2015 IEEE 14th International Conference on Machine Learning and Applications

Optimizing Attack Surface and Configuration Diversity Using Multi-Objective Reinforcement Learning

Bentz Tozer, Thomas Mazzuchi, Shahram Sarkani
The George Washington University
Department of Engineering Management and Systems Engineering
1776 G Street NW Suite 101, Washington, DC 20006
Email: {bentz, mazzu, sarkani}@gwu.edu

***Abstract*—Minimizing the attack surface of a system and introducing diversity into a system are two effective ways to improve system security. However, determining how to include diversity in a system without increasing the attack surface more than necessary is a difficult problem, requiring knowledge about the system characteristics, operating environment, and available permutations that is generally not available prior to system deployment. We propose viewing a system's components, interfaces, and communication channels as a set of states and actions that can be analyzed using a sequential decision making process, and using a multi-objective reinforcement learning algorithm to learn a set of policies that minimize a system's attack surface and execute those policies to obtain configuration diversity while a system is operating. We describe a methodology for designing a system such that its components and behaviors can be translated into a multi-objective Markov Decision Process, demonstrate the use of multi-objective reinforcement learning to learn a set of optimal policies using three different multi-objective reinforcement learning algorithms in the context of an online file sharing application, and show that our multi-objective temporal difference afterstate algorithm outperforms the alternatives for the example problem.**

I. Introduction

Traditional approaches to system security have focused on reducing errors in design and implementation that result in system vulnerabilities, along with selecting a configuration that exposes as little of the system as possible to potential attackers when it is deployed. The result of this effort should be a system that has a minimized attack surface, but despite these efforts, almost every system is still delivered with vulnerabilities that can be exploited by attackers. Also, the single, stationary configuration created by the process of minimizing the system's exposure to attack also reduces the amount of effort required by an attacker to assess potential vulnerabilities, Because of these issues, additional measures are needed to protect systems from attackers. One promising alternative is moving target defense [1], where a system's configuration is dynamically modified to introduce additional diversity, making an attacker's job more difficult. However, determining which changes to apply is a difficult problem due to the number of choices available and a dependency on knowledge about the operating environment, attacker behavior and capabilities, and runtime behavior of the system. One solution to this problem is the use of multi-policy multi-objective reinforcement learning to determine the set of system configurations that minimize its attack surface with respect to different measurements of security through information gathered as part of the system's normal interaction with its environment, while also maintaining configuration diversity by selecting from these optimal configurations at random as part of the decision making process.

In this paper, we discuss the relationships between attack surface measurement and moving target defense, describe the use of a multi-objective reinforcement learning algorithm to learn multiple policies that minimize a system's attack surface in different dimensions and control a system's behavior by selecting actions based on those policies, evaluate the performance of our methodology using three different multi-objective reinforcement learning algorithms with an example an online file sharing application, and discuss potential areas for future work. To the best of our knowledge, this is the first work to use multi-objective reinforcement learning as a method to create system diversity for moving target defense, and our multi-objective temporal difference afterstate algorithm outperforms the alternatives with respect to the rate of convergence on the set of optimal policies and total number of optimal policies learned, and total expected reward for the problem domain.

II. Background and Related Work

A. Multi-objective reinforcement learning

Reinforcement learning is a framework that allows an agent to learn an optimal set of actions to take based on feedback from the surrounding environment [2]. In practice, the states, actions, and rewards associated with the agent's movements in the environment are often modeled as a Markov Decision Process, which can be described as a tuple (S, A, P, R) where S is the set of states in the system, A is the set of actions that the agent can take, P is the probability $P(s'|s, a)$ that an action a from a state s results in a state s' and R is the value received from the reward function $R(s, a)$ for an action a from a state s. The agent learns by maximizing the expected reward based on rewards received as part of each state transition, determined by the action selected in a certain state and the stochastic behavior of the environment, a learning rate α that controls how fast the agent learns from the current reward, and a discount rate γ that controls the impact of future rewards on the learning process.

978-1-5090-0287-0/15 $31.00 © 2015 IEEE
DOI 10.1109/ICMLA.2015.144

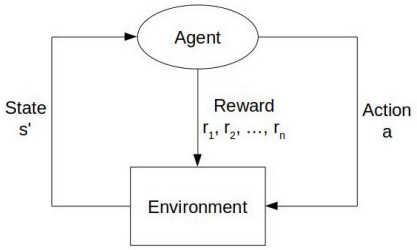

Fig. 1. Multi-objective Reinforcement Learning Environment

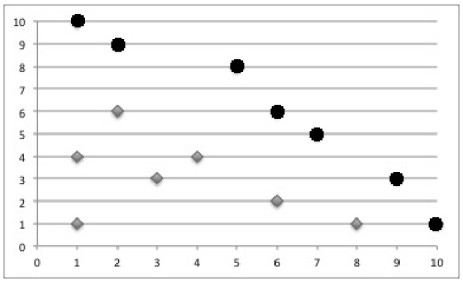

Fig. 2. Two Dimensional Pareto Frontier. Components of the Pareto frontier are designated as black circles.

Multi-objective reinforcement learning is the extension of the reinforcement learning paradigm to problems where there are multiple, usually conflicting, objectives, and a separate reward signal associated with each objective is received from the environment, as shown in Figure 1. Traditionally, implementations of multi-objective reinforcement learning algorithms have focused on the discovery of a single optimal policy for all objectives, usually through a scalarization algorithm that assigns weights to the reward from each objective. This effectively turns the problem into a single objective reinforcement learning process, so that a single policy is learned based on the scalarized reward values, and the scalarization algorithms can be linear [3] or non-linear [4] [5]. In cases where a user's preferences are clearly known, the solution space is well understood, or all the objectives are positively correlated, this method can be effective, but in many cases, these prerequisites are not met. The shortcomings of using weighted sums to scalarize reward vectors in multi-objective environments were first identified by White [6] and more recently addressed by Vamplew et al. [7].

In contrast to single policy methods, multiple policy methods learn a set of optimal policies, usually through the use of a multi-objective Markov Decision Process [8] where the reward received from a transition from one state to another is a vector rather than a scalar value. Calculating the optimal policies based on each component of the reward allows the algorithm to discover policies that may not have been previously known, and better supports scenarios where objectives are in conflict, which is frequently the case in real world applications. This is accomplished by utilizing the concept of Pareto dominance [9] to determine a set of Pareto optimal vectors, representing an action selection policy, that lie on the Pareto frontier. A vector v dominates vector v' if $v_i \geq v'_i$ for all i and $v_i > v'_i$ for at least one i, and the Pareto frontier is made up of vectors that are not dominated by any other vector in the problem space. An example Pareto frontier for a multi-objective problem with two dimensions can be seen in Figure 2.

There are several works that utilize the concept of the Pareto frontier to learn optimal policies. Barrett and Narayanan [10] use a value iteration algorithm to determine the convex hull of the Pareto front, while Lizotte et al. [11] use adaptive weights to learn the convex hull using an algorithm based on Q-learning, and Parisi et al. [12] use forms of policy iteration based on Pareto dominance and gradience ascent searches. Wang and Sebag [13] use the hypervolume quality indicator and Pareto dominance to extend Monte Carlo Tree Search to support multiple objectives, and Van Moffaert et al. [14] evaluate multi-objective reinforcement learning algorithms based on the hypervolume quality indicator, linear scalarizarion, and Chebyshev scalarization, then introduce Pareto Q-learning, which is a multi-policy algorithm that uses Pareto dominance for action selection and a variant of Q-learning where current and future rewards are learned separately. As we believe Pareto Q-learning is the only temporal difference based reinforcement learning algorithm in the literature capable of learning the complete set of Pareto dominant policies, rather than a convex hull or approximation based on a scalarization function, we used this algorithm as the baseline for our moving target defense analysis.

B. Attack Surface Measurement

A system's attack surface is the set of all possible options that can attacker could possibly use to exploit a system. Measuring a system's attack surface is a complex task since there are many potential ways to evaluate it, but the key metrics that should be considered when evaluating system security are identified, which are the number of potential attack vectors that exist, the minimum cost of a successful attack, the minimal number of steps required to execute an attack, and the probability of a successful attack [15]. One methodology that incorporates these metrics measures attack surface by determining the number of communication channels, function methods, and data items that are exposed to users as system entry points or exit points, as well as calculating a damage potential-effort ratio which measures the amount of harm that can be done with a specific resource compared to the amount of effort required by an attacker to exploit that resource [16].

C. Moving Target Defense

Traditional systems are deterministic, allowing attackers to methodically probe a system and develop a successful attack by finding the weaknesses that exist in every system over time. Also, because of the inherently asymmetrical balance between a defender that has to be prepared for an attack anywhere within the system and an attacker that only has to find a single vulnerability to exploit, defenders in this environment are at a major disadvantage. The goal of moving target defense is to turn what has traditionally been a stationary environment into a dynamic one, making a system more difficult to attack by modifying its characteristics. This increases the amount of

effort required by an attacker to evaluate a system for weaknesses, decreases an attacker's ability to obtain persistence, and reduces the chances that a system will be successfully attacked. Many approaches to moving target defense have been proposed in the literature [17], covering the introduction of changes to position, configuration, and behavior at all levels of granularity. The works that are most closely related to the methods in this paper utilize game theory or control theory to minimize attack surface and introduce diversity at the system level. Zhu and Başar [18] view the interactions between an attacker and defender as a zero sum game and uses feedback from the environment to implement a moving target defense strategy based on that information and the defender's goal to minimize its attack surface, and it relies on the concept of Nash equilibrium to determine optimal behavior, which may not result in Pareto optimal solutions. Manadhata [19] also treats the attacker and defender as players in a two person game, but in this case the defender is attempting to maximize its reward by finding adaptations that minimize its attack surface, and they also use Nash equilibrium rather than Pareto dominance to select optimal configurations. In both these papers, it is assumed that the defender has perfect information about the attack, which is an assumption we do not make. Rowe et al. [20] use control theory to determine adaptations that modify the attack surface based on an estimated state of the system and cost of a modification, but does not address methods for determining an optimal policy. Rasouli et al. [21] use dynamic programming to determine defender behavior based on a min-max performance criterion, rather than Pareto optimality, and all values associated with each state are represented by a scalar value, rather than a vector.

III. Methodology

To successfully deploy a dynamic system that supports configuration diversity, it must be adaptable at runtime while continuing to provide all required functionality. We accomplish this using a microservices architecture [22] where the system is built using independent components that provide one piece of functionality and communicate using standardized interfaces. This architecture allows us to create many functionally equivalent software components that have a different attack surface by modifying configuration parameters within individual components or replacing one component with another, while still allowing communication between components and enabling system modifications without degrading the system as a whole. Each of these heterogenous implementations have its own attack surface measurement providing tradeoffs between the number of communication channels, function methods, and untrusted data items, and the amount of effort required by an attacker to compromise a specific implementation. An example of this architecture can be seen in Figure 3, where each piece of required functionality in the system has been separated into an standalone service, and all services are capable of communicating with all other services as needed to provide desired functionality. Providing multiple instances of a service can be accomplished in numerous ways, such as intentionally developing multiple service implementations on different platforms, adding instances over time as existing implementations are upgraded, or by adjusting configuration parameters as part of the deployment process or at runtime.

We then treat each component in our system as a state

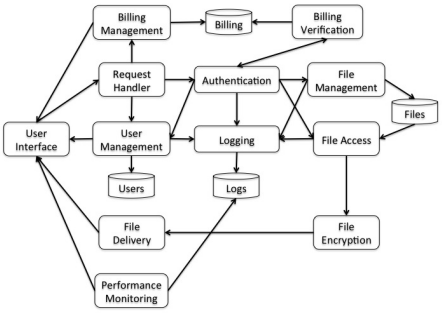

Fig. 3. Online File Sharing Microservices Architecture

in a multi-objective Markov Decision Process, the routing decisions made within the system as actions, the error rate within a state as the transition probability, the attack surface measurement for the selected component as the reward vector, and the path from the initial client request to a response as a learning episode. Since our goal is minimizing the attack surface, and our reinforcement learning algorithms are designed to maximize the expected reward, all attack surface values were converted to negative numbers. All state and reward information is forwarded to a central controller that is aware of all existing components and is running a multi-policy, multi-objective reinforcement learning algorithm which selects actions with the goal of learning which sets of actions result in a Pareto optimal sequence that minimizes the attack surfaces of the system components, and selecting those sets with increasing frequency over time, providing configuration diversity by selecting from the set of optimal actions at random. This is an improvement over randomly or arbitrarily modifying a system because components that provides less benefit from a security perspective than other functionally equivalent components will be selected less frequently over time, improving the security of the system while still providing sufficient diversity.

We evaluate the performance of three multi-objective reinforcement learning algorithms in this environment. All three algorithms rely on the use of Pareto dominance to evaluate potential actions available from a given state. One algorithm is an implementation of the Pareto Q-learning algorithm introduced in [14]. Q-learning is a model-free reinforcement learning algorithm that learns the optimal policy based on the action-value function Q(s,a). It is considered an off-policy learning algorithm because it learns based on the optimal action that could be taken from the current state, not the actual action taken.

To analyze the difference between on-policy and off-policy learning methods for our example scenario, we implemented a multi-objective version of SARSA [23], which is identical to Q-learning, other than learning future rewards based on the actions taken as part of the learning process, rather than learning based on the optimal action that could be taken from that state. We used the same action selection methodology as Pareto-Q-learning, based on Pareto dominance of the current estimates of the value of a state-action pair for all available actions in a given state.

As an alternative to the value-action based learning methods described above, we developed a multi-objective version of the temporal difference algorithm TD(0) [2] based on afterstates. Learning methods based on afterstates [2] separate the transition from the current state to the next state into two different steps, which is accomplished by evaluating the agent's position and the surrounding environment after an action has been taken by the agent, but before the impact of the environment on the state transition occurs, making it effective when the initial portion of a system's response to an action is known, but not the entire response. The afterstate method also reduces the complexity of the learning process by taking advantage of the fact that many state-action pairs lead to the same post-action state, allowing for more efficient learning and fewer values to maintain as part of the learning process. The implementation of our algorithm, which selects actions based on Pareto dominance of current estimates of the state's value function for each objective, similar to the Q-learning and SARSA algorithms described previously, can be seen below.

Algorithm 1 Multi-objective TD(0) Afterstate algorithm

Initialize $V(s)$ based on the number of objectives
for each episode i **do**
    Set s to initial state
    while s is not terminal **do**
        Choose action a based on Pareto dominance of current $V(s)$ values
        Take action a, receive reward vector r^T, afterstate s'_a, and next state s'
        $V(s_a) = V(s_a) + \alpha \left[r^T + \gamma V(s'_a) - V(s_a)\right]$
        Update $s_a = s'_a, s = s'$
    end while
end for

IV. Experimental Results

Using the microservices architecture described in the previous section, we modeled the online file sharing service shown in Figure 3 assuming the use of a lightweight Java HTTPS client and server, multiple interfaces to data stores, different configuration settings for client authentication, and multiple data access methods. All of these components have a well defined REST API, making the internal configuration of each component implementation independent of the interface, and allowing the controller to select any of the configurations of a component for any file request. It is assumed that a user can get a file, add a file, delete a file, or change permissions on a file, and each account has a distinct profile, which can also be modified by the account holder. Each component implementation was given a set of attack surface values based on the configured number of functions, communication channels, and untrusted files, the difficulty to attack relative to the damage to the system that could be done in the event that component was compromised, the requested action, and the user's profile.

For example, the file request process consists of a controller that receives the initial file request via HTTPS from a client application, components that support user authentication, billing, file access, file encryption, file delivery, and logging, and separate data stores that contain user information, billing information, and the files that will be delivered. Also, at any point in the sequence, a component can deny access to the file, either intentionally or due to the misconfiguration of a component, by entering an error state. This process can be seen in Figure 4.

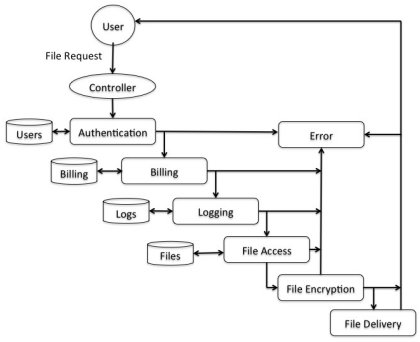

Fig. 4. File Request Sequence

The attack surface measurements for the all request types are dependent on the source of the request. For example, the compromise of an account with a large number of publicly available files stored within this service is likely to cause more damage than the compromise of an account with a few private files. Sample attack surface measurements based on the evaluation of a component's implementation and the user interacting with the system can be seen in Table 1.

TABLE I. Example Component Attack Surface Measurements

Component Type	Component Configuration	Requested Action	User Description	Attack Surface Measurements
File Access	Read-only file access	Get File	Trusted User	(4,10, 7)
File Access	Read-only file access	Get File	Anonymous User	(20,50, 35)
File Access	Read-write file access	Get File	Anonymous User	(200,500, 350)
Billing Access	SQL Injection Framework 1	Add File	Trusted User	(2,8,4)
Billing Access	SQL Injection Framework 2	Add File	Trusted User	(2,12,4)
Billing Access	SQL Injection Framework 3	Add File	Trusted User	(2,6,6)
Authentication	Key Based Authentication	Change File Permissions	Unknown User	(1,6,9)
Authentication	Password Based Authentication	Change File Permissions	Unknown User	(5,20,5)

Because of the uncertainty associated with user behavior and account configuration, the attack surface values are dynamic and not fully known in advance, requiring the use of learning algorithms to determine the optimal policies. A multi-objective Markov Decision Process representing the example file request scenario that we developed as a proof of concept for our multi-objective reinforcement learning algorithms can be seen in Figure 5.

This configuration consisted of 52,920 possible configurations, of which 216 were optimal policies based on the attack

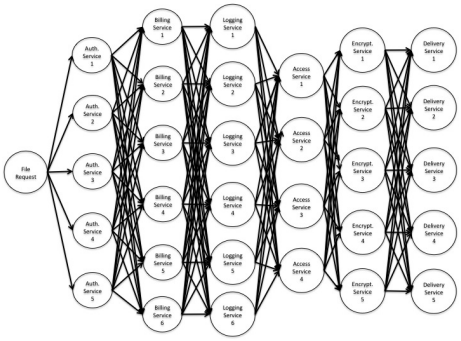

Fig. 5. File Request Markov Decision Process

surface values applied to each component. A learning rate of 0.9, discount factor of 0.5, and random exploration value of 0.1 for ϵ-greedy action selection were selected for each reinforcement learning algorithm through experimentation, and the error rate due to communication issues, platform stability issues, and coding or configuration errors was assumed to be 0.01. 30 runs of 2,500 episodes were executed for each algorithm, and the results of all runs were averaged together to account for anomalies in any single run. All results were compared using a two sample Wilcoxon rank-sum test [24] to evaluate the significance of the differences between the results from each algorithm, and in all cases, the p-value calculated by the test was less than 0.0001, demonstrating that the differences in performance were statistically significant for all measurements.

In Figure 6, the average number of optimal policies found by each algorithm per episode is shown. The Pareto TD(0) afterstate algorithm and Pareto-Q-learning algorithm both learned all 216 optimal policies within the 2,500 episodes on most of the 30 runs, resulting in an average number of optimal policies learned of approximately 216, but the Pareto TD(0) afterstate algorithm learned much more quickly, finding over 100 of the optimal policies by the 250th episode and over 200 of the optimal policies by the 1000th episode on average, while the Pareto-Q-learning algorithm took approximately 400 and 1400 episodes on average. The Pareto-SARSA algorithm learned significantly fewer solutions on average, and never learned all 216 optimal policies.

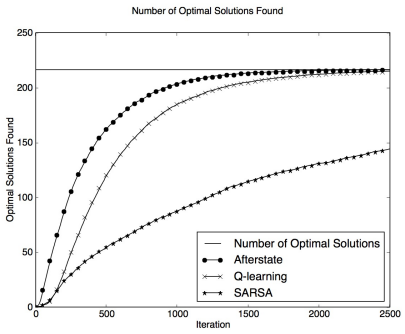

Fig. 6. Total Number of Optimal Policies Found

Figure 7 shows the average number of optimal actions selected at each episode for each algorithm type, and as with the average number of optimal policies learned, the Pareto TD(0) afterstate algorithm and Pareto Q-learning algorithm outperformed the Pareto SARSA algorithm, and the Pareto TD(0) afterstate algorithm learned significantly faster than the others. Because the action selection algorithm was designed to return a random exploratory action 10% of the time, selecting the optimal action approximately 90% of the time is the maximum threshold for correct action selection, and the Pareto TD(0) afterstate algorithm reached that benchmark prior to the 250th episode on average, while the Pareto Q-learning algorithm had not reached that threshold by the 2,500th episode but was still improving, and the Pareto SARSA algorithm only learned to select an optimal action about 73% of the time, and did not improve after 1,500 episodes.

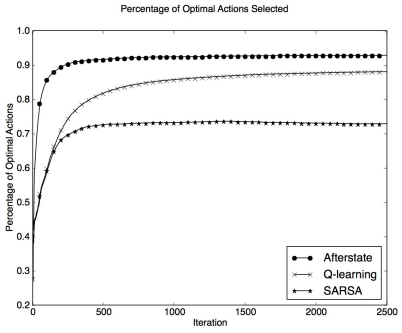

Fig. 7. Percentage of Optimal Actions Selected

Finally, we show the average cumulative reward at a specific episode for each algorithm in Figure 8, which was calculated by summing the reward received at each episode from each of the 3 objectives. Once again, the Pareto TD(0) afterstate algorithm and Pareto Q-learning algorithm outperformed the Pareto SARSA algorithm by a large margin, and the Pareto TD(0) afterstate algorithm was better than the Pareto Q-learning algorithm.

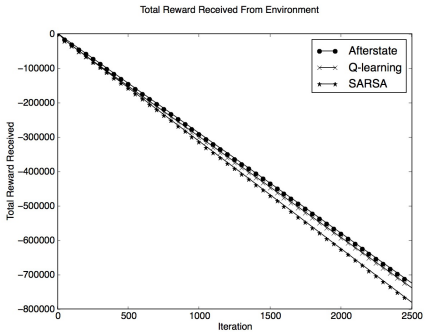

Fig. 8. Average Total Cumulative Reward per Episode

V. Conclusion and Future Work

In this paper, we presented the use of a multi-objective reinforcement learning algorithm to determine the Pareto optimal set of system configurations that minimize the system's

attack surface, and then execute policies selected at random from that set to create system diversity that deters attackers. We then evaluated this methodology using three different multi-objective reinforcement learning algorithms to control a simulated online file sharing application, demonstrating that the proposed framework performs as desired, and that our multi-objective TD(0) afterstate algorithm outperforms the other algorithms in the example environment. In the future, we plan to expand the size and scope of applications used to test the methodology and further evaluate its performance in non-stationary environments.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for their suggestions, which improved the paper.

REFERENCES

[1] S. Jajodia, A. K. Ghosh, V. Swarup, C. Wang, and X. S. Wang, *Moving target defense: creating asymmetric uncertainty for cyber threats*. Springer Science & Business Media, 2011, vol. 54.

[2] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press Cambridge, 1998, vol. 1, no. 1.

[3] S. Natarajan and P. Tadepalli, "Dynamic preferences in multi-criteria reinforcement learning," in *Proceedings of the 22nd international conference on Machine learning*. ACM, 2005, pp. 601–608.

[4] Z. Gábor, Z. Kalmár, and C. Szepesvári, "Multi-criteria reinforcement learning." in *ICML*, vol. 98, 1998, pp. 197–205.

[5] S. Mannor and N. Shimkin, "A geometric approach to multi-criterion reinforcement learning," *The Journal of Machine Learning Research*, vol. 5, pp. 325–360, 2004.

[6] D. J. White, "The set of efficient solutions for multiple objective shortest path problems," *Computers & Operations Research*, vol. 9, no. 2, pp. 101–107, 1982.

[7] P. Vamplew, J. Yearwood, R. Dazeley, and A. Berry, "On the limitations of scalarisation for multi-objective reinforcement learning of pareto fronts," in *AI 2008: Advances in Artificial Intelligence*. Springer, 2008, pp. 372–378.

[8] K. Chatterjee, R. Majumdar, and T. A. Henzinger, "Markov decision processes with multiple objectives," in *STACS 2006*. Springer, 2006, pp. 325–336.

[9] M. Voorneveld, "Characterization of pareto dominance," *Operations Research Letters*, vol. 31, no. 1, pp. 7–11, 2003.

[10] L. Barrett and S. Narayanan, "Learning all optimal policies with multiple criteria," in *Proceedings of the 25th international conference on Machine learning*. ACM, 2008, pp. 41–47.

[11] D. J. Lizotte, M. Bowling, and S. A. Murphy, "Linear fitted-q iteration with multiple reward functions," *The Journal of Machine Learning Research*, vol. 13, no. 1, pp. 3253–3295, 2012.

[12] S. Parisi, M. Pirotta, N. Smacchia, L. Bascetta, and M. Restelli, "Policy gradient approaches for multi-objective sequential decision making," in *Neural Networks (IJCNN), 2014 International Joint Conference on*. IEEE, 2014, pp. 2323–2330.

[13] W. Wang and M. Sebag, "Hypervolume indicator and dominance reward based multi-objective monte-carlo tree search," *Machine learning*, vol. 92, no. 2-3, pp. 403–429, 2013.

[14] K. Van Moffaert and A. Nowé, "Multi-objective reinforcement learning using sets of pareto dominating policies," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 3483–3512, 2014.

[15] L. Krautsevich, F. Martinelli, and A. Yautsiukhin, "Formal analysis of security metrics with defensive actions," in *Ubiquitous Intelligence and Computing, 2013 IEEE 10th International Conference on and 10th International Conference on Autonomic and Trusted Computing (UIC/ATC)*. IEEE, 2013, pp. 458–465.

[16] P. K. Manadhata and J. M. Wing, "An attack surface metric," *Software Engineering, IEEE Transactions on*, vol. 37, no. 3, pp. 371–386, 2011.

[17] K. A. Farris and G. Cybenko, "Quantification of moving target cyber defenses," in *SPIE Defense+ Security*. International Society for Optics and Photonics, 2015, pp. 94 560L–94 560L.

[18] Q. Zhu and T. Başar, "Game-theoretic approach to feedback-driven multi-stage moving target defense," in *Decision and Game Theory for Security*. Springer, 2013, pp. 246–263.

[19] P. K. Manadhata, "Game theoretic approaches to attack surface shifting," in *Moving Target Defense II*. Springer, 2013, pp. 1–13.

[20] J. Rowe, K. N. Levitt, T. Demir, and R. Erbacher, "Artificial diversity as maneuvers in a control theoretic moving target defense," in *National Symposium on Moving Target Research*, 2012.

[21] M. Rasouli, E. Miehling, and D. Teneketzis, "A supervisory control approach to dynamic cyber-security," in *Decision and Game Theory for Security*. Springer, 2014, pp. 99–117.

[22] M. Fowler and J. Lewis, "Microservices," *ThoughtWorks. http://martinfowler. com/articles/microservices. html [last accessed on July 30, 2015]*, 2014.

[23] G. A. Rummery and M. Niranjan, "On-line q-learning using connectionist systems," 1994.

[24] H. B. Mann and D. R. Whitney, "On a test of whether one of two random variables is stochastically larger than the other," *The annals of mathematical statistics*, pp. 50–60, 1947.