Balancing Security and Performance for Agility in Dynamic Threat Environments

***Abstract*—In cyber security, achieving the desired balance between system security and system performance in dynamic threat environments is a long-standing open challenge for cyber defenders. Typically an increase in system security comes at the price of decreased system performance, and vice versa, easily resulting in systems that are misaligned to operator specified requirements for system security and performance as the threat environment evolves. We develop an online, reinforcement learning based methodology to automatically discover and maintain desired operating postures in security-performance space even as the threat environment changes. We demonstrate the utility of our approach and discover parameters enabling an agile response to a dynamic adversary in a simulated security game involving prototype cyber moving target defenses. Our results establish a foundation on which future implementations of online adaptive control architectures in cyber security can build, and provide guidance on optimal parameter settings for enabling agile responses by these systems.**

I. Introduction

Network operators desire the ability to deploy new defensive technologies without impacting system performance. In reality, technological defensive mitigations typically have negative impacts on system performance. Achieving the desired balance between system security and system performance in dynamic threat environments is a long-standing open challenge for cyber security practitioners. The task of finding the optimal balance is a difficult one for network defenders, and involves subtasks such as making correct inferences from past historical experience, balancing complex requirements and criteria against each other, and accounting for a threat environment that changes moment to moment and reacts adaptively to defensive strategies and tactics. These tasks are beyond the scope of possibility for the majority of cyber defenders. Cascading consequences and path dependence in the decision space mean human operators cannot process enough of the relevant information on their own to come to optimal decisions and so most security decisions are made on a *ad hoc* basis, without the help of a coherent risk framework or rigorous decision criteria.

Automated decision making is needed both to place the security-related decisions of network defenders on a more rigorous and quantitative foundation and to keep pace with rapidly evolving adversaries. At a high level, it is typically the case that an increase in system security comes at the price of decreased system performance, and vice versa. This has the consequence that systems can easily become misaligned to the desired levels of security and performance as the threat environment evolves.

In recent years the need for frameworks and techniques to assist operators in balancing security gains against performance costs in information security has been recognized and researchers have begun addressing the challenge. A framework for adaptively balancing these tradeoffs for combinations of security services in wireless ad hoc and sensor networks was proposed in [1]. In [2] a framework for balancing security via encryption and network throughput in a wireless network environment is proposed and demonstrated both with and without modeled adversaries. A technique for achieving balance between security and performance in networked control systems based on a coevolutionary genetic algorithm technique has been developed in [3], [4]. The security-versus-performance challenge was addressed for robotic mobile wireless ad hoc networks in [5] using novel combinations of Petri nets and queueing networks. We propose a strictly online, light-weight, and conceptually simple adaptive control framework to automatically balance security gains against performance losses and demonstrate the proposed technique in the context of a strategic game of timing.

Our formulated strategic game captures several important aspects of the real world challenges cyber defenders face in optimizing system operations in the face of security challenges. Our game of timing reflects the essential logic of a large class of security scenarios in which security must be traded against system performance. We map our abstract game to a real-world prototype *Moving Target* technology [6], [7] that requires an explicit decision be made regarding the degree to which system security or performance will take precedence in system operations. Our management system leverages online, model-free technique to automatically discover optimal or near-optimal system operating points that balance security and performance. The major contributions of this work are as follows:

1) We develop an online, reinforcement learning based methodology to automatically discover and maintain desired operating postures in security-performance space.
2) We develop the *implicit-efficient attacker* model, a threat modeling framework that renders the modeling of sophisticated attackers tractable while capturing the attacker attributes essential to defender success.
3) We discover model parameters enabling agile responses to dynamic adversaries.
4) We provide a foundation on which future implementations of online adaptive control architectures in cyber security can build.

In the next section we develop our framework for reasoning about security decisions in the light of performance considerations and develop a strategic game scenario capturing challenges real world network defenders face in solving the security versus performance dilemma. In Section III we describe our attacker model and the metrics used to evaluate security

and performance. In Section IV we provide implementation details of our reinforcement learning solution methodology that achieves dynamic balance between desired security and performance levels in our security game scenario. In Section V we provide a unified overview of the system architecture implementing our reinforcement learning solution, followed by a discussion of experiment setup and model parameterization in Section VI. A series of simulation results are also presented and discussed, followed by concluding remarks in Section VII.

II. Strategic Game Scenario and Term Definition

We are interested in developing an automated technique to balance security and system performance in systems where the defender must actively elect to take a security enhancing action at the cost of immediate degradation of system performance. We develop notation here to capture this situation formally in a manner that can later be implemented in a computational engine for optimization and exploration. We define a *benefit* term B that captures the advantage a defender gains through achieving increased system security as a consequence of the security-enhancing action they have elected to undertake. We let a *latency* term L capture the performance costs that are incurred by the defended system as a consequence of the security-enhancing action. The benefit and latency terms are combined algebraically into an overall *reward* term R, expressed as,

$$R = B - L. \tag{1}$$

The objective of the defender is to discover a policy that will optimize the reward term R, which can be achieved by maximizing the benefit gained B while minimizing the latency cost L.

A. Game Scenario

The strategic scenario we study is inspired by a known attack scenario involving a computer's memory space. Address Space Layout Randomization (ASLR) [8] is a cyber moving target technique that has been adopted widely in modern operating systems [6]. In this technique a program is placed randomly in memory such that an attacker cannot predict the location of key components of a program's data. This protects an application from attacks that rely on detailed knowledge by the attacker of an applications memory space, such as buffer overflow and return oriented programming (ROP) attacks [6].

A class of attacks have recently received attention that exploit memory disclosures from a host system to overcome its ASLR defense. A *memory disclosure* occurs when a valid, active memory address is leaked from a system. Memory disclosures allow an attacker to re-map a program's memory space and thus circumvent ASLR obscurification. Once ASLR has been circumvented and a system compromised, the defender likely must engage in costly mitigation and restoration steps to return the system to a secure state.

Preventing all potentially damaging memory disclosures can be a daunting task. An alternative is to develop techniques to mitigate memory disclosures before they can be exploited by an attacker. This was recently demonstrated in [7], where the authors dynamically re-randomized system memory layout each time a system input followed system outputs. This scheme makes the strong assumption that all system outputs are potential memory disclosures on which an adversary may capitalize.

In reality, the vast majority of system outputs do not contain memory disclosures. Consequently there is an opportunity to improve performance by scanning system outputs for the presence of active, valid memory addresses. If a valid address is discovered in a system output, the mitigating step of re-randomizing the memory layout would then be taken. However, the act of searching and subsequent mitigating actions incurs a latency cost, which must be balanced against performance needs.

B. Latency Costs

We now derive the general conditions under which a strategy of searching system outputs for memory disclosures is sound policy for the defender. The simplest defender policy consists of taking mitigating steps each time there has been the possibility of a memory disclosure to the attacker, i.e., each time there has been a *write()* system call and a system output has been generated [7]. In many instances executing a policy of taking mitigation steps each time a system output is generated will incur a latency cost that exceeds the maximally allowed system latency, L_{tot}. In these instances the following inequality holds,

$$\sum_{i=1}^{O} L_{i,m} > L_{tot}, \tag{2}$$

with O the total number of system outputs during the period of interest, and $L_{i,m}$ the latency cost for mitigating memory disclosure i.

We seek to use a system output searching procedure with an associated latency cost ($L_{i,s}$) that is on average much less than the average latency cost of mitigating the memory disclosures,

$$\langle L_{i,s} \rangle_i \ll \langle L_{i,m} \rangle_i , \tag{3}$$

where $\langle\rangle_i$ is the average over i. If Eqt. 3 does not hold, then it would be preferable for the defender to simply mitigate the consequences of system compromise and dispense with searching system outputs.

We further note that, even given that Eqt. 3 holds, it can easily be the case that searching every system output for memory disclosures, and taking steps to mitigate the threat to the defended system that result from memory disclosures, incurs a latency cost that, while being less than the latency cost inherent in taking mitigation steps each time there is the possibility for memory disclosure, still exceeds the maximally desired system latency L_{tot},

$$\sum_{i=1}^{O} L_{i,m} > \sum_{j=1}^{O} (L_{j,s} + \delta L_{j,m}) > L_{tot}, \tag{4}$$

with δ set to 1 if the output search procedure discovers a memory disclosure, and 0 otherwise.

Given that the inequality in Eqt. 4 holds, our objective becomes the discovery of heuristics that selectively execute the searching of system outputs based on the past history of memory disclosure attacks such that,

$$\sum_{i=1}^{O} H\left(L_{i,s} + \delta L_{i,m}\right) < L_{tot}, \tag{5}$$

holds, with H representing the set of heuristic rules. The focus of this study is on the development of techniques for automated defender policy formulation.

III. Attacker and Defender Models

To keep the focus on defender policy development, we make simplifying assumptions with regards to the attacker model. We posit an *implicit attacker* model in which the effect of the attacker's actions are accounted for as opposed to the details of those actions. For the class of attacks we study here the effect of the attacker's actions is to cause memory addresses to be leaked from the system (i.e., memory disclosures). We use the term *implicit* because the effect explicitly modeled (i.e., memory disclosures) have implicit in them the actions of the attacker that led to those memory disclosures, though these detailed attacker actions are not included in our model.

We make an additional modeling assumption that all unmitigated memory disclosures harm the defender's security stance and advantage the attacker. We term this the *efficient attacker* assumption because no memory disclosure fails to harm the defender. Given the *efficient*, *implicit* attacker model we adopt here, the key property characterizing an attacker becomes the frequency and temporal clustering of memory disclosures caused by a given attack.

A. Security and Performance Metrics

We use simple, intuitive notions of *security* and *performance* to evaluate the defensive system's performance. We define *security* as the ratio of the number of memory disclosures that have been mitigated by the defender (Ω_{mit}), and the total number of memory disclosures (Ω_{tot}) caused by the *implicit* attacker's actions.

$$Security = \frac{\Omega_{mit}}{\Omega_{tot}}, \tag{6}$$

We characterize *performance* by the ratio of system outputs searched by the defender (O_s) to the total number of system outputs (O_{tot}) that have exited the system,

$$Performance = 1 - \frac{O_s}{O_{tot}}. \tag{7}$$

Our notion of security is defined with an implied assumption regarding the existence of an *oracle* that is able to inform the evaluation framework of the number of memory disclosures that have been missed up to the present round of attacker-defender interaction. This oracle construct is used strictly in the evaluation phase of our study. The defender's strategy evolution does not access oracle-class information on a policy's success. Q-function terms (see Section IV) that would capture this oracle-class information are explicitly excluded from the reinforcement learning framework.

We also note here that our definition of security and performance in this study assumes what might be termed an incrementalist approach to security, in which each memory disclosure that is not mitigated through re-randomization harms the defender's security stance by an equal quantitative amount. This can be contrasted with the notion of sudden, catastrophic security failure, in which a single attacker success leads to irrevocable compromise of the system's mission. While many examples of sudden, catastrophic attacks involving the neutralization of ASLR defenses can be easily imagined, a recent example of an incrementalist-style attack is the so called *Blind Return Oriented Programming* (BROP) attack outlined in [9], particularly the third phase of the BROP attack in which a binary is progressively dumped to the network (see [9] for details).

B. Model of Statistical Search

To detect memory disclosures in our model the defender inspects the payloads of system outputs for address signatures from a list of active, valid memory addresses that is maintained by the system. This task is accomplished via a sliding window technique in which 8-byte (16-hexadecimal character) segments of payload data are compared to the set of known active, valid memory addresses, which are already saved by the defensive system [7]. If the 8-byte payload segment matches a signature in the active address list then a memory disclosure has been discovered in the payload and mitigation steps are initiated to prevent exploitation of the leaked address.

While many techniques and strategies exist to accomplish payload inspection, an efficient means is to employ the *Bloom filter* probabilistic data structure [10]. In our model, a Bloom filter is programmed (i.e., populated) by generating a set of k hash values in the range $[1, ..., m]$ from each of n active system addresses. A vector of size m is initialized to null values. Each valid, active memory address is then hashed k times, and the corresponding location in the bit vector $[1, ..., m]$ is set to 1.

When the decision is taken to inspect a given system output, an 8-byte (16-character) window moves through the data byte-by-byte (skipping every other character) through the system output. Each 8-byte segment of payload data is hashed using hash functions identical to those used for programming the Bloom filter. If a 1 is discovered in every location queried, then the payload data segment being inspected is deemed to be an active system address (i.e., a memory disclosure). If, on the other hand, a queried vector entry is 0, the data segment is deemed not to contain a memory disclosure.

Strengths of a Bloom filter implementation of the search algorithm include a compact representation of the data set to be queried, and query times independent of the size of the signature set used to program the Bloom filter [11]. The source of these strengths is the simple representation of the data made possible by the hashing operation. Inherent in the compact data representation at the heart of a Bloom filter, however,

also comes a non-zero probability of *false positives*, given approximately by the expression [11],

$$P_{FP} = (1 - e^{-nk/m})^k. \tag{8}$$

Here P_{FP} is the probability of obtaining a false positive indication in a given query of the Bloom filter, n is the number of elements used to program the Bloom filter, k is the number of hash functions used in the Bloom filter, and m is the size of the Bloom filter's bit vector.

For a given n and m, the probability of obtaining a false positive result from querying the Bloom filter is minimized when

$$k = \frac{m}{n} ln2 \tag{9}$$

hash functions are used in the Bloom filter [11]. We use these expressions to parameterize our model of system output searching.

IV. Reinforcement Learning

The intelligent, adaptive capability of our proposed control architecture is accomplished via an agent architecture that leverages machine learning techniques. These methods allow the defender's control system to automatically adjust its operations to counter attacker attempts at compromise while considering the performance ramifications of policy choices. Many machine learning techniques rely on the existence on labeled examples to learn optimal mappings between inputs and outputs. This learning paradigm works well in many applications, but encounters difficulty in strategic situations of even moderate complexity where the decision space quickly grows to the point that any attempt by a human to label the good and bad moves by hand becomes infeasible [12]. The class of learning algorithms that attempt to circumvent this bottleneck by enabling agents to learn optimal courses of action, or policies, through interaction with an unknown environment that answers an agent's actions with rewards and punishments are termed *reinforcement learning* techniques.

Reinforcement learning [13] grew out of work on trial and error learning [14], [15] and optimal control [16] and today compromises a set of important and rapidly developing machine learning techniques that are used across a number of industries and academic fields [17]. Reinforcement learning comprises a class of machine learning algorithms that allow an agent to learn online and incrementally, proceeding without perfect knowledge of the environment or the reward structure.

A. Q-Learning

Q-learning [18] is a reinforcement learning technique that operates without needing a model of the environment. The goal of Q-learning is for the agent to discover a *policy*, π, giving an optimal mapping,

$$\pi : S \rightarrow A, \tag{10}$$

between states (S), and actions (A).

As the Q-learning algorithm proceeds a set of action-value, or Q, functions are incrementally learned. The Q values represent the utility an agent expects to achieve by taking the specified action in a given state and following an optimal policy afterwards. The Q values are initialized arbitrarily at the beginning of the learning process, and thereafter updated according to the value-iteration update equation,

$$\Delta Q(s,a) = \alpha[R + \gamma \, max_b Q(s',b) - Q(s,a)]. \tag{11}$$

Here a is the action taken in state s, and b is the optimal action taken from future state s'. The learning rate, $0 \leq \alpha \leq 1$, determines how quickly the decision rules are updated based on the latest information. The discount factor, $0 \leq \gamma < 1$, weights the importance of future rewards, with low values causing the Q-learning agent to focus on obtaining immediate payoffs, and high values of γ encouraging behavior to optimize future reward.

In Q-learning the agent estimates reward $Q(s,a)$ based on the estimate of future rewards $Q(s',b)$. This method has the advantage of allowing the Q-learning agent to estimate future rewards without having a knowledge of actual future actions and rewards/punishments [17].

In our Q-learning instantiation we use the reward function R to address the cost/benefit problem the defender must solve in devising a maximally effective set of search control heuristics. The reward R_i for inspecting a given system output i for memory disclosure is given by,

$$R_i = B_i - (L_{i,s} + L_{i,m}) \tag{12}$$

with B_i the benefit the defender gets from discovering a memory disclosure in system output i, $L_{i,s}$ the latency cost the defender pays for searching a system output i for memory disclosures, and $L_{i,m}$ the latency cost the defender pays for mitigating a memory disclosure in payload i once if it has been discovered. We note that R_i is the instantaneous feedback the defender receives due to the chosen action on system output i, while Q is the expected discounted reward the defender forecasts for a given course of action in the current round given that an optimal policy is followed afterwards.

We note that the *benefit term* B_i can be interpreted in two manners. In the first, B_i represents an empirically derived measure of inverse risk to the defended system of the attacker obtaining a memory disclosure from the system. The second interpretation of B_i is as a tuning parameter the system operator can use to tune the control architecture to a *security focused* or a *performance focused* posture. We adopt this later view of B throughout the present study.

In the implementation of Q-learning in this study the defender has access to the past 3 decisions that have been made, and whether or not the decision to inspect a system output (if such a decision has occurred in the previous 3 rounds of play) has resulted in a memory disclosure detection. If the defender has chosen not to inspect the system output in a given round, the defender cannot know whether or not the passed over system output contained a memory disclosure, though this information will be reflected in the *security* score, of which the

defender is not aware. The 3 step memory horizon comprises the state space (S) for our Q-learning defender agent. A single state in this space is given by,

$$s_1^n = \{(a_{n-1}, r_{n-1}), (a_{n-2}, r_{n-2}), (a_{n-3}, r_{n-3})\}, \quad (13)$$

with a the action on the $n - x$ ($x \in 1, 2, 3$) step, and r the result (i.e., discovery or no-discovery of a memory disclosure in the system output's payload) on the same step.

The set of all possible actions and consequent results makes up the state space S,

$$S = \{s_1, s_2, ...s_{27}\} \quad (14)$$

We note that the restriction that the Q-learning agent know only of the presence or absence of memory disclosures in payloads which the agent chooses to inspect reduces the state space from 64 to 27 states.

The *action space* for the defender (A^D) consists of the decision to inspect, and the decision to pass on a system output.

$$A^D = \{Inspect, Pass\} \quad (15)$$

The policy (π) is the set of learned heuristics H over the 3-step memory horizon,

$$\pi = H_n\left((a_{n-1}, r_{n-1}), (a_{n-2}, r_{n-2}), (a_{n-3}, r_{n-3})\right). \quad (16)$$

An adaptive agent in unknown or changing environments faces the *exploration-exploitation* dilemma [13]. An agent might choose to greedily exploit their understanding of the environment and take the action a^s expected to maximize future reward,

$$Q(a^s) = max_a Q(a). \quad (17)$$

Alternatively, an agent might choose to *explore* the environment, ignoring what it has learned of its environment in the past and choosing an action randomly from those available to it, observing the reward or punishment that results and adjusting its decision rules accordingly. Some adaptation techniques implicitly account for this tradeoff, for example the *mutation rate* in a genetic algorithm [19] drives the modeled population to greater exploration when set to a high value. The Q-learning algorithm makes the exploration/exploitation tradeoff explicit for each individual learning agent by augmenting the basic Q-update cycle with a scheme for allowing the agent to make exploration moves at some rate during its interaction with the environment.

In this study we address the *action-selection* dilemma by employing the ϵ-greedy scheme [13] in which agent's perform random, non-optimal exploration actions with a probability ϵ ($0 \leq \epsilon \leq 1$), and act greedily with probability $(1 - \epsilon)$. Unless otherwise indicated, at the start of a model run, ϵ is set to unity, and is then decreased uniformly on each subsequent step such that ϵ reaches zero on the final simulation step, thus maintaining at least a small probability of exploration throughout the model runs. By initializing ϵ to unity and then decreasing its value as the simulation proceeds we cause the agent to explore more often early on when the optimal policy is still unknown, while still acting according to the optimal policy with high probability later in the model run when the optimal policy is more certain.

V. System Architecture for Balancing Security and Performance

The system components described above are composed into the overall architecture for balancing security and performance depicted in Fig. 1. This architecture consists of a *decision* module (represented by the blue diamond in Fig. 1) that instantiates a set of inspection-decision heuristics for inspecting system outputs according to the estimated probability that a system output contains a memory disclosure given the recent history of disclosures. System outputs stream through the decision module to the inspection module, which either inspects the system output or lets it pass uninspected depending on the recommendation of the decision module. If the search is executed and a memory disclosure is discovered, the mitigation step is activated (indicated by the *green* box in Fig. 1).

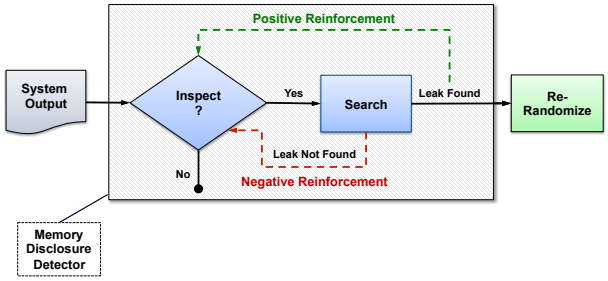

Fig. 1. Memory disclosure mitigation architecture for automatically discovering defensive policies that balance system *security* and system *performance*.

This architecture is general, and can be applied to problems across a broad spectrum of application areas where security must be balanced against system performance. The particular details of a given system enter the architecture in the form of the latency costs, $L_{i,s}$ and $L_{i,m}$, that paraemterize the reward function R that determines the feedback (negative or positive) to the *inspection* module and causes adaptation of the learned heuristics to the latest observed attacker behavior.

The human operator exerts a measure of control over the functioning of the architecture by selecting a value for the benefit term B_i in the reward function R_i. The value of B_i relative to the latency terms ($L_{i,s}$ and $L_{i,m}$) determines the characteristic behavior, what we term *bias*, of the system. For convenience, we define the *normalized benefit*, B_N, to be,

$$B_N = \frac{B}{L_s + L_m}, \quad (18)$$

the ratio of the user determined *benefit* term and the system-determined *latency* terms.

VI. Simulation Experiments

A. Model Parameterization

In order to quantitatively assess our framework for automatically balancing system security and performance we develop a scenario involving the specific cyber moving target technology known by the acronym TASR (Timely Address Space Randomization) [7]. The TASR technology mitigates memory leaks from the defended system by re-randomizing system memory at runtime. The default operation mode of TASR is to re-randomize memory just before processing an input system call after there has been a system output generated by the defended system [7]. For parsimony, we do not model input system calls, so TASR's default operation mode corresponds in our model to memory re-randomization after each system output exists the system.

The TASR technology provides a convenient example of a technology that executes a potentially costly mitigation in response to a specific threat scenario that would potentially benefit from adaptively predictive techniques to achieve security goals without unduly impacting system performance. In what follows, we use the TASR prototype as a surrogate system to derive realistic model parameters when they are needed to enable quantitatively evaluation of our framework's performance.

Our stand-in for mitigation in this study is the TASR prototype's runtime memory re-randomization, described in detail in [7]. The performance of the C-based programs in the SPEC CPU2006 benchmark with and without the TASR prototype in operation were studied and we have evaluated the execution time increases as measured in absolute time in the 10 programs examined. We normalized these times by the number of re-randomizations executed by the TASR prototype in the examined runs, and this analysis resulted in the *time per re-randomization* event for each of the studied programs in the SPEC benchmark. The median of this set is 0.1 seconds. We use this median number as our value for the latency cost of mitigation, $L_{i,m}$.

To generate a plausible set of parameters for representing the system output search task in our model we analyzed a sampling of system outputs from the Iceweasel [20] web browser application using a custom-built data capture tool [21]. This data sample gives a mean of 221 hexadecimal characters for the average system output payload size.

For the Iceweasel data sample we also obtained the list of valid, active system addresses for analysis. We note that the list of valid, active memory addresses is updated periodically throughout the application's operation. We ignore this dynamic factor in our search model and assume that a Bloom filter has been programmed prior to the start of the simulation, and further that the valid list of active address signatures does not change as the simulation proceeds. We calculate our n (i.e., number of address signatures used to program the Bloom filter) and m (i.e., size of Bloom filter in memory) values at the start of the data sample and hold these constant throughout our simulation.

In this formulation the operator selects an acceptable value for P_{FP}, taking into consideration the tradeoff between P_{FP} and k under the assumption of optimal k value, shown in Fig. 2. Unless otherwise indicated, we set the P_{FP} parameter to a value of 0.1 throughout this study, implying the use of $k = 4$.

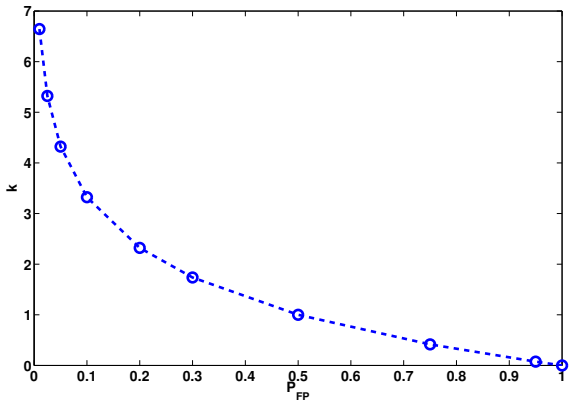

Fig. 2. Scaling of the number of hash functions (k) with the probability of false positives (P_{FP}) for a bloom filter under optimal assumptions.

Using 221 as the representatative system output size, we model the operation of the sliding window as proceeding byte-by-byte (i.e., sliding forward by two hexadecimal characters on each window advance through the data), and neglecting the final 7 bytes, as a valid system address must be made up from 8 bytes. This calculation gives 104 Bloom filter queries per system output inspected. Further, we assume our system is instantiated on a workstation with time duration per operation of $100\,ns$, a reasonable performance number for a standard PC workstation.

We then calculate $L_{i,s}$, the latency cost of searching a single system output i for the presence of memory disclosure as,

$$L_{i,s} = k * Z * F, \tag{19}$$

with $k = 4$ the number of hash functions used in the Bloom filter, which gives the scaling performance of the Bloom filter algorithm, $Z = 100\,ns/op$ the time duration per operation on the hypothetical system, and $F = 100$ queries/output the characteristic number of search queries that must be performed per system output payload inspected to scan the entire data payload for memory disclosures. Together these values give $L_{i,s} = 0.04\,ms$, holding $P_{FP} = 0.1$ throughout the remainder of this study unless otherwise noted.

To explore the sensitivity of model outputs to the value of the learning rate (α) and and discount factor (γ) we performed a factorial sweep over possible values for α and γ, the output of which is displayed in Fig. 3 in terms of simulated system security. We observe from Fig. 3 that for α values less than approximately 0.4, the value of the discount factor (γ) has minimal effect on overall security score obtained by the defender. As the learning rate (α) is increased beyond 0.4 the discount factor becomes increasingly important in overall system performance, with γ values between approximately 0.6 and 0.8 partially compensating for suboptimal tendencies in learned defender policy caused by a high α value. We adopt the values of $\alpha = 0.2$ and $\gamma = 0.2$ throughout the remainder of this study.

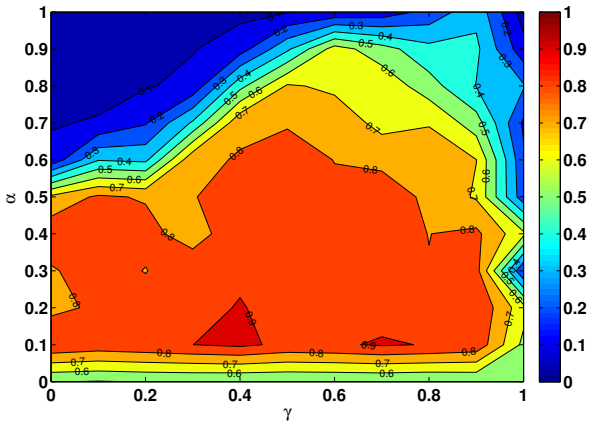

Fig. 3. System *security*, (Eqt. 6) as a function of Q-learning parameters α and γ. Color indicates the level of security achieved for a given value of α and γ, with redder colors indicating a higher level of security achieved. The mapping of colors to numerical results is given to the right of the figure.

Metrics quantifying the performance of the defender's learned policy, including *security* and *performance*, are reset periodically to facilitate a more accurate and timely characterization of the policy's current performance against the dynamic attacker. We select our chosen window size based on an analysis of the effect window size has on the stability of system effectiveness metrics across simulation runs.

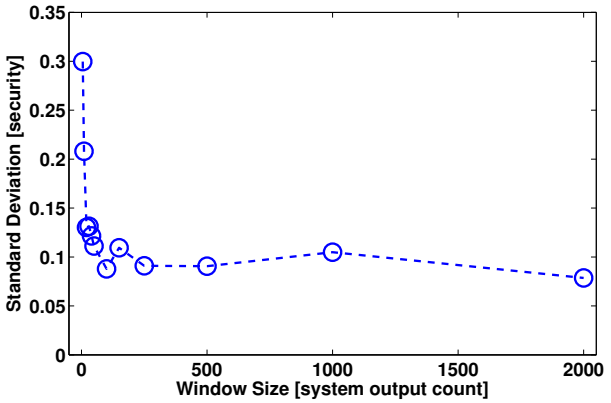

Fig. 4. Standard deviation of security as a function of metric window size. The displayed results are averages taken over 100 replicate runs of the simulation with a (randomized) attack strength of 0.50.

Figure 4 shows the standard deviation of the mean security score recorded on the final step of 100 replicate simulation runs. From these and other similar results we select a window size of 200 simulation steps for the resetting of system effectiveness metrics.

Table I summarizes the model parameter settings used throughout the simulation studies detailed in the next section.

B. Results

The purpose of our studies here are to begin to characterize the behavior of the proposed Q-learning architecture and provide preliminary guidance for users as to desirable model parameter values in various contexts. Due to space constraints, a complete characterization of model behavior is beyond the scope of the current work, but will be a topic that is revisited in future works.

TABLE I. SUMMARY OF BASELINE MODEL PARAMETER VALUES EMPLOYED THROUGHOUT THIS STUDY UNLESS OTHERWISE INDICATED IN THE TEXT.

Parameter	Meaning	Value
W	Metric window duration	200 model steps
P_{FN}	Probability of false positives	0.1
$L_{i,m}$	Latency cost of mitigation	101 ms
$L_{i,s}$	Latency cost of output search	0.04 ms
α	Q-learning learning rate	0.20
γ	Q-learning discount factor	0.20
D	Simulation duration (steps)	2000
C	Exploration decay constant	2000

We characterize the behavior of our Q-learning architecture when facing an attacker that causes memory disclosures uniformly at random at a fixed overall rate. A given attack strategy is characterized by a single parameter, A_p, that takes on values between 0 and 1 and characterizes the probability that a memory disclosure will be contained within any given system output. For efficiency, a single system output is produced on each simulation step. Simulations are run to 2000 steps, simulating 2000 system outputs for which the defender must decide to inspect or pass. Each attacker parameterization faces the defender over 100 repeated encounters (i.e., 100 replicate runs) of 2000 simulation steps each. Summary statistics characterizing defender success are collected and aggregated over each of the replicate runs.

Figure 5 shows the final security (blue) and performance (red) scores averaged over 100 replicate runs for a defender facing the attacker described above with $A_p = 0.50$. The x-axis in Fig. 5 gives the value of the *normalized benefit* term, B_N (defined in Eqt. 18), parameterizing the adaptive defense. For small *benefit* values the system obtains high performance scores and low security scores. On the other hand, large *benefit* values lead to high security scores by the defender and low performance scores.

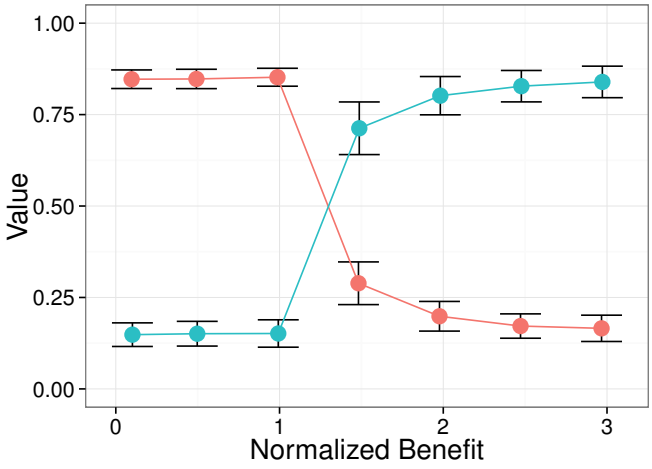

Fig. 5. Security (blue) and Performance (red) scores for the Q-learning architecture facing the $A_p = 0.50$ attacker. Markers denote mean final values of security and performance collected over 100 replicate runs. Vertical lines denote the standard deviation to the mean values.

In Fig. 5 the system undergoes a relatively abrupt transition from a focus on performance to a focus on security around $B_N = 1$. We note that this is the point that we have,

$$B_i \approx (L_{i,m} + L_{i,s}). \tag{20}$$

The observed transition implies that under the assumed attack strength we have two regimes of system behavior. When,

$$B_i < L_{i,m} + L_{i,s}, \tag{21}$$

the system is tuned for optimizing performance, while when,

$$B_i > L_{i,m} + L_{i,s} \tag{22}$$

the system is tuned for optimizing on system security.

Figure 6 shows the standard deviation of the mean security score as a function of the normalized benefit, B_N. A large relative increase in the standard deviation is evident in the transition region between the *security* and *performance*-focused regimes. Large variance at a relatively sharp transition between qualitatively distinctive behavioral regimes is a property often observed in both human and natural systems undergoing phase transitions [22]. This divergence in variance often entails an amplified agility on the part of systems poised on the edge of transitions between regimes of qualitatively distinctive behavior. It is our hypothesis that such agility can be harnessed in cyber control systems to counter dynamic, evolving threats. To investigate this hypothesis we study the distinctive characteristics of our Q-learning architecture poised between the security- and performance-biased regimes.

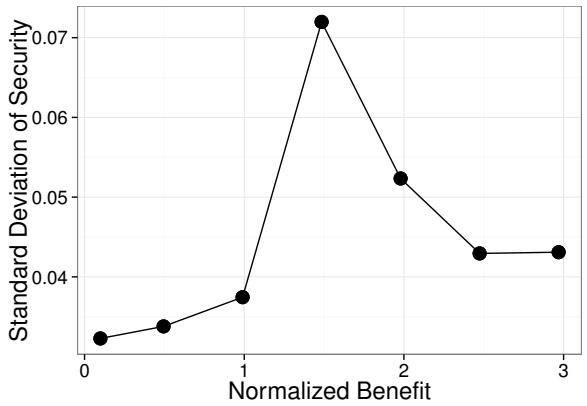

Fig. 6. The standard deviation of *security* exhibits a large increase in the vicinity of the transition between the security and performance focused security regimes

We instantiate 3 classes of Q-learning defender and study the performance of their learned defense policies. The first defender class, termed the *performance-biased* defender, is given a low relative reward value of $B = 25$. A second class of defender is given a large reward value, $B = 1000$, strongly biasing the defender to preferentially discover defensive policies that achieve high security. This is the *security-biased* defender. The third class of defender is positioned in the *transition* region by setting $B = 130$. We refer to this defender as the *critically-balanced* defender in analogy with the common terminology used in the physical sciences to describe systems undergoing phase transitions as *critical* systems.

The 3 classes of defender are made to face 3 severity levels of attack. Each defender faces a weak attack ($A_p = 0.10$), a medium-strength attack (($A_p = 0.50$), and a strong attack (($A_p = 0.90$). Results from theses studies are displayed in Fig. 7, with the strength of the attacker varied left to right (from weak to strong attacker instantiation), and the bias of the defensive architecture being varied from top to bottom (performance biased on top, security biased on bottom). Each marker in Fig. 7 indicates the final security and performance score achieved in a single replicate run of the model.

Looking across the top row in Fig. 7 we find that the *performance*-biased defender responds to all attack types with a policy that generates high levels of system performance but low levels of system security. In contrast, the *security*-biased defender, results from which are plotted along the bottom row of Fig. 7, consistently achieves high levels of system security against the *medium* and *strong* attacker variants, while achieving a mixed, but relatively high, level of system security when facing the *weak* attacker.

Opposed to both of these relatively uniform responses to the different attacker strengths, the *critical* defender achieves a largely performance focused result when facing the weak attacker, a security-performance balanced result when facing the medium strength attacker, and a high security response when facing the strong attacker.

Figure 8 provides an in-depth look at the response of the different defender types to the 3 classes of attacker. Figure 8 depicts the mean response (height of bar) of each defender type (x-axis) when facing each attacker strength (color). Error bars in Fig. 8 denote one standard deviation from the mean. The depicted values correspond to defender behavior during the final windowing period of 200 simulation steps, capturing the learned policy of each defender type during the final simulated period.

Figure 8 (left) shows that the performance-biased defender consistently inspects a low number of system outputs, while the security-focused defender consistently inspects a high number of system outputs. In contrast to these rigid behaviors, the critical-defender inspects a small number of packets when facing the weak attacker, a medium number of packets when facing the medium-strength-attacker, and a large number of system outputs when facing the strong attacker.

It is noteworthy that the *critical* defender learns to inspect a number of system outputs that nearly exactly matches the number of memory disclosures generated by the attacker on average in a given scenario. The $A_p = 0.10$ attacker causes 20 leaked addresses on average per 200 system outputs, to which the critical defender learns to inspect 20.85 system outputs on average per 200 simulation steps, or an inspection ratio of 0.104, very near to the 0.10 attack value. Contrast this with 9.74 inspected system outputs by the performance focused defender, and the 105.27 inspected packets of the securtiy-focused defender (an order of magnitude increase in inspections). A similar story holds for the responses to the *medium* and *strong* defenders.

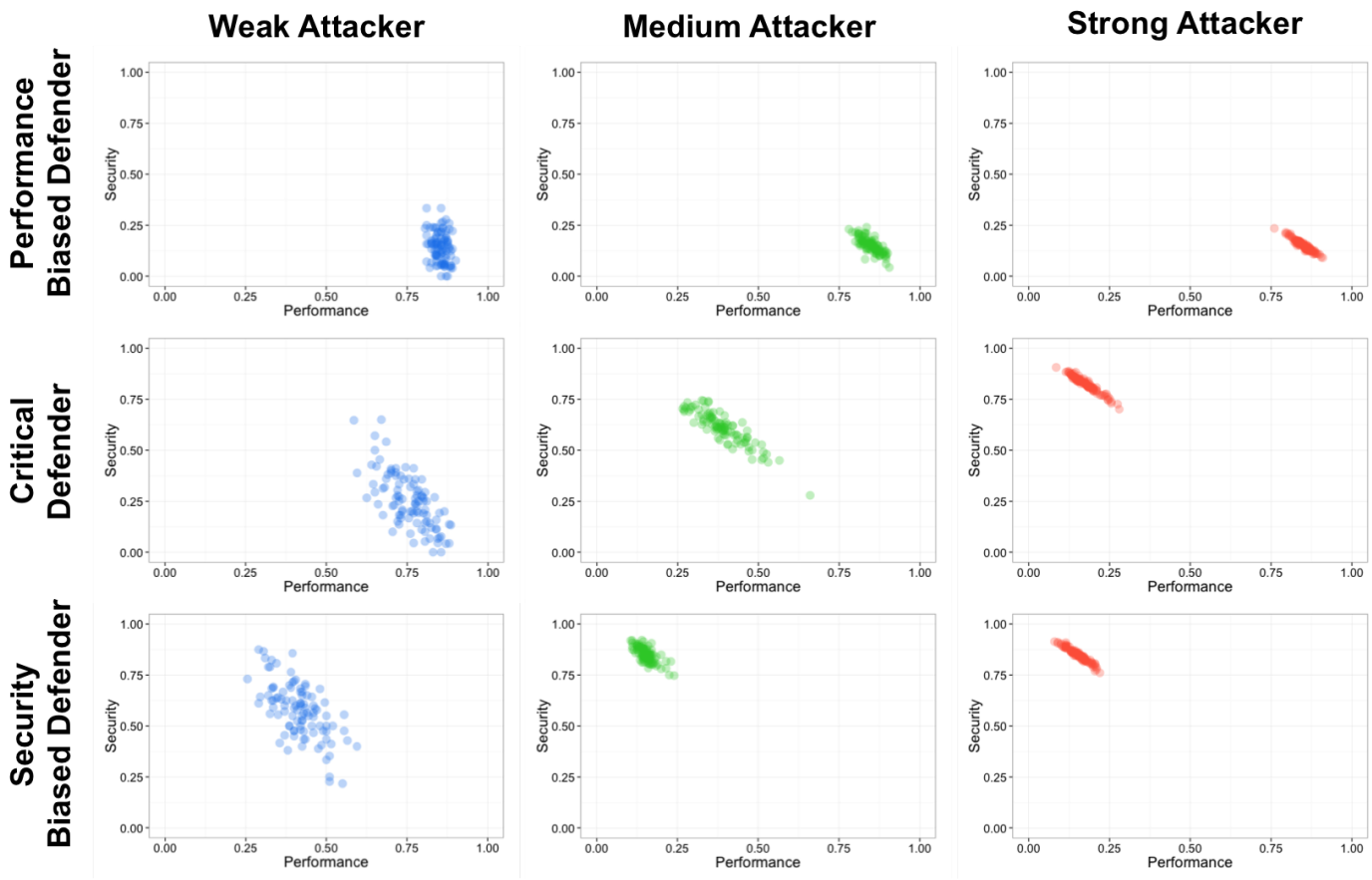

Fig. 7. Markers denote the value of *security* and *performance* obtained at the end of 100 independent replicate runs for performance ($B = 25$), critical ($B = 130$), and security ($B = 1000$) biased defenders facing a weak ($A_p = 0.10$), medium ($A_p = 0.50$), and strong ($A_p = 0.90$) attacker. Attacker strength is increased left to right, while the bias of the defensive architecture is varied top to bottom.

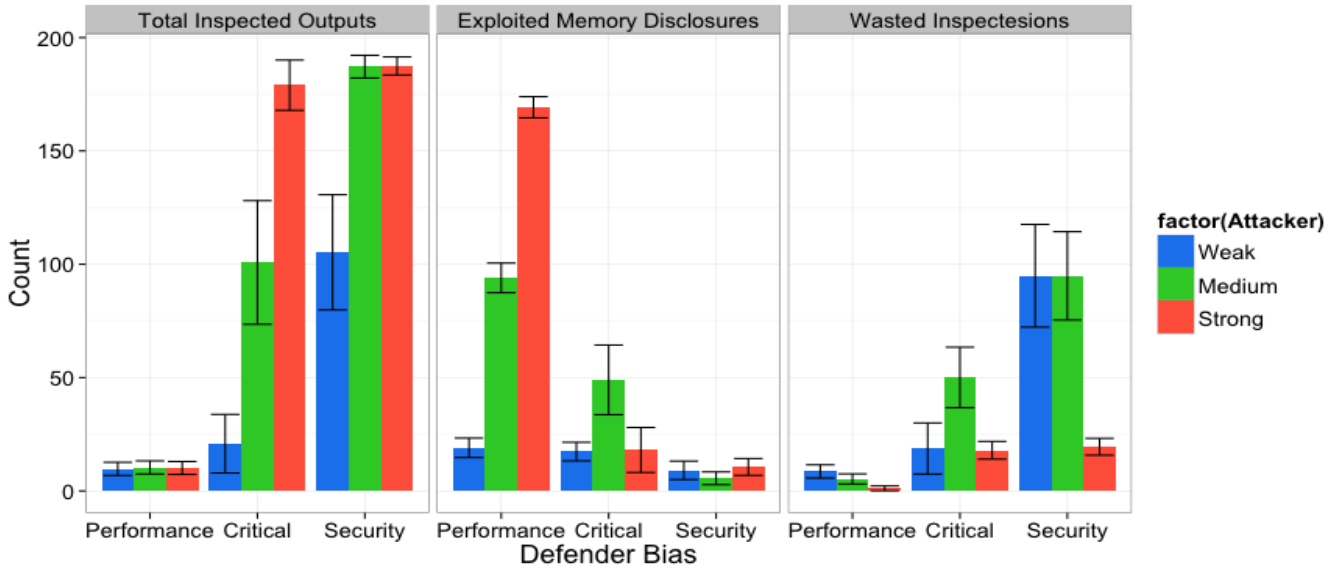

Fig. 8. Mean number of inspected outputs (left panel), exploited memory disclosures (center panel), and *wasted* inspections (right panel) over 100 replicate model runs each for the performance-, critical-, and security-focused defenders facing weak, medium, and strong attackers. Colors correspond between the current figure and Fig. 7.

The static, insensitive policies learned by the performance- and security-focused defenders lead to system inefficiencies, as depicted in Fig. 8 (center and right). This inefficiency can be demonstrated by comparing summary metrics for the security and performance-biased defenders with the those for the critical defender.

Figure 8 (right) indicates that when facing the *weak* attacker, the *critical* defender engages in approximately 76 fewer unnecessary inspections compared to the *security*-focused defender, a significant decrease in wasted system resources, while resulting in approximately 8 additional missed memory disclosures compared to the *security*-focused policy. The policy learned by the *critical* defender slashes system overhead due to system output searching by 90% from the baseline, *search everything* policy, compared to a 47% decrease in system overhead due to searching accomplished by the *security*-focused policy.

Parameterizing the Q-learning defense such that it is poised between the security-biased and performance-biased regimes gives the architecture an added level of adaptability and resilience to an attacker that might dynamically and perhaps adaptively adjust the strength of their memory disclosure attack. By the selecting a *critical* value for the reward term, *B*, the operator enables the control architecture to automatically adjust its level of operation to efficiently counter the current attack. This level of agility is only painstakingly achieved by human-operated systems, and so cannot be counted on to be achievable in the stresses of a rapidly unfolding cyber attack scenario. The proposed architecture achieves this high level of agility without the need for human intervention or supervision.

VII. Conclusion and Future Work

Balancing security and performance in an automated fashion while maintaining the requisite agility to move between these goals is a capability of clear utility in today's threat environment. We have proposed an architecture based on reinforcement learning to automatically achieve this objective and discovered a system parameterization setting to achieve defensive agility against an array of attacks. Adaptive control architectures combined with moving target defenses promise a new paradigm in resilient cyber defense. The leveraging of existing learning paradigms, and the invention of novel variants and techniques for the unique needs of cyber defenders is a vital direction for future effort.

The fact that learning techniques typically require the careful tuning of parameter values in order to operate effectively imposes significant barriers to operational adoption of systems built using these techniques. Ideally, operators would be spared the effort of manually tweaking and testing myriad parameter settings, and instead simply specify a desired operating point in performance-security space, for which the system would tune its own algorithms to achieve the specified level of security and performance, and dynamically adjust to changes in attacker behavior to maintain the requested levels of operation.

Automated tuning of algorithm parameters would have the added benefit of facilitating exploration of more complex and nuanced reward functions. We have adopted a simple reward function in this study to maximize the study's generality and applicability to the domain of adaptive control and strategy development,

$$R_i = B_i - (L_{i,s} + L_{i,m}) \tag{23}$$

with B_i the benefit the defender gets from discovering a memory disclosure in system output i, $L_{i,s}$ the latency cost the defender pays for searching a system output i for memory disclosures, and $L_{i,m}$ the latency cost the defender pays for mitigating a memory disclosure in payload i once if it has been discovered.

We note that by adopting an extended reward function we give the defender more parameters to adjust and tune to achieve a given effect. For example, if we have access to an *oracle* that informs the defender after-the-fact when a memory disclosure has been missed (or the defender is immediately able to discern this failure due to its effects in the defended system), we might incorporate a direct *cost* term (P) in the reward function to directly punish the defender for missing the said memory disclosure,

$$R_i = B_i - (P_i + L_{i,s} + L_{i,m}). \tag{24}$$

Here the defender is awarded B_i points for discovering a memory disclosure and taking mitigating steps before it harms system security, and punished P_i points each time a memory disclosure goes undetected and so is exploited by the attacker. Preliminary studies indicate that even a relatively small *punishment* term can impact strategy development nontrivially.

Other additional terms might include a reward for not inspecting a output, a punishment term for a certain threshold number of consecutive memory disclosure misses by the defender, and so on. Each of these terms give the policy tuning apparatus increased ability to fine tune the nature of the policy that is learned, but the combinatorial explosion caused by the increasing number of terms to be set by the user further necessitates a meta-adaptation capability to automatically set the parameters of the reward function and the learning algorithm parameters based on goals defined by the operator.

To further explicate this point, in this study we have examined model behavior under the variation of one, or at most two, model parameters. Important effects conceivably depend on subtle interactions between parameters that will be missed by this method. Brute force parameter space exploration methods rapidly lose feasibility as the number of parameters grow. Ideally we desire automated methods to find desirable operating points in parameter space, using techniques such as those described in [23], [24]. Developing such techniques would have the added significant benefit of potentially automating the tuning of the proposed architecture for operators that likely lack the requisite time and expertise to successfully tune a learning technique such as Q-learning. We set the investigation of self-tuning techniques as an item for future work.

References

[1] C. Chigan, Y. Ye, and L. Li, "Balancing security against performance in wireless ad hoc and sensor networks," in *IEEE Vehicular Technology Conference*, vol. 7. IEEE, 2004, pp. 4735–4739.

[2] M. Haleem, C. N. Mathur, R. Chandramouli, and K. Subbalakshmi, "Opportunistic encryption: A trade-off between security and throughput in wireless networks," *IEEE Transactions on Dependable and Secure Computing*, vol. 4, no. 4, pp. 313–324, 2007.

[3] W. Zeng and M.-Y. Chow, "Optimal tradeoff between performance and security in networked control systems based on coevolutionary algorithms," *IEEE Transactions on Industrial Electronics*, vol. 59, no. 7, pp. 3016–3025, 2012.

[4] ——, "Modeling and optimizing the performance-security tradeoff on d-ncs using the coevolutionary paradigm," *IEEE Transactions on Industrial Informatics*, vol. 9, no. 1, pp. 394–402, 2013.

[5] G. M. Miskeen, D. D. Kouvatsos, and E. Habibzadeh, "An exposition of performance-security trade-offs in ranets based on quantitative network models," *Wireless Personal Communications*, vol. 70, no. 3, pp. 1121–1146, 2013.

[6] H. Okhravi, T. Hobson, D. Bigelow, and W. Streilein, "Finding focus in the blur of moving-target techniques," *IEEE Security & Privacy*, vol. 12, no. 2, pp. 16–26, 2014.

[7] D. Bigelow, T. Hobson, R. Rudd, W. Streilein, and H. Okhravi, "Timely rerandomization for mitigating memory disclosures," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2015, pp. 268–279.

[8] "Pax address space layout randomization," http://pax.grsecurity.net/docs/aslr.txt, 2003.

[9] A. Bittau, A. Belay, A. Mashtizadeh, D. Mazieres, and D. Boneh, "Hacking blind," in *IEEE Symposium on Security and Privacy*. IEEE, 2014, pp. 227–242.

[10] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[11] P. Brass, *Advanced data structures*. Cambridge University Press Cambridge, 2008.

[12] S. Russell and P. Norvig, *Artificial intelligence: a modern approach*. Prentice Hall, 2010.

[13] R. S. Sutton and A. G. Barto, *Introduction to reinforcement learning*. MIT Press Cambridge, 1998.

[14] M. Minsky, "Theory of neural analog reinforcement systems and its application to the brain-model problem," in *Steps Toward Artificial Intelligence*. Princeton University, 1954, p. 830.

[15] B. Farley and W. Clark, "Simulation of self-organizing systems by digital computer," *Transactions of the IRE Professional Group on Information Theory*, vol. 4, no. 4, pp. 76–84, 1954.

[16] R. Bellman, *Dynamic Programming*. Princeton University Press, 2957.

[17] Z. Shi, *Advanced artificial intelligence*. World Scientific, 2011, vol. 1.

[18] C. J. C. H. Watkins, "Learning from delayed rewards," Ph.D. dissertation, University of Cambridge, 1989.

[19] J. H. Holland, *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[20] "Iceweasel," https://wiki.debian.org/Iceweasel, 2015.

[21] "D. Bigelow personal correspondence," 2015.

[22] P. Lamberson and S. E. Page, "Tipping points," *Quarterly Journal of Political Science*, vol. 7, no. 2, pp. 175–208, 2012.

[23] J. H. Miller, "Active nonlinear tests (ants) of complex simulation models," *Management Science*, vol. 44, no. 6, pp. 820–830, 1998.

[24] F. J. Stonedahl, "Genetic algorithms for the exploration of parameter spaces in agent-based models," Ph.D. dissertation, Northwestern University, 2011.