Defence Science Journal, Vol. 71, No. 1, January 2021, pp. 55-65, DOI : 10.14429/dsj.71.15780

DRLDO: A Novel DRL based De-obfuscation System for Defence Against Metamorphic Malware

Mohit Sewak[*], Sanjay K. Sahay[#], and Hemant Rathore[#]

[*]*Security and Compliance Research, Microsoft, Hyderabad, India*
[#]*Department of Computer Science & Information, Goa Campus, BITS Pilani, Goa - 403 726, India*
[*]*E-mail: mohit.sewak@microsoft.com*

ABSTRACT

In this paper, we propose a novel mechanism to normalise *metamorphic* and *obfuscated* malware down at the *opcode level* and hence create an advanced metamorphic malware de-obfuscation and defence system. We name this system as DRLDO, for deep reinforcement learning based de-obfuscator. With the inclusion of the DRLDO as a sub-component, an existing intrusion detection system could be augmented with defensive capabilities against 'zero-day' attack from obfuscated and metamorphic variants of existing malware. This gains importance, not only because there exists no system till date that use advance DRL to intelligently and automatically normalise obfuscation down even to the opcode level, but also because the DRLDO system does not mandate any changes to the existing IDS. The DRLDO system does not even mandate the IDS' classifier to be retrained with any new dataset containing obfuscated samples. Hence DRLDO could be easily retrofitted into any existing IDS deployment. We designed, developed, and conducted experiments on the system to evaluate the same against multiple-simultaneous attacks from obfuscations generated from malware samples from a standardised dataset that contain multiple generations of malware. Experimental results prove that DRLDO was able to successfully make the otherwise undetectable obfuscated variants of the malware detectable by an existing pre-trained malware classifier. The detection probability was raised well above the cut-off mark to 0.6 for the classifier to detect the obfuscated malware unambiguously. Further, the de-obfuscated variants generated by DRLDO achieved a very high correlation (of ≈ 0.99) with the base malware. This observation validates that the DRLDO system is actually learning to de-obfuscate and not exploiting a trivial trick.

Keywords: Adversarial artificial intelligence; Deep reinforcement learning; Metamorphic malware; De-obfuscation

1. INTRODUCTION

Metamorphism provides malware an effective mechanism of evading an Intrusion Detection Systems (IDS). The different metamorphic variants of a metamorphic malware are functionally equivalent, but their internal structures or source codes may differ. Code obfuscation methods like dead-code insertion are widely used for developing metamorphic malware[1]. The problem of de-obfuscation (/normalisation) at the opcode level, when transformed into a markov decision process (MDP), leads to a reinforcement learning (RL) task that involves high cardinality action space. Most of the popular RL or deep reinforcement learning (DRL)[2] agent algorithms like the Deep Q Networks (DQN)[3], or even its enhanced variants like the Double[4] or Dueling Deep Q Networks[5] could not efficiently learn an optimal policy under such constraints[6]. These systems also suffer from sample-inefficiency when applied to complex MDP. Such MDP(s) requires sophisticated DRL agents to effectively learn the policy and find effective and sample-efficient solutions to the given MDP. Training such DRL agents also requires sufficient data to ensure stable and robust learning. In this paper, we present a novel method for de-obfuscation of advanced metamorphic, oligomorphic, and polymorphic malware using DRL. We designed and developed a working system named DRLDO (deep reinforcement learning based de-obfuscator). The objective of the DRLDO system is to train DRL agents that can perform the task of transforming/normalising/de-obfuscating the obfuscated malware's opcode sequence features. Doing so a DRLDO based system could be fitted inside an IDS before the malware-classification system. The DRLDO system would de-obfuscated the incoming candidate file's features sufficiently so as the IDS' classification-system could detect these otherwise un-detectable feature-vectors correctly (as malicious) upon de-obfuscation without mandating any re-training or re-calibration of the IDS. Additionally, it is desirable that the transformed feature-vectors as generated by the DRLDO are similar (demonstrate high correlation between feature-vectors) to that of the original malware's to ensure that the DRLDO system is also compatible with IDS incorporating multinomial-classification-system and other advanced sub-systems that requires to identify the specific strain of malware for further processing. We conducted experiments with the developed system to gauge its effectiveness against multiple-simultaneous attacks from different obfuscated variants of malware extracted

Received : 30 April 2020, Revised : 14 September 2020
Accepted : 06 October 2020, Online published : 01 February 2021

from a standardised malware dataset[7]. Experimental results prove that the DRLDO system could normalise/ transform/ de-obfuscated the obfuscated malware's feature-vectors such that these malware could subsequently be correctly detected by existing IDS (which had failed to detect the metamorphic instance correctly) without mandating any re-training. Another significant achievement of the DRLDO system is that the de-obfuscations produced from it resulted in an opcode frequency vector which were very similar to that of the original malware variant's opcode frequency vector. This observation adds further credibility to the assertion that the DRLDO system is actually trying to de-obfuscate the malware's opcode frequency vector (of any junk opcode/ instruction insertion) and not just finding a trivial trick to ensure that IDS start detecting the metamorphic instances as malicious.

2 RELATED WORK

Recently there has been a lot of interest to improve the performance of IDS against unseen intrusions especially in network traffic and botnet attack areas. Most of the initial efforts in this regard were inspired by the generative adversarial networks (GAN) as formed by the combination of two different convolution neural networks (CNNs)[8] where the detection feedback from one network called the 'Discriminator CNN' (denoted as $'\mathcal{D}'$) is used to train the other CNN network called the 'Generator CNN' (denoted as $'\mathcal{G}'$). With the initial inspiration being drawn from the GAN networks of underlying CNN architecture, some aspects of network traffic were converted into a similar CNN map on which a GAN style methodology could be adopted to produce the necessary 'generative' perturbation by the 'Generator CNN' to create samples that the 'Discriminator' CNN could not identify correctly. This problem could be formulated as a min-max contest between $'\mathcal{D}'$ and $'\mathcal{G}'$, where, $'\mathcal{D}'$ is trying to maximise the cross-entropy error of detection of samples produced by $'\mathcal{D}'$ using $'\mathcal{G}'$ and $'\mathcal{G}'$ is trying to minimise it attractively. This could be defined mathematically as the optimisation function in equation 1:

$$\min_{\mathcal{G}} \max_{\mathcal{D}} E_p(x) \log \mathcal{D}(x) + E_p(z) \log\left(1 - \mathcal{D}\left(\mathcal{G}(z)\right)\right) \tag{1}$$

Since this is a differentiable equation, it could be optimised to train and weight of the $'\mathcal{D}'$ and $'\mathcal{G}'$ networks that converges gradually and then the samples from $'\mathcal{D}'$ are used for the intended purpose. To create an undetectable malicious entity (file or network traffic), the network $'\mathcal{D}'$ could be replaced by the corresponding IDS's classifier's approximation function. Such approximation function could be any Deep Learning (DL) network. Most of these systems work to create generative data distributions that supposedly mimic an undetectable malicious entity (file or network traffic). While using such a system to create malicious traffic/ botnet, the $'\mathcal{D}'$ network could be replaced by the corresponding IDS's classifier's associated trained DL network, to learn to create perturbations in the distribution of existing (malicious) data and creating new (malicious) data distributions that is unknown to the existing IDS ($'\mathcal{D}'$ network). Some prominent example of such approach could be MalGAN[9], IDSGAN[10], and by Usama[11], *et al.*. There are some non-GAN approaches which are based on differentiable objective functions, these are typically known as 'Gradient attack' based approaches. An example of a similar implementation is the fast gradient sign method (FGSM)[12]. Additionally, there are some other non-differentiable objective based methods in the area of reinforcement learning[13,14] that have been explored. These methods are designed with an underlying assumption that the distribution of the new data generated from the $'\mathcal{G}'$ network of these systems is significantly different form the ones that the IDS ($'\mathcal{D}'$ network) is trained on. Thus, on retraining the IDS on the data coming from this new distribution (in combination with their original training data) can improve the overall IDS system's response against an actual new attack for which the real data does not exist. But as pointed out in[9] these methods, especially the ones based on GANs make re-training of IDS ineffective, and others may even make the IDS over-fit[13] if trained on such generated data thus reducing their effectiveness. Therefore, it is not sufficient to just create a system that could generate malware samples that the IDS could not detect and hope that using these samples the IDS could be improved substantially in its ability to avert attacks from unknown intrusion attacks especially those by the complex obfuscations of existing malware. Also, it is not optimal to retrain the IDS with hypothetical data distributions generated with such perturbations, as this increases the risk of decreasing the effectiveness of the IDS on the actual existing malware detection. The above observation necessitates that instead of modifying the training of the existing IDS which is working well on the original/ un-obfuscated variants of the malware, the obfuscated variants of the threats need to be normalised to bring them close to the actual variant that the IDS was earlier trained upon and hence could probably detect.

3. DRLDO PROCESS FLOW

The process flow for the design of the DRLDO system is shown in Fig. 1. It has broadly 4 subsystems, namely:

- The obfuscated opcode repository and associated obfuscation generation system like the ADRLMMG (the detailed coverage of this system is out of scope of this paper)
- A pre-trained opcode frequency feature-vector generator and classification system (sub-components of existing IDS)
- A custom malware de-obfuscation training reinforcement learning environment and
- A DRL agent compatible with both the custom environment and also the type of reinforcement learning problem at hand (conceptually and mathematically).

In the Fig. 1 shown, the DRL agent interacts with the environment to train against *episodic* tasks. These tasks comprise of altering the presented opcode frequency vector. The opcode frequency vector is altered to achieve a net reduction of some of the selected opcode instructions, thus mimicking actions opposite to that in popular obfuscation techniques (i.e., of adding junk instructions). In each training step of every training episode, the environment receives and instruction from the DRL agent, and then transforms the opcode frequency structure as per the received instruction.

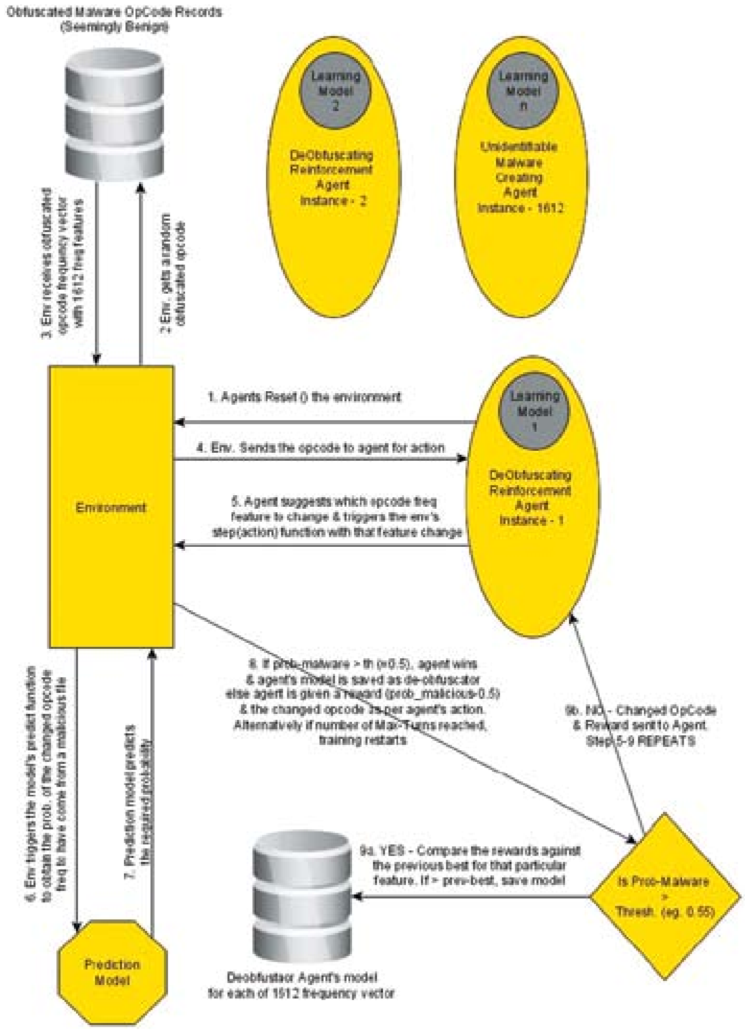
Obfuscated Malware OpCode Records
(Seemingly Benign)
Learning Model 2
DeObfuscating Reinforcement Agent Instance - 2
Learning Model n
Unidentifiable Malware Creating Agent Instance - 1612
3. Env receives obfuscated opcode frequency vector with 1612 freq features
2. Env. gets a random obfuscated opcode
1. Agents Reset () the environment
4. Env. Sends the opcode to agent for action
5. Agent suggests which opcode freq feature to change & triggers the env's step(action) function with that feature change
Environment
Learning Model 1
DeObfuscating Reinforcement Agent Instance - 1
8. If prob-malware > th (=0.5), agent wins & agent's model is saved as de-obfuscator else agent is given a reward (prob_malicious-0.5) & the changed opcode as per agent's action. Alternatively if number of Max-Turns reached, training restarts
9b. NO - Changed OpCode & Reward sent to Agent. Step 5-9 REPEATS
6. Env triggers the model's predict function to obtain the prob. of the changed opcode freq to have come from a malicious file
7. Prediction model predicts the required probability
9a. YES - Compare the rewards against the previous best for that particular feature. If > prev-best, save model
Is Prob-Malware > Thresh. (eg. 0.55)
Prediction Model
Deobfustaor Agent's model for each of 1512 frequency vector

Figure 1. DRL based de-obfuscation system - process flow.

The environment then uses the attached IDS to obtain the probability of the thus transformed opcode frequency vector to have come from a malicious file ($P_{malicious}$). A decent increment in $P_{malicious}$ from the initial state malicious detection probability (i.e., $P_{malicious} < 0.5$) of opcode frequency to a level where the classifier could subsequently detect the file instance as malicious (i.e. $P_{malicious} < 0.5$). The DRL agents train over multiple such training episodes to update and refine an action policy that could be applied on any obfuscated malware strain to de-obfuscate it. The trained DRL agent is used to create and store opcode frequency vectors representing abstractions that could be identified by existing detection systems as malware even when they could not identify the obfuscated variants of these malware in a situation typically posed under a zero-day attack. So, this system essentially works to negate a zero-day attack otherwise possible by using undetectable obfuscated instances of different malware variants. Multiple such DRL agents could be created and trained with varying degree of dissimilarity from other DRL agents trained on the same environment and subsequently producing probable de-obfuscation variants of the same malware that are dissimilar from each other at different levels. Such dissimilarities could range from the extremes of changing the underlying algorithm of the complete agent to just changing the random number seed of various instances of the same DRL agent. Working at the opcode level the process design is universal and could be implemented on malware of all popular systems ranging from Android, Windows, clients to Linux and Windows based servers. In production, this system could be implemented on any incoming file to these systems without having a prior assumption of the benign/ malicious nature of the file or its obfuscation status. Coupled with both the original opcode frequency vector and the probable de-obfuscated opcode frequency, either the existing classification system's detection potential could be enhanced, or more powerful single stage or hierarchical detection system be made.

3.1 Training Data and IDS used

The obfuscated (metamorphic) opcode frequency feature-vector of existing malware were produced by the ADRLMMG system which is an Adversarial Deep Reinforcement Learning based Metamorphic Malware Generator. The ADRLMMG and the DRLDO systems are isolated systems and they do not share weights and are unaware of the other's existence. The ADRLMMG system creates obfuscations of existing identified malware and stores the resultant opcode frequency vector of these obfuscations as the training data for any downstream de-obfuscation system like the DRLDO. The ADRLMMG system also maintains a dictionary of the original malware's identity and its opcode frequency feature-vector in a separate metadata for performance assessment of the downstream de-obfuscation system. The ADRLMMG system aims to mimics obfuscation techniques like junk instructions/ code insertion and also to preserves the functionality of the original malware and hence it uses strictly opcode additive action policy. Obfuscations of multiple malware and benign files were produced. The obfuscations of the malware that could not be identified by the existing IDS satisfactorily were retained and the remaining discarded. The original opcode frequency vector of each file was preserved for comparison with that of the final obfuscated variant as produced by the ADRLMMG and the resultant de-obfuscated version of each of the obfuscated variant as produced by the DRLDO systems. We choose the IDS system (including the pre-processing, feature selection and transformation and the classification sub-systems) as developed by[15-17] to augment it with Zero-Day-Defense[18] capabilities against metamorphic malware attack using the DRLDO system. The selected IDS had claimed to provide the best performance (with a combination of the highest accuracy accompanied with a very low false positive rates) over a standardised malware data[7] with mixed types and generation of malware. For the obfuscated malware samples, we used the same data on which the given IDS was trained (and claimed the associated performance). Since the chosen IDS is trained on the malware in this dataset and also claims to have performed well detecting these malware, we can safely assume all the malware in this dataset as 'existing' malware that are 'known' to the chosen IDS. We use the ADRLMMG system to produce multiple obfuscated variants of this data. Multiple obfuscated variants of each 'existing' malware could be generated. Each obfuscated variant thus produced was screened against the chosen IDS to ensure that it is incorrectly identifying the obfuscations of malware as non-malicious ($P_{non-malicious} < 0.5$) before using these obfuscated samples in our experiments with the DRLDO system and obtaining the corresponding results.

3.2 Preserving existing IDS and File Functionality

The Fig. 2 shows the existing workflow of any IDS system that could cater to both offline and online detection requirements either in batch or in real-time mode. Additionally, there are additional components marked that are required for the enablement of the DRLDO system to augment existing IDS with metamorphic malware detection capabilities. As could be found in this process, the existing components like the existing IDS setup, its related training workflows, the associated training data etc. does not require any change. The IDS do not even need to be retrained to accommodate any obfuscation data. The only change that is required is just having file's feature-vector (as extracted by the IDS's feature generator) transformed by the DRLDO system before sending it to the IDS's classifier for detection. Besides this single change in the deployment architecture, the entire deployment setup remains the same, thus preserving the functionality and also the training, scoring and the deployment setups of the existing IDS. Similarly, the associated treatment and the functionality of the files passing through the system is also preserved.

4 CUSTOM REINFORCEMENT LEARNING ENVIRONMENT

The environment serves a major role in reinforcement learning. As illustrated in algorithm 1 its role is to present a current state to the agent to act upon, and then subsequently give it an appropriate reward and the corresponding next state to the agent. The *current_state, action, reward, next_state* cycle continues until a terminal state is reached (for an episodic task) or until a predestined number of steps are completed. On reaching such scenario, the environment resets, itself and re-instantiates any default state and other necessary variables.

4.1 The Structure of 'State'

The state in our experiment is comprised of a vector of whole numbers ($S \in W^{|op-code|}$) corresponding to each of the unique opcode frequency for a given opcode in a file. We use the same unique opcode set as used by Sewak et. al.[16]. We also use the same IDS which had produced the best performance and as used in their work. Their system claimed an accuracy of 99.21% with a False Positive Rate of 0.19% on the Malicia dataset[7] which is to our information the best performance achieved on this standardised malware dataset till date. The opcodes generated are also from the Malicia dataset. We obfuscate the opcode frequency vector using another DRL based Obfuscation system and score the resulting opcode frequency vector as generated from this system with the earlier described classification system. The obfuscated variants that are detected by the detection system as benign with a probability $P_{benign} > 0.5$ are used as training dataset for our de-obfuscation system. This dataset along with the collected obfuscated opcode frequency vector from the work resulted in a set of 1612 unique opcodes. Correspondingly we have a state comprising of 1612 dimension 'Action' Space with a permissible range of $[0,10000] \in Z^{1612}$.

4.2 The Design of 'Action'

The reinforcement learning tasks (and hence the environments/ agents) could be broadly classified based on the type of action as discrete action and continuous action RL tasks. The high computational complexity of the continuous (and high cardinality) action space mandates use of specialised class of agents powered by special mathematical theorems[19] that could empower both the non-deep[19] and deep learning variants[6,20,21] of reinforcement learning agents. For each of the unique opcode in the state there are two decision/ action criteria. First is the direction of change in each, namely, to

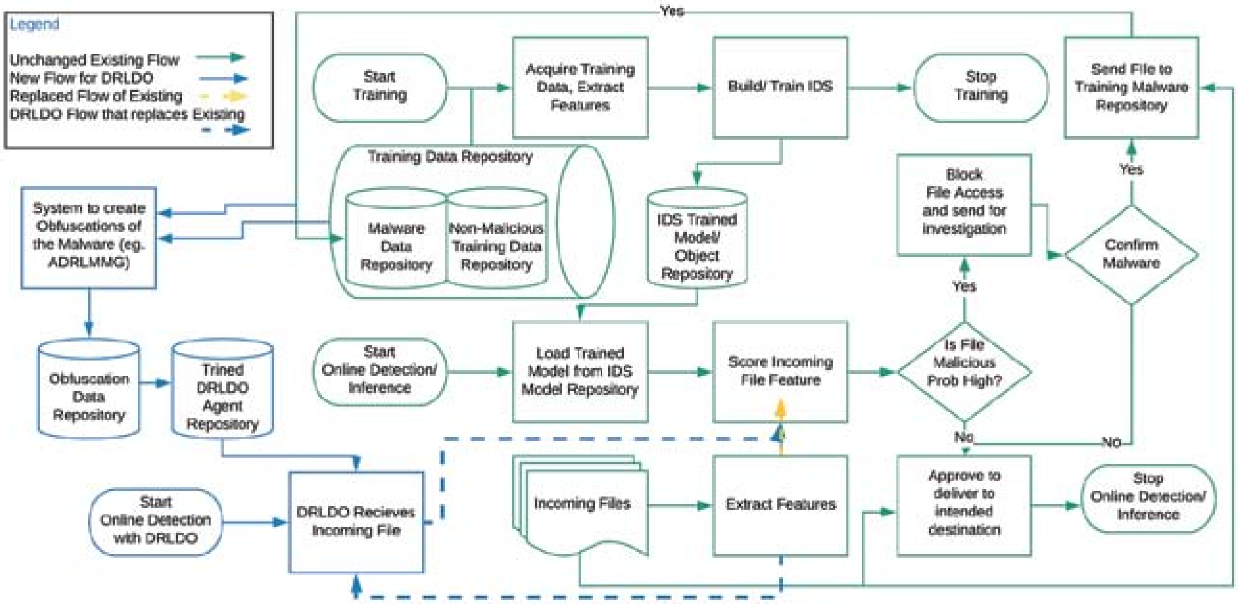
Legend
Unchanged Existing Flow
New Flow for DRLDO
Replaced Flow of Existing
DRLDO Flow that replaces Existing
Yes
Start Training
Acquire Training Data, Extract Features
Build/ Train IDS
Stop Training
Send File to Training Malware Repository
Training Data Repository
Malware Data Repository
Non-Malicious Training Data Repository
System to create Obfuscations of the Malware (eg. ADRLMMG)
IDS Trained Model/ Object Repository
Block File Access and send for investigation
Yes
Confirm Malware
Yes
Obfuscation Data Repository
Trined DRLDO Agent Repository
Start Online Detection/ Inference
Load Trained Model from IDS Model Repository
Score Incoming File Feature
Is File Malicious Prob High?
No
NO
Start Online Detection with DRLDO
DRLDO Recieves Incoming File
Incoming Files
Extract Features
Approve to deliver to intended destination
Stop Online Detection/ Inference

Figure 2: Preserving Functionality: DRLDO mandates no changes in the IDS not even re-training.

increase it or to decrease it, and second is the amount by which the increment/ decrease should occur. In this approach we have $N_{observation} \times 2$ actions, the first *N* actions corresponding to an increase in the specific opcode frequency by a constant $C_{increment}$ and the next *N* actions represent an act of decreasing the corresponding opcode frequency by a fixed amount τ; where, $C_{increment}$, $C_{decrement}$ $\in \mathcal{N}$. In our implementation we have kept $C_{increment}$ = $C_{decrement}$ =1. Also, since from the perspective of obfuscation, the easiest way of creating multiple obfuscation often increases the opcode frequency by adding junk code, instructions, indirect routing etc.[22]. Therefore, to mimic this effect we allow the agent's action only a *net* increase in individual opcode frequency from their initial level (as in original malware). An action with a net effect of decreasing an individual opcode below its original level results in returning the same state as before the action and a commensurate reward. We keep additional action constraints for our agent to ensure that its behaviour mimics the de-obfuscation action while preserving the original functionality. Since obfuscation is mostly created by techniques which generally increases the opcode frequencies in the resulting file, therefore a good de-obfuscation system should ideally reverse this effect. So, our agent could only take actions resulting in a net reduction of any specific opcode's frequency from its original level in the obfuscated file. Also, since a negative opcode frequency is not possible, so the least it could be decreased is to zero.

4.3 The Formulation of 'Reward' Function

What the agent learns and how quickly it converges is dependent upon the reward function (i.e., the reward/ penalty criteria and magnitude) to a considerable degree. Our primary objective is that the agent could alter the opcode frequency enough to substantially enhance the IDS's capability to detect it as malicious.

```
**Algorithm 1** Custom RL Environment Algorithm.
**Require:** RESET instructions from agent
**Ensure:** IDS, MalwareDateRepository is attached
$index \leftarrow random\ [1,\ N_{MalwareDataRepository}\ ]$
STATE $\leftarrow$ $IDSFeatureGenerator(index)$
REWARD $\leftarrow$ 0
IsCOMPLETE $\leftarrow$ False
$return \leftarrow$ (STATE, REWARD, IsCOMPLETE)
**while** $(Turns \le MaxTurns) \wedge (\neg IsComplete)$
**do**
$RESPONSE \leftarrow$ Agent (ACTION)
NEW STATE $\leftarrow$ $STATE(ACTION)$
$P_{malicious} \leftarrow IDSDetector(NEWSTATE)$
**if** $P_{malicious} \ge P_{MalThreshold}$ **then**
REWARD $\leftarrow$ REWARD $+ R_{Probability} + Reward_{Victory}$
$IsCOMPLETE \leftarrow True$
**else**
REWARD $\leftarrow$ REWARD $+\ R_{Probability} - Penalty_{Turn}$
$N \leftarrow n$
**end if**
$return \leftarrow$ (STATE, REWARD, IsCOMPLETE)
**end while**
```

Since the benign probability of the samples that we selected are ≥ 0.5 (i.e. $P_{malicious\_min\_required} = 0.5$), and since the maximum possible malicious probability is 1.0 (i.e. $P_{malicious\_\max\_attainable} = 1.0$), we take a mid-point of these two extremes (i.e. $P_{malicious} = 0.75$) as in equation 2 as the preliminary target for the system. This could be stated as (given $opcode \in W^{1612}$, and $P(M) = P(Mal|IDS)$):

$$P_{target} = \frac{P_{(M=Certain)} + P_{(M=Ambiguous)}}{2}$$

$$P_{target} = \frac{1}{2}(1.0 + 0.5) = 0.75$$

$$P\left(opcode_{file}\, file \subseteq \{maliciousfiles\}\right) > P_{target}$$

$$\text{or } P\left(opcodefile \subseteq \{maliciousfiles\}\right) > 0.75 \tag{2}$$

We penalise any resulting opcode frequency vector that has predicted probability of malicious ≤ 0.75 and reward the ones with probability of malicious ≥ 0.75 proportionally. So, in each step the instantaneous reward given to the agent could be stated as equation 3 (given):

$$P\left(op-codefile \subseteq \{maliciousfiles\}\right) = P_{malicious}$$

$$reward = P_{malicious} - 0.75 \tag{3}$$

But this reward mechanism has a drawback that it encourages long trajectories resulting in positive rewards instead of quickly reaching a very high P_{benign}. Since the 'discounting-factor' (γ) is only in the agent's control and not in environment's control, so the reward mechanism cannot take the help of lowering the discounting-factor enough so that quick high instantaneous rewards become more profitable than lower cumulative discounted-rewards. So, to overcome this effect, we have another (instantaneous) reward given by the environment to the agent (in addition to the one stated above), which is accrued when the agent to manage the alter the opcode frequency enough such that the file is almost unambiguously classified as malicious. This reward is high enough to easily surpass even multiple cumulative (even discounting given γ < 1) rewards and is similar to the malicious probability for original malware variants as detected by the system. This occurs when the $P_{benign} \geq P_{threshold}$; where, $P_{threshold}$ is a high threshold probability of malicious (say 0.99). Therefore, now the reward can be given as a step function as Eqn (4) below.

$$reward = \begin{cases} P_{malicious} - 0.75, & f\, P_{malicious} \\ & \leq P_{threshold} \\ R_{goal} & otherwise \end{cases} \tag{4}$$

where, R_{goal} could either be a fixed constant or one dependent upon the maximum steps allowed in the episode. An episode starts with a reset of the environment. During the 'reset', the environment fetches a random malware file's opcode. The episode ends when either the goal is achieved ($P_{malicious} > P_{threshold}$) or the maximum permissible steps for the episode is reached. Here we set $R_{goal} = Max\_Permissible\_Steps\_in\_an\_Episode$ so that we could balance the requirements for setups with large episodes. This allows for slow but steady convergence of complex agents with too many trainable parameters. In such setups the $max\_permissible\_step$ is adaptively set-in relation with the R_{goal}, such that it is always greater than any cumulative reward over even a long episode.

5. DRL AGENT(S) USED

Given the constraints of the design of the reinforcement learning as covered in section 4.2, we have a discrete action task with a very high action space (and also state space) cardinality. Some of the most popular DRL agents for discrete action agents like the 'Deep Q Networks' (DQN)[23,3], 'Double DQN' (DDQN)[4], and the 'Dueling DQN' (DDQN)[5]. These algorithms though could manage large state-space but perform poorly for large/ continuous action space. Deterministic Policy Gradient[19] based deep reinforcement learning approaches like the 'Deep Deterministic Policy Gradient' (DDPG)[6] claimed to be deliver the best-in-class performance on large, even continuous action-space based reinforcement learning tasks. The problem with such approaches is that their line-search based policy gradient update (as used during optimisation) either proves too big for updates involving non-linear trajectory. This results in the updates overshooting the target or slower convergence. Since in the deep reinforcement paradigm non-linear gradients are quite common so algorithms based upon line-search based gradient update are not very robust and cannot provide guarantees of near monotonic policy improvements. 'Trust Region Policy Optimisation' (TRPO)[20] algorithm which is based on 'trust-region' based policy updates using 'Minorise-Maximisation' (MM) (second order) gradient update, claims to solve this problem and provide guarantee for near monotonic general (stochastic) policy improvement even for non-linear policies like that approximated by (deep) neural networks. Additionally, TRPO uses a mechanism called 'Importance Sampling' to compute the expectancy of the policy from previous trajectories instead of only the current trajectory to stabilise the policy gradient. This method has an underlying assumption that the previous trajectory's distribution ($Q(x)$) is not very different from the current trajectory's distribution ($P(x)$). The policy gradient for a Stochastic Policy Gradient[24] method and associated algorithms like Actor Critic[25] is given as Eqn (5):

$$\nabla_{\theta}\left(J_{\theta}\right)=E_{\tau\sim\pi_{\theta}(\tau)}\left[\nabla_{\theta}\log\pi_{\theta}\left(\tau\right)r\left(\tau\right)\right] \quad (5)$$

In equation 5, the trajectory τ over which the samples for computing expectancy is gathered (to update the gradient ∇ of the policy-value-function J) is the same (current) trajectory of the policy as used in the policy π (parameterised over θ). But in the case of TRPO using importance sampling and the past trajectory for sampling, this policy-value-function update looks like Eqn (6) below:

$$\nabla_{\theta'}\left(J_{\theta}^{'}\right)=E_{\tau\sim\pi_{\theta}(\tau)}\left[\sum_{t=1}^{T}\nabla_{\theta}^{'}\log\pi_{\theta}^{'}\left(\prod_{t'=1}^{t}\frac{\pi_{\theta'}}{\pi_{\theta}}\right)\left(\sum_{t'=1}^{T}r\right)\right] \quad (6)$$

To avoid too large changes in gradient, a penalty needs to be added to equation 6 to make the optimisation more monotonic. With this penalty combined with the use of the advantage, the optimisation function is given as Eqn (7):

$$\max_{\theta}\hat{E}_t\left[\frac{\pi_{\theta}\left(a_t\mid s_t\right)}{\pi_{\theta old}\left(a_t\mid s_t\right)}\hat{A}_t-\beta KL\left[\pi_{\theta old}\left(.\mid s_t\right),\pi_{\theta}\left(.\mid s_t\right)\right]\right] \quad (7)$$

The problem with the β based penalty as in equation 7 is that it is difficult to choose a single value of β that aligns well to different types of problems, or even for a single problem as the learning progress. Therefore, to resolve this issue, TRPO instead of using β based penalty, uses KL based constraints as shown in Eqn (8), thus requiring a second order optimisation solution.

$$\max_{\theta}\hat{E}_t\left[\frac{\pi_{\theta}\left(a_t\mid s_t\right)}{\pi_{\theta old}\left(a_t\mid s_t\right)}\hat{A}_t\right] \quad (8)$$

$$\text{Subject to: }\hat{E}_t\left[KL\left[\pi_{\theta old}\left(.\mid s_t\right),\pi_{\theta}\left(.\mid s_t\right)\right]\right]\le\delta$$

In TRPO, the second order gradient update computation is complicated and also very expensive, and hence for real size-able tasks it is seldom use. The 'proximal policy optimisation' (PPO)[21] algorithm instead of using a constrained form of solution (as shown in Eqn (8)), clips the surrogate objective to ensure that the updates are not unconstrained. This is as given in Eqn (9).

$$L^{CPI}\left(\theta\right)=\hat{E}_t\left[\frac{\pi_{\theta}\left(a_t\mid s_t\right)}{\pi_{\theta old}\left(a_t\mid s_t\right)}\hat{A}_t\right]=\hat{E}_t\left[r_t\left(\theta\right)\hat{A}_t\right] \quad (9)$$

The original surrogate objective L^{CPI} for TRPO as discussed in equation 9. This in the 'clipped' form could be reformulated as equation 10; where, ϵ is a hyper-parameter. The default value of ϵ is set to 0.2.

$$L^{CLIP}\left(\theta\right)=$$

$$\hat{E}_t\left[\min\left(r_t\left(\theta\right)\hat{A}_t,clip\left(r_t\left(\theta\right),1-\int,1+\int\right)\hat{A}_t\right)\right] \quad (10)$$

Another suggested variant of the PPO algorithm is based upon adaptive β penalty as given in equation 11. But in various experiments, the clipped penalty form of equation 10 performed better than the adaptive penalty form of equation 11, and hence we use the clipped form in our system.

$$\beta/2 \qquad if\ d\le d_{targ}\div1.5$$

$$\beta=\ \beta\times2 \qquad if\ d\ge d_{targ}\times1.5 \quad (11)$$

$$\text{where, }d=\hat{E}_t\left[KL\left[\pi_{\theta}\left(.\mid s_t\right),\pi_{\theta old}\left(.\mid s_t\right)\right]\right]$$

The PPO algorithm works similar to TRPO and is much easier to compute as it uses a linear variant of the gradient update called the 'Fisher Information matrix' (FIM). In equation 6, the trajectory is sampled from the policy as it existed in previous time (t) as $\pi_{\theta}=Q\left(x\right)$. The expectancy over such collected samples is used to update the policy at next time step ($\pi_{\theta 0}=P\left(x\right)$). When the ratio of expectancy over the two trajectory distributions ($P\left(x\right)/Q\left(x\right)$) vary significantly as in the case of linear gradient update in PPO, the previously stated assumption may not hold, leading to high variance in policy updates. To avoid this there are two methods that the PPO algorithm recommends. The first one uses a 'Adaptive KL Penalty' and the second one use 'Objective Clipping'. As per the original PPO paper[21], the 'Objective Clipping' variant, with the clipping factor $\epsilon=0.2$ provided the best result. We use similar mechanism in our experiment. In this mechanism if the probability ratio between the two trajectory's policies is not in the range $\left[(1-\epsilon),(1+\epsilon)\right]$ the 'estimated advantage' is clipped. We use the Proximal Policy Optimisation algorithm (PPO)[21]. PPO is an improvement over Trust Region Policy Optimisation algorithm (TRPO)[20]. The deep learning model that we use for the PPO algorithms actor and critic network comprise of 2 hidden layers each, with each hidden layer having 64 neurons and a 'tanh' activation function.

6. EXPERIMENTS AND RESULTS

We conducted over 2000 (episodic) experiments where the PPO algorithm based Deep Reinforcement Learning (DRL) agent would attempt to de-obfuscate a malicious file and validate if the associated IDS could then detect the de-obfuscated version of the obfuscated malicious file. Each such experiment is constituted of an episode consisting of several steps. In the first step the environment extracts a random new obfuscation of any malicious file from the opcode frequency vector feature repository (as generated by the ADRLMMG system) and sends it to the agent to process it. The agent then alters the frequency of one of the opcodes in each subsequent step and the environment correspondingly rewards the agent as per the mechanism described in section 4.3. For each episode, we record the malicious probability that was finally reported by the existing IDS on the de-obfuscated version of the opcode frequency vector as generated by the agent in the final step of every episode (each episode starts with a new obfuscated feature-vector). We use the existing IDS as-is without altering or retraining it. These probability $P_{malicious}$ trends across episodes are reported in the plot in Fig. 3. As shown in this figure, the detected $P_{malicious}$ (moving average) has crossed the critical point of $P_{malicious} > 0.5$ very early and has reached $P_{malicious} > 0.6$ in around 2000 episodes. In each step of every episode the agent gets some instantaneous rewards. The rewards received earlier in an episode could be discounted by a discounting factor to give more importance to more recent awards. The total of all instantaneous rewards or the discounted rewards indicates how well the agent is learning to achieve its objectives as converted into the rewards using the defined reward function for the agent. As shown in Fig. 4, as the episodes progress the agent is able to accrue higher total instantaneous rewards and discounted rewards (left y-axis) indicating that the agent

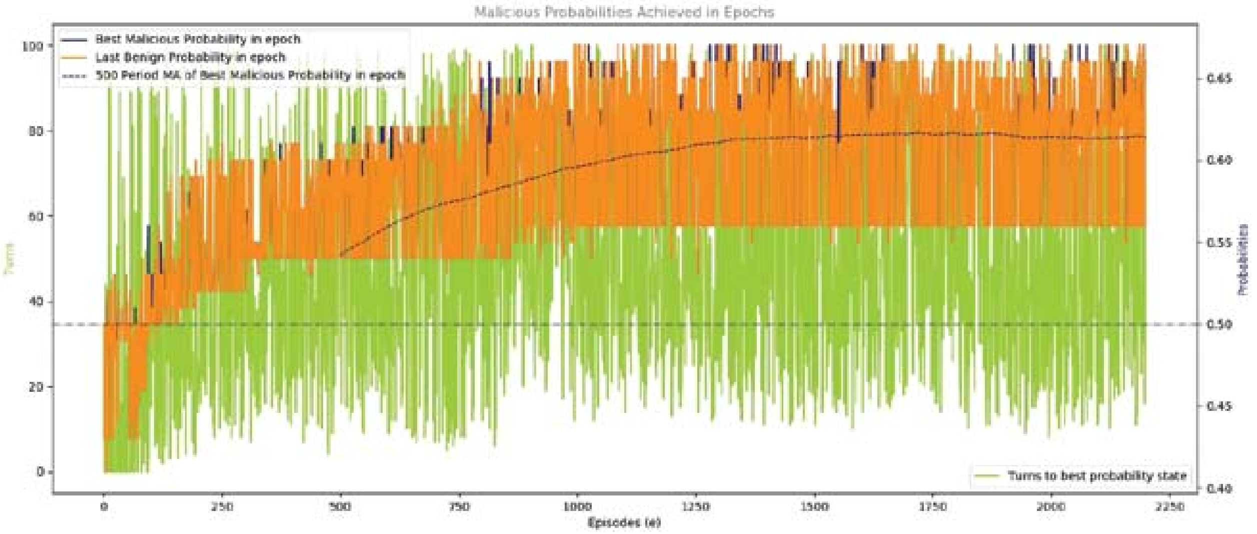

Figure 3. Training Statistics - Malicious Probability detected by the IDS after De-obfuscation frequency vector across episodes.

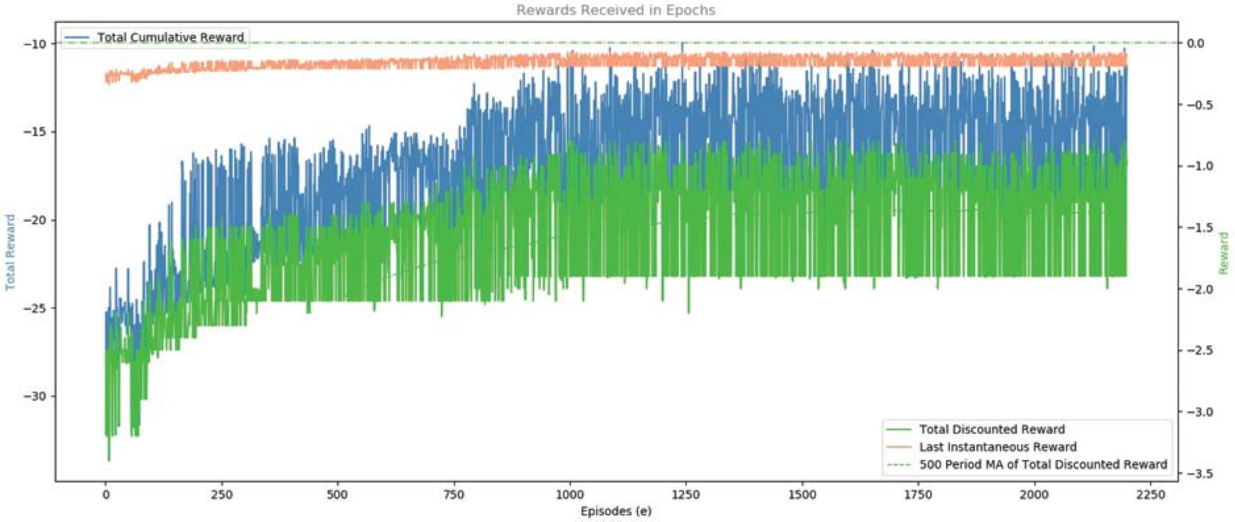

Figure 4. Training statistics - Total instantaneous and discounted rewards accrued by agent across episodes.

is able to effectively learn the desired policy. Also, the last instantaneous reward for most of the episodes (right y-axis) is high indicating that the episodes are ending in successful detection of the transformed opcode frequency features as malicious by the IDS. The total rewards may just be a single indicator of how well the agent is learning the mathematical abstraction of the tasks formulated into a reward function. Additionally, to ensure that the agent's learning is aligned well with the desired outcome that the IDS should be able to detect the malicious file correctly after de-obfuscation, we present the histogram of all the final $P_{malicious}$ detection probability of the de-obfuscated feature (opcode frequency) vector of the obfuscated/ metamorphic malware file by the existing IDS without re-training or modifying the IDS in Fig. 5. As shown in this figure the mean malicious probability ($\frac{1}{n}\sum^{n} P_{malicious}$) was uplifted to ≈ 0.6 (where, $P^{initial}_{malicious} \in [0.0, 0.5]$), indicating that the IDS could now effectively detect the generated de-obfuscated variants as malicious with high $P_{malicious}$ probability.

7. DISCUSSION

As covered in section 6, the IDS can detect the obfuscated malicious file after the resultant de-obfuscation by the DRLDO agent as malicious, which helps probably in evading a 'zero-day attack' by a metamorphic/ obfuscated variant of an existing malware. But still, we are not sure whether the agent is actually learning to de-obfuscate/normalise the obfuscated variant of the malware or learning some nonreproducible tricks or exploiting some trivial loopholes to help the IDS raise the $P_{malicious}$ of some specific variant. To ascertain that the agent is actually learning to de-obfuscate the metamorphism in the malware we compute and record the correlation similarity of the final output (de-obfuscated) feature-vector (opcode frequency vector) as produced by the agent with the opcode feature-vector of the obfuscated variant that the agent received as an input. We also compute the correlation similarity of the final output feature-vector with that of the original malware's corresponding feature-vector. Figures 6, and 7 shows the histogram of the overall correlation similarities between the feature-vector of the de-obfuscations produced by the agent

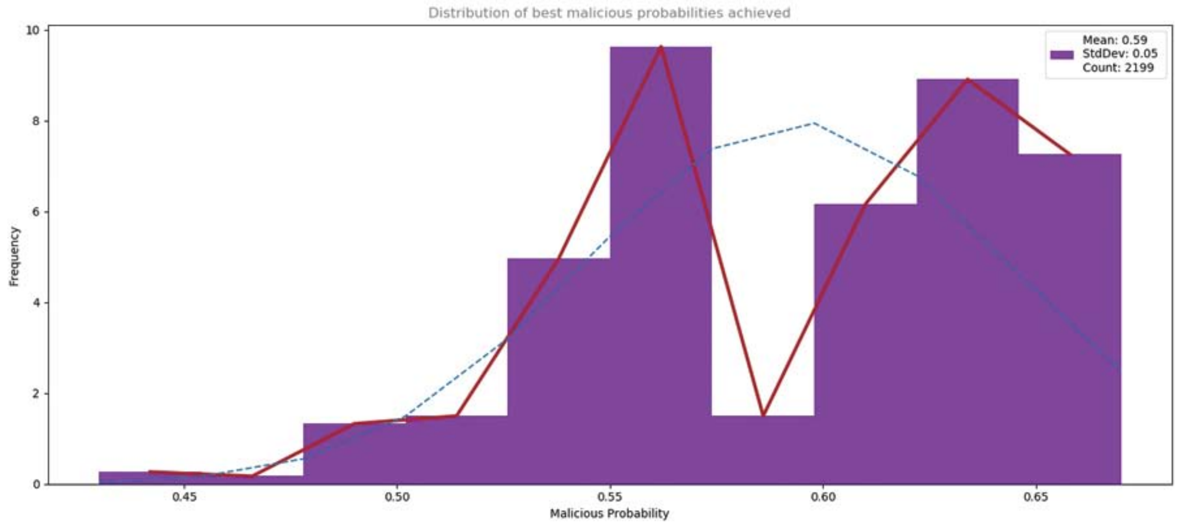

Figure 5. Training statistics - $P_{malicious}$ as detected by the IDS for the de-obfuscated version of the malware by the agent.

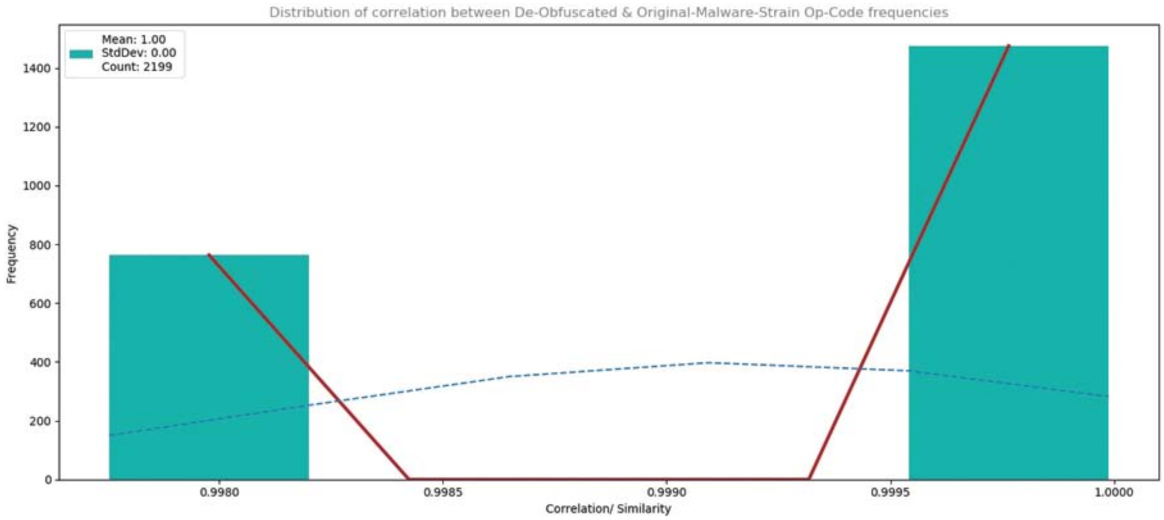

Figure 6. Training statistics - Histogram of similarity of the feature-vector of the de-obfuscations produced by the agent with the obfuscated Variant feature-vector of the Malware as given as input to the agent.

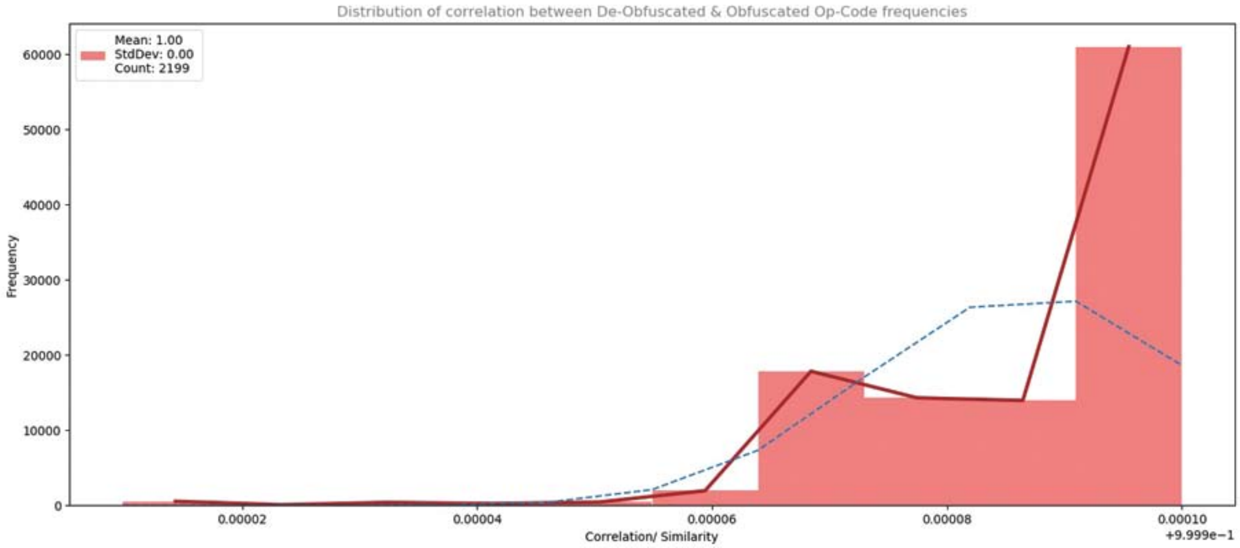

Figure 7. Training statistics - Histogram of Similarity of the feature-vector of the de-obfuscations produced by the agent with the feature-vector of the original Malware (that was not exposed to the agent).

with the feature-vector of the original malware's and between feature-vector of the de-obfuscations produced by the agent with that of its obfuscated variant's feature-vector as submitted to the agent respectively. We use Pearson product-moment correlation coefficients between the opcode vectors to generate these similarities. The correlation is taken from the correlation matrix R, whose relationship with the co-variance matrix, C, is as given as $R_{ij} = \frac{C_{ij}}{\sqrt{C_{ii} * C_{jj}}}$. Another interesting observation is related to the resultant output's opcode frequency vector's correlation similarity. We measure two types of correlations, the first is the similarity between the de-obfuscated opcode frequency vector as created by the DRLDO system and that of the obfuscated opcode frequency vector as provided as an input to the DRLDO system, and the second similarity is measured between the de-obfuscated opcode frequency vector as generated by the DRLDO system and that of the opcode frequency vector of the original malware. The opcode frequency of the original malware is not known to the DRLDO system and we extract it from the metadata created by the ADRLMMG system which produced these obfuscations. The feature-vector of the de-obfuscated variants as produced by the DRLDO system were quite similar to (≥ 0.99) their original malware's feature-vector and very dissimilar to (≤ 0.001) their obfuscated variants as submitted to the DRLDO system. This observation establishes that the desired enhancement in the detection probability $P_{malicious}$ of these feature-vectors after processing them via the DRLDO system is actually due to the probable de-obfuscation carried out by the DRLDO system in which it removed some of the additional opcode/ instructions that were inserted in the original malware to evade its detection by the IDS and the results are unlikely to be because of any trivial and nonreproducible trick that the agent might have unintentionally learnt. The insights from these observations are very significant especially since we never exposed the original malware strain's opcode frequency vector to the DRLDO system. Attaining such high similarity with the original malware's opcode frequency indicates that the resultant de-obfuscations thus created by our system could not only be now detected by most existing IDS as malicious, but it could also be identified that the incoming malware is an obfuscated variant of one of the existing malware variants that the IDS has in its training repository. So, besides enhancing binary IDS that could just detect whether a file is benign or malicious the de-obfuscations created from our system is also compatible with and would produce correct results with a multinomial IDS that also detects the family of the malware variants. Alternatively, in a binary IDS subsequently by using the similarity between the outputs of the DRLDO system with the stored feature-vectors of the existing malware variants in the system's repository the family of the obfuscated malware could be identified thus enhancing the insights generated from the detection system. The above observations on the similarity with the original malware strain also indicates that following de-obfuscation, the file size and opcode frequency distribution does not change substantially. This has other significant implications as this would also prevent any malware prediction probability creep/ enhancement even when subjected to any IDS which first segregates the files into different categories based on either their file size[26] or on the outcomes of machine learning methods like clustering[27] before scoring/ predicting them for their maliciousness, to enhance their respective prediction/ detection effectiveness/ accuracy.

8. CONCLUSION

We designed and developed an advance Deep Reinforcement Learning based system named DRLDO that could learn how to de-obfuscate and normalise a metamorphic (or otherwise obfuscated) malware. Unlike some other systems that could work only at the binary level and hence the transformations from these are intractable and non-functionality-preserving, the DRLDO is the first system that could perform de-obfuscations at the opcode-sequence level. Additionally, the DRLDO system offers unique advantages as it does not mandate any change in the IDS' classification-system and does not even require a re-training of the classifier. Thus, the DRLDO system could be easily retrofitted into an existing IDS setup.

The experiments conducted with the DRLDO system, and the corresponding results obtained, proves that PPO algorithm based DRL agents, as used in the DRLDO system, could be effectively trained using our custom developed RL environment. The so trained DRL agents could effectively de-obfuscate the (opcode-sequence) feature-vector of an incoming obfuscated malware. The resulting transformed feature-vector could be correctly detected by an existing IDS with a detection probability of up to 0.6 for previously un-detectable obfuscated intrusions. In the entire process no re-training, re-configuration or re-calibration of the IDS is required.

Thus, the DRLDO system could effectively provide an existing IDS the augmented capabilities of defense against (even multiple-simultaneous) attack from metamorphic variants of existing malware. Doing so, the DRLDO system can enhance an IDS with unique defensive capabilities against any probable 'zero-day attack' by a metamorphic attack from obfuscated variants of an existing malware.

REFERENCES

1. Mirzazadeh, R.; Moattar, M.H. & Jahan, M.V. Metamorphic malware detection using linear discriminant analysis and graph similarity. *In* Proceedings of the International Conference on Computer and Knowledge Engineering, IEEE, 2015, pp. 61–66. doi:10.1109/ICCKE.2015.7365862
2. Sewak, M., Deep Reinforcement Learning: Frontiers of Artificial Intelligence. Springer, 2019, pp. 128-143. doi: 10.1007/978-981-13-8285-7
3. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D. & Riedmiller, M., Playing atari with deep reinforcement learning. https://arxiv.org/abs/1312.5602, [Accessed on January 2020].
4. Van Hasselt, H.; Guez, A. & Silver, D., Deep reinforcement learning with double q-learning. https://arxiv.org/abs/1509.06461, [Accessed on January 2020].
5. Wang, Z.; Schaul, T.; Hessel, M.; Hasselt, H.; Lanctot, M. & Freitas, N. Dueling network architectures for deep reinforcement learning. *In* Proceedings of the International conference on machine learning. 2016, pp. 1995–2003.
6. Lillicrap, T.P.; Hunt, J.J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D. & Wierstra, D., Continuous control with deep reinforcement learning. https://arxiv.org/abs/1509.02971, [Accessed on January 2020].
7. Nappa, A.; Rafique, M.Z. & Caballero, J. Driving in the Cloud: An Analysis of Drive-by Download Operations and Abuse Reporting. *In* Proceedings of the Conference on Detection of Intrusions and Malware & Vulnerability Assessment. 2013. doi:10.1007/9783-642-39235-1_1
8. Sewak, M.; Karim, M.R. & Pujari, P., Practical Convolutional Neural Networks: Implement Advanced Deep Learning Models Using Python. Packt Publishing Ltd, 2018, pp. 144-162.
9. Hu, W. & Tan, Y., Generating adversarial malware examples for black-box attacks based on gan. https://arxiv.org/abs/1702.0598, [Accessed on January 2020].
10. Lin, Z.; Shi, Y. & Xue, Z., IDSGAN: Generative adversarial networks for attack generation against intrusion detection. https://arxiv.org/abs/ 1809.02077, [Accessed on January 2020].
11. Usama, M.; Asim, M.; Latif, S.; Qadir, J. *et al.* Generative adversarial networks for launching and thwarting adversarial attacks on network intrusion detection systems. *In* Proceedings of the International Wireless Communications & Mobile Computing Conference, IEEE, 2019, pp. 78–83. doi:10.1109/IWCMC.2019.8766353.
12. Grosse, K.; Papernot, N.; Manoharan, P.; Backes, M. & McDaniel, P., Adversarial perturbations against deep neural networks for malware classification. https://arxiv.org/abs/1606.04435, [Accessed on January 2020].
13. Anderson, H.S.; Kharkar, A.; Filar, B.; Evans, D. & Roth, P., Learning to evade static pe machine learning malware models via reinforcement learning. https://arxiv.org/abs/1801.08917, [Accessed on January 2020].
14. Wu, D.; Fang, B.; Wang, J.; Liu, Q. & Cui, X. Evading machine learning botnet detection models via deep reinforcement learning. *In* Proceedings of the IEEE International Conference on Communications, IEEE, 2019, pp. 1–6. doi: 10.1109/ICC.2019.8761337
15. Sewak, M.; Sahay, S.K. & Rathore, H. An investigation of a deep learning based malware detection system. *In* Proceedings of the International Conference on Availability, Reliability and Security, 2018, pp. 1–5.
16. Sewak, M.; Sahay, S.K. & Rathore, H. Comparison of deep learning and the classical machine learning algorithm for the malware detection. *In* Proceedings of the IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, IEEE, 2018, pp. 293–296. doi:10.1109/SNPD.2018.8441123
17. Rathore, H.; Agarwal, S.; Sahay, S.K. & Sewak, M. Malware detection using machine learning and deep learning. *In* Proceedings of the International Conference on Big Data Analytics, Springer, LNCS, 2018, pp. 402–411. doi: 10.1007/978-3-030-047801_28
18. Sahay, S.K.; Sharma, A. & Rathore, H., Evolution of malware and its detection techniques. *In* Proceedings of the Information and Communication Technology for Sustainable Development, Springer, 2020, pp. 139– 150. doi: 10.1007/978-98113-7166-0_14
19. Silver, D.; Lever, G.; Heess, N.; Degris, T.; Wierstra, D. & Riedmiller, M. Deterministic policy gradient algorithms. *In* Proceedings of the International Conference on International Conference on Machine Learning, 2014.
20. Schulman, J.; Levine, S.; Moritz, P.; Jordan, M. & Abbeel, P. Trust region policy optimization. *In* Proceedings of the International Conference on International Conference on

Machine Learning, 2015.

21. Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A. & Klimov, O., Proximal policy optimization algorithms. https://arxiv.org/abs/1707.06347, [Accessed on January 2020].
22. Behera, C.K. & Bhaskari, D.L., Different obfuscation techniques for code protection. *Procedia Computer Science*, 2015, pp. 757–763.
    doi: 10.1016/j.procs.2015.10.114
23. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A.A.; Veness, J.; Bellemare, M.G.; Graves, A.; Riedmiller, M.; Fidjeland, A.K.; Ostrovski, G. *et al.*, Human-level control through deep reinforcement learning. *Nature*, 2015, pp. 529–533.
    doi: 10. 1038/nature14236
24. Sutton, R.S.; McAllester, D.; Singh, S. & Mansour, Y. Policy gradient methods for reinforcement learning with function approximation. *In* Proceedings of the International Conference on Neural Information Processing Systems, 1999, pp. 1057–1063.
25. Mnih, V.; Badia, A.P.; Mirza, M.; Graves, A.; Lillicrap, T.P.; Harley, T.; Silver, D. & Kavukcuoglu, K., Asynchronous methods for deep reinforcement learning. https://arxiv.org/abs/1602.01783, [Accessed on January 2020].
26. Sahay, S.K. & Sharma, A., Grouping the executables to detect malwares with high accuracy. *Procedia Computer Science*, 2016, pp. 667–674.
    doi: 10.1016/j.procs.2016. 02.115
27. Rathore, H.; Sahay, S.K.; Chaturvedi, P. & Sewak, M. Android malicious application classification using clustering. *In* Proceedings of the International Conference on Intelligent Systems Design and Applications, Springer, 2018, pp. 659–667.
    doi: 10.1007/978-3-030-16660-164

CONTRIBUTORS

Mr Mohit Sewak received his MTech in Software Systems from BITS Pilani and BE in Marine Engineering from Jadavpur university in 2015 and 2003, respectively. Currently he is serving as the Principal Data Scientist in the Security and Compliance Research team of Microsoft, India R&D, and is also pursuing part-time PhD at BITS Pilani. Mohit has more than 15 years of experience in Research, Design and Development of Machine Learning and Artificial Intelligence products in the industry. His current research interests are security, compliance, and artificial intelligence.

In the current study he conceptualised the idea, executed experiments as per supervision, investigation, writing - original draft, writing - review and editing.

Dr Sanjay K. Sahay presently working as Associate Professor in the Department of Computer Science and Information System in BITS, Pilani, K.K. Birla Goa campus. He is also a Visiting Associate of IUCAA, Pune. His research interests are Malware identification, Artificial Intelligence, Authentication, Data Mining and Gravitational Waves. He also served as a program committee member of various reputed conferences and has been a reviewer for journals. He published more than 50 scientific articles and edited the SKM-209 proceeding, published in Springer, Communications in Computer, and Information Science.

In the current study he supervised/conceptualised the entire paper and helped in formulating the methodology and reviewing/writing the final draft.

Mr Hemant Rathore received his BE and ME in computer science from RGTU, India and BITS Pilani, India in 2010 and 2013, respectively. Currently, he is pursuing PhD at Birla Institute of Technology and Science, Pilani, K. K. Birla Goa Campus. He has three years of industrial experience at Symantec, India. His current research interests are malware analysis and detection, machine learning and network security. He is also a student member of the IEEE and ACM.

In the current study he conceptualised the idea, methodology, investigation, writing - original draft, writing - review and editing.