Dynamic Scheduling of Cybersecurity Analysts for Minimizing Risk Using Reinforcement Learning

RAJESH GANESAN, SUSHIL JAJODIA, and ANKIT SHAH, George Mason University
HASAN CAM, Army Research Laboratory

An important component of the cyber-defense mechanism is the adequate staffing levels of its cybersecurity analyst workforce and their optimal assignment to sensors for investigating the dynamic alert traffic. The ever-increasing cybersecurity threats faced by today's digital systems require a strong cyber-defense mechanism that is both reactive in its response to mitigate the known risk and proactive in being prepared for handling the unknown risks. In order to be proactive for handling the unknown risks, the above workforce must be scheduled dynamically so the system is adaptive to meet the day-to-day stochastic demands on its workforce (both size and expertise mix). The stochastic demands on the workforce stem from the varying alert generation and their significance rate, which causes an uncertainty for the cybersecurity analyst scheduler that is attempting to schedule analysts for work and allocate sensors to analysts. Sensor data are analyzed by automatic processing systems, and alerts are generated. A portion of these alerts is categorized to be *significant*, which requires thorough examination by a cybersecurity analyst. Risk, in this article, is defined as the percentage of *significant* alerts that are not thoroughly analyzed by analysts. In order to minimize risk, it is imperative that the cyber-defense system accurately estimates the future significant alert generation rate and dynamically schedules its workforce to meet the stochastic workload demand to analyze them. The article presents a reinforcement learning-based stochastic dynamic programming optimization model that incorporates the above estimates of future alert rates and responds by dynamically scheduling cybersecurity analysts to minimize risk (i.e., maximize *significant* alert coverage by analysts) and maintain the risk under a pre-determined upper bound. The article tests the dynamic optimization model and compares the results to an integer programming model that optimizes the static staffing needs based on a daily-average alert generation rate with no estimation of future alert rates (static workforce model). Results indicate that over a finite planning horizon, the learning-based optimization model, through a dynamic (on-call) workforce in addition to the static workforce, (a) is capable of balancing risk between days and reducing overall risk better than the static model, (b) is scalable and capable of identifying the quantity and the right mix of analyst expertise in an organization, and (c) is able to determine their dynamic (on-call) schedule and their sensor-to-analyst allocation in order to maintain risk below a given upper bound. Several meta-principles are presented, which are derived from the optimization model, and they further serve as guiding principles for hiring and scheduling cybersecurity analysts. Days-off scheduling was performed to determine analyst weekly work schedules that met the cybersecurity system's workforce constraints and requirements.

Categories and Subject Descriptors: C.2.3 Network Monitoring [**I.2.6 Learning**]: I.2.8 Scheduling

General Terms: Cybersecurity, Analysts, Dynamic Scheduling

R. Ganesan, S. Jajodia, and A. Shah were partially supported by the Army Research Office under Grants No. W911NF-13-1-0421 and No. W911NF-13-1-0317 and by the Office of Naval Research under Grants No. N00014-15-1-2007 and No. N00014-13-1-0703.
Authors' addresses: R. Ganesan, Department of Systems Engineering and Operations Research, George Mason University, Mail Stop 4A6, Fairfax, VA 22030-4422; email: rganesan@gmu.edu; S. Jajodia and A. Shah, Center for Secure Information Systems, George Mason University, Mail Stop 5B5, Fairfax, VA 22030-4422; emails: {jajodia, ashah20}@gmu.edu; H. Cam, Army Research Laboratory, 2800 Powder Mill Road, Adelphi, Maryland 20783-1138; email: hasan.cam.civ@mail.mil.

DOI: http://dx.doi.org/10.1145/2882969

Additional Key Words and Phrases: Cybersecurity analysts, dynamic scheduling, genetic algorithm, integer programming, optimization, reinforcement learning, resource allocation, risk mitigation

ACM Reference Format:
Rajesh Ganesan, Sushil Jajodia, Ankit Shah, and Hasan Cam. 2016. Dynamic scheduling of cybersecurity analysts for minimizing risk using reinforcement learning. ACM Trans. Intell. Syst. Technol. 8, 1, Article 4 (July 2016), 21 pages.
DOI: http://dx.doi.org/10.1145/2882969

1. INTRODUCTION

The task of a cybersecurity analyst includes examining the alerts generated by an intrusion detection system (IDS) such as SNORT or a Security Information and Event Management (SIEM) tool such as Bro [Paxson 1999] and identifying those that are considered as significant and require further investigation. Table X in the Electronic Appendix provides an example of cyber incident and event categories used by the U.S. Department of Defense [Chief Information Officer 2008]. All alerts are categorized on a scale of 1–9 with categories 1, 2, 4, and 7 (incidents) being severe ones. Category 1, 2, 4, and 7 alerts are analyzed and reported to a watch officer and then a report has to be written.

Dynamic scheduling to manage cybersecurity analysts to minimize risk is a critical infrastructure problem that poses several operational challenges and garners importance at the level of national security. In this article, we view cybersecurity analysts as a resource that must be allocated to the process of examining alerts in an optimal way that minimizes risk while meeting the resource constraints. Such resource constraints include the number of sensors on which an analyst can be trained or assigned, the expertise mix that the cybersecurity defense organization wishes to have, the expected utilization for the analysts, the time taken by an analyst to investigate an alert (translates to analyst workload), and preferences such as shift hours and days-off in a week for the analysts. Hence, existing adaptions of dynamic scheduling in manufacturing and service applications that are explained in the next section are inadequate to address the problem of scheduling cybersecurity analysts, and new adaptions must be designed, tested, and implemented. Also, a precise notion of risk is used, which is defined as the percentage of the significant alerts that were not thoroughly analyzed. This insufficient analysis of alerts could happen due to the following reasons such as sub-optimal scheduling of analysts, not enough personnel detection tools, and signature rules to analyze the alerts when there are varying rates of alert generation, lack of time to analyze a significant alert, and/or not having the right mix of expertise in the shift in which the alert occurs. For the remainder of the article, the term alert refers to the significant alert that must be thoroughly investigated.

Recently, there has been some work that has focused on the need to improve efficiency of cybersecurity analysts [Erbacher and Hutchinson 2012; Zimmerman 2014]. Optimal sensor-to-analyst allocation and scheduling for a static workforce size at a fixed alert generation rate has been investigated by Ganesan [Ganesan et al. 2015]. The above article presents a heuristic-based optimization model that is useful in three ways: (1) given an organization size and sensor-to-analyst allocation, one can determine the risk and analyst utilization (baseline); (2) given both risk and analyst utilization needs, one can determine the optimal size of the organization, optimal sensor-to-analyst allocation, and a static workforce schedule; and (3) given an organization size that reports to work, one can determine the optimal sensor-to-analyst allocation to minimize risk and maximize analyst utilization. While several insights on cybersecurity analyst allocation to sensors and scheduling are obtained in the above work, the static model cannot react to the uncertainties in alert generation rates from the broad range of vulnerabilities that the system is subjected to on a daily basis, which results in higher

risk. The above calls for dynamic (on-call) workforce scheduling that is capable of mitigating the risk in the face of the above uncertainties while adhering to the constraints of the operating environment.

The objective of the article is to formulate and test an adaptive and dynamic analyst scheduling strategy for effective cyber-defense that is capable of using an estimate of the varying future alert generation rates and scheduling an optimal number of cybersecurity analysts at different expertise levels that minimizes the risk and maintains risk under a pre-determined upper bound for a set of system defined parameters and constraints. The 1-day look-ahead dynamic scheduling strategy includes the determination of (1) an estimated workload for the next day based on an estimate of the number of alerts to be investigated from each sensor, (2) the static workforce required based on a historical daily-average alert generation rate, (3) the dynamic (on-call) workforce required based on the additional workload anticipated above and beyond the historical daily-average workload, (4) the sensor-to-analyst allocation of both static and dynamic workforce, (5) the desired expertise mix of the combined static and dynamic analyst workforce, (6) the days-off schedule for both static and dynamic workforce, and (7) the on-call schedule for the dynamic workforce. The article considers three levels of analysts based on their work experience: senior L3, intermediate L2, and junior L1 level analysts. An analyst can be allocated to more than one sensor based on their experience, and the time taken to analyze an alert is dependent on the experience of the analysts.

There are several contributions in this article. The first contribution of the article is a mixed-integer programming model for static allocation, which differs from the heuristic approach used in Ganesan et al. [2015]. The heuristic approach is useful in determining the best sensor-to-analyst allocation solution among several good-enough (near-optimal) solutions for every shift of the day, which could speed up the computational time for large size problems with hundreds of sensors. In this article, the need was felt to develop a static mixed-integer programming model to determine the *optimal* long-term organization size and static schedule of a regular workforce, which is then fed into the dynamic optimization algorithm for determining the daily on-call workforce size and their expertise. The combined static and on-call workforce is then used by a heuristic model to obtain the sensor-to-analyst allocation for each shift. The second contribution of the article is a stochastic dynamic programming optimization model for dynamically scheduling cybersecurity analysts to minimize risk and keep risk below a given pre-set level that is desirable for the cybersecurity system. The above risk minimization capability is a direct measure of the goodness of the analyst resource allocation system. It should be noted that the optimization algorithm is a generic one and is independent of the number of sensors; however, the algorithm's output adapts to the estimated workload from the varying alert traffic per sensor and the number of sensors in the system. The dynamic optimization model is solved using reinforcement learning (i.e., approximate dynamic programming), and the model uses the workforce size and scheduling output of the above static mixed-integer programming model as one of the inputs. The third contribution is another mixed-integer programming algorithm for static days-off scheduling that can schedule the cybersecurity analysts based on their expertise mix and shift-hours and days-off preferences, which is further used in the determination of the on-call schedule for the analysts. The above mixed-integer programming algorithm for static days-off scheduling differs from the days-off scheduling heuristic approach used in Ganesan et al. [2015] by providing a precise full-day (12-hour) and part-day (8-hour) schedule for all the analysts. The other contributions include a set of meta-principles that provide the general guidelines for developing and implementing an efficient dynamic scheduling system for cybersecurity analysts. The novelty lies in intertwining the static optimization and scheduling models

(both mixed-integer programming models), the dynamic optimization and scheduling models (reinforcement learning-based stochastic dynamic programming models), and a sensor-to-analyst allocation heuristic with human intervention capability to arrive at a functional and implementable framework that meets the objective of the article given above.

The article is organized as follows. Section 2 presents the related literature in cybersecurity analyst scheduling. In Section 3, the stochastic dynamic optimization model is presented. Individual algorithms for each model have been developed and are presented in the Electronic Appendix section of the article. Section 4 presents the results from testing the above model. Finally, in Section 5, the conclusions and future research directions are presented.

2. RELATED LITERATURE

Intrusion detection has been studied for over three decades, beginning with the pioneering works by Anderson [1980] and Denning [1986, 1987]. Much of the work has focused on developing automated techniques for detecting malicious behavior [Northcutt and Novak 2002; Di Pietro and Mancini 2008; Albanese et al. 2014; Ovelgonne et al. 2015]. Some of the early research focused on misuse detection models (developing signatures of known attacks; any matched activity is flagged as an attack) or anomaly detection models (characterizing normal behaviors of all users and programs; any significant deviation from the normal profile is considered suspicious). As the volume of alerts generated by intrusion detection sensors became overwhelming, a great deal of later research work focused on developing techniques (based on machine learning [Sommer and Paxson 2010] or data mining [Barbara and Jajodia 2002], for example) for reducing false positives by developing automated alert reduction techniques. Indeed, there are open-source [Paxson 1999] and commercially available [Zimmerman 2014] SIEM tools that take the raw sensor data as input, aggregate and correlate them, and produce alerts that require remediation by cybersecurity analysts. The article differs from the above literature by focusing on the cybersecurity analysts who are viewed as a critical resource. It develops a generic dynamic optimization algorithm that provides the flexibility to optimally schedule the cybersecurity analysts by splitting the workforce into two components: static and dynamic (on-call) workforces.

The dynamic scheduling in this article, in comparison with extensive work in the fields of reactive scheduling, real-time scheduling, online scheduling, dynamic scheduling for parallel machines and multi-agents, would apparently appear to be similar in terms of the overall goal where in scheduling decisions are done under uncertainty; however, dynamic scheduling in the cybersecurity field poses several new challenges and, to our knowledge, this is the first time where it has been applied to the cybersecurity area. One of the first differences with the current literature is the severity aspect of sub-optimal scheduling of cybersecurity analysts that has the potential for devastating consequences, ranging from an organization level to national and world levels twith regard to their security, financial integrity, and economic stability. Unlike any other published literature on scheduling, the cybersecurity scheduling problem is unique in terms of the factors that affect its implementation, namely sensor deployment, alert generation rates, 24/7 work time, shift periods, occurrence of unexpected events affecting analysts' workload, broad scope of cybersecurity vulnerabilities and exploits, and analyst experience. Another important difference with the existing scheduling literature is that the article's objective is to minimize risk, whereas the literature is focused on minimizing cost or tardiness or completion time.

Some of the literature pertaining to dynamic scheduling includes the work by Lesaint et al. [2003], where the authors discuss a heuristic dynamic scheduler to generate long-term schedules in the field of network technicians with the objective

of minimizing cost. Examples of dynamic scheduling from freight handling and airline fleet and crew scheduling are also geared toward reducing operational costs to improve customer satisfaction [Nobert and Roy 1998]. In comparison to the dynamic scheduling work in manufacturing, distribution, and supply chain management that uses multi-agents, the article's dynamic aspects differ considerably [Reis and Mamede 2002; Zhou et al. 2012]. For example, in the multi-agent environment, the coordination and communication mechanism is essential for autonomic decisions by the agents, whereas in cybersecurity the factors given earlier, such as the broad scope of cybersecurity vulnerabilities and exploits, and analyst experience are critical in making dynamic scheduling decisions. In job-shop and parallel machine scheduling, a reinforcement learning-based dynamic scheduler optimizes a cost function that is related to the weighted tardiness or completion time in the presence of uncertainty arising from future job arrivals [Paternina-Arboleda and Das 2005; Aydin and Oztemel 2000]. It is clear from the above that the practice of dynamic scheduling must be adapted to a new context of cybersecurity with the objective to minimize risk under several system and operational constraints, which is novel and complements existing literature.

3. CYBERSECURITY ANALYST DYNAMIC RESOURCE ALLOCATION MODEL

The following section presents the details of the cybersecurity analyst dynamic resource allocation model, which is built using the theory of stochastic dynamic programming, which is used for applications with sequential decision making under uncertainty in a dynamic environment that evolves and changes over time. The sequential decision-making problems with continuous-time states and decisions are studied under control theory [Wonham 1979], whereas problems with discrete states and decisions are studied under dynamic programming: Markov decision process, in particular, if the problem is stochastic [Bellman 1957; Puterman 1994]. Finding solutions to stochastic dynamic programming problems is challenged by the *curse of modeling*, that is, the need for state transition probabilities. However, this challenge is mitigated via stochastic approximation, which forms the foundation of Approximate Dynamic Programming (ADP) (a triangulation of dynamic programming principles, reinforcement learning, and statistics) [Sutton and Barto 1998; Gosavi 2003; Powell 2007].

3.1. Dynamic Optimization Model and Constraints

The alert generation, estimation, general requirements, and modeling assumptions for cybersecurity analyst scheduling are provided in the Electronic Appendix section of the article. The objective of the dynamic optimization model is to optimize the number of on-call analysts and their expertise that is needed daily, while keeping a long-term view of the potential need for on-call analysts in the future. As in any resource allocation problem, having too little or too many of the resource would make the solution trivial. However, with an adequate number of analysts, dynamic optimization is needed to manage the resource allocation optimally over the planning horizon.

A framework for the dynamic optimization model is provided in Figure 1. The dynamic optimization model consists of four models, which are executed in the following order: (1) a static mixed-integer programming model for obtaining the minimum number of analysts (static or regular workforce) for a historical daily-average alert generation rate calculated over the past 2-week period, (2) a second static mixed-integer programming model for obtaining the static days-off schedule that indicates which analyst is at/off work, (3) a dynamic model based on stochastic dynamic programming to obtain the minimum number of additional workforce and their expertise level that is needed (dynamic or on-call workforce) based on the estimated additional alerts per sensor for the next day, and (4) a genetic algorithm heuristic model that allocates sensors to analysts (both static and dynamic combined) subject to the constraints on

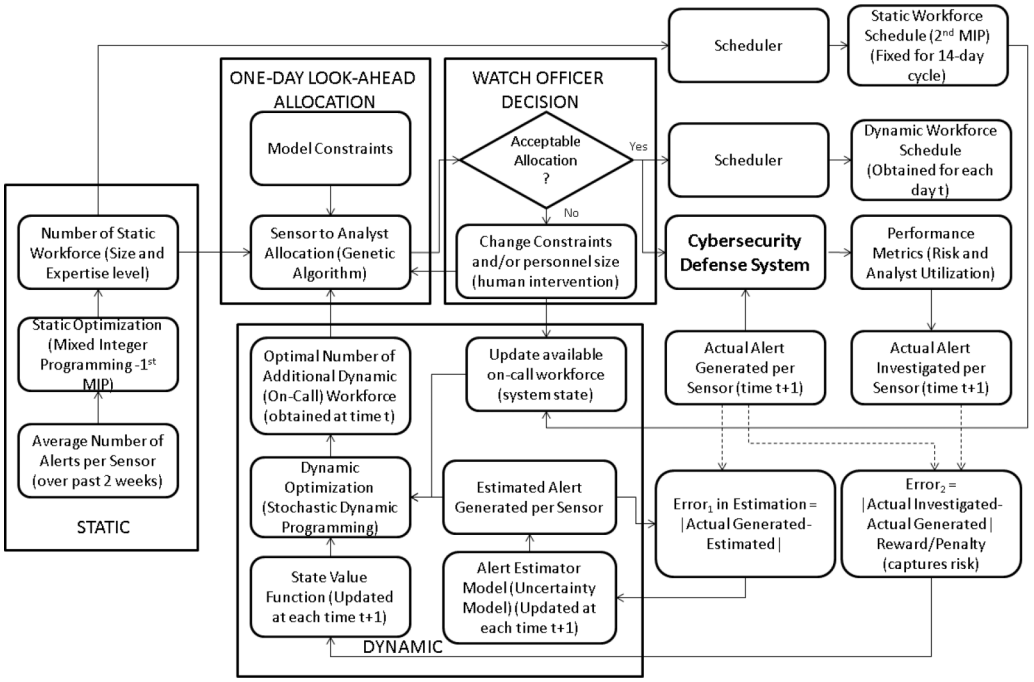

Fig. 1. Dynamic optimization model framework.

analyst utilization, upper bound on risk, system constraints on analyst workload, and the desired experience level mix that is specified by the organization. An acceptable allocation is decided by the watch officer, who has the capability to override algorithm decisions. The static model is run once at the beginning of each 14-day work period cycle, and the dynamic model is run each day of the operation. The 14-day cycle is motivated by the assumption that analysts work a 12-hour shift for 6 days and an 8-hour shift for 1 day (total 80 hours in 2 weeks), and the static cycle repeats every 14 days. Days-off scheduling is performed first for the static workforce. Among those analysts who are not at work in the static days-off schedule, the on-call or dynamic workforce is selected, henceforth referred to as *available* on-call workforce, which also considers those who have already served on-call within the 14-day window. The scheduling of the dynamic workforce is done within the dynamic programming framework to determine on-call rotation schedules.

3.2. Static Workforce Optimization and Scheduling

The main input to the static optimization model is the number of sensors and a historical daily-average alert generation rate calculated over the past 2-week period. All sensors are treated equally and the historical daily-average alert generation rate is assumed to be the same for all sensors over the next 14-day period. The static optimization algorithm begins with the assumption that a large number of analysts are available at each experience level, among which the minimum number will be selected such that the model constraints are met. It should be noted that all calculations are done per day (24-hour period) in the optimization model, whose output is now the input to the static scheduling model. Since regular (static) analysts work in 12-hour and 8-hour shifts as per the model assumptions, the static scheduling model will determine the final workforce size of the organization and a feasible long-term days-off schedule

for the analysts. The above optimization and scheduling formulation is modeled and solved as two separate mixed-integer programming algorithms as shown in Figure 1. The mathematical formulation and the steps of the mixed-integer programming algorithms are given in the Electronic Appendix [Winston 2003; Nemhauser and Wolsey 1999; Chen et al. 2010].

3.3. Dynamic Workforce Optimization and Scheduling

The three main inputs to the dynamic optimization model are the estimated additional number of alerts per sensor per hour for the following day, the available on-call analyst resource that must be optimally allocated, and the state-value function (explained later), which is derived from the error in alert estimation. The additional number of alerts per sensor per hour is the number that is over and above the historical daily-average per sensor per hour that was used in the above static optimization. Also, alert rates for a sensor could drop below the average per hour. All sensors are not treated equally and the alert generation rate is assumed to differ for all sensors both within a day and between days over the next 14-day period. The above estimation is provided by the alert estimator model on a daily basis; however, such a model was not developed in this article. Instead of an alert estimator model, the article assumes distributions for alert prediction, which could be replaced with the outputs of an alert estimator model. The dynamic optimization algorithm uses the information on next-day alert estimation, available on-call resource, the number of days left in a 14-day cycle, and its own state-value functions to determine the optimal number of dynamic (on-call) workforce needed along with their expertise level. See the dynamic block in Figure 1. As explained later, the state-value function plays a very important role by avoiding a myopic decision of reacting to completely fulfill all immediate analyst needs and running out of on-call analysts in the future when the estimated alert is high. Instead, the state-value function guides the decision-making process to be optimal overall by taking a long-term view that effectively manages the limited on-call resource.

3.3.1. Sensor to Analyst Allocation Using Heuristics. The sensor-to-analyst allocation for the following day is done by a genetic algorithm heuristic that considers the total workforce (static and dynamic) that reports to work and allocates them to sensors such that the model constraints are met as shown in Figure 1 under the 1-day look-ahead allocation block. If the allocation is not acceptable, then the constraints could be relaxed and/or the size and expertise mix of the on-call workforce could be overridden by a watch officer until an acceptable and feasible solution is found. In the long-run, it is expected that the alert estimation would improve and the dynamic programming model would have learned to find the optimal actions (optimal number of on-call workforce per day) so the genetic algorithm would also find an acceptable sensor-to-analyst allocation that meets the constraints of the model. Also, it is very important to note as a word of caution that (1) allowing the heuristic to perform the job of dynamic programming in selecting the size of on-call workforce and their expertise level would make the decision-making process myopic, which results in the risk of running out of on-call analysts in the future, and (2) excessive human intervention that changes the available on-call workforce will curtail the ability of the dynamic programming algorithm to learn action policies that makes the overall system optimal in the long-run. Decoupling the on-call decision-making process by dynamic programming and the allocation process by heuristic has a computational advantage because the dynamic programming model is driven by the need to minimize and balance risk over the 14-day period (risk is captured by $error_2 = \|\text{actual alerts investigated - actual alerts generated}\|$), and the computational complexity of finding a feasible sensor-to-analyst allocation subject to the constraints will not slow down the dynamic programming's decision-making

process. Besides, another advantage is that human intervention can be modeled separately, where the decision to override the dynamic programming's on-call workforce size decision will only affect the available on-call resource for the next day but not the current optimal decision of the dynamic programming model.

Once an acceptable sensor-to-analyst allocation is implemented for the following day based on estimated alert generation, at the end of that day, performance metrics on risk and analyst utilization are obtained using the actual alert generated and investigated by the analysts. The $error_2$ is also calculated, which captures risk. An accurate estimation of alerts per sensor would result in excellent performance metrics for risk (kept under the upper bound) and analyst utilization (kept within the requirement). However, if the number of actual alerts exceeds the estimated alerts, then the risk would increase. Similarly, if the number of actual alerts fell below the estimated alerts, then the analyst utilization would decrease. Hence, it is imperative to develop an excellent alert estimator model and to determine the optimal static and on-call workforce size in the presence of uncertainty in alert generation. The error in estimation $error_1 = \|$actual alerts generated - alerts estimated$\|$) is used to update the alert estimation model so the parameters of the model can be fined tuned, and $error_2$ is used as a reward/penalty for updating the state-value functions in the dynamic optimization model whose decision-making process will learn to bring in the optimal number (both size and expertise mix) of additional on-call workforce in the long-run.

3.3.2. Stochastic Dynamic Programming Formulation. Stochastic Dynamic Programming (SDP) models for dynamic resource allocation problems exploit the fact that for complex systems with no well-defined analytical models and no closed-form solutions, their *evolving properties can be studied through their interactions with the environment*. For the cybersecurity analyst dynamic scheduling problem, the availability of analysts is the dynamic (on-call) resource, and uncertainty is modeled by interacting with the dynamic alert generating environment. The generic SDP model involves defining the following elements: (i) system state variable B_t at time t, which are all recurrent states in a Markov chain; (ii) decision variable g_t to quantify the amount of resource(s) allocated at t; (iii) exogenous information from a real process or a process simulator W_{t+1}, which is observed between t and $t+1$ after decision g_t is taken; (iv) state transition function $B_{t+1} = h(B_t, g_t, W_{t+1})$, where B_{t+1} is the next system state at $t+1$, which depends on (B_t, g_t, W_{t+1}); (v) contribution function $C(B_t, g_t)$ that calculates the reward/cost for taking action g_t in state B_t; (vi) B_t^g and B_{t+1}^g are post-decision states at time t and $t+1$, respectively, which are all recurrent states in a Markov chain; and (vii) an objective function that minimizes (cost) or maximizes (reward) the total contribution (value of a state $V(B_t)$) over a long period (infinite horizon) of time. In the context of dynamic resource allocation of cybersecurity analysts the above variables are defined as follows.

(1) System state B_t: It is a two-dimensional (2D) vector that indicates the *available* number of on-call L2 and L3 analysts for allocation at time t where $t = \{1 \ldots 14\}$. L1 (junior) analysts are not used on-call for this article, because the purpose of an on-call workforce is to bring in the minimum and most efficient workforce to meet the additional workload demand. Time t is indexed at the end of each day (24-hour period) for this article.
(2) Decision g_t: It is a 2D vector that indicates the number of on-call L2 and L3 analysts allocated at time t in state B_t. The decision is verified for feasibility by the genetic algorithm heuristic, which allocates the sensors to the analysts (both static and dynamic as in Figure 1 subject to all the constraints of the model. If the decision was found infeasible, then the dynamic programming will adjust the decision until a feasible solution is found. This is referred to as the learning phase of dynamic

programming. It should be noted that the dynamic programming decision and subsequent verification of feasibility by a genetic algorithm is based on an estimated value of alert generation for the next day.

(3) Post-decision system state B_t^g: It is a 2D vector that indicates the *available* number of on-call L2 and L3 analysts at time t after decision g_t is taken.

(4) Exogenous information (uncertainty) W_{t+1}: The alert estimator model captures the uncertainty in alert generation for each sensor by estimating the number of additional alerts over and above the historical daily-average rate of alert generation. In this research, an alert estimator model is not developed. Instead, probability distributions are used in place of the estimator model.

(5) State transition function $B_{t+1} = h(B_t, g_t, W_{t+1})$: This function defines how the next system state at time $t + 1$ is evolved. Once a decision on the number of additional dynamic (on-call) workforce is made, the decision could be altered by the shift manager as the uncertainty W_{t+1} unfurls in the form of actual alert generation. Also, state transition probabilities are not known due to the large number of states in the system. Hence, a reinforcement learning-based approach is used for the dynamic optimization model.

(6) Contribution function $C(B_t, g_t)$: It is measured at the end of the following day after the actual alerts have occurred (uncertainty has unfurled), which is the same as $error_2$. The error captures risk, which is a function of the state and action that was taken from that state. The error would be negligible if the action taken to add on-call workforce based on an accurate number of estimated alerts was optimal for the actual number of alerts that were investigated.

(7) The objective function for the dynamic programming is measured as the long-run total discounted value of the states $V^j(B)$ as the iteration index $j \rightarrow \infty$, which is derived using the recursive Bellman's optimality Equation (1) shown below [Bellman 1957]. $V^j(B)$ is a cumulative sum of discounted $C(B_t, g_t)$ errors for the learning phase whose iterations are indexed from 1 to j. It should be noted that index t is reset to 1 after every 14-day cycle. The learning phase goes through several iterations (indexed with j) of 14-day cycles. Since the value of the state is measured in terms of errors in alert estimation, the objective function will be to minimize the long-run total discounted value of the states $V^j(B)$. In other words, the lower the value of $V^j(B)$, the better the system state. The dynamic programming algorithm will strive to move from one good state to another by making a decision under uncertainty, which is guided by the lowest value of the future states that are reachable at any given time t.

3.3.3. Phases of Stochastic Dynamic Programming. To achieve the objective of the article, the stochastic dynamic programming formulation proceeds in three phases—exploration, learning, and learned. The recursive Bellman's optimality equation that updates the value of the states is given as follows:

$$V^j\left(B_{t-1}^g\right) = (1 - \alpha^j)V^j\left(B_{t-1}^g\right) + \alpha^j \eta^j, \tag{1}$$

$$\eta^j = \left[\min_{g_t \in \mathcal{G}} \{C(B_t, g_t) + \beta V^j(B_t^g)\}\right], \tag{2}$$

where α^j is the learning parameter that is decayed gradually, $\mathcal{G}$ is the set of all feasible decisions from which the dynamic programming algorithm will choose a decision at every iteration, and β is the fixed discount factor that allows the state values to converge in the long run. It should be noted that for a decision g_t taken at the end of today from state B_t, the update of the value of post-decision state B_{t-1}^g from yesterday at $t - 1$ that had put the system into state B_t is executed at the end of the next day at $t + 1$ when

the actual alerts have been investigated. This is a class of time-lagged information acquisition problems where we know the result of a decision on a state by determining the next reachable post-decision state but do not know the value of the current state until it is updated after the uncertainty in alert rates is unfurled (information on actual alert investigated is known and acquired) at the end of the next day. Such problems occur in the real world such as when travel and hotel reservation decisions are done today for a future date, which affects the availability of the resource, but the value of such reservations is not known until the date has occurred, as given in Powell [2007].

(1) Exploration Phase: In this phase, the dynamic programming algorithm would explore several non-optimal decisions (but feasible as verified by genetic algorithm) and acquire the value of system states that are visited. Equation (1) is executed without the min operator in Equation (2) by taking random decisions for the number (and their expertise) of additional on-call workforce, and the value of $V^j(B_t^g)$ and $V^j(B_{t-1}^g)$ is used from the previously stored values if the state was visited earlier or 0 otherwise. Since the algorithm begins with all $V^0(B) = 0\ \forall B$ at $j = 0$, exploration helps to populate the values of some of the states that are visited. Exploration is stopped after a certain number of iterations, which depends on the size of the state-space and number of iterations planned for the learning phase. Also, during this phase, the alert estimator is also updated and its accuracy of alert estimation is improved.
(2) Learning Phase: In this phase, the dynamic programming algorithm would take (near-) optimal decisions at time t, which is obtained from Equation (2) with the min operator under the assumption that better alert estimates are available from the alert estimator ($error_1 = 0$) and all alerts can be investigated (Contribution function $= error_2 = 0$). The value of the previous post decision state is updated at time $t + 1$ as per Equation (1) with the known value of error (Contribution function $= error_2$). After several iterations, learning is stopped when convergence of the value of the states is achieved, as measured in terms of the mean-square error of the stochastic gradient, which is given in Powell [2007].
(3) Learned Phase: This is the implementation phase of the dynamic programming. The inputs to this phase include the value of the states at the time when learning was terminated. The alert estimator is also expected to have significantly improved its accuracy of alert estimation per sensor. In this phase, the dynamic programming algorithm would take optimal decisions at each time t, which is obtained from Equation (2) with the min operator under the assumption that better alert estimates are certainly available from the alert estimator ($error_1 = 0$) and all alerts can be investigated (Contribution function $= error_2 = 0$). The dynamic programming algorithm will evaluate all its feasible actions and chooses an action that takes the system to the post-decision state with the lowest value (minimization problem).

4. RESULTS

The following section presents the results of the integer programming optimization model for a static sensor to analyst assignment with no estimation of future alerts, which serves as a baseline for comparison with the learning-based stochastic dynamic programming model that includes the estimation of future alert rates. The following input parameters and constraints were provided.

(1) Number of sensors = 10
(2) An organization that aims to have a mix of personnel with L1-, L2-, and L3-level experience should aim to find the right mix of expertise for a given upper bound

Table I. Static Sensor-to-analyst Allocation by Integer Programming

Sensor →		1	2	3	4	5	6	7	8	9	10
Level ↓	Analyst ID ↓										
L3	1	1	1	0	1	0	1	0	0	0	1
	2	1	0	1	0	0	0	1	0	1	1
	3	0	1	0	1	0	0	1	1	1	0
	4	1	0	0	0	1	1	0	0	1	1
L2	5	0	0	1	0	0	1	1	1	0	0
	6	0	0	1	1	1	0	0	0	0	1
	7	1	0	1	1	1	0	0	0	0	0
L1	8	0	0	0	1	0	0	0	0	0	0
	9	0	0	0	0	1	0	0	0	0	0
	10	0	0	0	0	0	0	1	0	0	0

on risk level that it wishes to maintain. The required analyst proportion in the organization was set to 20–40% L1, 30–50% L2, and 30–40% L3 levels.

(3) Number of days to optimize per run of the algorithm - 2 weeks (14 days). The 2-week run is used to obtain $12 * 6 + 8 * 1 = 80$ hours of work in a 14-day period.
(4) Risk upper bound = 5%
(5) Analyst utilization = 95–100%
(6) Number of sensors to be allocated per analyst: one to two for L1, three to four for L2, and four to five for L3.
(7) Alert generation rate per sensor per hour. The significant alert generation rate was 1% of the entire alerts generated per day. The remaining alerts are considered insignificant. Due to lack of real-world data (highly sensitive), an alert estimator model was not developed. Instead, the alert predictions were generated for 14 days using a combined Uniform distribution U(0,13) and Poisson(2) distribution with a mean value of $\lambda = 2$ alerts per hour/sensor (referred to as baseline for this research). The Poisson distribution provided a wide range of variability for the uncertainty model. After combining the above distributions, the actual alert generation was approximately nine alerts per hour per sensor and was drawn from (U(0,18)) per hour per sensor, with each sensor having a different rate of alert generation. The above process for predicted and actual alert generation was repeated for each 14-day run of the integer programming model. In the real world, the actual rate will come from the process itself and the predicted rate from the statistical model developed by the organization, which is part of the future work of this research.
(8) Average alert analysis rate for each level of analyst is specified in time units: 12 min/alert for L1, 7.5 min/alert for L2, and 5 min/alert for L3.

4.1. Results from Mixed-Integer Programming Model (Static Model)

The mixed-integer programming model determines the optimal number of static analysts at each expertise level per 12-hour shift and optimally allocates the sensors to analysts. The objective is to minimize the total number of analysts in the organization. Table I provides the sensor-to-analyst allocation using the static mixed-integer programming model for a sample run of the algorithm. It can be observed from the output of the mixed-integer programming model that there are three L1-, three L2-, and four L3-level analysts needed per 12-hour shift for the above average alert prediction rate (14-day average of approximately 2164 alerts in a day as explained in Table II (also the baseline for this research)). The 1's in the 0-1 matrix indicate that the sensor is allocated to the analyst. The constraint on the number of sensors to be allocated to each level of analyst is also met. The allocation scheme is maintained the same for a 14-day period. At the end of the 14-day period, the historical daily-average alert

Table II. Number of Alerts Investigated and Risk % for a Sample 14-day Run Using a Static Sensor-to-Analyst Allocation

Day →	1	2	3	4	5	6	7	8	9	10	11	12	13	14	Average
Total # actual alerts →	2275	2064	2256	2066	2125	2250	2165	2230	2110	2165	2089	2225	2165	2105	2164
# of alerts investigated	2088	2077	2088	2053	2088	2088	2088	2088	2088	2088	2077	2088	2088	2088	2084
Risk %	8.0	0.0	8.0	0.0	2.8	7.0	3.9	7.0	0.6	2.8	0.0	6.0	3.9	0.6	3.6

generation rate is updated based on actual alert generation that was observed, and a new sensor to analyst allocation scheme is obtained by rerunning the mixed-integer programming model. Another observation is that the number of L3 analysts is chosen to be the maximum among its allowable range of 30–40% in an organization. This is because the objective is to minimize the total number of analysts, and L3 analysts are the most efficient.

Table II shows a sample output of the actual number of alerts generated, actual number analyzed, and risk % for the sensor to analyst allocation in Table I as determined by the mixed-integer programming model. It can be observed that the risk was $\leq$5% on days 5, 7, 10, and 13 when the number of actual alerts were statistically indifferent to the 14-day historical daily-average alert generation rate reported in the last column of the table (14-day average = 2164). The above outcome is because the mixed-integer programming model was run with a 5% risk upper bound and a fixed 14-day historical daily-average alert generation rate as input. On days 1, 3, 6, 8, and 12, when the actual number of alerts exceeded the historical daily average (statistically significant), the risk % was higher because the integer programming model has a static sensor-to-analyst allocation and could not adapt the workforce size to match the increase in alert generation. Similarly, on days 2, 4, 9, 11, and 14 with alert generation being significantly below the historical daily-average alert generation rate, the risk was closer to the ideal value of 0%, because the mixed-integer programming optimization model for determining the minimum number of analysts and their sensor-to-analyst allocation was run with a 5% upper bound on risk. Also, it must be noted that the utilization requirement of all analysts was kept between 95 and 100% during the run of the optimization model. However, in a rare occurrence, if the actual number of alerts falls far below the historical daily-average alert generation per day, then one can expect some underutilization of analysts.

Clearly, the above sample result indicates that the static mixed-integer programming model cannot adapt to the uncertainty in alert generation rate and, at best, can provide a static sensor-to-analyst allocation for a given 14-day historical daily-average alert generation rate. Consequently, on days when the alert generation is higher than the historical daily average, the risk will be higher than the 5% upper bound. Therefore, a dynamic workforce scheduling model is needed, which estimates the 1-day look-ahead uncertainty in workload and schedules the analysts to meet the workload demand. The results of the dynamic model are presented next.

4.2. Results from Stochastic Dynamic Programming Model (Dynamic Model)

The dynamic model is run in three stages: exploration, exploitation (learning), and learned (implementation or validation). All of the inputs as specified above in the mixed-integer programming model are valid for the dynamic model, except there is a change in how the alert generation is processed by the model. Unlike the integer programming model, which uses only a historical average alert generation rate, the dynamic model estimates the 1-day look-ahead alert generation rate at the end of the current day. If the estimate exceeds the historical daily average, then analysts from the dynamic (on-call) workforce is called in to work for the next day. The historical daily-average alert generation is handled by the daily scheduled workforce, who is

Table III. Input for Dynamic Programming: Estimate of Total Number of Alerts per Sensor per Day

Day of week →		Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat
Day →		1	2	3	4	5	6	7	8	9	10	11	12	13	14
Sensor ↓	Alerts/day ↓ (U(0,13)/ hour)	Total Alerts/day ↓ (U(0,13)+Poisson(2) /hour)													
1	166	190	214	262	190	166	214	262	286	166	214	214	166	214	214
2	117	141	141	237	165	237	141	237	237	237	213	117	141	213	117
3	200	248	224	200	224	224	200	224	200	224	296	320	296	320	224
4	155	203	251	179	179	251	179	179	179	155	179	155	203	275	203
5	179	275	203	227	275	203	227	275	227	203	179	203	203	179	203
6	189	285	237	237	213	237	237	213	237	285	213	213	285	213	213
7	171	291	171	219	195	171	219	219	195	195	291	219	291	195	291
8	171	291	291	267	171	195	267	171	219	195	171	195	195	171	195
9	152	152	176	248	272	248	272	176	200	248	176	176	248	176	176
10	169	193	169	193	169	217	289	217	265	193	217	265	193	217	265
Est. Alert Total →	1669	2269	2077	2269	2053	2149	2245	2173	2245	2101	2149	2077	2221	2173	2101
Actual Alert Total →	U(0,18) →	2275	2064	2256	2066	2125	2250	2165	2230	2110	2165	2089	2225	2165	2105
$Error_1$ in estimation		6	−13	−13	13	−24	5	−8	−15	9	16	12	4	−8	4

referred to as static workforce in both the stochastic dynamic programming model and the static mixed-integer programming model. The results of the learned phase of the dynamic programming model are given below.

4.2.1. Alert Processing by Dynamic Programming. In order to compare the dynamic programming results with the mixed-integer programming results, the predicted alert generation rate was maintained at U(0,13) + Poisson(2) /hour/sensor, which is also the number of alerts for the static model that has no prediction. Myopically, the decision of a shift manager would have been to determine the required number of additional analysts at each level of expertise that are needed to investigate fully the predicted additional demand in the workload due to the alerts that are over and above the historical daily-average number of alerts. Thus, a long-term view is not taken, and there is a potential danger to run-out of on-call workforce toward the later part of the 14-day work cycle. Since the dynamic workforce is limited, the dynamic programming decision obtained from Equation (2) may not necessarily provide all of the required number of analysts to meet the next-day's additional workload demand fully. Instead, the dynamic programming algorithm will aim to balance risk among the 14 days by taking a long-term view (includes the value of the next system state), which in turn avoids the situation of running out of on-call resources when there is a critical need. At the end of the next day, the total number of actual number of alerts investigated is known, and ($error_1$) is calculated, which can be used to update an alert estimator model (one of the future work of this research), and ($error_2$) is used as the reward/penalty scheme to update state-value functions within the dynamic programming model.

Table III shows the input alert data for a 14-day operation. For a comparison between integer and dynamic programming results, it can be noted that the total number of actual alerts per day for dynamic programming was kept the same as those in Table II, which was used in the mixed-integer programming algorithm (static model).

4.2.2. Interpretation of Results from Dynamic Programming. Table IV presents the sensor-to-analyst allocation decision by the dynamic model for a particular day of operation in which the estimated alerts generated matched (no statistical difference) the historical daily-average alert generation per day that was used in the static mixed-integer programming model. For example, days 5, 7, 10, and 13 in Table III have estimated alert generation that is statistically indifferent to the 2-week historical daily-average alert generation per day of 2,164 shown in Table II. It can be noted that the number of analysts needed on the above days is three L1, three L2, and four L3 per

Table IV. Sensor-to-Analyst Allocation by Dynamic Programming for Average Alert Generation Days (Three L1, Three L2, and Four L3 Analysts)

Sensor →		1	2	3	4	5	6	7	8	9	10
Level ↓	Analyst ID ↓										
L3	1	1	1	1	1	0	0	0	0	0	0
	2	1	1	1	0	1	0	1	0	0	0
	3	1	1	0	0	1	1	0	0	0	1
	4	1	1	0	0	1	0	1	0	0	1
L2	5	1	0	1	0	1	0	0	0	1	0
	6	1	0	0	1	0	0	0	1	0	1
	7	1	1	1	0	1	0	0	0	0	0
L1	8	1	0	0	0	0	1	0	0	0	0
	9	0	1	0	0	1	0	0	1	0	0
	10	0	0	0	0	1	0	0	0	1	0

Table V. Sensor-to-Analyst Allocation by Dynamic Programming for 5% Increase in Alerts over Average Alert Generation (Three L1, Three L2, and Five L3 Analysts)

Sensor →		1	2	3	4	5	6	7	8	9	10
Level ↓	Analyst ID ↓										
L3	1	1	1	0	0	0	1	0	1	1	0
	2	1	1	0	0	0	1	0	1	1	0
	3	1	1	0	0	0	1	0	1	1	0
	4	1	1	0	0	0	1	0	1	1	0
	5	0	0	1	1	0	1	0	1	1	0
L2	6	0	0	0	0	0	1	1	1	0	1
	7	0	0	1	0	1	0	1	0	0	1
	8	0	0	1	0	1	0	1	0	0	1
L1	9	0	0	0	1	0	0	1	0	0	0
	10	0	0	0	1	1	0	0	0	0	0
	11	0	0	0	0	1	0	0	0	1	0

12-hour shift, which is the same as the mixed-integer programming solution. Although, Tables I and IV differ in the allocation of sensors to analysts, there is no statistical difference in the number of alerts investigated on the above days between the static integer and dynamic programming models, which highlights the existence of multiple feasible allocations. Also, all the inputs and constraints of the model, such as the upper bound for risk and analyst utilization, were met.

Table III indicates that on days 1, 3, 6, 8, and 12, the number of alerts estimated has significantly exceeded the 2-week historical daily average of 2,164 alerts per day, which was found to statistically differ using a student t-test at a 95% confidence level. Therefore, the static sensor to analyst allocation from integer programming (Table I) cannot achieve a risk below the 5% upper bound as shown in Table II. In order to bring risk below the 5% upper bound, the dynamic (on-call) workforce is needed. The stochastic dynamic programming algorithm uses the estimate of the additional alerts per day (over and beyond the historical daily-average per day) and makes a decision to add one more L3-level analyst, thus bringing the total required analysts to three L1, three L2, and five L3 analysts, which in turn brings the risk below 5% as shown in Table VI. Table V provides the sensor-to-analyst allocation for the combined static (three L1, three L2, and four L3) and dynamic (1 L3) workforce for the above days. Also, analyst utilization was greater than 95%.

Table III indicates that on days 2, 4, 9, 11, and 14, the number of alerts estimated is significantly less than the historical daily average of 2,164 alerts per day

Table VI. Risk % over a 14-day Period Using Dynamic Programming with Dynamic (On-Call) Workforce

Day of week →	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat
Day →	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Total # actual alerts →	2275	2064	2256	2066	2125	2250	2165	2230	2110	2165	2089	2225	2165	2105
Total # of alerts investigated	2218	2057	2218	2057	2057	2218	2057	2218	2057	2057	2057	2218	2057	2057
Risk %	2.5	0.3	1.7	0.4	3.2	1.4	5.0	0.5	2.5	5.0	1.5	0.3	5.0	2.3
Analyst mix	3L1	3L1	3L1	3L1	3L1	3L1	3L1	3L1	3L1	3L1	3L1	3L1	3L1	3L1
	3L2	3L2	3L2	3L2	3L2	3L2	3L2	3L2	3L2	3L2	3L2	3L2	3L2	3L2
	5L3	4L3	5L3	4L3	4L3	5L3	4L3	5L3	4L3	4L3	4L3	5L3	4L3	4L3

(statistically different). Obviously, no additional dynamic workforce is needed and the regular workforce of three L1, three L2, and four L3 will continue to work on the above days as per the allocation in Table IV. There is the possibility that analyst utilization might be slightly less than 95% for a few analysts in some shifts, even after balancing the workload among a group of analysts that are allocated to a group of sensors; however, due to scheduling constraints, there is no effort in this article to reduce the daily (static) workforce size or rearrange the sensor to analyst by reallocation within a shift (future work).

Finally, the actual number of alerts per sensor per hour was obtained using $U(0, 18)$, and the total number of actual alerts per day from all sensors is shown in Table III. Using a pairwise student t-test, it was observed with a 95% confidence level that there was no statistical significant difference between the total predicted and the total actual number of alerts generated ($error_1$) per day. In the real world, the alert prediction model per sensor must be constructed from historical actual alert patterns, and actual alerts per sensor must be obtained directly from the intrusion detection system. The risk percentage is shown in Table VI, which is obtained by comparing the number of alerts investigated to the actual number of alerts generated. Since the risk percentage was kept below 5%, it can be concluded that the optimization decision to bring in the additional analysts (dynamic workforce) was optimal subject to the constraints in the model.

4.2.3. Response to Temporal Patterns and Spikes in Alert Generation. The Poisson(λ) part of the alert prediction distribution was changed to induce high variations in the number of alerts generated per hour per sensor while keeping U(0,13) fixed. For example, λ values were increased from 2 to 5. Consequently, the total number of alerts predicted per day increased on average by about 28.7%. A total of four case studies were conducted in which the spike in alert generation (with Poisson(5)) was created at the beginning (days 1–4), middle (days 6–9), end (days 11–14), and at random for 5 days of the 14-day cycle, and on the remaining days Poisson(2) was used along with U(0,13). In each of the above cases, the dynamic programming algorithm responded with additional on-call workforce. The decision to bring on-call workforce is dependent on the available workforce, the number of days left in the 14-day period, the predicted number of alerts, and the accuracy of prediction. The main advantage of the dynamic programming model's decision is that it is not myopic, and it will not respond with an on-call workforce to any and all alert predictions that are given as input because doing so has the risk of catering to all immediate needs without thinking of the future on-call workforce needs. Moreover, the prediction accuracy might differ from the actual. By incorporating the value of the future state of the system in decision making and by learning from the errors in decision making (known as interaction with the environment in reinforcement learning literature), the dynamic programming model is capable of taking the best decisions over a long period of time. If there are several unpredictable spikes in alert generation, then the current static system in today's practice would immediately fail and the risk will increase. The dynamic programming model would first exhaust its

Table VII. On-Call Analysts Needed for Spikes in Alert Generation

Poisson(λ)	% increase in average alert generation over baseline	Static workforce	Dynamic workforce	Risk
2	0	3L1, 4L2, 4L3	none	$\leq$5%
3	5.5	3L1, 4L2, 4L3	1L3	$\leq$5%
4	14	3L1, 4L2, 4L3	1L3 and 1L2	$\leq$5%
5	28.7	3L1, 4L2, 4L3	2L3 and 2L2	$\leq$5%

on-call workforce and would eventually fail beyond a limit but it would have offered a good cushioning effect that absorbs the unpredictable spikes to a reasonable extent as shown in Table VII. However, if the unpredictable spikes are sustained, then the static model would have to be re-executed with an increase in the size of the workforce so the unpredictability in alert generation is within a manageable range for the given analyst resource. Table VII provides the number of on-call analysts needed for an increase (spike) in the predicted number of alerts over the daily average number of alerts (baseline) for any given day of operation.

For a further increase in λ value beyond 5, the dynamic programming did not increase the number of on-call analysts. This is an important observation of the learning process of dynamic programming because the algorithm has trained itself to respond only to a certain percentage increase in alert prediction in order to conserve the on-call resource for a future use in the 14-day period. For the case study where spikes were noticed toward the end of the 14-day period, the algorithm was willing to allocate more on-call workforce; however, there were practical limitations on the number of available on-call workforce per day as shown in Table XII of the Appendix.

4.2.4. Exhausting On-Call Workforce. Another experiment was performed to determine the value of λ at which the on-call workforce would be completely exhausted, given the constraint that L3- and L2-level analysts can work on 2 on-call days in a 14-day period. Unlike the previous experiment with spikes on certain days, here the value of λ was kept fixed for the entire 14-day period. It was observed that at $\lambda = 5$ the entire on-call workforce was exhausted. Some of the days in the 14-day period recorded an average of 28.7% increase in alert generation over the baseline and on those days the risk was near 0% with 2 L3 and 2 L2 additional on-call analysts. The other days in the same period had an alert generation that was as high as 45% above baseline average alert generation in day (summed over all sensors). For days with very high alert generation ($>$28.7% over baseline), the risk was also above 0% because the dynamic programming algorithm did not provide all the required on-call staff to meet all of the high demand for alert investigation (the algorithm provided two L3 and two L2 additional on-call analysts only). However, the overall maximum risk across all 14 days did not exceed 12% with Poisson($\lambda = 5$). By allowing risk to be close to its upper bound of 5%, about an additional 33% of the alerts were investigated over and above the average baseline (obtained from U(0,13)+Poisson(2)/hour/sensor) by the additional two L3 and two L2 on-call analysts. The results obtained above support intuition because if an analyst works on 2 on-call days (24 hours extra), then this is about 30% more work hours over a 2-week period (80 hours of work). Therefore, if all on-call analyst resources are exhausted over 2 weeks, where each analyst has worked on 2 on-call days, then an additional 30% increase in alert investigation can be expected. In other words, an overall increase of about 30% in alert generation can be handled effectively by the on-call workforce. Since L1 analysts are not on-call, the additional increase in alerts that can be handled effectively by the on-call workforce is expected to $<$30%.

4.2.5. Scalability. A scalability test with 20, 30, and 40 sensors was performed to test the adaptability of the dynamic scheduling framework to the increase in the number of sensors. The arrival rate of significant alerts and service rate by analysts were kept

constant. The static model was run to obtain the optimal number of analysts at each level of expertise, which was found to proportionally increase with the increase in the number of sensors. The dynamic model was then applied, and it was observed that the decision to bring in on-call workforce adapted to the changes in significant alert generation rates.

4.3. Scheduling of Analysts

Days-off scheduling for cybersecurity analysts was run separately with inputs from the static mixed-integer optimization model on the number of L1-, L2-, and L3-type analysts required per day. Since the static optimization model provides the number of analysts that are required to be working per day, the real-world system will have additional analysts in each of the L1-, L2-, and L3-level expertise to account for those who are on the dynamic (on-call) workforce and those who are having days off in any given day.

For the average alert generation days with 3 L1, 3 L2, and 4 L3 analysts per 12-hour shift or 6 L1, 6 L2, and 8 L3 per day, the scheduling heuristic in the Electronic Appendix (Table XII) suggests a payroll number of 12 L1, 12 L2, and 16 L3 analysts (total 40) for a set of given scheduling constraints. The above assumes that each of the analyst works for 12-hour shifts, which will yield $12 * 7 = 84$ hours of work, whereas the scheduling constraint calls for $12 * 6 + 1 * 8 = 80$ hours of work in a 2-week period. There are three ways to handle the above.

(1) Keep the workforce at 40 (12 L1, 12 L2, and 16 L3), and pay the analysts for an additional 4 hours over and above the 80-hour limit in a 14-day period.
(2) Keep the workforce at 40 (12 L1, 12 L2, and 16 L3), and each analyst works for only 80 hours. In this case, the risk will increase, which depends on the number of analysts that are on 8-hour shifts and the total number alerts generated in a particular day.
(3) Increase workforce size such that the risk is maintained below 5% and all analysts work for 80 hours in a 14-day period (Table XI).

Table XI in the Electronic Appendix shows the output of the scheduling algorithm where mixed-integer programming is used for scheduling, which precisely determines the full-day (12-hour) and part-day (8-hour) work schedules for the analysts. The actual number of additional dynamic (on-call) workforce that reported to work (determined by dynamic programming) over and above the static workforce of three L1, three L2, and four L3 analysts per 12-hour shift is shown in Table VI.

4.4. Sensitivity Analysis

One of the main sensitivity analyses is to test the efficacy and adaptability of the stochastic dynamic programming model to minimize risk subject to the variations in the prediction of alert generation. The total number of actual alerts is calculated from the sum of alerts generated at U(0,13) /sensor/hour and the additional alerts generated from Poisson(λ)/sensor/hour for capturing the uncertainty in alerts. Again, the sum of $U(0, 13)$ *and* $Poisson(2)$ was used as the baseline historical daily-average alert generation rate. To perform the sensitivity analysis, the value of λ is increased very gradually (instead of steps of 1 for inducing spikes as shown earlier). The predicted value of alerts, the available workforce, and the number of days left in the 14-day cycle were used to trigger a decision of how many additional dynamic (on-call) workforce was needed and at what level of expertise. As performance metrics, the changes in the risk percentage and workforce mix were observed. Table VIII provides the outcome of the above study. As the alert generation was increased in steps (measured in percentage increase over the historical daily-average alert generation rate per day), there are

Table VIII. Sensitivity Analysis

Estimated number of alerts	Static workforce per 12-hour shift	Additional dynamic workforce needed per 12-hour shift	Risk %
Number of alerts below average	3L1,3L2,4L3	No dynamic workforce	less than 5%
Average number of alerts	3L1,3L2,4L3	No dynamic workforce	5
2.5% increase in alerts above average	3L1,3L2,4L3	1L3	0
9.3% increase in alerts above average	3L1,3L2,4L3	1L3	5
9.3% increase in alerts above average	3L1,3L2,4L3	1L3 and 1L2	0
18.4% increase in alerts above average	3L1,3L2,4L3	1L3 and 1L2	5
18.4% increase in alerts above average	3L1,3L2,4L3	2L3	0
24.8% increase in alerts above average	3L1,3L2,4L3	2L3	5
24.8% increase in alerts above average	3L1,3L2,4L3	2L3 and 1 L2	0
28.7% increase in alerts above average	3L1,3L2,4L3	2L3 and 1 L2	5
28.7% increase in alerts above average	3L1,3L2,4L3	2L3 and 2L2	0
33% increase in alerts above average	3L1,3L2,4L3	2L3 and 2L2	5

Table IX. Confidence Interval on Risk and Analyst Utilization

Number of sensors	10
Static workforce per 12-hour shift	3 L1, 3L2, 4 L3
Number of simulation runs	50
Length of each simulation run	14 days
Additional dynamic workforce needed per 12 hr shift	No dynamic workforce
Average number alerts per day	2110
Variation allowed over average	2.50%
95% confidence Interval on Risk %	1.5%–4.9%
95% confidence Interval on Analyst Utilization	97.3%–100%

certain intervals when the risk would increase from 0% towards 5% for a constant analyst mix. Any further increase in predicted alert generation would then increase the risk above 5%; however, by adding an analyst from the dynamic workforce, the risk is reduced to 0%. For example, between 2.5% and 9.3% increase in alert generation over the above historical daily average, one additional L3 analyst is enough to maintain the risk below 5%; however, any increase above 9.3% in alert generation over the historical daily average would require an additional L2 analyst to keep the risk under 5%. The combined effort of one additional L3 and one additional L2 analyst can keep the risk under 5% until the additional alerts generated increase above 18.4% of the historical daily average alert generation rate. The above observations show that the stochastic dynamic programming algorithm is adaptable to increases in historical daily-average alert generation rates by drawing on a dynamic workforce (on-call analysts) to meet the uncertain demands in alert investigation such that the risk remains below the preset upper bound of 5%.

4.5. Validation of Optimization using the Learnt Phase

The sensor-to-analyst allocation from the dynamic programming algorithm (Table IV) was tested on several alerts that were sampled from the combined U(0,13) and Poisson(2) distributions using 50 simulation runs. In each instance of the simulation run, actual alerts were generated over a 14-day period, alerts for the next day were predicted using the combined Uniform and Poisson distributions, and the risk percentage and overall analyst utilization were measured. Also, 95% confidence intervals were calculated for the above performance metrics as shown in Table IX. The 50 simulations gave an average of 2,110 alerts per day. The solution of three L1-, three L2-, and four L3-level analysts was valid up to an additional 2.5% increase in alerts over and above

the historical daily-average alert generation rate of $U(0, 13) + Poisson(2)$/sensor/hour. For all days in each of the 50 simulations, the risk was maintained under 5% and analyst utilization above 95%. Thus, the optimization results were validated for up to 2.5% increase in alert generation over the baseline. Beyond 2.5%, Equation (2) with contribution function $error_2 = 0$ provided the number of on-call analysts required for the next day. Similar validation studies were conducted for a >2.5% increase in alert generation over the baseline.

5. CONCLUSION

The article presents a stochastic optimization model for dynamic scheduling of cybersecurity analysts. The problem is modeled as a dynamic programming model and solved using reinforcement learning with the objective to minimize the number of analysts and provide a sensor-to-analyst allocation, which minimizes risk and maintains risk under a given upper bound, achieves a given expertise mix, and obtains an utilization value for the analysts as desired by the cyber-defense organization. The model is very generic and can be used with any number of sensors with various alert generation rates. The output is (sensor-to-analyst allocation) adaptable to increasing alert generation rates by drawing from a pool of available dynamic analyst workforce (on-call). Risk as defined in this article can be further mitigated with the above dynamic workforce that complements the static workforce to meet the increase in the workload demand for alert investigation. A combination of uniform and Poisson distributions is used as the 1-day look-ahead prediction of alert generation per sensor. A days-off scheduling model for static workforce is presented using mixed-integer programming from which the dynamic workforce is determined. The sensitivity study, scalability study, and validation study confirms the viability of the dynamic model for generating efficient sensor to analyst allocation under the uncertainty of alert generation and model constraints.

There are several future extensions to the research article. Uncertainty modeling (alert estimation or prediction) is a key component that drives the dynamic workforce scheduling. Hence, accurate estimate of the uncertainty is very critical. The article used an uncertainty model by combining two distributions to test the concept of dynamic scheduling; however, sophisticated models of uncertainty (alert prediction models) could be built that identify patterns on a time-frequency scale and in space (location of sensor). Also, major world events, sporting events, and national holidays could be used as triggers for scheduling additional analysts. Shift scheduling was not performed, and analysts were assumed to have non-overlapping shifts. However, alert generation rate can change from time to time. Consequently, more analysts are needed at certain hours of the day. In such a case, overlapping shifts with different shift lengths can provide the required number of analysts to meet the hourly demand using shift-scheduling algorithms [Pinedo 2009]. The sensor-to-analyst allocation model in this article is dynamic to the extent that it is based on an estimated workload for the next day. Consequently, a dynamic workforce may or may not be called. Once fixed at the end of the day, the sensor-to-analyst allocation model is static for the rest of the next day and does not adapt to changes in alert generation rates within a shift due to unforeseen reasons such as absenteeism of analysts or excessive workload from an intrusion in one or many sensors. A new model for within-shift reallocation of sensors to analysts is required to address variations in alert generation between sensors in a shift. The model will adapt to changing demands and ensure that the optimal workforce is maintained from hour to hour on a daily basis. The above extensions combined with the article's dynamic (on-call) workforce component will further increase the efficiency of the cybersecurity workforce to minimize the overall risk from threats, which will provide the maximum readiness capability to the cyber-defense organization.

ELECTRONIC APPENDIX

The electronic appendix for this article can be accessed in the ACM Digital Library.

ACKNOWLEDGMENTS

The authors thank Dr. Cliff Wang of the Army Research Office for suggesting this problem to us. Also, we thank Dr. Satinder Baveja and Dr. Kareem Amin for their comments that improved the clarity of the article.

REFERENCES

M. Albanese, C. Molinaro, F. Persia, A. Picariello, and V. S. Subrahmanian. 2014. Discovering the top-k unexplained sequences in time-stamped observation data. *IEEE Trans. Knowl. Data Eng.* 26, 3 (2014), 577–594.

J. P. Anderson. 1980. *Computer Security Threat Monitoring and Surveillance*. Technical Report. James P. Anderson Co., Fort Washington, PA.

M. E. Aydin and E. Oztemel. 2000. Dynamic job-shop scheduling using reinforcement learning agents. *Robot. Autonom. Syst.* 33, 2 (2000), 169–178.

Daniel Barbara and Sushil Jajodia (Eds.). 2002. *Application of Data Mining in Computer Security*. Advances in Information Security, Vol. 6. Springer, Berlin.

R. Bellman. 1957. *Dynamic Programming*. Princeton University Press, Princeton NJ.

Der-San Chen, Robert G. Batson, and Yu Dang. 2010. *Applied Integer Programming*. Wiley, New York, NY.

CIO Chief Information Officer. 2008. *DON Cyber Crime Handbook*. Dept. of Navy, Washington, DC.

Dorothy E. Denning. 1986. An intrusion-detection model. In *Proceedings of IEEE Symposium on Security and Privacy*. Oakland, CA, 118–131.

Dorothy E. Denning. 1987. An intrusion-detection model. *IEEE Trans. Software Eng.* 13, 2 (1987), 222–232.

Roberto Di Pietro and Luigi V. Mancini (Eds.). 2008. *Intrusion Detection Systems*. Advances in Information Security, Vol. 38. Springer, Berlin.

Robert F. Erbacher and Steve E. Hutchinson. 2012. Extending case-based reasoning to network alert reporting. In *2012 ASE International Conference on Cyber Security*. 187–194.

Rajesh Ganesan, Sushil Jajodia, and Hasan Cam. 2015. Optimal scheduling of cybersecurity analyst for minimizing risk. *ACM Transactions on Intelligent Systems and Technology (under review)* (2015).

A. Gosavi. 2003. *Simulation Based Optimization: Parametric Optimization Techniques and Reinforcement Learning*. Kluwer Academic, Norwell, MA.

D. Lesaint, C. Voudouris, N. Azarmi, I. Alletson, and B. Laithwaite. 2003. Field workforce scheduling. *BT Technol. J.* 21, 4 (2003), 23–26.

George L. Nemhauser and Laurence A. Wolsey. 1999. *Integer and Combinatorial Optimization*. Wiley-Interscience, New York, NY.

Y. Nobert and J. Roy. 1998. Freight handling personnel scheduling at air cargo terminals. *Transport. Sci.* 32, 3 (1998), 295–301.

Stephen Northcutt and Judy Novak. 2002. *Network Intrusion Detection, 3rd Edition*. New Riders Publishing, Thousand Oaks, CA.

M. Ovelgonne, V. S. Subrahmanian, T. Dumitras, and A. Prakash. 2015. *Global Cyber-Vulnerability Report*. Springer, Berlin.

C. D. Paternina-Arboleda and T. K. Das. 2005. A multi-agent reinforcement learning approach to obtaining dynamic control policies for stochastic lot scheduling problem. *Simul. Model. Pract. Theor.* 13, 5 (2005), 389–406.

Vern Paxson. 1999. Bro: A system for detecting network intruders in real-time. *Comput. Networks* 31, 23–24 (1999), 2435–2463.

Michael Pinedo. 2009. *Planning and Scheduling in Manufacturing and Services*. Springer, New York, NY.

W.B. Powell. 2007. *Approximate Dynamic Programming: Solving the Curses of Dimensionality*. Wiley-Interscience, New York, NY.

M. L. Puterman. 1994. *Markov Decision Processes*. Wiley Interscience, New York, NY.

J. Reis and N. Mamede. 2002. *Multi-Agent Dynamic Scheduling and Re-Scheduling with Global Temporal Constraints*. Kluwer Academic Publishers, Amsterdam.

Robin Sommer and Vern Paxson. 2010. Outside the closed world: On using machine learning for network intrusion detection. In *Proceedings of IEEE Symposium on Security and Privacy*. 305–316.

R. Sutton and A. G. Barto. 1998. In *Reinforcement Learning*. The MIT Press, Cambridge, MA.

El-Ghazali Talbi. 2009. *Metaheuristics*. Wiley-Interscience, New York, NY.

Wayne Winston. 2003. *Operations Research*. Cengage Learning, New York, NY.

W. Wonham. 1979. *Linear Multivariable Control: A Geometric Approach*. Faller-Verlag.

F. Zhou, J. Wang, J. Wang, and J. Jonrinaldi. 2012. A dynamic rescheduling model with multi-agent system and its solution method. *J. Mech. Eng.* 58, 2 (2012), 81–92.

Carson Zimmerman. 2014. *The Strategies of a World-Class Cybersecurity Operations Center*. The MITRE Corporation, McLean, VA.

Received September 2015; revised November 2015; accepted January 2016