent that it could be unambiguously detected as non-malicious. Also, we want to attain this in the minimum number of turns. However, the first aspect is partially, and the second aspect is completely missing in reward criterion 1. Therefore, we introduce another reward criterion, which gets triggered only when the detection probability reaches a higher threshold (say $\mathbb{P}(\text{File}_{\text{Ransomware}} < 0.1)$, and thereby generates a disproportionate higher reward. It is given in Eq. 3.

$$\begin{aligned} \mathbb{R}_{\text{criteria-2}} : \mathbb{P}(\text{Detection}) &= \mathbb{P}(\text{File}_{\text{Ransomware}}) \\ &\leq \mathbb{P}_{\text{Threshold}} = 0.1 \\ \mathbb{R}_2 = F(\text{Turns}_{Max}) &= \text{Turns}_{Max} \times \delta \\ &= \text{Turns}_{Max} \times \mathbb{P}(\text{AmbiguousDetection}) \end{aligned} \tag{3}$$

In Eq. 3, we kept the ultimate-objective reward as a function of Turns_{Max} (maximum number of turns allowed to the agent for a candidate file). This is to balance two opposing factors as under:

1. We want to have this reward disproportionately high enough such that the agent would weigh quicker termination more favourably than hunting in a range $\mathbb{P}(\text{File}_{\text{Ransomware}}) < 0.5$.
2. In cases where the agent is not able to reach this disproportionately high reward, it should still try to achieve reward criteria 1 and maintain it instead of completely disregarding it.

Here detection probability of ≤ 0.1 has been kept as the threshold to indicate unambiguous detection.

Another aspect in Eq. 3 is the constant multiplier δ, which is kept as $\mathbb{P}(\text{AmbiguousDetection})$ to enhance the balance against the reward-criteria 1 further. Combining these reward criteria, the final reward function ($\mathcal{R}_{\text{A-DRL}}$) is given in the Eq. 4.

$$\mathcal{R}_{\text{A-DRL}} = \begin{cases} 0.5 - \mathbb{P}(\text{Detection}), & \text{if } \mathbb{P}(\text{Detection}) \\ & \geq \mathbb{P}_{\text{Threshold}} \\ \text{Turns}_{Max} \times \delta, & \text{otherwise} \end{cases} \tag{4}$$

For the C-A-DRL agents, there are similar constraints as that of A-DRL, but their directions and limits change. Since we start from the ransomware samples generated corresponding to Eq. 2, for reward-criteria 1 for C-A-DRL, their default non-ambiguous probability range does not start at 0.0. Accordingly, the first reward criteria for C-A-DRL is given in the Eq. 5.

$$\begin{aligned}
&\mathbb{P}(\text{AmbiguousDetection})_{\text{C-A-DRL}} = \\
&\quad \mathbb{P}(\text{File}_{\text{Ransomware}} \mid File \subseteq \{\text{Ransomware}\}) = \\
&\mathbb{P}(\text{File}_{\text{Ambiguous_A-DRL}} \mid File \subseteq \{\text{Non-Ransomware}\}) \\
&\quad = 0.75 \\
&\mathbb{R}_{\text{criteria-1}} : \mathbb{P}(\text{Detection}) = \mathbb{P}(\text{File}_{\text{Ransomware}}) \\
&\quad \geq \mathbb{P}(\text{AmbiguousDetection})_{\text{C-A-DRL}} \\
&\mathbb{R}_1 = \mathbb{P}(\text{Detection}) - \mathbb{P}(\text{AmbiguousDetection})_{\text{C-A-DRL}} \\
&\quad or, \mathbb{R}_1 = \mathbb{P}(\text{Detection}) - 0.75
\end{aligned} \tag{5}$$

The reward-criteria 2 remains the same for C-A-DRL, except for the threshold for the criteria-trigger which becomes $\mathbb{P}_{\text{Threshold\_C-A-DRL}} = \mathbb{P}(\text{File}_{\text{Ransomware}} \mid File \subseteq \{\text{Ransomware}\}) - \mathbb{P}_{\text{Threshold\_A-DRL}} = 0.9$. Combining the reward criteria, the reward function for C-A-DRL is given in the Eq. 6.

$$\mathcal{R}_{\text{C-A-DRL}} = \begin{cases} \mathbb{P}(\text{Detection}) - 0.75, & \text{if } \mathbb{P}(\text{Detection}) \\ & \geq \mathbb{P}_{\text{Th.\_C-A-DRL}} \\ \text{Turns}_{Max} \times \delta, & \text{otherwise} \end{cases} \tag{6}$$

5 A-DRL and C-A-DRL Agent(s) Based on PPO Algorithm

Figure 2 shows the interaction between the A-DRL system's agent with its environment (via the agent interface) as per the training process. The C-A-DRL agent also similarly interacts with its environment.

The action space of the Markov Decision Process (MDP) for both the A-DRL and C-A-DRL problems consists of a vector of instruction set for the underlying kernel and processor platform. This makes the MDP for both of our DRL environments constrained to an exceptionally large cardinality discrete action space. As compared to classical RL/DRL, which could handle large state spaces very efficiently. Popular value-based DRL [27] algorithms like the DQN [13,14] and variants [8,40] claims to handle states representing multiple image-frame in Convolution Neural Network [29] tensors. Still, these also do not scale appropriately to accommodate high cardinality action spaces. Therefore, we appraised several policy-based DRL algorithms and finally used the PPO-based algorithm.

A policy-approximation-based algorithm needs to compute the expectancy of a particular policy to formulate the policy-utility function, for which it intends to find the gradient. The policy gradient for a Stochastic Policy Gradient [38] is

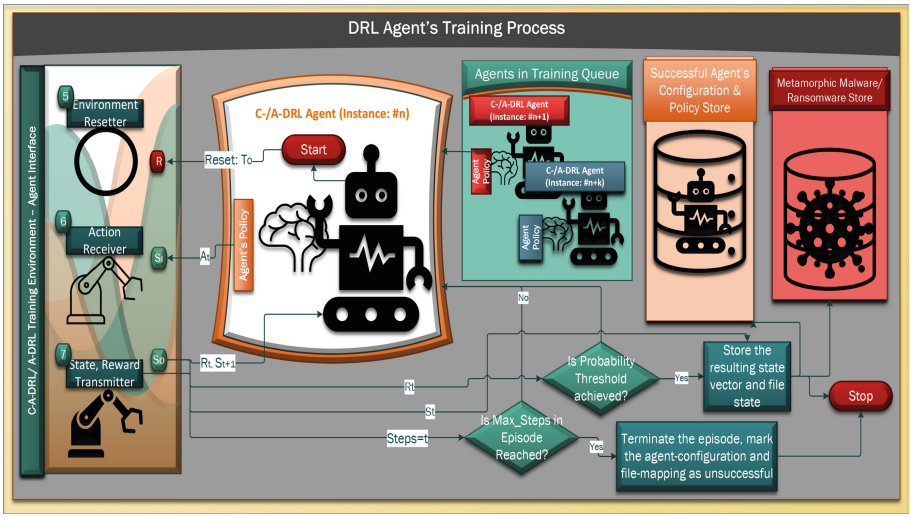

Fig. 2. DRL system's agent interaction with the environment.

given in Eq. 7, in which J is the policy-utility function that needs to be maximized and to maximize it, we need to compute the gradient ∇ of the policy-utility function. This requires finding expectancy $\mathbb{E}$ over different trajectories τ under this policy. Here $r(\tau)$ represents reward in trajectories parameterized by τ over which the expectancy is computed, and the one which is updated based on the resulting expectancy, and thereby the gradient of the policy-utility function.

$$\nabla_\theta(J_\theta) = \mathbb{E}_{\tau\sim\pi_\theta(\tau)}[\nabla_\theta \log \pi_\theta(\tau) r(\tau)] \tag{7}$$

A very potent policy-based algorithm Trust Region Policy Optimization (TRPO) [25] instead samples expectancy from these trajectories differently as $\pi_{\theta\prime}$, and π_θ, and use their ratio as an additional term in the gradient of the utility function. Such a gradient is given in Eq. 8.

$$\nabla_{\theta\prime}(J_{\theta\prime}) = \mathbb{E}_{\tau\sim\pi_\theta(\tau)}[\sum_{t=1}^{T} \nabla_{\theta\prime} \log \pi_{\theta\prime}(\prod_{t\prime=1}^{t} \frac{\pi_{\theta\prime}}{\pi_\theta})(\sum_{t\prime=t}^{T} r)] \tag{8}$$

To define the trust region and ensure that the updates are confined to this region, TRPO adds additional penalties to the optimization function to make the updates nearly monotonically, improving the utility function. With this, the optimization function for the gradient of J is given in Eq. 9.

$$\max_\theta \hat{\mathbb{E}}_t[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta old}(a_t|s_t)} \hat{A}_t - \beta.\mathrm{KL}[\pi_{\theta old}(.|s_t), \pi_\theta(.|s_t)]] \tag{9}$$

Here, a_t is the action corresponding to the states s_t at time t, r_t is the instantaneous reward at time t and $\hat{A}_t$ is the Advantage. The Eq. 9 parameterize the trust-region penalties with a coefficient β. This could lead to inefficient optimization, as before computing any policy updates, complex computations for establishing an optimal value of β update are required. Therefore, for optimization computation, instead of the optimizable parameter β based penalty, a Kullback Leibler (KL) divergence-based approach is used. This approach directly computes the KL divergence between the distributions of $\pi_{\theta\prime}$ (distribution of trajectories sampled from the current policy that needs to be updated) and π_θ (distribution of the trajectories importance-sampled from previous policies) as a proxy for the limiting conditions for applicability of the trust-region bounds. The simpler optimization problem with KL constraints is given in Eq. 10.

$$\begin{aligned} &\max_{\theta} \hat{\mathbb{E}}_t[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta old}(a_t|s_t)}\hat{A}_t] \\ &\text{subject to } \hat{\mathbb{E}}_t[\mathrm{KL}[\pi_{\theta old}(.|s_t), \pi_\theta(.|s_t)]] \leq \delta. \end{aligned} \tag{10}$$

The PPO algorithm offers a more efficient solution to implement the KL penalty constraint for optimization under the trust-region bounds. The PPO algorithm removes the KL penalties from the optimization function and simplifies the associated updates. Instead of using the KL penalties, it clips the surrogate objective as given in Eq. 11, ensuring that the updates are not completely unbound.

$$L^{CPI}(\theta) = \hat{\mathbb{E}}_t[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta old}(a_t|s_t)}\hat{A}_t] = \hat{\mathbb{E}}_t[r_t(\theta)\hat{A}_t]. \tag{11}$$

The original surrogate objective L^{CPI} in the 'clipped' form as used in TRPO can be reformulated and is given in the Eq. 12, where, ϵ (with a default value of 0.2) is an adjustable hyper-parameter.

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t[\min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)]. \tag{12}$$

Alternatively, Eq. 13 provides another mechanism as a replacement for the KL divergence-based penalty for optimization. In this alternative mechanism, instead of replacing the optimizable β parameter with KL penalties, an adaptive β penalty is used. Experimental results, though, favour the clipped penalty variant (Eq. 12) over the adaptive penalty variant as given in Eq. 13. Hence, most practical applications of PPO default to the clipped KL penalty forms for the implementation of the PPO algorithm. We also use this form of PPO algorithm implementation as the agents for both the A-DRL and C-A-DRL systems.

$$\begin{aligned} &\beta = \begin{cases} \beta/2, & \text{if } d \leq d_{targ}/1.5 \\ \beta \times 2, & \text{if } d \geq d_{targ} \times 1.5 \\ \beta & \text{otherwise} \end{cases} \\ &\text{where, } d = \hat{\mathbb{E}}_t[KL[\pi_\theta(.|s_t), \pi_{\theta old}(.|s_t)]] \end{aligned} \tag{13}$$

PPO uses the popular actor-critic [12,23] framework for implementation, with options of DL models for service as actor and critic. We use a DNN with hidden layers for both the A-DRL's and C-A-DRL's actor and critic networks. In our chosen DNN network, each hidden layer consists of 64 neurons and uses a 'tanh' activation function.

6 Dataset and Challenger Detection System

Threat experts identify the threat actor by discovering the modus operandi of the threat vector and the attack surface and correlating these with similar discoveries existing in historical databases. Often to evade quick detection, the malware developer prefers to create metamorphic variants from an older malware repository which preferably is neither public nor available in any recent private databases of the malware defenders; or is otherwise not easy to correlate with. While metamorphism enables the malware to evade detection even from the classifiers trained on their base variants, the lack of recency effect gives the malware developers an opportunity to hide their tracks and avert an expedited discovery of compromise. Therefore, to mimic such a scenario, we use an old. Still, a popular standardized malware dataset named Malicia [16], which has different types of malware samples, viz. ransomware, viruses, etc., belonging to different generations and families; and is also not public anymore (potentially to prevent malware developers from using samples from it to perpetuate any further actual attack). Therefore, this dataset is ideal for the intended purpose. Our data collection, processing, and base classifier training and selection process is illustrated in Fig. 3.

Using this dataset, the A-DRL system attempt to generate malware that could not be detected by a challenger malware detection system which is pre-trained using a dataset that contains its base variants. But to avoid any bias related to the challenger system, we first experiment with multiple challenger systems and identify the best one to embed in the A-DRL/C-A-DRL environments. The challenger system is described and evaluated in Subsect. 6.1.

6.1 Multiple Challenger Malware Detection Systems

We use three series of classifiers, namely the ML, the sequential DL architectures based Deep Neural Networks (DL-DNN), and the recurrent DL architecture based Long-Short-Term-Memory networks (DL-LSTM). With each series, we use compatible and appropriate feature selection/ reduction/ transformation algorithms. From the ML classifier series, we use the Random Forest (RF) classifier, which is one of the most potent classical ML classifiers used for malware detection [31]. With RF, we use variance threshold (VT) as the feature selection/reduction mechanism. For the DL-DNN classifier series, we use the Deep Stacked Auto-Encoders (AE) for dimension reduction [33]. For the DL-LSTM series, besides using feature transformation/reduction using embeddings (AE-based), we need some additional pre-processing to reduce the variation

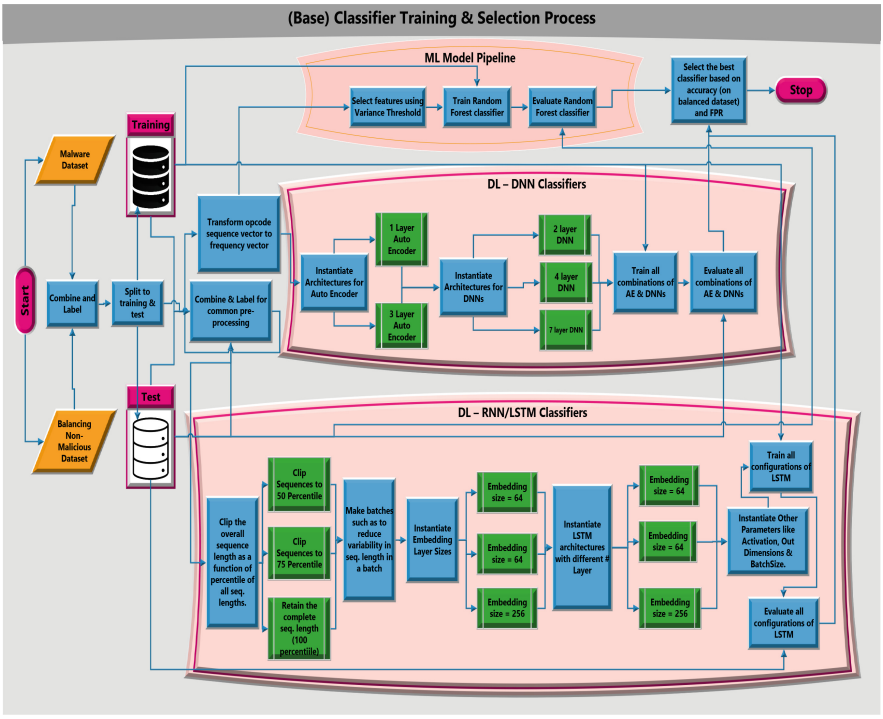

Fig. 3. Flow of the data processing, challenger classifier training, and selection process.

in the sequence length across the samples included in each batch of training. This is essential, as LSTMs are sensitive toward excessive pruning/padding of sequences within a training batch. Next, we train each of the models using (several batches/epochs) of the training dataset. For DL models, the training is stopped on convergence to avoid over-fitting (early stopping). Then, we evaluate the different models on the described metrics using the validation dataset. The performance of the classical RF classifier is given in Table 1. The RF model, with or without the VT augmentation, works well. With VT feature-selection enhancement, the accuracy is slightly better. Also, VT decreases the overall computational complexity of the model due to the reduced size of the feature vector. We chose this configuration for embedding inside the A-DRL and C-A-DRL environment as the challenger malware detection system.

We also experimented with different combinations of the AE, DL-DNN, and DL-LSTM classifiers. The obtained performance is given in the Table 2 and 3. From the considered hyper-parameters and their levels, many combinations of LSTM network configurations are possible. But all configurations did not converge optimally and hence are not reported. Also, some configurations did not lead to superior performance on monotonically varying a given parameter, so these configurations are also not reported.

Table 1. Performance of the classical RF in the pipeline.

Feature	Classification	Acc.	FPR
None	RF	0.994	0.0022
VT	RF	0.995	0.0025

Table 2. Performance of DNN with AE and different layers in the pipeline.

AE Layers	DNN Layers	Acc.	FPR
1	2	0.965	0.0242
1	4	0.977	0.0129
1	7	0.978	0.0165
3	2	0.905	0.0889
3	4	0.928	0.0751
3	7	0.931	0.0897

Table 3. Performance of LSTM classifier in pipelines for the considered hyper-parameters.

No.	Seq.Ln.	Emb.Sz.	No.Lyr	Out.Dim.	Act.	DO	Bt.Sz.	Val.Loss	Acc.
1	100P	64	2	100P	tanh	0.3	32 6	0.7014	0.5625
2	75P	256	2	75P	tanh	0.3	32	0.6933	0.5312
3	75P	256	1	256	sigmoid	0.5	128	0.6989	0.5227
4	75P	128	4	75P	tanh	0.3	64	0.6881	0.5781
5	75P	128	2	75P	tanh	0.3	128	0.6886	0.5625
6	75P	128	2	256	sigmoid	0.5	128	0.7094	0.4218
7	75P	128	1	256	sigmoid	0.5	128	0.7005	0.5111
8	50P	256	1	256	sigmoid	0.5	128 5	0.7082	0.5117
9	50P	128	1	256	sigmoid	0.5	128 4	0.6826	0.5319

Based on these results, we chose the entire malware detection pipeline of the VT-enhanced RF model (along with the feature extraction module). We plugged it into our A-DRL and C-A-DRL training environments to suggest appropriate rewards for the training of the corresponding DRL agent. The added advantage of opting for the VT and ML-RF combination classification pipeline is that both the algorithms are extremely robust to both the adversarial-DL attacks and the gradient attacks and hence could not be fooled by random perturbation insertions. This scenario is ideal for training a DRL agent to favour learning to generate a metamorphic malware instead of trivially adding noise to it to evade detection. Further, we also use additional checks in our system (discussed in Sect. 7 and 8) to ensure that the agents are learning to infuse metamorphism instead of exploiting other tricks even beyond noise insertions.

7 A-DRL Experiments and Result Analysis

We conducted multiple experiments in two phases. In the first phase, we instantiated several A-DRL agents with a custom-designed environment (discussed in Sect. 4). The environment encapsulates the entire malware detection pipeline (Sect. 6.1). The interaction between the A-DRL agent and the environment resulted in augmenting a repository of metamorphic malware/ransomware, as shown in Fig. 2. In each episode, an A-DRL agent is offered to obfuscate and infuse evasive metamorphism into randomly chosen (existing) ransomware. For each of the ransomware offered to the agent to metamorph, the associated classifier pipeline is pre-verified to detect the base variant of the same un-ambiguously as malicious with a probability $\mathbb{P}_{Detection} > 0.95$. Each episode consists of several steps of interaction across multiple episodes between an agent and a randomly instantiated environment. In each episode, the agent was allowed a maximum of 1,500 turns/steps to generate an undetectable metamorphic variant of an existing (base) ransomware.

The experimental results of A-DRL agent training (Fig. 4) show a rising trend in the evasion probability ($\mathbb{P}_{Metamorphism}$) that the agent could achieve, which indicates that the agent is learning a generalized policy to achieve metamorphism. Also, from the results, we observe that in less than 100 episodes, the agent was briefly able to reach the desired threshold of $\mathbb{P}_{Metamorphism} > 0.5$ for the first time. In this instance, the malware classification pipeline started classifying it as non-malicious. Further, at ≈150 episode, the agent has learned an effective generalized policy to evade the malware classifier consistently and repeatedly in invariably all subsequent episodes. In addition, we also observe that as the number of episodes progress, the total discounted reward consistently increases. This again indicates that the agent is learning a generalized policy that is applicable for infusing metamorphism in any unseen (not known to the agent) ransomware as well.

The trends of instantaneous and discounted rewards received by this agent.

Often, DRL agents end up learning a trivial trick/exploits to accomplish the rewards instead of learning a desired generalized policy that can be replicated across different scenarios/ransomware. In the context of the A-DRL system, one such trivial exploit could be to transform all ransomware into a standardized feature vector that the agent knows (has learned during training) that will be reported as non-malicious by the classifier. Learning such undesired exploits could defeat the objective of the system because then the evasion cannot be completely attributed to infusion of metamorphism/obfuscations, and such exploits could even modify the file functionality. Therefore, to ensure that the A-DRL learned a generalized and effective policy to infuse metamorphism and not a trivial trick/exploit, we compute the similarity (Pearson Correlation [4]) of the final obfuscated feature vector with the original malware that the environment was instantiated with when the episode began (Fig. 5). A higher similarity indicates that the A-DRL agent has learned to insert minimum junk code(s) in that specific malware so as to enable it to evade the detection successfully, and

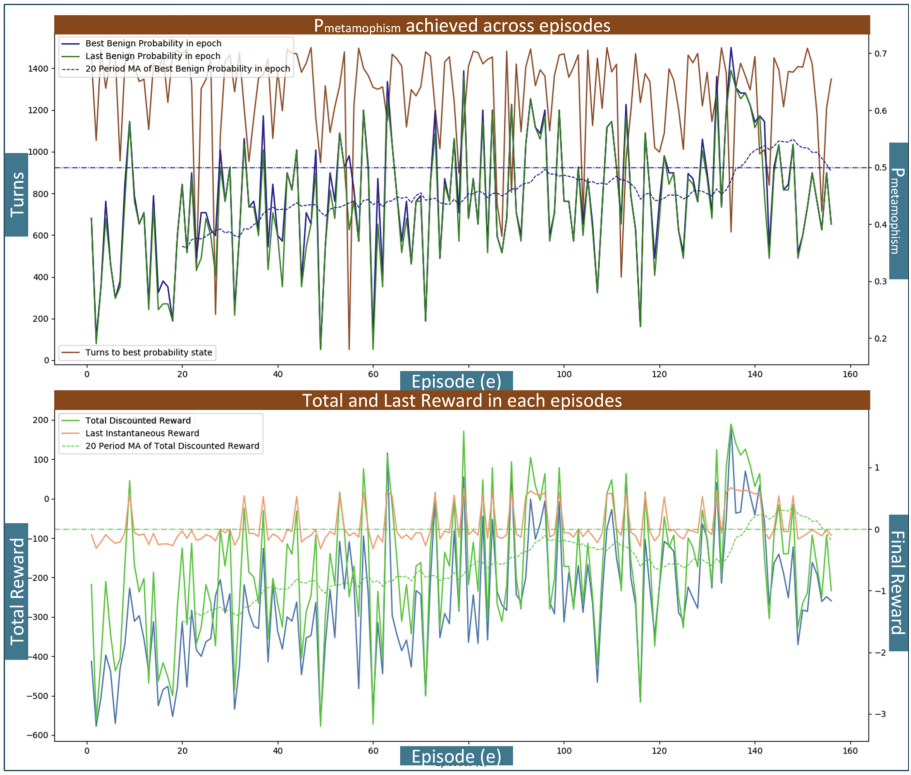

Fig. 4. Training statistics of an A-DRL agent.

the A-DRL agent attained a remarkably elevated level of similarity such that ransomware achieves a decent level of metamorphism.

The feature vectors of the generated metamorphic variant of the ransomware that can successfully evade detection are stored in a repository of the metamorphic ransomware that is shared with the training environment of the C-A-DRL. For each existing (base) ransomware, there could be multiple metamorphic variants of the same as generated by different A-DRL agents using diverse action policies. Also, since these variants are generated using diverse policies (from different agents), such metamorphic variants offer a lot of variety to the C-A-DRL agents to learn a generalized policy of normalizing diverse types of obfuscations and metamorphism. Therefore, next, in the second phase of experiments, we train the C-A-DRL agents using the custom C-A-DRL environment.

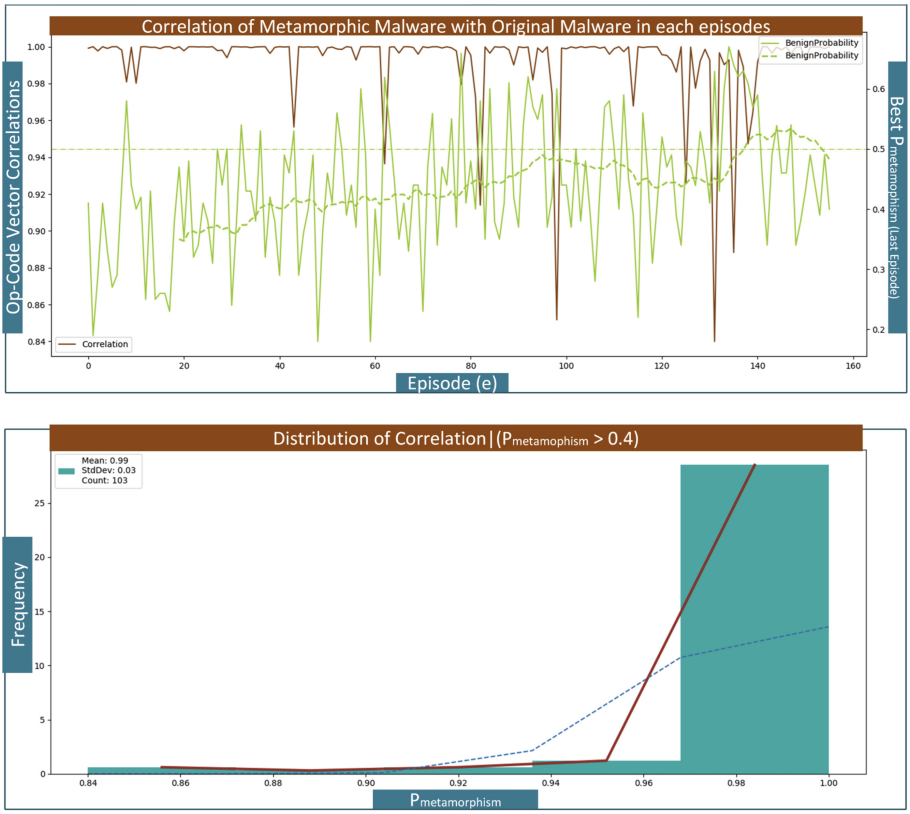

Fig. 5. Similarity trend across the episode and the achieved metamorphism for an A-DRL agent.

8 C-A-DRL Experiments and Analysis

The C-A-DRL agents learn to normalize the obfuscations generated by junk-code insertions from the metamorphic samples generated by A-DRL. Now we intend to train the C-A-DRL agents to learn generalized metamorphism-normalization policy. Therefore, like the experiments with the A-DRL system, we train the C-A-DRL agents over multiple episodes. In each episode, the C-A-DRL agent has been given a randomly selected variant of one of the ransomware to normalize so that it can be subsequently detected by the challenger detection system. Compared to the A-DRL, the C-A-DRL agent took more episodes exponentially to converge. Although the incremental improvement across episodes is slow, it increases monotonically. The experimental analysis shows that the C-A-DRL agent could achieve a detection probability post-metamorphic normalization of $P_{Metamorphism} > 0.6$. Still, it took ≈1000 episodes to improve the $P_{Metamorphism}$ by 0.1 asymptotically.

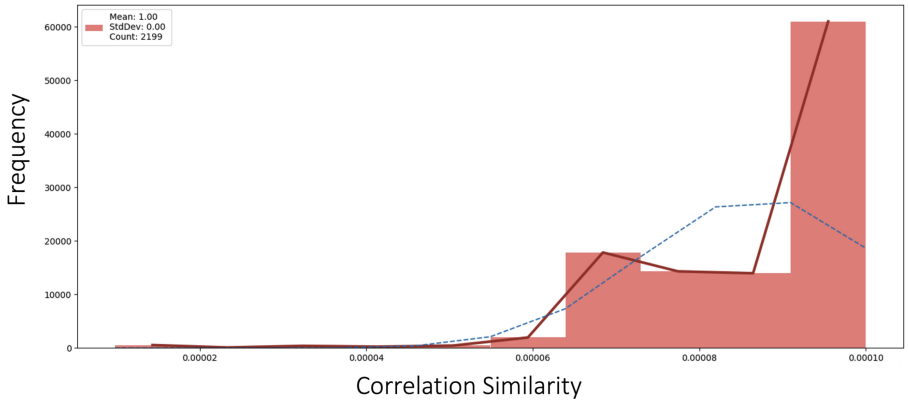

Fig. 6. Similarity of the feature-vector of the normalized ransomware variant generated by the C-A-DRL system with its metamorphic variant generated by the A-DRL system.

Next, we qualify the learning of the C-A-DRL agent through different checks to ensure that it is not leveraging any trivial exploits to attain higher rewards and is genuinely learning a desired generalized metamorphism-normalization policy. With reasoning like that in the case of A-DRL, we can claim that C-A-DRL is also learning a generalized policy for normalization and not just a trivial trick. On a similar line, like that in the case of A-DRL, we can claim that C-A-DRL is also learning a generalized policy for normalization and not just a trivial trick. But just the rising trends in the $P_{Metamorphism}$ for A-DRL and corresponding rising trends in the $P_{Normalization}$ for the C-A-DRL do not guarantee that we have successfully created a malware/ ransomware normalization system. In addition, a ransomware normalization should not only lead to a rise in $P_{Normalization}$ but shall also revert the ransomware to its base variant. So that normalization will ensure that the observed $P_{Normalization}$ is classifier agnostic, and if the malware could be normalized to its original variant form, then any classifier that can detect the base variant can also be able to detect the metamorphic variant post normalization. Therefore, to implement this additional check on the C-A-DRL system, we computed the similarity between the normalized variant (normalized by C-A-DRL) of the metamorphic ransomware and its existing base variant from which A-DRL produced the specific metamorphism. Here also we observe a high degree of similarity ($correlation \approx 1.0$) which indicates that we have successfully developed a metamorphic malware/ransomware normalization/de-obfuscation system.

Similar to the discussed agent analysis for which we have produced the detailed training and similarity statistics, many more A-DRL and C-A-DRL agents were trained to ensure that we exhaustively cover multiple types of obfuscation and metamorphism strategies and explore effective ways to normalize them. The analysis of the aggregate distribution of $\mathbb{P}_{Normalization}$ has been achieved across different metamorphic ransomware variants, and we find that the

system achieved a mean $\mathbb{P}_{Normalization} \approx 0.6$, and for most of the variants the system succeeded in normalizing them enough ($\mathbb{P}_{Normalization} \approx 0.5$) to ensure that they can not evade the existing classifier.

9 Conclusion

We used policy-based A-DRL to successfully create the first-ever metamorphic ransomware using obfuscations at the assembly code level. Next, we trained several such A-DRL agents to create a swarm of such metamorphic ransomware to perpetuate a *zero-day metamorphic swam attack*. The ransomware detection system we used had been shown to be robust against perturbations and gradient-based attacks and hence cannot be evaded using popular adversarial-DL techniques. Next, we also show that our A-DRL system generates metamorphic ransomware by learning a generalized policy to infuse obfuscations in any ransomware and does not exploit any trivial trick/exploit for evasion. Further, we developed a C-A-DRL system as a counter-adversary to our A-DRL system. This system learned to solve the problem of normalizing metamorphic malware at the assembly level. By doing so, the C-A-DRL system was able to normalize metamorphic ransomware to their base variants, which existing detection systems can detect. Moreover, we show that the C-A-DRL successfully enhances the detection probability of the metamorphic ransomware by successfully normalizing its underlying metamorphism and does not rely on trivial exploits for this accomplishment. Also, due to the inherent complexity of the MDP, no popular RL/DRL techniques could provide a feasible solution. Therefore, we used advanced PPO algorithm-based agents that efficiently handle such a large action space and guarantee monotonic improvement.

References

1. Baysa, D., Low, R.M., Stamp, M.: Structural entropy and metamorphic malware. J. Comput. Virol. Hacking Tech. **9**(4), 179–192 (2013)
2. Behera, C.K., Bhaskari, D.L.: Different obfuscation techniques for code protection. Procedia Comput. Sci. **70**, 757–763 (2015)
3. Bilge, L., Dumitraş, T.: Before we knew it: an empirical study of zero-day attacks in the real world. In: ACM Conference on Computer and Communications Security (CCS), pp. 833–844 (2012)
4. Freedman, D., Pisani, R., Purves, R.: Statistics. Norton & Company (1998)
5. Gazet, A.: Comparative analysis of various ransomware virii. J. Comput. Virol. **6**(1), 77–90 (2010)
6. Goodfellow, I., et al.: Generative adversarial networks. Commun. ACM **63**(11), 139–144 (2020)
7. Grosse, K., Papernot, N., Manoharan, P., Backes, M., McDaniel, P.: Adversarial perturbations against deep neural networks for malware classification. arXiv preprint arXiv:1606.04435 (2016)
8. van Hasselt, H., Guez, A., Silver, D.: Deep reinforcement learning with double q-learning. CoRR abs/1509.06461 (2015)

9. Kolosnjaji, B., et al.: Adversarial malware binaries: evading deep learning for malware detection in executables. CoRR abs/1803.04173 (2018)
10. Madry, A., Makelov, A., Schmidt, L., Tsipras, D., Vladu, A.: Towards deep learning models resistant to adversarial attacks. In: International Conference on Learning Representations (ICLR) (2018)
11. Meng, D., Chen, H.: Magnet: a two-pronged defense against adversarial examples. In: ACM SIGSAC Conference on Computer and Communications Security, pp. 135–147 (2017)
12. Mnih, V., et al.: Asynchronous methods for deep reinforcement learning. CoRR abs/1602.01783 (2016)
13. Mnih, V., et al.: Playing Atari with deep reinforcement learning. CoRR abs/1312.5602 (2013)
14. Mnih, V., et al.: Human-level control through deep reinforcement learning. Nature **518**, 529–533 (2015)
15. Mohurle, S., Patil, M.: A brief study of wannacry threat: ransomware attack. Int. J. Adv. Res. Comput. Sci. **8**(5), 1938–1940 (2017)
16. Nappa, A., Rafique, M.Z., Caballero, J.: The MALICIA dataset: identification and analysis of drive-by download operations. Int. J. Inf. Secur. **14**, 15–33 (2015)
17. Papernot, N., McDaniel, P., Jha, S., Fredrikson, M., Celik, Z.B., Swami, A.: The limitations of deep learning in adversarial settings. In: IEEE European Symposium on Security and Privacy (Euro S&P), pp. 372–387 (2016)
18. Rathore, H., Bandwala, T., Sahay, S.K., Sewak, M.: Adversarial robustness of image based Android malware detection models. In: Krishnan, R., Rao, H.R., Sahay, S.K., Samtani, S., Zhao, Z. (eds.) SKM 2021. CCIS, vol. 1549, pp. 3–22. Springer, Cham (2022). https://doi.org/10.1007/978-3-030-97532-6_1
19. Rathore, H., Nikam, P., Sahay, S.K., Sewak, M.: Identification of adversarial Android intents using reinforcement learning. In: International Joint Conference on Neural Networks (IJCNN), pp. 1–8. IEEE (2021)
20. Rathore, H., Samavedhi, A., Sahay, S.K., Sewak, M.: Robust malware detection models: learning from adversarial attacks and defenses. Forensic Sci. Int.: Digit. Invest. **37**, 301183 (2021)
21. Rathore, H., Samavedhi, A., Sahay, S.K., Sewak, M.: Towards adversarially superior malware detection models: an adversary aware proactive approach using adversarial attacks and defenses. Inf. Syst. Front. **25**, 567–587 (2022)
22. Rathore, H., Sasan, A., Sahay, S.K., Sewak, M.: Defending malware detection models against evasion based adversarial attacks. Pattern Recogn. Lett. **164**, 119–125 (2022)
23. Rathore, H., Sharma, S.C., Sahay, S.K., Sewak, M.: Are malware detection classifiers adversarially vulnerable to actor-critic based evasion attacks? EAI Endorsed Trans. Scalable Inf. Syst. **10**(1), e6 (2023)
24. Ren, K., Zheng, T., Qin, Z., Liu, X.: Adversarial attacks and defenses in deep learning. Engineering **6**(3), 346–360 (2020)
25. Schulman, J., Levine, S., Abbeel, P., Jordan, M., Moritz, P.: Trust region policy optimization. In: International Conference on Machine Learning (ICML), pp. 1889–1897. PMLR (2015)
26. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. CoRR abs/1707.06347 (2017)
27. Sewak, M., Sahay, S.K., Rathore, H.: Value-approximation based deep reinforcement learning techniques: an overview. In: IEEE 5th International Conference on Computing Communication and Automation (ICCCA), pp. 379–384 (2020)

28. Sewak, M.: Deep Q Network (DQN), double DQN, and dueling DQN. In: Sewak, M. (ed.) Deep Reinforcement Learning, pp. 95–108. Springer, Singapore (2019). https://doi.org/10.1007/978-981-13-8285-7_8
29. Sewak, M., Karim, M.R., Pujari, P.: Practical Convolutional Neural Networks: Implement Advanced Deep Learning Models Using Python. Packt Publishing (2018)
30. Sewak, M., Sahay, S.K., Rathore, H.: Policy-approximation based deep reinforcement learning techniques: an overview. In: Joshi, A., Mahmud, M., Ragel, R.G., Thakur, N.V. (eds.) Information and Communication Technology for Competitive Strategies (ICTCS 2020). LNNS, vol. 191, pp. 493–507. Springer, Singapore (2022). https://doi.org/10.1007/978-981-16-0739-4_47
31. Sewak, M., Sahay, S.K., Rathore, H.: Comparison of deep learning and the classical machine learning algorithm for the malware detection. In: 19th IEEE/ACIS SNPD 2018, pp. 293–296. IEEE (2018)
32. Sewak, M., Sahay, S.K., Rathore, H.: Assessment of the relative importance of different hyper-parameters of LSTM for an IDS. In: IEEE Region 10 Conference (TENCON), pp. 414–419. IEEE (2020)
33. Sewak, M., Sahay, S.K., Rathore, H.: An overview of deep learning architecture of deep neural networks and autoencoders. J. Comput. Theor. Nanosci. **17**(1), 182–188 (2020)
34. Sewak, M., Sahay, S.K., Rathore, H.: Adversarialuscator: an adversarial-DRL based obfuscator and metamorphic malware swarm generator. In: International Joint Conference on Neural Networks (IJCNN), pp. 1–9. IEEE (2021)
35. Sewak, M., Sahay, S.K., Rathore, H.: DRLDO: a novel DRL based de-obfuscation system for defence against metamorphic malware. Def. Sci. J. **71**(1), 55–65 (2021)
36. Sewak, M., Sahay, S.K., Rathore, H.: DRo: a data-scarce mechanism to revolutionize the performance of DL-based Security Systems. In: IEEE 46th Conference on Local Computer Networks (LCN), pp. 581–588. IEEE (2021)
37. Sewak, M., Sahay, S.K., Rathore, H.: Deep reinforcement learning for cybersecurity threat detection and protection: a review. In: Krishnan, R., Rao, H.R., Sahay, S.K., Samtani, S., Zhao, Z. (eds.) SKM 2021. CCIS, vol. 1549, pp. 51–72. Springer, Cham (2022). https://doi.org/10.1007/978-3-030-97532-6_4
38. Sutton, R.S., McAllester, D., Singh, S., Mansour, Y.: Policy gradient methods for reinforcement learning with function approximation. In: International Conference on Neural Information Processing Systems, pp. 1057–1063. MIT Press (1999)
39. Usama, M., Asim, M., Latif, S., Qadir, J., Ala-Al-Fuqaha: Generative adversarial networks for launching and thwarting adversarial attacks on network intrusion detection systems. In: 15th International Wireless Communications Mobile Computing Conference (IWCMC), pp. 78–83 (2019)
40. Wang, Z., Schaul, T., Hessel, M., Van Hasselt, H., Lanctot, M., De Freitas, N.: Dueling network architectures for deep reinforcement learning. In: International Conference on International Conference on Machine Learning, pp. 1995–2003 (2016)