Proceedings of the 39th International Conference on Machine Learning (ML4Cyber workshop),
17-23 July 2022, Baltimore, Maryland, USA
https://icml.cc/

ACD-G: Enhancing Autonomous Cyber Defense Agent Generalization Through Graph Embedded Network Representation

Josh Collyer [1] **Alex Andrew** [1] **Duncan Hodges** [2]

Abstract

The adoption of autonomous cyber defense agents within real-world contexts requires them to be able to cope with differences between their training and target environments, bridging the simulation to real gap to provide robust, generalized defensive responses. Whilst the simulation to real gap has been studied in-depth across domains such as robotics, to date there has been minimal research considering generalizability in the context of cyber defense agents and how differences in observation space could enhance agent generalizability when placed into environments that differ from the training environment. Within this paper, we propose a method of enhancing agent generalizability and performance within unseen environments by integrating a graph embedded network representation into the agent's observation space. We then compare agent performance with and without a graph embedded network representation based observation space within a series of randomized cyber defense simulations. We find that there is a trade-off between the effectiveness of the graph embedding representation and the complexity of the graph, in terms of node count and number of edges.

1. Introduction

AI-based cyber-attacks are no longer science fiction and have been fuelled by the recent advances in artificial intelligence technologies (Kaloudi & Li, 2020) with a growing body of work exploring areas such as autonomous penetration testing (Chaudhary et al., 2020; Chowdhary et al., 2020; Tran et al., 2021). These works have driven the research and development of autonomous cyber defense systems which are able to respond in a relevant timescale and at a sufficient scale. Modern networks however are non-heterogeneous and dynamic with new application deployment paradigms such as serverless and containerisation becoming common place alongside standard server deployments. This raises challenges when trying to train and deploy decision making agents due to the high likelihood of differences being present between the training and target environments. This problem, typically referred to as the 'simulation to reality gap', is not something that is unique to a cyber defense setting and has been researched heavily within contexts such as robotics (Zhao et al., 2020) and autonomous driving (Balaji et al., 2020). To date however, generalizability has not been a specific consideration within the autonomous cyber defense literature with a limited number of papers discussing it as a key barrier to real-world deployment.

Current state of the art approaches towards autonomous cyber defense have typically been focused on proving the feasibility of agents in a range of settings; such as Industrial Control Systems (ICS) (Mern et al., 2021) and Software Defined Networks (SDN) (Akbari et al., 2020). In addition, a significant body of work exists that has considered specific problem formulations such as; Optimal Stopping for Intrusion Prevention (Hammar & Stadler, 2021), autonomous network defender performance when faced with adversarial perturbations (Molina-Markham et al., 2021), as well as several works which have developed flexible and re-usable agent training environments (Li et al., 2021; Standen et al., 2021; Microsoft Defender Research Team., 2021). This extensive body of work demonstrates the appetite for autonomous cyber defense agents, yet we still do not fully understand how to increase the generalizability of autonomous defense agents.

Generalizability and the ability to transfer an agent from its training environment to unseen environments is likely to be key for the real-world adoption of autonomous cyber defense agents. This is a complex problem to tackle, given that the cyber security domain is both complex and adversarial in nature (Kott et al., 2016). Cyber environments are non-stationary by their very nature, which is a known challenge for Reinforcement Learning (RL) approaches (Igl et al.,

[1]Defence Science and Technology Laboratory (Dstl) [2]Defence Academy, Cranfield University. Correspondence to: Joshua Collyer <jcollyer@dstl.gov.uk>.

Presented at the Workshop on Machine Learning for Cybersecurity (ML4Cyber) as part of the Proceedings of the 39^{th} International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022.

2020). Several works propose approaches to tackle this problem such as introducing data augmentation (Kostrikov et al., 2020) or a representational learning step (Agarwal et al., 2021) to enhance the agents observation space and increase agent performance. Unfortunately, the vast majority of these approaches are image based and therefore are not directly transferable to a cyber security setting. Evidence does however suggest that graphs are ideal representations of computer networks (Dawood, 2014) and this is further reinforced with several works using graphs as the underlying representation of RL training simulations (Ridley, 2018; Microsoft Defender Research Team., 2021). We am to build upon this previous body of work by exploiting the graph representation and supply it as an input into autonomous agents as a means of providing a map or diagram of the network to be defended.

In this paper, we investigate the relationship between agent performance and the composition of an agent's observation space, using unseen environments that are structurally different from the training environment. The agent is placed within an intrusion response setting across a three different environment sizes. The agent's objective is to prevent a stochastic attacker from compromising the whole of the network with the reward function favoring minimisation of compromise. A stochastic attacker was chosen over a deterministic attacker in order to provide an abstract representation of different attackers across episodes and to ensure the behavior was diverse across training and evaluation. We formulate two observation space conditions; one which incorporates an adjacency matrix representation of the network to be defended, and another which incorporates a graph embedded network representation. We then train two sets of agents each with a different network representation as part of their observation space within a static training environment for five million training timesteps, before comparing agent performance within a series of randomized evaluation environments.

We find that whilst agents under both observation space conditions exhibit similar training time performance, the agents that observe an embedded network representation perform significantly better in unseen 20 node environments whilst performing similarly or worse in both the 10 and 40 node environments. This empirical evidence suggests that there is a trade-off between the environments network complexity and the effect of exploiting the graph structure inherent of cyber security environments.

The remainder of this work has the following structure. §Section 2 describes the methods used to generate the empirical results, covering aspects such as the agent's observation space and action space, training and evaluation protocols and metrics collected. The experimental results are then presented in §Section 3 before finishing with §Section 4 describing the key conclusions and then §Section 5 detailing potential future work directions.

2. Methods

2.1. Simulation Environment

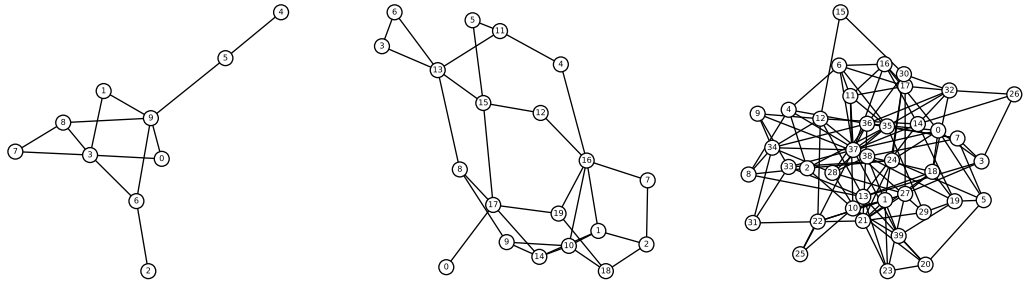
A custom, highly abstract, flexible training environment called YAWNING TITAN (henceforth YT) was developed to provide a means of training autonomous cyber defense agents [1]. YT is an OpenAI Gym (Brockman et al., 2016) based simulation which can represent a range of cyber scenarios, such as intrusion response and crown jewels defense. As the simulation is OpenAI compatible, it provides a means of seamless experimentation with open source RL algorithm libraries such as Stable Baselines 3 (Raffin et al., 2021).

Each scenario within this environment is composed of two components. The *first* is an undirected graph $G = (V, E)$ where V is a set of vertices and E is a set of edges. Each vertex represents a computing device within the computer network and each edge represents a connection between computing devices. This undirected graph G represents the network that is going to be defended by the autonomous cyber defender. The *second* is a configuration file which contains a range of parameters used for the simulation environment. This provides a means of fine grained control of key aspects of the simulation, such as what actions form the defending agents action space, what reward function should be used and how many time-steps each episode should be (Sutton & Barto, 2018).

2.2. Scenario Initialization

For the purposes of this work, three environment sizes were chosen, 10 nodes, 20 nodes and 40 nodes respectively. This range was chosen primarily to provide a means of determining the effectiveness of the graph based observation space as well as providing the agents with an incrementally increasing challenge. The undirected graphs for all three environment sizes were generated using an amended Erdős-Rényi (Paul & Alfréd, 1959) graph generator which begins by creating a random graph with n nodes and an edge probability value of p. Once created, the amended generator then checks that every vertex within the graph has at least one edge and for those vertices without edges, randomly creates an edge. This ensures that there are no vertices without any edges. This is important because if any vertex is inaccessible and the lose criteria for the blue agent is total compromise, the red agent would be unable to compromise nodes with no edges and therefore, the blue agent would win consistently. A selection of sample graphs can be seen in Fig. 1. The edge probability value p was fixed throughout all of the ex-

[1] YAWNING TITAN is available at: https://github.com/dstl/YAWNING-TITAN



(a) 10 Node (b) 20 Node (c) 40 Node

Figure 1. Example Simulated Network Topologies

periments outlined within this paper to $p = 0.1$. This value was chosen to balance between having a sufficient number of edges present within each environment graph while also ensuring that every node was not connected to every other node, creating highly connected, dense graphs.

During training, a single YT scenario was created at the beginning of training and then used as the environment for the training duration. However during evaluation we use a collection of different YT scenarios created as evaluation environments. The only variable change between the evaluation environment initialization and training environment initialization was the underlying topology of the graphs. This provided a means of creating unseen environments, ensuring that if there were differences between both observation space conditions, it could be attributed to the observation space and not other parameter changes.

Each vertex of the graph has two attributes. The first is a node vulnerability score $nv \in \mathbb{R} : nv \in [l, u]$, where l and u denote the vulnerability score lower and upper bounds. This is defined in the scenarios configuration as described in Section 2.2. This can be considered a highly abstract representation of the number of vulnerabilities a computing device currently has, and is used as an attack multiplier by the attacking agent. The second attribute c denotes the binary compromised status where

$$c = \begin{cases} \text{uncompromised}, & \text{if } c = 0, \\ \text{compromised}, & \text{if } c = 1. \end{cases}$$

At the beginning of an episode all vertices are set to $c = 0$, denoting uncompromised status. The node vulnerability score for each vertex is sampled uniformly from $nv \sim \mathcal{U}[l, u]$, where l and u are the lower and upper bounds. In addition to these two steps, an entry node is selected and acts as the red agent's primary route into the network. This entry node can be viewed as an externally facing device which is the initial target for compromise.

Table 1. Blue Agent Action Descriptions

Action	Description
patch	Reduces the vulnerability score of the targeted node by 0.2
recover	Recovers a compromised node to its initial state at the beginning of the episode
noop	Performs no operation - effectively skipping a turn

Once the environment is created, one blue, defending agent and one red, attacking agent is placed into the environment. The blue agent is a *learner* and uses RL to learn how to achieve the task at hand, whereas the red agent is stochastic and is described in more detail in Section 2.5. The objective of the blue agent is to successfully stop the red agent from compromising all of the nodes within the network. The red agent's objective is to achieve total compromise. These objectives were chosen due to their similarities with the staging phase of a ransomware attack, where the attacker wants to impact the highest percentage of assets as possible as well as ensuring that training episodes would not terminate too early.

2.3. Defending Agent Configuration

2.3.1. Action Set

The defenders action space, a, consists of three actions, *patch*, *recover* and *noop*. Each of these actions are described in Table 1.

This action space composition has been formulated to provide the agent with a means of both proactive (*patch*) and reactive (*recover*) response. The $patch$ value of 0.2 was chosen to provide agents a means of reducing node vulnerability by a meaningful amount, whilst restricting their ability to create invulnerable hosts rapidly and 'game' the simulation. In reality, these actions would both likely have different costs and associated timescales, which would add another layer of complexity and would need to be incorporated into either the agent's reward function or the simulation itself. However this is outside the scope of this work.

2.3.2. Observation Space

We compose two different observation spaces which we labelled $standardObs$ and $graphObs$. The conditions for both observation spaces share a number of common features and include all node vulnerability values NV, where $NV = nv_1, nv_2, ..., nv_n$ and compromised statuses C, where $C = c_1, c_2, ..., c_n$. Alongside these shared features, both sets of agents are provided with a different structural representation of the network they are defending. $standardObs$ provides the agents with a flattened adjacency matrix network representation A where $\|A\| = n \times n$ and n is the number of nodes within the network. $graphObs$ provides the agents with a fixed length graph embedded network representation F, generated using the FEATHER-G whole graph embedding algorithm (Rozemberczki & Sarkar, 2020) where $\|F\| = 500$.

Table 2. Observation Space Conditions

Observation Space	NV	C	A	F
standardObs	✓	✓	✓	
graphObs	✓	✓		✓

NV: Node Vulnerability Scores C: Compromised Statuses A: Adjacency Matrix F: FEATHER-G Graph Embedding

The FEATHER-G algorithm is a graph embedding approach which is capable of embedding non-attributed whole graphs. FEATHER-G calculates graph characteristic functions using the probabilities derived from the transition probabilities of random walks in a computationally efficient manner. Its computational efficiency was one of the reasons it was chosen for this experimentation. FEATHER-G does not require any computationally intensive training because it derives the embeddings from the statistical proprieties of the inputted graph. This is well suited to an RL use-case due to the number of training experiences needed to generate useful policies. In addition, it provides a means of generating graph embedding for the observation space quickly. Whilst no computational limitations were encountered, due to the limited number of actions that could make changes to the underlying graph, if agents are able to take actions such as isolating nodes, generating a new graph embedding representation should be computationally inexpensive.

The second reason is that, due to not requiring training and deriving the embeddings from the statistical properties of the input graph, the graph embeddings are deterministic. This means that when faced with the same graph, FEATHER-G will output an identical graph embedding. This property was viewed favorably as it could support generalizability by providing the agent with a common looking map of the network it was defending at any given moment. This observation space transformation was implemented using an OpenAI Observation Wrapper[2].

2.4. Reward Function

The reward function used for all experiments was a score based on the percentage of nodes compromised. If we let N be the total number of nodes within the network and Co be the total number of nodes compromised, then the reward function R is:

$$R = \frac{N - Co}{N}.$$

This reward function incentivizes the blue agent to minimize the number of compromised nodes within the network and thus increase it's per-timesteps reward. The terminal episode reward is 100 for successfully surviving until the episode end and -100 for unsuccessfully surviving until episode end. This means that for a 500 timestep episode, the optimal reward is 599.

2.5. Attacking Agent Configuration

The simulation would not be complete without an attacking agent. Each episode has one red agent which is represented by a probabilistic attacker with two possible actions - a basic attack action or a zero-day action. The basic attack action is an attack on a node where the 'attack strength', as, of the attack is based upon both the attacker's skill level, and the target node's vulnerability score, nv_n. The attacker skill, s, is defined at the the beginning of a scenario to be between $0 < s \leq 100$, giving the attack strength as:

$$as = \frac{s^2}{s + (1 - nv_n)}.$$

Once the attack strength as has been calculated, a threshold value t is uniformly sampled from $t \sim \mathcal{U}[0, 100]$ and the

[2]Observation space wrappers are used to transform OpenAI Gym Observation Space objects - An example implementation can be found here - https://github.com/openai/gym/blob/master/gym/wrappers/flatten_observation.py

attacks success is determined by:

$$c = \begin{cases} \text{unsuccessful}, & \text{if } as < t, \\ \text{successful}, & \text{if } as/geqt. \end{cases}$$

The second action is a zero day action and is a guaranteed node compromise, regardless of the node's vulnerability score. At the beginning of each episode the red agent starts with a single zero day action and prioritizes its usage. When the agent has no zero-days available, the agent will use its basic attack action to randomly target connected, uncompromised nodes. The red agent then periodically gains additional zero days at a rate of one per three timesteps. This rate of zero day generation was chosen to ensure that a red agent could consistently make progress within a network even when a blue agent had reduced the node vulnerabilities significantly. In addition, a stochastic attacker was chosen (as opposed to a deterministic attacker) to provide an abstract representation of different attackers across episodes, and ensure the attacker behavior was diverse across training and evaluation.

2.6. Algorithm

All the agents within the experiments used the Proximal Policy Optimization (PPO) (Schulman et al., 2017) algorithm which has been implemented as part of the Stable Baselines 3 Deep RL library. PPO was chosen primarily for its strong performance across a range of different tasks. The default hyperparameters provided within Stable Baselines 3 were used, no hyperparameter optimization was undertaken and the same hyperparameters were used across all observation space conditions and environment sizes.

2.7. Training and Evaluation Protocol

The training and evaluation protocol outlined below was used for both the $StandardObs$ and $GraphObs$ agents. All agents were trained and evaluated using Amazon Web Services (AWS) Sagemaker Notebook Instances using a compute optimized ml.c5.9xlarge with 36 CPUs and 72 GiB RAM.

2.7.1. Training Protocol

At the beginning of training, three random seeds were chosen. These random seeds were then used to create three uniquely seeded agents for each of the observation space conditions resulting in six PPO agents. Each of these agents were then trained individually for a total of 5 million timesteps within a training environment unique to the agent. This number of training timesteps was chosen to provide agents with sufficient training time, especially those within the larger environment sizes. It is shown in Fig. 2 that across all environment sizes the mean reward for agents is typically still increasing at five million timesteps, suggesting the agents are still learning. Techniques such as domain randomisation were not implemented as part of the training process in order to allow us to measure the raw effectiveness of the agent's observation space composition when faced with unseen environments.

2.7.2. Evaluation Protocol

Once trained, all agents were set to deterministic mode and then placed into 50 random environments for 10 episodes, each resulting in 500 evaluation episodes. Each of the 50 evaluation environments used a unique, randomly initialized graph generated using the amended Erdős-Rényi graph generator described in Section 2.2, with an edge probability value of $p = 0.1$ as the environments topology. For each of the evaluation episodes, both the episode length and reward attained were recorded.

2.8. Metrics Reported for Training and Evaluation

In order to provide a means of comparison between different training environments and between different agent's performance, a number of evaluation metrics were collected during agent training and evaluation. During agent training, an expanding window mean was calculated for both Episode Reward and Episode Length. This provided a means of tracking the progress of training and the data collected is shown in Fig. 2.

During the evaluation phase, we collected both Episode Reward and Episode Length for each agent evaluation episode. This data was then used to calculate a range of summary statistics such as max, min, standard deviation and mean. Following the procedure outlined in the work of (Henderson et al., 2017) these evaluation metrics were then aggregated in several ways to produce a robust representation of agent performance.

Alongside reporting the mean, we also report the truncated mean where scores from the upper and lower quartile were removed and then mean was recalculated. This procedure ensures that any outliers are removed (such as those produced by a very highly performing agent) and provides a robust representation of the central tendency across all agents trained within the same conditions. Using this truncated mean, a 95% confidence interval was also calculated to provide a means of comparing variability and robustness.

3. Results

3.1. Training

To understand the training performance, both the Mean Reward and Mean Episode Length calculated during agent

Figure 2. Mean Reward and Mean Episode Length during five million training timesteps across all three environment sizes

training can be found in the Fig. 2. Each line depicted within Fig. 2 is the mean across all three agents trained within that environment and observation space condition. Each line has a shaded area which depicts the width of the standard deviation about the mean. Fig. 2 shows that when comparing agents within the same environment size, the composition of the observation space neither aids or hinders the agents, with both agent types exhibiting highly similar scores for both reward and episode length. The data also suggests there is a relationship between agent training and environment size. We find that when examining episode reward, both $StandardObs$ and $GraphObs$ agents within the 10-node and 20-node environments begin by rapidly improving performance until approximately 750,000 timesteps and then continue to make smaller incremental improvements until the end of training.

In contrast, the agents within the 40-node environment size begin strongly and then make incremental progress throughout training. This behavior is partially explained within the right hand panel of Fig. 2. Again, for agents trained within the 10 and 20 node environment sizes a similar trend is exhibited as seen in the mean reward data. The agents start poorly, rapidly improve and then have a longer period of incremental improvement. However, when viewing the mean episode data, we can see that the for the 40-node environments, both $StandardObs$ and $GraphObs$ agents are achieving a near optimal score in terms of episode length from the very beginning. It is reasonable to conclude this is likely a function of the environment's size and the number of attackers within the simulation. The single attacker simply cannot compromise the whole of the 40 node network quickly enough. The fundamental insight however is that regardless of the observation space composition used, the agents are performing comparably and therefore provides a useful baseline to take into agent evaluation.

3.2. Evaluation

Table 3 shows the experimental results for this study. For the 10-node environment, the $standardObs$ agents typically perform better in terms of both reward and episode length across all measures. In contrast, $graphObs$ agents within 20-node environment size clearly benefit from the addition of a graph embedded network representation with the significant differences between metrics such as mean reward, truncated mean reward and confidence intervals.

However, this trend changes within the 40-node environment size where the results are mixed. Interestingly, the results from the 40-node environment suggest that whilst both $standardObs$ and $graphObs$ agents are performing comparably in terms of episode length, the $graphObs$ agents are performing better in terms of reward. The most notable difference is seen in the maximum reward, with $graphObs$ successfully achieving the maximum - significantly higher than that achieved by $standardObs$ in the same environment size.

These results individually provide insight into agent performance within specific environment sizes, but when considered together also tell another story. The results suggest that there is a trade-off between when the graph embedded network representation is helpful or not based on environment complexity in terms of number of nodes and edges. Fig. 1 provides visualisation of example environment graphs used during experimentation.

The results presented in Table 3 suggest that the graph embedded network representation makes little difference within the 10-node environment size. This theoretically makes

	10-node		20-node		40-node	
	standardObs	*graphObs*	*standardObs*	*graphObs*	*standardObs*	*graphObs*
Reward						
Mean	**150.4**	125.19	138.38	**181.94**	100.79	**122.59**
Max	599	599	599	599	430.05	**599**
Min	-93.4	**-93**	-85.9	**-83.1**	-69.23	**-67.25**
Std	233.71	**213.19**	177.84	**178.95**	**127.92**	173.25
Trunc. Mean	**137.06**	117,35	143.33	**187.19**	103.26	**123.14**
99% CI	(70.38, 203.73)	(57.31, 177.4)	(99.52, 187.13)	(146.38, 228)	(72.42, 134.1)	(77, 169.29)
Ep. Length						
Mean	**295.38**	279.21	351.8	**396.04**	**359.33**	355.37
Max	500	500	500	500	500	500
Min	**16**	15	37	**40**	70	**73**
Std	**231.08**	232.52	208.6	**185.24**	**185.17**	187.02
Trunc. Mean	**285.8**	277.9	373.45	**421.1**	**372.7**	366.56
99% CI	(223.18, 348.42)	(214.53, 341.28)	(328.32, 418.58)	(389.43, 452.77)	(328.45, 416.93)	(325.35, 407.76)

Table 3. Evaluation Metrics for each observation space condition and environment size. Bold numbers are used to highlight the highest scores between observation space conditions in each environment size.

sense because of how simple the environment is in terms of number of nodes and number of edges. A 500 dimension network representation is likely overkill and an adjacency matrix representation is more succinct. As the complexity grows in the 20-node environment, the graph embedded network representation significantly helps and provides the agents with an enhanced view of the cyber terrain it is defending. The effect of the enhanced network representation becomes mixed when moving to the 40-node environment size and can likely be attributed to the increased number of edges and density of the graph.

4. Conclusion

To conclude, this paper has explored the impact of including a graph embedded network representation into the observation space of an autonomous cyber defense agent. Two observation space conditions were created, one with an adjacency matrix network representation and one with a graph embedded network representation. Agents using both observation space conditions were then trained within a unique environment for five million timesteps before being evaluated within a series of unseen randomized environments.

The primary finding of this work suggests that there is a trade-off between the effectiveness of introducing graph embedded network representations and the complexity of the network to defend. The results suggest that the graph embedding may become less useful when the environment is either simple (in terms of number of nodes and number of edges) as its greatly increasing complexity when compared to an adjacency matrix or highly dense (in terms of number of edges) as it could be over simplifying the relationships. However, when faced with a network which is of modest node count and edge density, the graph embedded network representation provides a significant boost in performance within unseen environments. This demonstrates the utility of including the enhanced network representation as part of an autonomous cyber defense agent and raises some interesting questions related to agent observation space composition, especially when the focus is on developing generalizable agents.

5. Future Work

The results from this paper highlight several future work areas. Only three different environment sizes were used within this study. One possible avenue of further study would be to explore the identified trade-off between environment complexity and potency of a graph embedded network representation in more detail. This could be done by conducting a broader set of experiments across a larger number of environment sizes.

Another area to investigating would be the impact that domain randomisation has on the generalizability performance when added to the training environment generation. This could change aspects of the environment generation process such as the values for the node vulnerability lower and upper bounds or the edge probability within the graph generator. This enhanced variety of training experience could enhance performance across both observation conditions, and it would be interesting to see if the trade-off is still exhibited.

Moreover, exploring the possibility of creating global vulnerability and compromised status observation features alongside the network embedding could also be a potential avenue for future work. This could provide insight into whether it is possible to train agents which can generalize across different environment sizes. This could open the door to

the possibility of agents being able to perform well within a range of different environments in terms of both size and topology.

References

Agarwal, R., Machado, M. C., Castro, P. S., and Bellemare, M. G. Contrastive behavioral similarity embeddings for generalization in reinforcement learning, 2021. URL https://arxiv.org/abs/2101.05265.

Akbari, I., Tahoun, E., Salahuddin, M. A., Limam, N., and Boutaba, R. Atmos: Autonomous threat mitigation in sdn using reinforcement learning. In *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*, pp. 1–9. IEEE, 2020.

Balaji, B., Mallya, S., Genc, S., Gupta, S., Dirac, L., Khare, V., Roy, G., Sun, T., Tao, Y., Townsend, B., et al. Deepracer: Autonomous racing platform for experimentation with sim2real reinforcement learning. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2746–2754. IEEE, 2020.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. Openai gym, 2016.

Chaudhary, S., O'Brien, A., and Xu, S. Automated post-breach penetration testing through reinforcement learning. In *2020 IEEE Conference on Communications and Network Security (CNS)*, pp. 1–2. IEEE, 2020.

Chowdhary, A., Huang, D., Mahendran, J. S., Romo, D., Deng, Y., and Sabur, A. Autonomous security analysis and penetration testing. In *2020 16th International Conference on Mobility, Sensing and Networking (MSN)*, pp. 508–515. IEEE, 2020.

Dawood, H. A. Graph theory and cyber security. In *2014 3rd International Conference on Advanced Computer Science Applications and Technologies*, pp. 90–96. IEEE, 2014.

Hammar, K. and Stadler, R. Learning intrusion prevention policies through optimal stopping. In *2021 17th International Conference on Network and Service Management (CNSM)*, pp. 509–517. IEEE, 2021.

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., and Meger, D. Deep reinforcement learning that matters. *CoRR*, abs/1709.06560, 2017. URL http://arxiv.org/abs/1709.06560.

Igl, M., Farquhar, G., Luketina, J., Boehmer, W., and Whiteson, S. Transient non-stationarity and generalisation in deep reinforcement learning. *arXiv preprint arXiv:2006.05826*, 2020.

Kaloudi, N. and Li, J. The ai-based cyber threat landscape: A survey. *ACM Comput. Surv.*, 53(1), feb 2020. ISSN 0360-0300. doi: 10.1145/3372823. URL https://doi.org/10.1145/3372823.

Kostrikov, I., Yarats, D., and Fergus, R. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels, 2020. URL https://arxiv.org/abs/2004.13649.

Kott, A., Swami, A., and West, B. J. The fog of war in cyberspace. *Computer*, 49(11):84–87, 2016. doi: 10.1109/MC.2016.333.

Li, L., Fayad, R., and Taylor, A. Cygil: A cyber gym for training autonomous agents over emulated network systems. *arXiv preprint arXiv:2109.03331*, 2021.

Mern, J., Hatch, K., Silva, R., Brush, J., and Kochenderfer, M. J. Reinforcement learning for industrial control network cyber security orchestration. *arXiv preprint arXiv:2106.05332*, 2021.

Microsoft Defender Research Team. Cyberbattlesim. https://github.com/microsoft/cyberbattlesim, 2021. Created by Christian Seifert, Michael Betser, William Blum, James Bono, Kate Farris, Emily Goren, Justin Grana, Kristian Holsheimer, Brandon Marken, Joshua Neil, Nicole Nichols, Jugal Parikh, Haoran Wei.

Molina-Markham, A., Winder, R. K., and Ridley, A. Network defense is not a game. *arXiv preprint arXiv:2104.10262*, 2021.

Paul, E. and Alfréd, R. On random graphs i. *Publicationes Mathematicae (Debrecen)*, 6:290–297, 1959.

Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., and Dormann, N. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. URL http://jmlr.org/papers/v22/20-1364.html.

Ridley, A. Machine learning for autonomous cyber defense. *The Next Wave*, 22(1):7–14, 2018.

Rozemberczki, B. and Sarkar, R. Characteristic Functions on Graphs: Birds of a Feather, from Statistical Descriptors to Parametric Models. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM '20)*, pp. 1325–1334. ACM, 2020.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms, 2017. URL https://arxiv.org/abs/1707.06347.

Standen, M., Lucas, M., Bowman, D., Richer, T. J., Kim, J., and Marriott, D. Cyborg: A gym for the development of autonomous cyber agents. *arXiv preprint arXiv:2108.09118*, 2021.

Sutton, R. S. and Barto, A. G. *Reinforcement learning: An introduction*. MIT press, 2018.

Tran, K., Akella, A., Standen, M., Kim, J., Bowman, D., Richer, T., and Lin, C.-T. Deep hierarchical reinforcement agents for automated penetration testing. *arXiv preprint arXiv:2109.06449*, 2021.

Zhao, W., Queralta, J. P., and Westerlund, T. Sim-to-real transfer in deep reinforcement learning for robotics: a survey. In *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 737–744. IEEE, 2020.