Actor Critic Deep Reinforcement Learning for Neural Malware Control

Yu Wang, Jack W. Stokes, Mady Marinescu
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052
{wany, jstokes, mady}@microsoft.com

Abstract

In addition to using signatures, antimalware products also detect malicious attacks by evaluating unknown files in an emulated environment, *i.e.* sandbox, prior to execution on a computer's native operating system. During emulation, a file cannot be scanned indefinitely, and antimalware engines often set the number of instructions to be executed based on a set of heuristics. These heuristics only make the decision of when to halt emulation using partial information leading to the execution of the file for either too many or too few instructions. Also this method is vulnerable if the attackers learn this set of heuristics.

Recent research uses a deep reinforcement learning (DRL) model employing a Deep Q-Network (DQN) to learn when to halt the emulation of a file. In this paper, we propose a new DRL-based system which instead employs a modified actor critic (AC) framework for the emulation halting task. This AC model dynamically predicts the best time to halt the file's execution based on a sequence of system API calls. Compared to the earlier models, the new model is capable of handling adversarial attacks by simulating their behaviors using the critic model. The new AC model demonstrates much better performance than both the DQN model and antimalware engine's heuristics. In terms of execution speed (evaluated by the halting decision), the new model halts the execution of unknown files by up to 2.5% earlier than the DQN model and 93.6% earlier than the heuristics. For the task of detecting malicious files, the proposed AC model increases the true positive rate by 9.9% from 69.5% to 76.4% at a false positive rate of 1% compared to the DQN model, and by 83.4% from 41.2% to 76.4% at a false positive rate of 1% compared to a recently proposed LSTM model.

Introduction

The detection of malware (*i.e.*, malicious software) is a significant problem for the world's computer users, and deep learning-based systems have been proposed for this task (Dahl et al. 2013; Pascanu et al. 2015; Athiwaratkun and Stokes 2017). To avoid infecting computers in a production test environment, unknown files are either emulated within the antimalware engine itself or executed in a special virtual machine (VM) (*i.e.*, virtualization) which includes additional monitoring. In the remainder of this paper, we focus on emulation, but the methods can be directly applied to virtualization as well.

One important aspect of a malware detection system is determining when to halt the execution of the unknown file during emulation. Malware often delays the execution of its malicious actions in an attempt to evade detection by the emulation engine. The antimalware engine's emulator employs heuristics to determine when to halt the execution of a file. Recently, the authors in (Wang, Stokes, and Marinescu 2019) proposed a model based on deep reinforcement learning (DRL) which learns when to halt the file's execution during emulation. Their system uses a Deep Q-Network (DQN) to learn a policy to either continue or halt execution using the operating system's application programming interface (API) calls made by the file. In addition to being able to halt the execution of 91.3% files earlier than the antimalware engine's heuristics, the inclusion of the DQN model improved the binary classification of unknown files by over 61.5% compared to a recent recurrent deep learning model.

In this work, we propose a new deep reinforcement learning model which uses an Actor Critic (AC) model instead of the DQN model (Wang, Stokes, and Marinescu 2019). The Actor Critic method utilizes two interactive models: an actor and a critic. The actor determines the best action to take based on the current environment information. In addition, the critic plays an evaluation role for the actor by considering the environment state and returning a score that represents how good the action is for that state. Unlike previous models employing RNNs (Pascanu et al. 2015; Athiwaratkun and Stokes 2017) or a DQN (Wang, Stokes, and Marinescu 2019) which cannot handle adversarial learning-based attacks (Hu and Tan 2017), this new structure fits much better into our antimalware detection scenario, particularly for adversarial cases where the malware can dynamically change its action based on the antimalware system's performance. In this paper, we propose a new modified critic structure which can simulate the malware's responses with adversarial attacks, and then train the actor to handle these simulated adversarial learning-based attacks generated by the critic.

The improvements for the new AC-based malware execution halting system are significant compared to the antimalware engine's heuristics and the previously proposed DQN-based system. The AC-based model halts the execution of 2.5% of the files earlier than the DQN-based model and 93.6% earlier than antimalware engine's heuristics. In addition for the task of malware classification, the new model has a true positive rate of 76.4% compared to 69.5% for the DQN system and 41.2% for a baseline file classifier (Athiwaratkun and Stokes 2017).

The contributions made by this work include the following:

1) We propose an actor critic, deep reinforcement learning-based model to halt the execution of an antimalware engine's processing of an unknown file.

2) We evaluate this new model on a large corpus of malicious and benign files. The results show significant improvement over all previously proposed models.

Emulation System Call Events

During scanning, the antimalware engine processes and records all of the assembly-level instructions of an unknown executable file. Some of these instructions correspond to operating system API calls. Since malware often utilizes the native operating system functions to implement malicious activity, these system API calls can be used as events to detect malware. The functionality employed by malware includes using network functions to implement command and control, reading and writing to the registry, and saving temporary files to the hard drive.

Analysts provided the data for this study which was collected by scanning malware and benign files in a production pipeline. In total, there are 114 events corresponding to various high-level representations of operating system activity.

Malware often employs polymorphism in an attempt to avoid detection. Multiple APIs can be called to accomplish the same task, and polymorphic malware uses different combinations of these APIs to make one instance appear to be different from another. To combat this threat, the emulator in the antimalware engine reports a high-level event whenever a low-level API call which corresponds to the event is made by the file.

The event sequence lengths of the malware tend to be longer than those for the benign files. The distribution of the lengths of the malicious files is shown in Figure 1, while Figure 2 depicts of the lengths of the benign files.

In order to better understand the underlying behavioral nature of the files, we plot the event sequences of two malware files in Figure 3 and two benign files in Figure 4. All of these files were randomly selected from our dataset. The x-axis is the time index (*i.e.*, sequence event number) in a file, and the y-axis represents the event IDs corresponding to the high-level events. Similar to results from Figures 1 and 2, the malware traces tend to contain more events than the benign files. Furthermore, malware tends to contain long loops after executing the first several events, *i.e.*, the parallel lines in the figures. Benign files, on the other hand, tend to exhibit more random behavior. While they do contain loops, these tend to be shorter and less repetitive. Ideally, our execution control

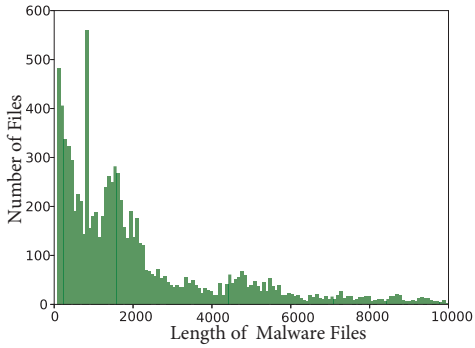

Figure 1: Distribution of the lengths of the malware files.

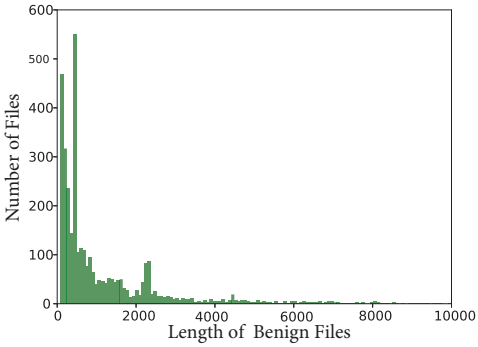

Figure 2: Distribution of the lengths of the benign files.

model can leverage these patterns to decide when to stop the emulator and perform the evaluation. Our main hypothesis is that the DRL algorithm will be able to learn these malicious patterns and be able to halt the execution of an unknown file earlier than the antimalware engine's heuristics.

System Overview

In this paper, we extend the deep reinforcement learning system originally proposed by (Wang, Stokes, and Marinescu 2019) which uses a Deep Q-Network to halt a file's emulation. One of the issues of this DQN-based structure is that it is still vulnerable to adversarial attacks as stated in their paper, *i.e.* "malware can learn the antimalware engine's behavior and change its attack strategy adaptively". In our work, we instead replace the DQN with a modified Actor Critic model for DRL. In this new model, we leverage a modified critic structure to simulate the adversarial learning-based attacks, and train the actor based on this signal. Hence, the system can handle these adversarial attacks much better than the DQN model.

The proposed system is depicted in Figure 5. Initially, the antimalware engine processes an unknown input file and extracts a sequence of file events, E, which correspond to individual system API calls. The system contains two high-level modules, the *execution control model* and the *improved in-*

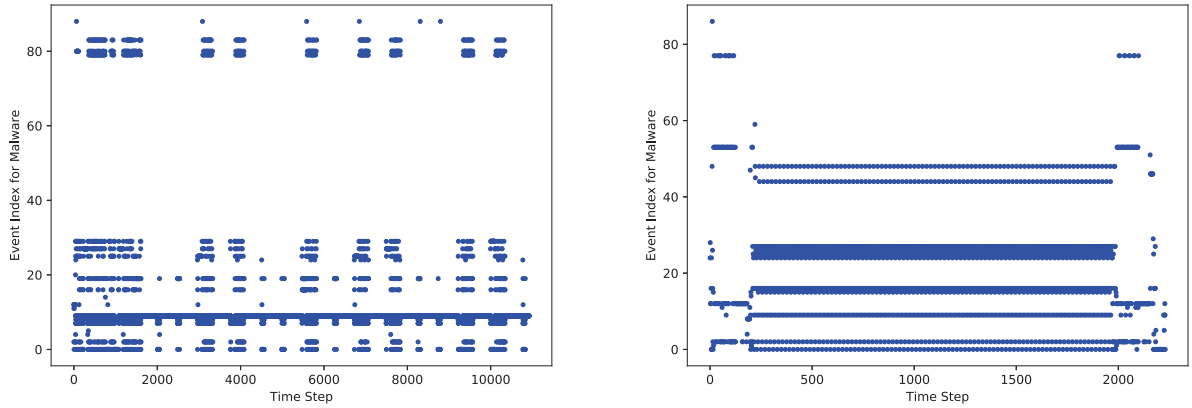

Figure 3: Event sequences of two malware files. These files were randomly selected from the test dataset. The y-axis represents the event IDs while the x-axis is the time index of each event.

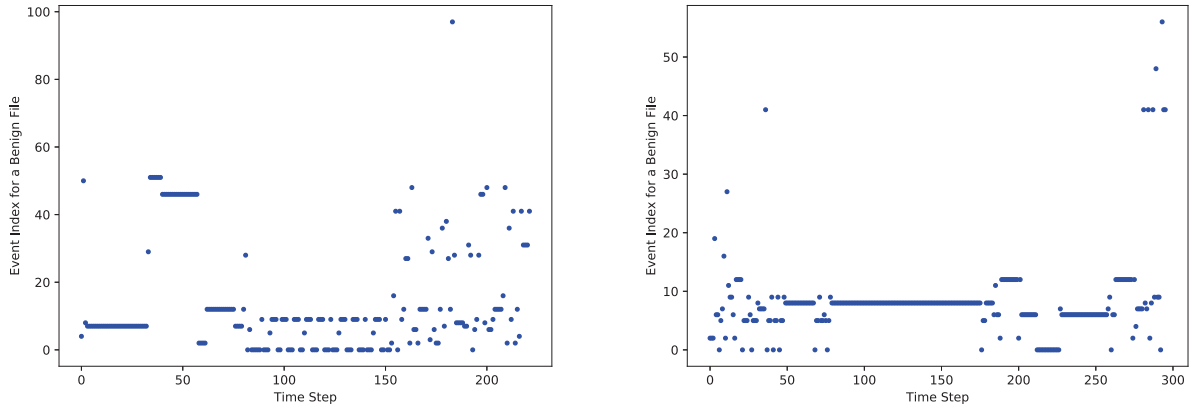

Figure 4: Event sequences of two benign files chosen at random from the test dataset.

ference model, which process these file events. The *execution control model* is responsible for halting the emulation of the file, while the *improved inference model* produces an improved probability estimate of whether or not the unknown file is malicious.

Execution Control Model. The individual file events e_t are processed by an *event classifier* which provides the probability, $y_{e,t}$, that the event, e_t, is malicious. The event classifier includes three components: a long, short-term memory (LSTM) language model, a Max Pooling layer and a classifier which processes the embedding generated by the LSTM and Max Pooling layer (Pascanu et al. 2015; Athiwaratkun and Stokes 2017). The embedding consists of three parts: a $D_{LM} = 1500$ section corresponding to the language model's output, a $D_{MP} = 1500$ section for the output of the Max Pooling layer, and a $D_{BW} = 114$ section for the bag of words of the recent events (Pascanu et al. 2015).

The output of the *event classifier* is the probability that an individual file event is malicious based on its recent history. However as noted by (Wang, Stokes, and Marinescu 2019), we found that the performance of *event classifier* is not adequate to halt the execution of the file, $y_{e,t}$, and therefore we use a deep reinforcement model to augment the *event classifier's* prediction. To do so, we first construct the DRL's state, s_t. The state and $y_{e,t}$ are input to the Actor Critic DRL model to construct its reward function. We discuss the AC DRL model in detail in the following section. The *execution control model* also produces an execution control signal, h_t, which indicates whether or not the file's emulation has been halted.

Improved Inference Model. Once a file's emulation has been halted, it is then classified using the *improved inference model* which refines the event classifier's score to produce the final improved file classifier score, $y_{RL,t}$. The *improved inference model* also includes a baseline file classifier which produces a score y_f based on the first 200 events and provides an initial probability estimate if the file is malicious. If h_t is equal to 1,

$$\begin{aligned} y_{RL,t} &= \max\{y_{e,t-K+1}, \cdots, y_{e,t}\} \quad \text{if } y_f > 0.5 \\ y_{RL,t} &= \min\{y_{e,t-K+1}, \cdots, y_{e,t}\} \quad \text{if } y_f \leq 0.5 \end{aligned} \tag{1}$$

where $y_{e,i}$ is the event classifier's prediction at step i, K

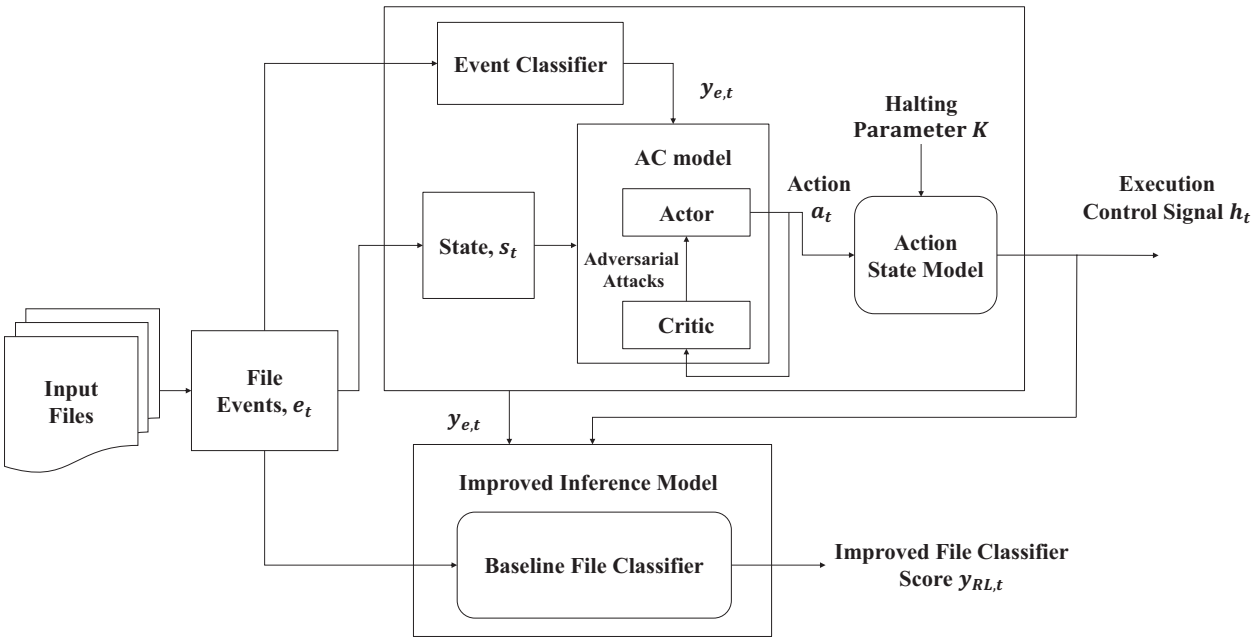

Figure 5: The full system structure based on DRL for halting the execution of an unknown file and improved malware classification, with its input as e_t and outputs are h_t and $y_{e,t}$.

is the number of most recent event probabilities to consider and $y_{RL,t}$ is the improved inference model's output, *i.e.*, the final prediction probability that the unknown file is malicious.

Actor Critic Model

In this section, we present how to construct an actor critic-based, deep reinforcement learning model to control the antimalware engine's execution. Also, we show that the critic can be treated as an adversarial learning-based attack simulator. Hence, the system can train a stronger actor to handle these dynamic attacks in addition to commodity malware behavior.

Our actor critic system generates two actions $\{a^1, a^2\}$ to control the file's execution in the sandbox environment. One action is to continue running the software and another is to halt the execution, $a^1 =$ **Continue**, $a^2 =$ **Halt**. It is possible for many different models to generate the actions including RNN-based models or the DQN-based model (Wang, Stokes, and Marinescu 2019). However, there is one difficulty that has not been previously addressed. After our antimalware system takes an action a_t, the executing file may change its next event behavior based on this action. Thus, it is very difficult for the system to run a static model to handle these dynamic behaviors without training. One reason why the actor critic system is useful for our purposes is because it has two models: an actor and a critic. The critic model can take the malware's dynamic adversarial behavior into consideration and output an evaluation score based on the action in the last step, while the actor can be trained against it. Another reason why we use an actor critic model is because it is known to converge more smoothly and have better training performance on a system with a large state space like our scenario (Wawrzyński and Tanwani 2013; Lillicrap et al. 2015).

Unlike a value-based or policy-based DRL model, an AC model leverages two neural networks to model the actor and critic, separately. At each step, the actor model takes a state s_t as its input and generates a sample action at state s_t. After performing the action, the system reaches a new state s_{t+1}. Next, the critic model looks back and estimates the sum of expected rewards using a value function by taking any possible actions a_t^i from state s_t to s_{t+1}. Then, we can generate the adversarial-attack modified signal $V'(s_t|\theta_t)$ based on the output of the value function, $V(s_t|\theta_t)$:

$$V'(s_t|\theta_t) = V(s_t|\theta_t) - \epsilon \cdot sign(\nabla_{a_t} V(s_t|\theta_t)). \quad (2)$$

The new adversarial-attack, modified value function signal $V'(s_t|\theta_t)$ is generated by subtracting an action-based perturbation $\epsilon \cdot sign(\nabla_{a_t} V(s_t|\theta_t))$ in the direction of the negative gradient, where $\epsilon \in (0, 1)$ is a user selected parameter corresponding to the strength of the adversarial attack. The gradient of V is the feedback to the actor model and is used as an attack signal.

Formally, an action-value function in a conventional actor critic model is given as:

$$Q(s_t, a_t^i|\theta_t) = r(s_t, a_t^i) + \hat{V}^{\pi_t}(s_{t+1}|\theta_t) - \hat{V}^{\pi_t}(s_t|\theta_t) \quad (3)$$

where $r(s_t, a_t^i)$ is the reward from taking action a^i at time step t and state s_t. $\hat{V}(s_t|\theta_t)$ is the estimated value function

for state s_t, *i.e.* the maximum reward that can be obtained at state s_t. Hence, the adversarial-attack modified action-value function estimator is derived as:

$$Q'(s_t, a_t^i|\theta_t) = r(s_t, a_t^i) + \hat{V'}^{\pi_t}(s_{t+1}|\theta_t) - \hat{V'}^{\pi_t}(s_t|\theta_t) \quad (4)$$

where $\hat{V'}(s_t|\theta_t)$ is the estimated adversarial-attack modified value function for state s_t as defined in (2).

As mentioned previously, the actor model and the critic model are both neural network-based estimators. The actor neural network is used to estimate the policy π_t at time step t. The critic model estimates the value function $\hat{V}(s_t|\theta_t)$, hence generating the adversarial-attack modified signal $\hat{V'}(s_t|\theta_t)$. The gradient for adjusting its parameters θ_t by leveraging the policy generated by the actor network and the adversarial-attack modified action-value function $Q'(s_t, a_t^i|\theta_t)$, is given as:

$$\begin{aligned} \nabla_\theta J(\theta_t) &= \sum_i \nabla_\theta \log \pi_\theta(a_t^i|s_t) Q'(s_t, a_t^i|\theta_t) \\ \theta_{t+1} &= \theta_t + \alpha \nabla_\theta J(\theta_t). \end{aligned} \quad (5)$$

It can be observed that an AC model calculates its action-value function $Q'(s_t, a_t^i|\theta_t)$ by using the adversarial-attack modified estimated value function $\hat{V'}(s_t|\theta_t)$, instead of using $\hat{V}(s_t|\theta_t)$ to estimate $Q(s_t, a_t^i|\theta_t)$ directly. This potentially gives the system an ability to handle the generated adversarial attacks based on the estimated rewards or penalty.

States, Actions and Rewards. As a deep reinforcement learning system, our actor critic model contains several key parameters, including the state (s_t), the action (a_t) and the reward (r_t). Their detailed definitions are given below and follow (Wang, Stokes, and Marinescu 2019).

States s_t: Our system's state takes three types of information into consideration including the event type, the location of this event in the file and the information of all previously seen events in the file. A graphical illustration of an individual state is given in Figure 6. The first item contains the event information itself, and the concatenation of the other two can be considered as its feature embedding. The event position in the file is represented by ρ_t, which will be later used to define the reward as well. The information of all events observed in the file up to time t is then represented by the histogram containing the frequency of these events.

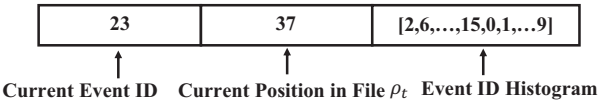

Figure 6: A graphical representation of states s_t.

Actions a_t: As mentioned previously, the actor model is making a decision at each timestep t to choose one action out of two: one is the **Continue** action (*i.e.* a_t = **Continue**), and the other is the **Halt** action (*i.e.* a_t = **Halt**). This decision generation procedure is also called policy generation. With the state s_t and adversarial critic model's output as the input, the actor model learns a policy π to decide which action to take at each timestep.

To represent this mathematically:

$$\pi_t = \arg\max_{a_t^i} Q'(s_t, a_t^i|\theta_t) \quad (6)$$

where $i = \{1, 2\}$, *i.e.* a_t^1 = **Continue** and a_t^2 = **Halt**. As illustrated in Figure 5, the actor network selects an action a_t for timestep t based on the policy as given in (6).

Rewards r_t: In our actor critic model, the reward is defined to guide our system to achieve two targets: 1) We assign a larger reward if the true label is correctly predicted by the event classifier, and 2) One of our performance criteria is based on the number of events executed before we make the *correct* decision. Hence, we assign a larger reward if a correct decision is made earlier rather than later. The reward is defined as:

$$r_t = (0.5 - |y_{e,t} - L|)e^{-\beta\rho_t} \quad (7)$$

where $y_{e,t}$ is event prediction generated by the most recent 200 events, $L \in \{0, 1\}$ is the ground truth file label, and β is a decay factor which is between 0 and 1. For $L = 0$, the file is benign, while $L = 1$ corresponds to a malicious file.

Action State Model. The action state model in Figure 5 takes the sequential action outputs from the actor model and then generates the execution control signal, h_t, to stop the program's execution. Although the actor model can already generate the **Halt** action, it is not a perfect response with high confidence if the AC model is not fully trained and noisy (considering the long training time for a DRL system). Hence, we use a trace-back setup to only halt the antimalware system's execution if there are at least K consecutive actions with value **Halt**, *i.e.* a_{t-K+1} to a_t are all equal to **Halt**. In this case, the action state model then stops the file's execution.

Experimental Results

In this section, we evaluate the performance of the proposed actor critic-based deep reinforcement learning model. The two baselines used in this work include the DQN system presented in (Wang, Stokes, and Marinescu 2019) and the antimalware engine's heuristic model. We first describe the setup and the datasets that are used for the experiments. Next, we evaluate the performance of the system in halting the execution of the unknown file. Finally, we compare the classification performance of the AC system with the DQN system and a recurrent deep learning malware classifier which classifies the events directly.

Setup. When possible, we follow the experimental setup used in (Wang, Stokes, and Marinescu 2019) for our DQN experiments and the hyperparameters which are shared between our AC model and the DQN model. For all experiments, the deep learning system is implemented with Keras (Chollet and others 2015) and Theano (Al-Rfou et al. 2016). The DQN model uses a deep neural network with 3 dense hidden layers of dimension 200. The last layer is a softmax layer which generates two outputs representing the expected rewards by taking the two actions.

For the actor critic model, there are two networks: one is for the actor model and the other is for the critic model. Both networks also contain three dense layers with a layer size of 200. The action model is followed by a softmax layer which generates the action to perform.

The minibatch size in all experiments is $B_{RL} = 50$. We set μ = 50,000 for the replay memory length. For the reward function, we set the decay factor $\gamma = 0.01$ in both models.

Datasets. The event datasets used in this study are derived from a collection of 75,000 emulation scans which is evenly split between malware and benign files. All of the files have distinct sequences. This collection is then randomly split into 50,000, 10,000 and 15,000 for training, validation, and testing, respectively.

Improved Stopping Performance. We next compare the halting performance of the AC DRL model to the DQN DRL model as well as the antimalware engine's heuristics. We use the production, Microsoft Windows Defender antimalware engine to collect our datasets. All the events are generated before the antimalware engine terminates its execution. Hence, the file length represents the number of events that were executed based on the engine's heuristics. When the file's execution is controlled by a deep reinforcement learning model, one possibility is that the model does not terminate before reaching the end of the event sequence. In this case, the DRL model was not able to make a decision beforehand, and the antimalware engine would have been permitted to continue execution after the heuristics would have stopped the processing of the file.

Table 1 provides the statistics about the percentage of files which were halted earlier than the engine's heuristics using the Actor Critic model and Deep Q-Network model (Wang, Stokes, and Marinescu 2019). In the table, N is the number of training files, and K is the number of consecutive **Halt** actions generated by the actor model. The percentage of files that each of the DRL models halts execution before the end of the file, α, is computed as:

$$\alpha = \frac{100\% * (\text{Total number of early halted files})}{(\text{Total number of files})}. \quad (8)$$

We make three observations from the results presented in the table. First, the percentage of files whose execution is halted by both the AC DRL model and the DQN DRL model earlier than by the heuristics continues to increase as the number of training file N increases. A better trained model allows the engine to halt execution earlier. Second, the modified AC model's performance is better than the DQN model for all K and N values. Third, the proportion of early halted files decreases with a larger K. Recall that K is number of consecutive actions where a_t = **Halt** before the DRL model stops the file's execution. Therefore, both DRL models fail to achieve larger number of K consecutive halting actions before reaching the end of the sequence. However, it can be observed that the AC model's performance is still very good. Over 93.6% of the files in the test set are halted early when $K = 20$ and the AC DRL model is only trained with N = 2000 files.

Improved File Classification. While the results in Table 1 indicate that emulation of the majority of the files can be halted earlier than the heuristics employed in the engine by using AC deep reinforcement learning, it is important to understand how early stopping affects the detection performance.

Model Structure	# of Files	K=10	K=15	K=20
DQN	N=30	71.5%	64.1%	58.3%
	N=200	82.9%	75.2%	69.2%
	N=2000	98.2%	95.1%	91.3%
Actor Critic	N=30	73.2%	69.6%	63.5%
	N=200	85.4%	77.8%	73.2%
	N=2000	99.4%	96.9%	93.6%

Table 1: The percentage of files that are halted earlier by using the proposed Actor Critic model and the DQN model, compared to the production antimalware engine heuristics.

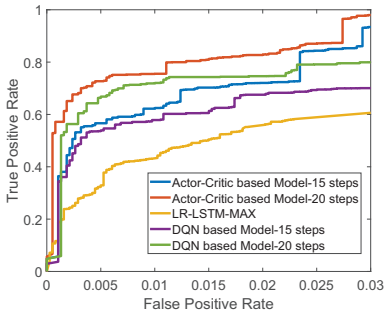

Figure 7: Comparison between the proposed DRL-based (DQN and Actor Critic) model for $K \in \{15, 20\}$ and the baseline file classifier.

We compare the LSTM-based baseline file classifier in (Athiwaratkun and Stokes 2017), the DQN model in (Wang, Stokes, and Marinescu 2019) and the Actor Critic model, where the two different DRL-based models choose $K \in \{15, 20\}$ as shown in Figure 7. The figure clearly indicates that the AC-based model with $K = 20$ offers significantly better performance compared to other systems, including both the DQN based models and the LSTM-based model. In particular, the Actor Critic DRL-based model with $K = 20$ offers a relative improvement of 83.4% for the true positive rate (TPR) at an false positive rate (FPR) of 1% compared to the LSTM-based file classifier, and a relative improvement of 9.9% for the TPR at an FPR of 1% compared to the best performing DQN model.

Related Work

Prior work that is related to the proposed DRL-based malware detection system generally falls into two areas: deep learning for malware classification and deep reinforcement learning.

Deep Learning for Malware Classification. A number of authors have explored using deep learning for malware classification, and these works typically use either stan-

dard deep neural networks or recurrent models. The first paper on malware classification used a shallow neural network (Kephart 1994). Dahl et al. (Dahl et al. 2013) explored the performance of a deep neural network for malware classification. This architecture was extended by (Huang and Stokes 2016) to include multi-task learning. Another DNN model was proposed for malware classification by (Saxe and Berlin 2015).

Behavioral-based malware detection using recurrent neural networks and echo state networks was first proposed by Pascanu et al. (Pascanu et al. 2015). The baseline file classifier and event classifier employed in this work use the LSTM-based recurrent neural network architectures proposed by (Athiwaratkun and Stokes 2017). Recurrent models which proposed combining a CNN and an LSTM were proposed by (Kolosnjaji et al. 2016) and (Agrawal et al. 2018).

Deep Reinforcement Learning. Deep reinforcement learning has been an active research topic since the 1990s, and most of the early works are in the system and control field, including optimal control and robotic system control (Sutton 1984; Williams 1992; Littman 1994; Kaelbling, Littman, and Moore 1996; Sutton and Barto 1998).

One of the main reasons limiting the applicability of reinforcement learning in other domains is due to the computational complexity when the state-space becomes large. During the last decade, researchers have leveraged deep neural networks to tackle this issue, such that reinforcement learning can be used in other applications. For example, a deep neural network-based Q-learning (DQN) model, proposed by Mnih, et al. (Mnih et al. 2013; 2015), has been successfully used to train an AI system to play Atari games. By utilizing Monte Carlo tree search, Silver, et al. (Silver et al. 2016; 2017) proposed an actor critic deep reinforcement learning model which can handle large state-spaces. The model has been successfully trained to play Go games and even beat human Go masters for the first time in history.

Progress has also been made in all three types of deep reinforcement learning: value-based algorithms (Van Hasselt, Guez, and Silver 2016; Wang et al. 2016), policy-based algorithms (Schulman et al. 2015; Duan et al. 2016; Gu et al. 2017; O'Donoghue et al. 2017), and actor critic based algorithms (Lillicrap et al. 2015; Mnih et al. 2016; Haarnoja et al. 2018). Some recent work also leverages the concepts of multi-agent and multi-model to boost a DRL system's performance through multiple model or agent cooperation and information sharing (Narendra, Mukhopadyhay, and Wang 2015; Narendra, Wang, and Mukhopadhay 2016; Foerster et al. 2018; Wang and Jin 2018; Zhang et al. 2018).

In (Wang, Stokes, and Marinescu 2019), Wang et al. propose a Deep Q-Network reinforcement learning model for malware classification. This model achieves good performance when there are no adversarial attack signals. The actor critic DRL model introduced in this paper is the first deep reinforcement learning model that can handle adversarial attacks for file emulation control.

Conclusion

A recently proposed system (Wang, Stokes, and Marinescu 2019) utilizing a Deep Q-Network offered significant improvement over the heuristics employed in a production antimalware engine for the task of determining when to halt the emulation of an unknown file. This DQN model also led to significant improvements for the task of predicting if the file is malicious or benign.

In this work, we extend (Wang, Stokes, and Marinescu 2019) by replacing the DQN with an actor critic-based, deep reinforcement learning model. This new AC DRL execution halting model yields significant gains compared the the DQN approach. First for the task of halting execution, 93.6% of the files are stopped earlier than when controlled by the engine's heuristics. Similarly, the AC model halts 2.5% earlier than the DQN model. In addition, our results for the task of file classification, the AC model shows an improvement in the true positive rate of 9.9% from 69.5% to 76.4% at a false negative rate of 1% compared to the DQN model. Thus, we believe that the AC-based execution halting model can lead to increased protection for an extremely large numbers of computer users.

References

Agrawal, R.; Stokes, J. W.; Marinescu, M.; and Selvaraj, K. 2018. Robust neural malware detection models for emulation sequence learning. In *Proceedings of the IEEE Military Communications Conference (MILCOM)*. IEEE.

Al-Rfou, R.; Alain, G.; Almahairi, A.; Angermueller, C.; Bahdanau, D.; Ballas, N.; Bastien, F.; Bayer, J.; Belikov, A.; Belopolsky, A.; et al. 2016. Theano: A python framework for fast computation of mathematical expressions. *arXiv preprint arXiv:1605.02688* 472:473.

Athiwaratkun, B., and Stokes, J. W. 2017. Malware classification with lstm and gru language models and a character-level cnn. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2482–2486.

Chollet, F., et al. 2015. Keras. https://github.com/keras-team/keras.

Dahl, G. E.; Stokes, J. W.; Deng, L.; and Yu, D. 2013. Large-scale malware classification using random projections and neural networks. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 3422–3426. IEEE.

Duan, Y.; Chen, X.; Houthooft, R.; Schulman, J.; and Abbeel, P. 2016. Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning (ICML)*, 1329–1338.

Foerster, J. N.; Farquhar, G.; Afouras, T.; Nardelli, N.; and Whiteson, S. 2018. Counterfactual multi-agent policy gradients. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Gu, S.; Lillicrap, T.; Ghahramani, Z.; Turner, R. E.; and Levine, S. 2017. Q-prop: Sample-efficient policy gradient with an off-policy critic. In *5th International Conference on Learning Representations (ICLR)*.

Haarnoja, T.; Zhou, A.; Abbeel, P.; and Levine, S. 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, 1856–1865.

Hu, W., and Tan, Y. 2017. Generating adversarial malware examples for black-box attacks based on gan. *arXiv preprint arXiv:1702.05983*.

Huang, W., and Stokes, J. W. 2016. Mtnet: A multi-task neural network for dynamic malware classfication. In *Proceedings of Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*, 399–418.

Kaelbling, L. P.; Littman, M. L.; and Moore, A. W. 1996. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research* 4:237–285.

Kephart, J. O. 1994. A biologically inspired immune system for computers. In *In Artificial Life IV: Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems*, 130–139. MIT Press.

Kolosnjaji, B.; Zarras, A.; Webster, G.; and Eckert, C. 2016. Deep learning for classification of malware system call sequences. In *Australasian Joint Conference on Artificial Intelligence*, 137–149. Springer International Publishing.

Lillicrap, T. P.; Hunt, J. J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; and Wierstra, D. 2015. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

Littman, M. L. 1994. Markov games as a framework for multi-agent reinforcement learning. In *International Conference on Machine Learning (ICML)*, volume 157, 157–163.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; and Riedmiller, M. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

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. 2015. Human-level control through deep reinforcement learning. *Nature* 518(7540):529–533.

Mnih, V.; Badia, A. P.; Mirza, M.; Graves, A.; Lillicrap, T.; Harley, T.; Silver, D.; and Kavukcuoglu, K. 2016. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning (ICML)*, 1928–1937.

Narendra, K. S.; Mukhopadyhay, S.; and Wang, Y. 2015. Improving the speed of response of learning algorithms using multiple models. *arXiv preprint arXiv:1510.05034*.

Narendra, K. S.; Wang, Y.; and Mukhopadhay, S. 2016. Fast reinforcement learning using multiple models. In *IEE Conference on Decision and Control (CDC)*, 7183–7188. IEEE.

O'Donoghue, B.; Munos, R.; Kavukcuoglu, K.; and Mnih, V. 2017. Pgq: Combining policy gradient and q-learning. In *5th International Conference on Learning Representations (ICLR)*.

Pascanu, R.; Stokes, J. W.; Sanossian, H.; Marinescu, M.; and Thomas, A. 2015. Malware classification with recurrent networks. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 1916–1920. IEEE.

Saxe, J., and Berlin, K. 2015. Deep neural network based malware detection using two-dimensional binary program features. *Malware Conference (MALCON)*.

Schulman, J.; Levine, S.; Abbeel, P.; Jordan, M.; and Moritz, P. 2015. Trust region policy optimization. In *International Conference on Machine Learning (ICML)*, 1889–1897.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. 2016. Mastering the game of go with deep neural networks and tree search. *Nature* 529(7587):484–489.

Silver, D.; Schrittwieser, J.; Simonyan, K.; Antonoglou, I.; Huang, A.; Guez, A.; Hubert, T.; Baker, L.; Lai, M.; Bolton, A.; et al. 2017. Mastering the game of go without human knowledge. *Nature* 550(7676):354.

Sutton, R. S., and Barto, A. G. 1998. *Introduction to reinforcement learning*, volume 135. MIT Press Cambridge.

Sutton, R. S. 1984. Temporal credit assignment in reinforcement learning. *PhD thesis, University of Massachusetts*.

Van Hasselt, H.; Guez, A.; and Silver, D. 2016. Deep reinforcement learning with double q-learning. In *AAAI*, volume 16, 2094–2100.

Wang, Y., and Jin, H. 2018. A boosting-based deep neural networks algorithm for reinforcement learning. In *2018 Annual American Control Conference (ACC)*, 1065–1071. IEEE.

Wang, Z.; Schaul, T.; Hessel, M.; Hasselt, H.; Lanctot, M.; and Freitas, N. 2016. Dueling network architectures for deep reinforcement learning. In *International Conference on Machine Learning (ICML)*, 1995–2003.

Wang, Y.; Stokes, J. W.; and Marinescu, M. 2019. Neural malware control with deep reinforcement learning. In *Proceedings of the IEEE Military Communications Conference (MILCOM)*.

Wawrzyński, P., and Tanwani, A. K. 2013. Autonomous reinforcement learning with experience replay. *Neural Networks* 41:156–167.

Williams, R. J. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning* 8(3-4):229–256.

Zhang, K.; Yang, Z.; Liu, H.; Zhang, T.; and Basar, T. 2018. Fully decentralized multi-agent reinforcement learning with networked agents. In *International Conference on Machine Learning*, 5867–5876.