Approximate State Space Abstraction for Autonomic Intrusion Response Systems

By

Andrew L Kerby

A Report
Submitted to the Faculty of
Mississippi State University
In Partial Fulfillment of the Requirements
for the Degree of Master of Science
in Computer Science
in the Department of Computer Science and Engineering

Mississippi State, Mississippi
October 2018

Approximate State Space Abstraction
for Autonomic Intrusion
Response Systems

By

Andrew L Kerby

Approved:

_____

Dr. Stefano Iannucci
(Major Professor)

_____

Dr. Maxwell Young
(Committee Member)

_____

Dr. Christopher Archibald
(Committee Member)

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Autonomic computing, inspired by the autonomic nervous system in humans, was proposed by Kephart & Chess in 2003 as a means by which computer systems could configure, optimize, heal, and protect themselves with minimal human intervention [9]. In a 2005 whitepaper by IBM, the Monitor, Analyze, Plan, and Execute (MAPE-K) control loop was proposed for an autonomic manager that would continuously watch a system for changes, analyze its alerts and events, plan the best course of action to take, and execute the planned action(s) in keeping with a high-level, adaptable policy [1]. Cyber-attacks on healthcare systems are extremely prevalent, with a survey by KPMG of executives in the healthcare industry revealing that over 80% have had their companies compromised [8]. The autonomic approach described in [1, 9] is an excellent fit for the protection of enterprise systems from cyber-attacks. In the cybersecurity space, an intrusion detection system (IDS) is a tool that monitors network activity and/or the operating systems of computing resources on the network, while an intrusion response system (IRS) receives alerts from an IDS, plans, and executes a response in defense of the system to bring it to a safe, stable state [13]. Various types of machine-learning approaches have been used to increase the accuracy and efficiency of intrusion detection systems, such as k-nearest neighbor, genetic algorithms, and artificial neural networks [10]. In an experiment by

Iannucci & Abdelwahed in 2016, Markov Decision Processes were used to model the defensive actions an IRS might take in response to attacks on a network. Using Value Iteration (VI) and a rollout-based Monte-Carlo search algorithm called UCT, the response-planning phase of the MAPE-K control loop was completed quickly enough to be practical in real-world applications [14].

Recent work by Iannucci & Abdelwahed in 2018 evaluated the performance of both single-threaded VI (via BURPLAP library [4]) and multi-threaded VI (via extension of BURLAP library [15]) on the planning time required to optimally solve an IRS system modeled using MDPs [15]. VI finds an optimal policy for scenarios modeled using MDPs, but a significant drawback of this approach is that as the system the IRS intends to defend grows increasingly large, the size of the MDP state-space grows exponentially [15]. A mathematical approach exists to create an abstraction of an MDP from the original MDP, which reduces the size of the state-space at the cost of trading an optimal policy for a near-optimal one [7]. The MDP state-space reduction techniques used in [7] has been used in wireless resource scheduling of software-defined radio access networks to significantly reduce the computational overhead of packet scheduling planning [17], therefore it is promising that the same technique could be used to increase the speed of IRS response-planning.

CHAPTER II

CONTRIBUTION OF PROJECT

The novel contributions in this directed project were three-fold. The first is the implementation of the Approximate State Abstraction algorithms described in [7] using BURLAP 3. Prior implementations used BURLAP 2, and only one approximate abstraction function, φ_{Q^*}, was implemented out of the four approximate state abstraction functions described in [7]. The second novel contribution is the optimization of the φ_{Q^*} abstraction function by utilizing alternate data structures from the original BURLAP 2 implementation, as well as caching of previously obtained results. This is explained further in the implementation section below. The final contribution of this paper was the application of approximate state abstraction to an IRS in a modeled healthcare environment. These contributions are described in more detail in the sections below.

CHAPTER III

MODEL SYSTEM

Markov Decision Processes are tasks in reinforcement learning defined by a set of states the agent can exist in and a set of actions that allow the agent to move between these states [12]. In addition, MDPs also contain a transition probability function that determines the probability that the agent will successfully carry out an action and move to another state. For each action, there is also a reward function that determines the amount of reward for each action. Lastly, the γ parameter in an MDP is the discount rate, which determines whether the agent has a preference for seeking to maximize rewards in the near term or, on the other hand, makes choices that maximizes long-term rewards [12].

The Value Iteration (VI) algorithm was utilized to solve the MDPs in this project. The Brown-UMBC Reinforcement Learning and Planning (BURLAP) Java library [4] was used to model the MDP and carry out VI on the IRS model. The set of states chosen in this project are discrete. The model system used in this work was patterned after a hypothetical healthcare system, as attacks on healthcare systems have become widespread in recent years [5], thus would stand to benefit greatly from the protection of an autonomic IRS. This model healthcare network consists of three open-source components: an Apache 2.2.29 web server [2], an Apache Tomcat 9.0.0.M1 application server [3], and a MySQL 5.6.41 database [7]. These specific application

versions were chosen because they are vulnerable to a number of security holes detailed in the Common Vulnerabilities and Exposures (CVE) list [5]. See Tables 1 and 2 for the specific CVE entries [5] associated with the web server and database components in the model healthcare network.

Table 1 details the states available in the MDP, as well as the possible values of each attribute. With the exception of the state attribute LOG_VERB, all of the attributes can either have a value of “true” or “false”. The value of LOG_VERB can be an integer between 0 and 5 (inclusive). It is assumed in the model system that there is an IDS that can notify the IRS of specific cyberattacks. The alerts are represented by attributes such as CACHE_POISONED. A value of “true” for the CACHE_POISONED attribute, set in the start state, would simulate the IDS alerting the IRS that its DNS cache is poisoned. The full list of actions that the agent can take to transition from state to state is shown in table 2. Many of these attributes were found in [15] and expanded upon in this project. Similarly, the terminal state for the MDP was chosen to emulate a “clean” state of the system, as was done in prior research regarding autonomic intrusion detection systems [15].

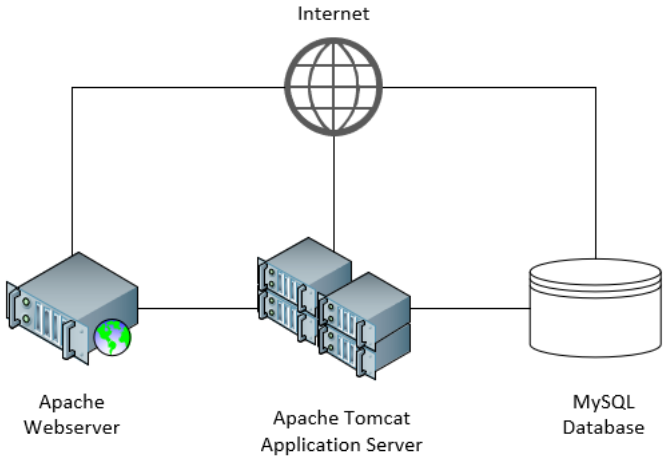

Figure 3.1 Architecture of Model Healthcare Network

Whether or not an action can be taken by the agent is dictated by a set of preconditions and postconditions for each action. In other words, the intrusion response actions are only available to be taken by the IRS if it is in a specific state. For example, for the IRS to be able to execute the "restore" action in order to cleanse itself of a ransomware infection, it must satisfy the Boolean precondition ALERTED && BACKED_UP &&! SYSTEM_RESTORED && LOG_VERB > 4. If it is in a state that satisfies the precondition, then it can transition to the state that is identical to its current state except the attributes SYSTEM_RANSOMED will be equal to "false" and SYSTEM_RESTORED will be equal to "true". In this way, the IRS can execute response actions to move it towards its "clean" goal state.

Table 3.1 Description of State Attributes in MDP [5]

State Attributes	Possible Values	Description
BACKED_UP	{true, false}	Indicates whether the system has been backed up by the IRS
ACTIVE	{true, false}	Indicates whether the system is actively running
SOFTWARE_UPDATED	{true, false}	Indicates whether any software has been updated on the system by the IRS
RESTARTED	{true, false}	Indicates whether the system has been restarted by the IRS
FIREWALL_ACTIVE	{true, false}	Indicates whether the firewall is currently active
LOG_VERB	{0,1,2,3,4,5}	Indicates the current level of verbosity for the logging module
ALERTED	{true, false}	Indicates whether a notification has been sent to a human system administrator
SYSTEM_RESTORED	{true, false}	Indicates if the system has been restored from backup by the IRS
WEBSERVER_ADDED	{true, false}	Indicates if a new instance of the webserver has been instantiated in the cloud
DATABASE_RESTORED	{true, false}	Indicates whether the database has been restored from an archived version
HONEYPOT_ACTIVE	{true, false}	Indicates if the IRS is actively directing attackers to a honeypot system
IP_PREVIOUSLY_BLOCKED	{true, false}	Indicates if the IP of the attacker has been previously blocked by the network
SQL_MAPPED	{true, false}	Indicates whether a SQL map is being run by the attacker
PORT_SCANNED	{true, false}	Indicates if the attacker has executed a port scan on the network
SLOW_LORRIS_ATTACKED	{true, false}	Indicates if system is currently experiencing a slow lorris attack
AUTH_BYPASSED	{true, false}	Indicates whether the IDS has detected an authentication bypass for Apache (CVE-2017-3167) [5]
HTTP2_VULNERABILITY	{true, false}	Indicates whether the IDS has detected an exploit of an HTTP2 vulnerability
SQL_INJECTED	{true, false}	Indicates whether the IDS has detected a SQL injection attack
REPEATABLE_CRASH_ATTACKED	{true, false}	Indicates whether the IDS has detected a repeatable crash attack (CVE-2018-2600) [5]
MY_SQL_DOS_ATTACKED	{true, false}	Indicates whether the IDS has detected a denial of service exploit for MySQL (CVE-2018-2703) [5]
CACHE_POISONED	{true, false}	Indicates whether the IDS has detected that its DNS cache was poisoned
PRIVILEGE_ESCALATED	{true, false}	Indicates whether the IDS has detected a privilege escalation exploit targeting Tomcat (CVE-2017-6712) [5]
OPTIONS_BLEED	{true, false}	Indicates whether the IDS has detected an options bleed exploit (CVE-2017-9798) [5]
SYSTEM_RANSOMED	{true, false}	Indicates whether the IDS has detected a ransomware attack

For each action chosen by the agent that is not the goal state, it receives a penalty of -10. When the agent reaches a state that satisfies the terminal condition where LOG_VERB = 0 and the system is not under attack and its cloud web server and honeypot are decommissioned, it will receive a reward of 1000. The transition function returns a transition probability of 1 for every action in the MDP, so at every time steps when the IRS chooses an action to execute, the IRS agent will transition to the next state with absolute certainty.

Table 3.2 Description of Actions in MDP [5]

Actions	Description
Backup	Create a backup of the system
Restart	Restart the system
ActivateFirewall	Activate firewall to mitigate a cyberattack
LogVerbIncrease	Increase the verbosity of the logging module
LogVerbDecrease	Decrease the verbosity level of the logging module
Alert	Send an alert to a system administrator
BlockIp	Block a malicious IP address
Restore	Restore the affected computer from backup
RebuildDns	Restore the DNS cache from a clean state
UnblockIp	Unblock an IP address that was previously blocked
SendToHoneypot	Redirect user to honeypot system
RemoveFromHoneypot	Remove user from honeypot system
UpdateAuthBypass	Patch authentication bypass vunerability in Apache; CVE-2017-3167
UpdateHttp2	Patch HTTP/2 vulnerability in Apache Tomcat; CVE-2017-7675
UpdateMySqlDos	Patch a low privilege DOS vulnerability in MySQL; CVE-2018-2703
UpdateOptionsBleed	Patch unauthenticated OPTIONS HTTP request for Appache; CVE-2017-9798
UpdateRptCrash	Patch repeatable crash vulnerability for MySQL; CVE-2018-2600
UpdatePrivEscalation	Patch Cisco ESC controller vulnerability in Tomcat; CVE-2017-6712
RestoreDatabase	Restore the database from most recent backup
AddWebServer	Deploy additional web server using cloud provider
DecommissionWebServer	Shut down and remove web server from network

An example of the response plan executed will now be provided. In hypothetical scenario in which the intrusion response system detects that its DNS cache has been poisoned, the model

IRS will start with a state of CACHE_POISONED equal to “true” and all other Boolean attributes of the state set to “false”. The response planned by the IRS using the MDP modeled in BURLAP 3 for this project is LogVerbIncrease -> ActivateFirewall -> Alert -> LogVerbIncrease -> RebuildDNS -> LogVerbDecrease -> LogVerbDecrease. This demonstrates that the model MDP has the ability to plan a logical sequence of IRS response actions from an “under attack” start state to a safe terminal state.

CHAPTER IV

STATE ABSTRACTION FUNCTION IMPLEMENTATION

The original, *ground* MDPs can be abstracted, reducing the size of the state space and creating an *abstract* state space [7, 18]. There were four approximate state abstraction functions implemented as part of this project, three of which were not previously implemented in the original BURLAP 2 project found on GitHub [6]. This is due to the reducibility of any of the other approximate abstraction functions into ϕ_{Q^*} [7]. Each state abstraction function aggregates states from the ground MDP into clusters of states. Every state in the abstract clusters are deemed approximately equivalent based on the ϕ function, so the many ground states in the cluster can be mapped to one abstract state. Therefore, the size of the state space can be reduced to the number of clusters resulting from the ϕ function, while still preserving near-optimality [7]. This approximate state grouping based on similarity allows for the state space of the ground MDP to be compressed more than when using exact state equivalency, as few pairs of states in the state space of an MDP could be considered exactly the same, but many could be considered similar [7].

The four approximate state abstraction functions are implemented as part of this project are ϕ_{Q^*}, ϕ_{bolt}, ϕ_{model}, and ϕ_{mult}. Each of the approximate state abstraction functions, represented by ϕ, take a pair of states that are candidates for being clustered into a single abstract state, as well

as a mutual action that these states could execute to get to the next state, and perform some computation to determine if these states should be collapsed. There is a threshold parameter ε that determines the level of abstraction between the ground MDP and the abstract MDP. For example, if the value of $\varepsilon = 0$, then then the two states must be exactly equivalent with each other in order to be collapsed into the same state. A higher value of ε will provide a higher level of abstraction up to certain bound [7].

ϕ_{Q^*} first calculates the optimal actions for each action in the pair. It then compares the optimal Q-values for both of the states and any shared optimal actions, and it they are within the threshold set by ε, then they are collapsed into the same state. The implementation of ϕ_{Q^*} for BURLAP 3 provides two performance improvements over the BURLAP 2 implementation. The first improvement is that the list of calculated Q-values used in ϕ_{Q^*} are stored in a tree, such that any values outside of the bounds of ε from the root node of the tree do not need to be traversed to. The second performance improvement, which provided a substantial speedup over the original implementation, was the caching of the list of optimal actions after they were computed. Additionally, in order to allow the abstract state space to be reorganized into a coherent state space, bound by the same preconditions and postconditions of the ground MDP, a minor extension was required for the BURLAP 3 implementation of VI to allow the abstract state space to be inserted as a replacement value function at runtime, so that VI could be run on the abstract state space.

The other three approximate state abstraction functions were implemented as specified in the paper by [7]. As previously mentioned, these were not implemented in the BURLAP 2 implementation. ϕ_{model} does not rely on optimal Q-values, but rather rely on the computation of the differences between the reward values and transition probabilities for each of the state pairs

that share the same action, therefore value iteration does not need to be run in order to obtain optimal Q-values for computing whether the two states can be collapsed together. ϕ_{bolt} and ϕ_{mult} are interesting from an abstraction perspective because they compare optimal Q-values over Boltzman and multinomial distributions, rather than directly comparing the optimal Q-values for shared optimal actions. In the following section, the results of the runtime performance and abstraction ability of each of the four ϕ functions is compared with VI.

CHAPTER V

RESULTS

The testbed for running the performance experiments was a system with two 16-core Intel Skylake Xeon 6142 CPUs running at 2.6 GHz and 384GB of RAM. Although a large amount of RAM was present on the system, only 64 GB of RAM was allocated to the Java runtime environment. Furthermore, the VI implementation in BURLAP 3 was single-threaded in nature and therefore none of the algorithms benefitted from the multi-core environment of the test system. The δ value chosen for VI was 0.1, and the γ value was set to 0.99 to have the agent prefer long-term rewards over short-term rewards. In addition, the same ε value of 30.0 was used for all of the approximate state abstraction function experiments. This value of ε was chosen after examining the Q-values output by VI. The VI algorithm was executed from various start states that produced state spaces of increasing size, in order to get a baseline for comparing its runtime and reward numbers against the abstract MDPs generated by the four approximate state abstraction functions. Following this, the four ϕ functions were each used to generate an abstract MDP, and then planning was executed from the same start state for 10000 trials to gather data on the average reward collected.

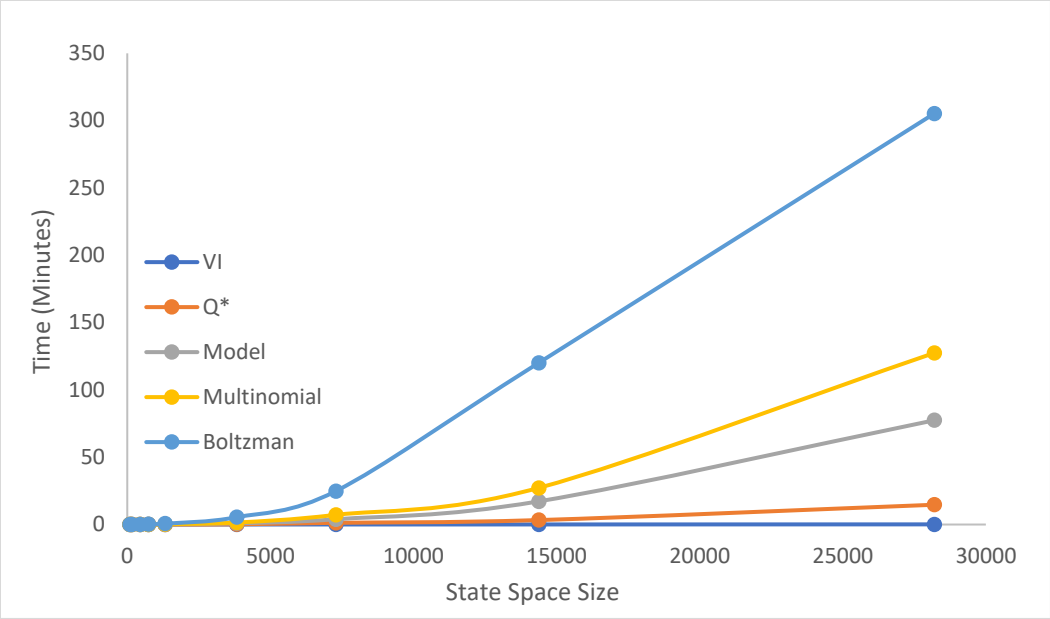

Figure 5.1 Runtime of Approximate State Abstraction Functions Compared to VI

As shown in Figure 5.1, the runtime for abstracting state spaces increased very quickly, especially beyond ground MDPs of larger than 7,500 states, making it unlikely that the approximate state space functions will be able used for speeding up IRS response planning, as was the original intent of the project. While the state-space abstraction methods provided a means of collapsing substantially large MDPs with a reasonable distance from the ground MDP in terms of reward gap, the experimental runtime increased far too quickly to be of use in real-world scenarios where speed is concerned. However, as shown in Table 5.1, the ϕ functions performed very well at abstracting relatively large state spaces down to even single digit sizes, in some cases, while still staying nearly optimal in terms of the reward, as shown in Figure 5.2.

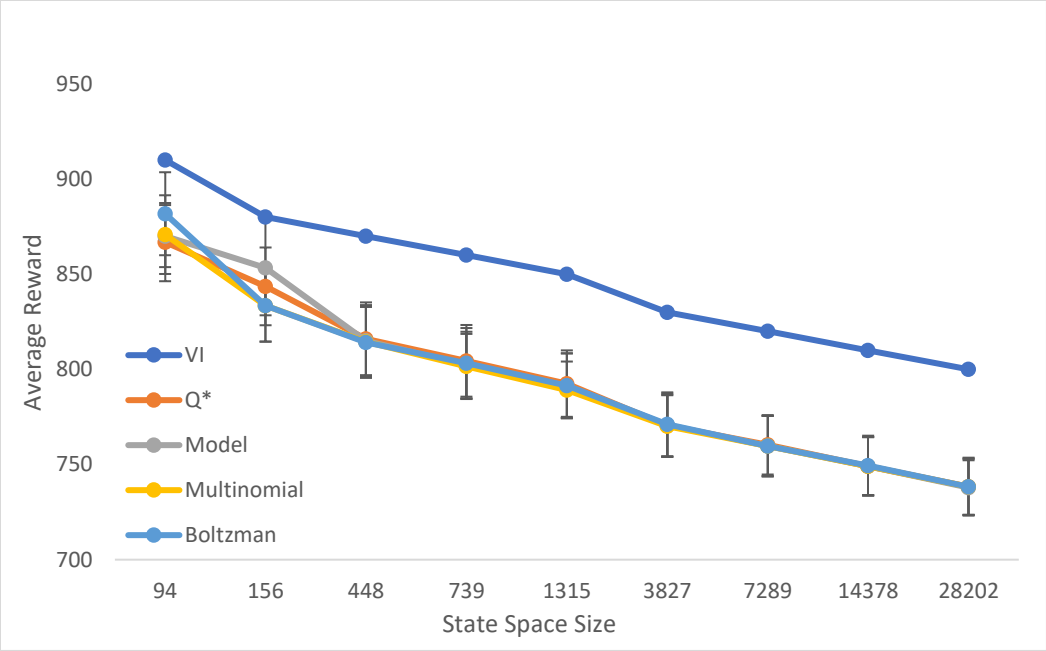

Figure 5.2 Reward Gap Between Ground MDP and Abstract MDP After Running VI

Table 5.1 Size of Ground MDPs Compared to Abstract MDPs

VI	Q*	Model	Multinomial	Boltzman
94	9	8	17	6
156	12	6	14	8
448	14	9	20	8
739	11	7	19	9
1315	13	10	18	9
3827	9	10	22	9
7289	10	8	20	8
14378	11	11	21	10
28202	12	9	20	10

CHAPTER VI

CONCLUSION

In conclusion, a healthcare network defended by an autonomic IRS was modeled as an MDP using BURLAP 3. The autonomic IRS was able to plan a response to a hypothetical attack from a specified start state using VI, and it was demonstrated that the four ϕ functions were able to abstract the ground MDP of the model healthcare system down to a fraction of its original size, while still maintaining near-optimality in terms of the reward produced by single-threaded VI. The main drawback of this project is that the runtime of the four ϕ functions are not fast enough to be used in conjunction with VI to solve very large MDPs more quickly than VI alone.

REFERENCES

[1] "An architectural blueprint for autonomic computing", *IBM White Paper*, 2006.

[2] "Apache HTTP Server Project", *The Apache Software Foundation*, 2018, http://httpd.apache.org/ (accessed Oct 2nd, 2018).

[3] "Apache Tomcat", *The Apache Software Foundation*, 2018, http://tomcat.apache.org/ (accessed Oct 2nd, 2018).

[4] Brown-umbc reinforcement learning and planning (burlap), 2018, http://burlap.cs.brown.edu/ (accessed Oct 2nd, 2018).

[5] "Common Vulnerabilities and Exposures", The MITRE Corporation, https://cve.mitre.org/ (accessed Oct 2th, 2018).

[6] D. Abel, "Code for abstracting, evaluating, and visualizing Markov Decision Processes", *Github*, 2018, https://github.com/david-abel/state_abstraction (accessed Oct 2th, 2018).

[7] D.Abel, D.E. Hershkowitz, and M.L. Littman, "Near Optimal Behavior via Approximate State Abstraction", *Proceedings of the 33rd International Conference on International Conference on Machine Learning*, ICML'16, June 2016, pp. 2915-2923.

[8] G. Bell and M. Ebert. "Health Care & Cyber Security: Increasing Threats Require Increased Capabilities", *KPMG*, 2015.

[9] J. Kephart and D.M. Chess. "The vision of autonomic computing." *Computer*, vol. 36, no. 1, 2003, pp. 41-50.

[10] M. Zamani and M. Movahedi, "Machine Learning Techniques for Intrusion Detection", *CoRR*, 2015.

[11] "MySQL", *Oracle*, 2018, https://www.mysql.com/ (accessed Oct 2nd, 2018).

[12] R. Sutton and A.G. Barto, "Reinforcement Learning: An Introduction", *MIT Press*, 2015.

[13] S. Anwar, et. al. “From Intrusion Detection to an Intrusion Response System: Fundamentals, Requirements, and Future Directions”, *Algorithms*, vol. 10, no. 39, 2017.

[14] S. Iannucci, Q. Chen and S. Abdelwahed, "High-Performance Intrusion Response Planning on Many-Core Architectures," *2016 25th International Conference on Computer Communication and Networks (ICCCN)*, Waikoloa, HI, 2016, pp. 1-6.

[15] S. Iannucci and S. Abdelwahed, "Model-based Response Planning Strategies for Autonomic Intrusion Protection", *ACM Transactions on Autonomous and Adaptive Systems*, 2018.

[16] S. Iannucci and S. Abdelwahed, "Towards Autonomic Intrusion Response Systems," 2016 *IEEE International Conference on Autonomic Computing (ICAC)*, Wurzburg, 2016, pp. 229-230.

[17] X. Chen, Z. Han, H. Zhang, G. Xue, Y. Xiao and M. Bennis, "Wireless Resource Scheduling in Virtualized Radio Access Networks Using Stochastic Learning," in IEEE Transactions on Mobile Computing, vol. PP, no. 99, pp. 1-1.

[18] D. Abel, et al., “Goal-based action priors”, ICAPS, pp. 306–314, 2015.