Bridging Automated to Autonomous Cyber Defense: Foundational Analysis of Tabular Q-Learning

Andy Applebaum
Camron Dennler
Patrick Dwyer
Marina Moskowitz
aapplebaum@apple.com
cdennler@apple.com
pnd@apple.com
mmoskowitz@apple.com
Apple
Cupertino, CA, USA

Harold Nguyen
Nicole Nichols
Nicole Park*
harold_nguyen@apple.com
nicole_nichols@apple.com
nicolepark@berkeley.edu
Apple
Cupertino, CA, USA

Paul Rachwalski
Frank Rau
Adrian Webster
Melody Wolk
prachwalski@apple.com
frau@apple.com
adwebster@apple.com
melody_wolk@apple.com
Apple
Cupertino, CA, USA

ABSTRACT

Leveraging security automation and orchestration technologies enables security analysts to respond more quickly and accurately to threats. However, current tooling is limited to automating very finely scoped and hand-coded situations, such as quarantining known malware and blocking traffic from known malicious domains. Recent research has sought to bridge the gap between this kind of automated security and *autonomous* cyber defense, leveraging reinforcement learning (RL) on top of basic automation to enable *intelligent* response. This paper provides foundational analysis of autonomous agents trained with Tabular Q-Learning through a series of experiments examining a range of network scenarios. Our results demonstrate that off-the-shelf Tabular Q-Learning does not offer a single, superior solution across all scenarios. However, we also find that modifying the underlying state encoding and update function can influence the robustness of the defensive agent to generalize to unseen evaluation environments without a significant loss in accuracy. These results highlight potential optimizations for more advanced RL techniques as well as provide a baseline for others leveraging RL for defensive cyber automation.

CCS CONCEPTS

• **Security and privacy** → **Network security**; • **Computing methodologies** → **Reinforcement learning**; *Modeling and simulation*;

KEYWORDS

autonomous cyber defense, reinforcement learning, intrusion response, network security

*Work done while employed at Apple.

AISec '22, November 11, 2022, Los Angeles, CA, USA

ACM ISBN 978-1-4503-9880-0/22/11.
https://doi.org/10.1145/3560830.3563732

ACM Reference Format:
Andy Applebaum, Camron Dennler, Patrick Dwyer, Marina Moskowitz, Harold Nguyen, Nicole Nichols, Nicole Park, Paul Rachwalski, Frank Rau, Adrian Webster, and Melody Wolk. 2022. Bridging Automated to Autonomous Cyber Defense: Foundational Analysis of Tabular Q-Learning. In *Proceedings of the 15th ACM Workshop on Artificial Intelligence and Security (AISec '22), November 11, 2022, Los Angeles, CA, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3560830.3563732

1 INTRODUCTION

Typical enterprise security deploys monitoring software that generates alerts, triaged by human analysts. Advances in security automation technologies have brought efficiencies by automatically triggering a response or executing a *playbook* [32] in response to these alerts. Such automation saves analyst time, reduces personnel load, enables faster response to intrusions, and limits damage. Industry collaborations have defined standardized response/playbooks to further simplify automation; e.g. the Integrated Adaptive Cyber Defense (IACD) playbook specification [14] and the Incident Response Consortium's playbook policy engine [15].

Current state-of-the-art playbook technologies focus on encoding the *mechanics of how* to respond to intrusions and less on the underlying *decision*. While both problems are challenging, automating decisions is particularly difficult given the abundance of false-positives generated by benign user activity and the potentially high productivity cost to incorrect defensive actions. Consequently, current automation technologies only handle alerts with known low false-positive rates. Technologies that process alerts with high false-positive rates rely heavily on human operators to hardcode if-then decision logic to correlate specific alerts and responses.

Instead, by leveraging machine learning (ML) techniques, we can train an agent that is able to autonomously defend a system, minimizing self-damage from responses that use noisy sensor data. Research in this space – termed *Autonomous Cyber Defense* (ACD) – has spanned the past few decades and grown from early work using expert systems [5] to most recently using reinforcement learning (RL) [28]. These latter types of systems are advantageous in that they *learn* effective policies that maintain system performance in the face of persistent adversaries, without being *told* what those policies should be. Recent advancements in this space have shown success applying state-of-the-art RL to ACD [8, 11, 12, 22].

This work builds on prior research by providing *foundational* analysis of the Tabular Q-Learning RL algorithm as a baseline to compare other ACD solutions to. Tabular Q-Learning is well-suited to foundational analysis due to its simple design, allowing us to describe its implementation with high detail. By contrast, more advanced approaches can produce stronger results, but can also be harder to analyze. Moreover, because of its simplicity, we can easily customize Tabular Q-Learning to test different variations. We specifically experiment with modifying the state encoding and update function, showing how doing so can produce a more effective defender in a variety of network topology and noise scenarios, as well as when evaluated in unseen environments.

In testing and evaluating Tabular Q-Learning, we are (1) able to identify areas of future work and enhancement, (2) understand the utility of more advanced approaches, and (3) provide interpretable baselines to compare to, contributing the following:

Comprehensive Testing. We provide a deep dive into Tabular Q-Learning as applied to network defense through an extensive series of experiments on a modified version of *CyberBattleSim* [20]. We vary network topology, attacker win condition, and detection noise rate, comparing performance of different Tabular Q-Learning variations against fixed defensive strategies and each other.

Abstraction and Reward. As part of our comprehensive testing, we show how RL-based cyber defenders can use an abstracted state space and a customized update function to provide a more robust defense. Depending on the application, this enables ACD end-users to encode a trade-off between accuracy and resiliency.

Transfer Learning. We run a set experiments showing the performance of different Tabular Q-Learning variations in scenarios where the training and evaluation environments are different. These results show the viability of using an abstract state encoding to hedge against differences in training and evaluation.

Paper Organization. Section 2 presents our formal model and summarizes related work; Section 3 describes our extensions to CyberBattleSim; Section 4 walks through our Tabular Q-Learning approach; Section 5 details our experimental design; Section 6 provides results from our experiments; Section 7 discusses takeaways from our results; and Section 8 highlights areas of future research.

2 BACKGROUND

This work focuses on the *autonomous response* problem, wherein a defensive agent needs to automatically execute an action in response to imperfect observations about an attacker. This section provides background on this problem, covering our formal model, the basics of Tabular Q-Learning, and an overview of related work.

2.1 Formal Model

We use a *Markov Decision Process (MDP)* to formally model the problem, describing it as a 4-tuple $< \mathcal{S}, \mathcal{A}, \mathcal{R}_a, \mathcal{P}_a >$ where:

- $\mathcal{S}$ is the set of states; in this case encoding the defender's observation and other network information.
- $\mathcal{A}$ is the set of actions the defender can take; e.g. isolate a host, reissue a user's password, reimage a host, etc.
- $\mathcal{R}_a : \mathcal{S} \times \mathcal{S} \rightarrow \mathbb{R}$ is the immediate reward from executing action a in state s and arriving in state s', with the reward based on some measure of compromise and availability.
- $\mathcal{P}_a : \mathcal{S} \times \mathcal{S} \rightarrow \{x \in \mathbb{R} \mid 0 \leq x \leq 1\}$ is the probability that executing action a in state s will result in state s'.

By design $\mathcal{P}_a$ encodes the probabilities of state transitions resulting from defender *and* attacker actions and the probabilities of state changes due to receiving new (potentially incorrect) alerts.

2.2 RL and Tabular Q-Learning

Reinforcement learning (RL) techniques can provide approximate solutions to MDPs, learning policies that maximize expected reward over a finite time horizon. RL uses the same *state*, *action*, and *reward* components but rather than conduct an exhaustive state space search for the optimal policy, an agent interacts with the environment and chooses actions based on its current state and a reward/penalty from historic actions.

Our work has the agent interacting with a *simulation* of a cyber environment as opposed to a real network, enabling more rapid learning of policies. However, using a simulator necessarily requires a simplification that leaves an extrapolation step to use the learned policy in a real network. By contrast, *emulated* environments have greater realism but require significant upfront cost to design and build, with learned policies not guaranteed to generalize.

The *Tabular Q-Learning* class of RL methods seeks to learn the expected reward for each state-action pair, maintaining a table of these rewards referred to as the *Q-matrix*. Formally, we define $Q : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ such that $Q(s, a)$ is the expected discounted reward for executing action $a \in \mathcal{A}$ in state $s \in \mathcal{S}$. To populate Q, we use an iterative Bellman equation [37] update such that for timestep t with the agent in state $s_t \in \mathcal{S}$, after executing action $a_t \in \mathcal{A}$, receiving reward r_t, and transitioning to new state $s_{t+1} \in \mathcal{S}$:

$$Q'(s_t, a_t) \leftarrow Q(s_t, a_t) + l_r \cdot (r_t + \gamma \cdot \max_a Q(s_t, a) - Q(s_t, a_t)) \tag{1}$$

where $Q'(s_t, a_t)$ is the new Q-value for (s_t, a_t); l_r is the learning rate controlling how fast the values in Q change at each update; γ is the discount factor controlling the trade-off between future and immediate rewards; and $\max_a Q(s_{t+1}, a)$ is the estimate of the optimal value over all actions if taken in the new state s_{t+1}.

Tabular Q-Learning also uses an *exploration strategy* that defines when to choose actions by *exploiting* the currently learned policy/Q-matrix or *exploring* new actions and outcomes. We focus on the ϵ-*greedy* strategy [38] as it provides some guarantees of convergence to an optimal policy: at each time step we select the action with the highest possible reward (*exploitation*) with probability $1 - \epsilon$, otherwise choosing an action at random (*exploration*).

2.3 Related Work

Drawing from the cyber resiliency [4] community, [28] justifies using RL in ACD by applying Tabular Q-Learning to an abstract cyber simulation environment where the attacker propagates probabilistically through a network and the defender can do nothing, isolate a host, or patch a host. Their experiments show the learned policy can successfully defend the network, and is one of the few works to examine Tabular Q-Learning.

[6] also examines Tabular Q-Learning. This work uses an abstract network simulator encoding lateral movement, giving both the attacker and defender partial observability at each step. The paper

tests many RL algorithms for both attacker and defender, including Tabular Q-Learning, four Monte-Carlo techniques (ϵ-greedy, Softmax, Upper Confidence Bound 1 [2], and Discounted Upper Confidence Bound [9]), a neural network, and a linear network. Q-Learning results as one of the worst algorithms for the attacker, but is one of the *best* algorithms for the defender.

FARLAND [22] implements more sophisticated RL solutions by leveraging the RLlib [17] interface, integrating it into a hybrid simulation-emulation environment. Using this interface, they show positive results against a simple attacker for Ape-X Deep Q-Networks (APEX-DQN) [13], Proximal Policy Optimization (PPO) [30], and Asynchronous Advantage Actor-Critic (A3C) [21]. However, each approach's results drop against a more stealthy and deceptive attacker, likely a result of FARLAND's complexity; [28] and [6] had different conclusions but used heavily abstracted simulation environments that left little room for attacker strategy deviation.

The CybORG environment [3] also uses a hybrid simulation-emulation environment, and as a publicly-released simulator[1] later became the basis for the CAGE Challenge [1], a competition to build effective autonomous cyber defense agents. In the most recent competition the winning solution [8] used PPO with Curiosity [26] for exploration. Their solution was also hierarchical: the defending agent used an auxiliary model to classify the attacker strategy and deploy a defense tuned to that strategy.

[11] uses RL with self-play for ACD, leveraging an additional open-source simulator[2] that uses a more abstract network representation than FARLAND or CybORG. The work tests PPO, REINFORCE [27], and a new approach termed *Auto-Regressive PPO (PPO-AR)* which exploits sequential decisions to reduce the action space. Each approach is effective, with PPO-AR appearing strongest. However, similar to [22], both PPO-AR and PPO struggle to converge against an advanced and dynamic attacker.

Relative to [28] and [6], our experiments test a wider set of network scenarios and use a more fine-grained network simulator that more closely matches a real environment. [22] and [8] use a more advanced simulator and sophisticated RL, but do not test against the less-complex Tabular Q-Learning and only consider a limited number of scenarios. Lastly, as opposed to [6, 8, 11, 22, 28], our work features more baseline results to compare to.

Within the ACD domain, there is related work that does not use RL or focus on network response. *FlipIt* [34] is a simple game where an attacker and defender both have partial observability and battle for control of a single resource. The original work derived solutions via game theory, but has since been solved with Deep-Q Networks [10] and a temporal-difference Q-learning approach called *QFlip*[3] [24]. The mechanics underneath FlipIt rely on the timing of attacks/defenses and feature a smaller state space. In contrast, network-oriented ACD has exponentially-sized state space and is concerned with responding to noisy alerts while maintaining network performance and availability.

Other game-theoretic solutions include [39], which implements an automated response and recovery system by modeling the attacker-defender interaction as a two-player Stackelberg stochastic game [25], with the core of the approach being to construct an attack-response tree for the protected system, convert the tree to a competitive Markov decision process (CMDP) [7], and then solve the resultant CMDP. Another example is the work of [23], which models the autonomous response problem as a Partially Observable Markov Decision Process (POMDP), constructed through conversion of an attack graph and using a more descriptive state space. The resultant POMDP is then solved using Monto-Carlo planning [31], using a game-theoretic cyber simulator to perform the search.

We consider *autonomous red teaming* [16, 33] research to be out of scope due to asymmetry in modeling the attacker vs defender. Interested readers can consult the survey in [36], which also provides a more extensive survey of autonomous response.

3 SIMULATION DESIGN

Our work expands Microsoft's CyberBattleSim[4] simulation environment [20], which utilizes the OpenAI Gym interface to train RL-based attacker agents and comes pre-packaged with two RL attacker agents. In this section, we provide an overview of the originally released capabilities and detail our extensions.

3.1 Overview and Original Capabilities

Simulations within CyberBattleSim are organized into attacker-defender games (*episodes*) over a specific network topology. Each episode has a maximum number of turns (*iterations*), with an episode ending when the attacker achieves its objective or reaching the maximum number of iterations. Each turn the attacker executes an action and then receives an observation of the environment – including new information learned from executing the action – as well as a reward, which is dictated by hardcoded values associating actions, outcomes, and host importance. If the attacker achieves its objectives, it receives a large reward and the episode ends.

The attacker can choose one of three action types: (1) exploit a local vulnerability; (2) exploit a remote vulnerability; or (3) connect to a remote host using lateral movement. Each action requires different parameters, such as the specific vulnerability to use for exploitation or the specific user credentials to use for connect actions. Actions also have preconditions – e.g., having already discovered the target host, knowing the credentials for a user account, etc. – with postconditions such as discovering new hosts, leaking credential information, gaining a foothold on a new host, etc.

CyberBattleSim networks are explicitly declared with what individual hosts, vulnerabilities, ports/protocols, and properties are present on the network. Additionally, for each host the specification declares pairs of credentials and ports that are authorized for remote authentication and firewall rules to identify which ports accept inbound or outbound traffic. To ensure the attacker is operating realistically, the CyberBattleSim environment cleanly separates the attacker's *knowledge* of the network and the *specification* of the network, forcing partial observability and preventing the attacker from trivially constructing an attack graph of the network.

3.2 Defender Interface

The original CyberBattleSim implementation only had stochastic defenders that initiate random defensive actions, which include

[1] https://github.com/cage-challenge/cage-challenge-2/tree/main/CybORG
[2] https://github.com/Limmen/gym-idsgame
[3] https://github.com/lisaoakley/gym-flipit
[4] https://github.com/microsoft/CyberBattleSim

re-imaging a host, adding/removing vulnerabilities from hosts, starting/stopping services, and changing firewall rules. The defender's success is measured by a *network availability* score, which is calculated each iteration as a scalar value derived from the number of running hosts and services.

Our extension replaces the stochastic defender with an interface to train a dynamic defensive agent. On each turn the attacker and defender both make a decision based on their own individual current observation, *declare* their respective actions (the attacker's action resolves first, then the defender's action), then both receive an updated observation. This paradigm allows both attacker and defender to observe consequences of their respective actions, as well as to infer or perceive the actions of the other.

We also replaced the original defensive actions with a new set, only reusing *reimage* from the original set. Our new set of actions was chosen to provide parity with the attacker's action sophistication, and provide new analysis insight without significant modification. Each action is either *global*, taking no input parameters, or *targeted*, taking action against a specific target. Each turn the defender must pick *exactly one* of the following actions:

Wait (Global). The defender does nothing for the turn.

Reimage (Targeted). This action takes a target host and sets it as offline, removing the attacker's foothold if present. While offline, neither the defender or attacker can use the host, and the overall network availability score will be reduced.

Rebuild All (Global). This action allows the defender to re-image all hosts on the network at once, "resetting" the network to a known good state but reducing availability to 0, similar to [23]. When executed, all online hosts are put into the re-imaging state, and any hosts in the process of re-imaging continuing that process.

Revoke Credentials (Targeted). This action takes a target credential and marks it as revoked, treating any usage (benign or malicious) of that credential as invalid. After a pre-set number of turns, the credential is made valid (*active*) again. The attacker will only learn a credential has been revoked after using it, and must "refresh" a previously learned credential if it was revoked after being learned (similar to a user changing the credential.).

3.2.1 Observations and Noise. Each timestep the defender receives an observation O with the following fields:

- $owned$: an array of size $|\mathcal{N}|$, where $\mathcal{N}$ is the set of all hosts, such that $owned[i] = 1$ implies an alert on host i indicating it is compromised, with $owned[i] = 0$ for no alerts.
- $online$: an array of size $|\mathcal{N}|$, where $\mathcal{N}$ is the set of all hosts, with $online[i] = 1$ if host i is online, and $online[i] = 0$ if not.
- $recent$: a list of variable size, where each element is a pair (i, j) indicating that in the last timestep sensors identified credential i logging into host j.
- $active$: an array of size $|C|$, where C is the set of all user credentials, with $active[i] = 1$ if credential i is currently active, and $active[i] = 0$ if not.

We also introduce noise parameters to impact observations:

- False positive rate: fp_rate. At each timestep, if the attacker does not have a foothold on host i, we generate an erroneous alert on that host (i.e., $owned[i] = 1$) with probability fp_rate, reporting $owned[i] = 0$ correctly otherwise.
- False negative rate: fn_rate. At each timestep, if the attacker does have a foothold on host i, we do *not* generate an alert on that host (i.e., $owned[i] = 0$) with probability fn_rate, reporting $owned[i] = 1$ correctly otherwise.
- Benign login activity rate: $login\_rate$. For each credential c, we generate a valid login event to host h with probability $login\_rate$. To allow for more noise, this process is repeated each time an event is successfully generated, moving to the next credential otherwise.
- Missed login activity rate: $missed\_rate$. The $missed\_rate$ controls how likely it is a (benign or malicious) login event will be successfully recorded.

3.2.2 State and Action Wrapper. The original attacker agent featured a wrapper that contained (1) a *state tracking* object that provides stateful memory; (2) a set of functions that maps the state tracker to well-defined *features*; and (3) an action abstraction that converts the set of actions for a given environment into discrete, fixed integers and back. This interface enables subject matter experts to encode relevant features they want their learners to use.

The added defender agent follows a similar process, creating a separate state tracker that stores historical (past 10 timestep) host status and host-credential connections, defender-oriented features, and a defensive action abstraction. We discuss these features and the action abstraction later in Section 4.

3.2.3 Reward Function. The defensive agent's reward function – referred to as the *blue reward* – is based on network availability: after each turn the defender receives a modified network availability score, or a penalty if the attacker has achieved its objective. We made two modifications to the defender's network availability score: first, if a credential was revoked or a host was re-imaged and the *attacker* had access to that credential/host, then the *defender* will receive full availability, to represent a successful defense, despite that object being revoked/offline. Second, if an attacker has control of a host at the end of a turn, then that host will be treated as being offline for network availability purposes. This scheme is similar to others in prior work (e.g. [3, 8]) but differs in the first modification. Note the blue reward is independent from the attacker, or *red*, reward, which allows us to encode different rewards for each agent.

3.3 New Environments

Three types of network environments are used in this paper: *Chain*, *Flat*, and *Structured*. *Chain* is included in CyberBattleSim and dynamically creates a network where each host is a "link" in a lateral movement chain; i.e., host 0 can laterally move to host 1, host 1 can laterally move to host 2, and so on. The Chain environment specifies two "types" of hosts, such that every other link in the chain is either using the same Linux or Windows profile.

Flat is new and encodes a simple network that is easy to traverse, featuring a highly connected credential topology with many paths for lateral movement. The network has an *administrator* account that can be used to move to any host, in addition to other accounts that are randomly seeded. By design, the network has only one host profile and has no"disallow" firewall rules.

Structured is also new and encodes a more challenging environment for the attacker, having one core attack path with multiple

(a) Attacker baseline results.

Attacker Agent	Environment		
	Chain	Flat	Struct.
Random	305	215	224
Cred-Exploiting	982	212	739
Tab. Q-Learning	835	207	293
Deep Q-Learning	1278	214	320
Greedy	1340	207	902

(b) Defender baseline results.

Defender Agent	Environment		
	Chain	Flat	Struct.
Cred Hunter	364	327	368
Zapper	364	332	367
Threshold	278	252	330
Blue-RAND	91	75	89
None	50	5	79

Table 1: Average rewards for different attacker (1a) and defender (1b) strategies, averaged over 20 episodes of length 400. Each attacker is measured against the *None* defender, while each defender is measured against the *Greedy* attacker. Note that 1a reports the average *attacker* reward, which is different than the defender reward used in 1b and the rest of this paper.

opportunities to deviate, such as vulnerability frequency differences, credential provenance, and restrictive firewall rules. To further add to the network's complexity, the *Structured* environment has seven types of host profiles, as opposed to just two for *Chain* and one for *Flat*. These design choices make the network *structured* in the attack topology, but *unstructured* in host design.

3.4 Fixed-Policy Agents

3.4.1 Attacker Agents. CyberBattleSim comes with two fixed-policy (*Random* and *Cred Exploiting*, the latter of which prioritizes lateral movement with leaked credentials) and two learning-based attacker agents (*Tabular Q-Learning* and *Deep Q Network*); note that the Tabular Q-Learning agent referenced here applies exclusively to the attacker and is independent from our approach in Section 4. We add a new fixed-policy attacker called **Greedy** that follows a simple script for action selection, prioritizing lateral movement, followed by host discovery, and then credential discovery; if none of these three are possible, it chooses an action at random. *Greedy* also has several mechanisms for dealing with repeated actions; i.e., in some cases it may try a previously executed lateral movement step, but it will not repeat any host or credential discovery steps.

Table 1a shows results comparing each of the attackers in the *Chain*, *Flat*, and *Structured* network configurations with no defensive agent, with the goal of performing a quick test to understand the new *Greedy* attacker's efficacy relative to the built-ins; we choose to compare to no defender for ease of comparison and to analyze the *best-case* scenario for the attacker. Results show that the new *Greedy* attacker is competitive with the built-in ones; as an added benefit, being a fixed-policy approach it runs much more quickly than the learning-based agents, which each had 200 training episodes beforehand. Because of its speed and efficacy, we only use this attacker agent when testing our defensive agents later in Sections 3.4.2, 5, and 6. Looking at the environments, *Flat* appears easiest – its maximum possible reward is ~225 – *Chain* seems to be of middle difficulty, and *Structured* appears to be hardest, with the fixed-policy agents significantly out-performing the learning ones.

3.4.2 Defender Agents. We introduce five fixed-policy agents to be used for defensive baselining.

Zapper. Randomly selects a reportedly compromised host (if any) and *reimages* it. Uses the *owned* portion of the observation.

Credential Hunter. Revokes any credentials that have login events – generated either by lateral movement or a benign event from the $login\_rate$ noise parameter – and re-images any host that credential recently logged into. Because the defender can only issue one move per turn, keeps a list of "to-do" actions it will need to execute, as each login event yields at minimum two actions. Uses the *active* portion of the observation.

Threshold. Issues a *rebuild all* action if 50% or more hosts report as compromised. Uses the *owned* portion of the observation.

Random (Blue-RAND). Chooses one of *wait*, *reimage*, *rebuild all*, and *revoke credentials* at random with equal probability.

None. Issues the *wait* command each timestep.

Table 1b shows the results for each defender in each of the three network configurations with no noise and against the *Greedy* attacker, reporting the average blue reward over 20 episodes of length 400; we choose no noise to simplify the environment to baseline the defenders, but choose the Greedy attacker to make sure they are sufficiently stress-tested. *Cred Hunter* and *Zapper* score the highest in each scenario, with *Threshold* third, then *Blue-RAND*, and then *None*. Looking at environments, in general rewards are less in the *Flat* network versus the other two, which is due to how quickly the attacker can move through that network.

4 TABULAR Q-LEARNING IMPLEMENTATION

Our Tabular Q-Learning implementation follows the specification in Section 2.2, including the update function in Equation 1. Extending this specification, we introduce several small modifications that help the algorithm perform slightly better in CyberBattleSim, including (1) how we model states and actions (i.e., $\mathcal{S}$ and $\mathcal{A}$); (2) how we decide between exploitation and exploration; and (3) how we perform updates to the Q-matrix.

4.1 State Space

To define the state space we create a function $S : O \rightarrow \mathcal{S}$ that maps every observation to a state. In total we use four representations, referred to as follows: *vector*, *count*, *percentile*, and *none*. The most common approach in the literature [6, 8, 23, 39] is to map each observation to a fixed-size vector where each bit denotes some property about each individual host or credential, enabling fine-grained tracking of each object in the environment. To implement this in CyberBattleSim, we designate $\mathcal{S} = O$, concatenating the *owned*, *online*, *recent*, and *active* vectors into a single state[5]. We refer to this representation as *vector*.

Another way to represent S is to *count* the number of compromised and online hosts and the number of recently used and active accounts in the observation, and then map that to a discrete state; i.e., $\forall s \in S, s = s_1 \| s_2 \| s_3 \| s_4$ where $s_1 = \sum_{i=1}^{n} owned[i]$, $s_2 = \sum_{i=1}^{n} online[i]$, etc. As opposed to the *vector* representation, the *count* representation is more abstract and has a smaller overall size ($n^2 \cdot c^2$ vs. 2^{cn} where n is the number of hosts and c the number of credentials). However, multiple observations can map to the same state, losing descriptiveness.

A third representation is to use the *percentile* of positive values in each observation vector. This scheme explicitly designates a set of *buckets*, scoring if the vector has 0% positive values, 1−25%, 26−50%, 51 − 75%, or 76 − 100%. With four observation vectors, regardless of

[5] In this case we convert *recent* to an array of size c where c is the number of credentials, with $recent[i] = 1$ if credential i was recently used and $recent[i] = 0$ if not.

the environment this guarantees a state space of exactly $|\mathcal{S}| = 625$, with many observations mapping to the same state(s).

With these three representations – as well as a fourth, where we ignore the observation vector (referred to as *none*) – we can construct 256 different state space variations, mapping each of the four observation vectors to one of the four representations.

Note that throughout this paper we often refer to the vector representation as a *specific* state space representation, and the percentile, count, and none representations as *abstract*.

4.2 Action Space

Most RL algorithms have an action set $\mathcal{A}$ such that every action in the environment has a representative action. For CyberBattleSim, this would yield a total of $2 + n + c$ actions; one for *wait*, one for *rebuild all*, n hosts that can be re-imaged, and c credentials that can be revoked. Using this approach would result in a Q-matrix of size $(2 + n + c) \cdot |\mathcal{S}|$, pairing each action to each state.

Instead, we follow the CyberBattleSim attacker scheme and encode actions as *features* coupled with *abstract actions* that are translated on execution. Formally, we define an action $a = a_F \| a_A$ where a_F is a set of features to match – i.e., $a_F = a_{f_1} \| a_{f_2} \| a_{f_3} \| \ldots$ – and $a_A \in \{0, 1, 2, 3\}$ is the abstract action, with each integer representing a specific defensive action. To execute $a = a_F \| a_A$, we find an object that matches the features in a_F and apply the action a_A to it. Note that the feature set a_F varies based on specific action a_A, as e.g. relevant features for hosts and credentials are different.

We provide a wide set of features to build a_F, including ones that look at host/credential history, recent compromise status, credential location usage, etc. In this work, we only consider two feature-space variations, each coupled with a specific state space representation. The first variation matches the case of using a vector state space, with the only feature being the host or credential ID; this results in an action in $\mathcal{A}$ for each action in the environment, and does not reduce the size of the Q-matrix. We call this approach *target-based*.

The second variation – *feature-based* – is used for the abstract state space representations (count, percentile, none). Depending on the action, we use the following features:

- For *re-imaging a host*: whether or not the host is compromised; whether or not it is online; whether or not it had a recent logon event; and whether or not it is reimagable.
- For *revoking a credential*: whether or not the credential is active; whether or not the credential was recently used on a compromised host; and whether or not the credential was used recently on any host.

4.3 Exploitation and Exploration

During exploitation, we choose the action with the highest expected discounted reward; i.e., given state s, we select $a \in \mathcal{A}$ such that $\forall a' \in \mathcal{A} \mid a \neq a'; Q(s, a) >= Q(s, a')$. When $\mathcal{A}$ is target-based, a contains both the specific action type and target and can be immediately executed. For feature-based actions, we use post-processing where given $a = a_F \| a_A$, we create a list of all targets that match the feature set in a_F, choosing one at random. If no target matches that feature set, the agent chooses the action with the second highest value for the current state, repeating if the action is not valid and defaulting to exploration after five unsuccessful attempts.

For exploration the agent does not select an action completely at random. Instead, it selects an action that it has executed the least number of times in the current state, which encourages more varied exploration. It is also restricted to only explore *rebuild all* at most three times for each state, since the action has consistent end state.

To split between exploitation and exploration, we leverage the ϵ-greedy strategy, modifying it to use three training phases:

(1) *Exploration*. Trains over a large number of episodes of shorter duration with $0.9 \leq \epsilon \leq 1$ and mostly choosing exploration.
(2) *ϵ-greedy*. Trains on normal ϵ-greedy search, decaying ϵ from 0.9 to 0.1. Uses a mid-size set of episodes with the normal number of iterations.
(3) *Exploitation-only*. Trains with $\epsilon = 0$ – i.e. only exploitation – with a small number of episodes and normal iteration count.

4.4 Updating and Loss Avoidance

The last change is motivated by the difficulty balancing the penalty for losing an episode with that episode's intermediate rewards. To combat this, we introduce *loss avoidance*, where the update function handles losses as a special case. We build two variations; assuming an initial state of s_t, action a_t, next state of s_{t+1}, and reward of r_t:

Avoidance. If r_t is negative, we set $Q(s_t, a_t)$ to a large negative value and skip the Bellman update. Any state-action pair that was previously assigned a negative value is never updated.

***High*-Avoidance**. Implements the same conditions as for *Avoidance*. In addition, if an episode results in a loss with terminal state s_{t+1}, for all $a = a_F \| a_A \in \mathcal{A}$, if $a_A \neq$ *rebuild all*, then we set $Q(s_{t+1}, a)$ to be a large negative value.

Both approaches avoid investigating "losing" states in the future: with *Avoidance* once a state-action pair has produced a loss, that action cannot be chosen again in that state in the future. *High-Avoidance* does the same and adds that if a game has ended in state s_{t+1}, then if in that state again in the future we should select *rebuild all* and reset the network to a known good state. For both, by modifying the update function we run the risk of violating the Bellman update's optimality guarantee, but believe this is acceptable given how long it might take the traditional update to converge.

5 EXPERIMENTAL SETUP

Our experiments leave most simulator settings constant (Appendix A, Table 5), varying instead environment details and defender strategy. The attacker agent is fixed to use the *Greedy* policy due to its balance of speed and performance. To enable more interesting interplay between attacker/defender, we prevent the defender from re-imaging the attacker's start host, which forces the episode to run until the attacker achieves its objectives or the episode hits the maximum number of iterations. Evaluation episodes consist of 500 time iterations, bounding the defender reward to a maximum loss penalty of -1000 (instantly losing) to 500 (no attacker foothold).

Nomenclature. An *episode* refers to a single game consisting of multiple turns, which are referred to as *iterations* (Section 3). We refer to a specific CyberBattleSim environment configuration – i.e., combination of noise profile, topology and win condition – as a *scenario*. A specific defender strategy being trained/evaluated in a specific scenario is referred to as a *trial*. We disambiguate defender strategies using Tabular Q-Learning but with different settings by

(a) Noise profiles.

Name	*fp*	*fn*	*login*	*missed*
No Noise	0.00	0.00	0.00	0.00
No Host Alerts	0.00	1.00	0.00	0.00
No Cred Alerts	0.00	0.00	0.00	1.00
Low Noise	0.10	0.20	0.10	0.35
Some Noise	0.30	0.50	0.30	0.60
High Noise	0.45	0.75	0.60	0.80

(b) Topologies.

Topology	Size
Chain-4	4
Flat-7	7
Structured	13
Chain-10	10
Flat-17	17

(c) Win conditions.

Name	% Owned
Quick	30
Two-Thirds	66
No Penalty	110

(d) Tabular Q-Learning variations.

Named Tabular Q-Learning Variation	State Representation		Avoidance
	Hosts	Creds	
Targeted	Vector	Vector	None
Host-Targeted	Vector	Percent	None
Host-Targeted (**h**)	Vector	Percent	**High**-Avoidance
Count	Count	Count	None
Host-Count	Count	Percent	None
Host-Count (a)	Count	Percent	Avoidance
Host-Count (**h**)	Count	Percent	**High**-Avoidance
Percentile	Percent	Percent	None
Percentile (**h**)	Percent	Percent	**High**-Avoidance
Percentile, No Creds (**h**)	Percent	-	**High**-Avoidance

Table 2: Experiment design, including for noise (2a), network topology (2b), and attacker win condition (2c). 2d shows the tested Tabular Q-Learning variations, listing the state space representations for hosts, credentials, and loss avoidance strategy. Similar named variations are distinguished with (a) and (h) to correspond to the distinguishing avoidance strategy. Specific values were chosen based on preliminary testing to ensure environment diversity.

referring to these strategies as Tabular Q-Learning *variations*. When we train a particular Tabular Q-Learning variation in a variety of scenarios, that variation learns different *policies* customized to each. We refer to the set of all Tabular Q-Learning variations as *learners*.

5.1 Scenario Design

A key advantage of an RL-based agent vs. fixed policy is the potential to learn an effective defensive policy regardless of environment. Agents such as *Zapper* depend on detection accuracy and are unable to change. With this idea, we seek to answer the following:

Question 1. *Can Tabular Q-Learning provide an effective defender regardless of the specific noise profile, network topology, and attacker win condition?*

The span of values for the noise profiles, network topologies, and attacker win conditions, are defined in Tables 2a, 2b, and 2c; we use the combination of these values to define the defense agent test scenarios. For the listed noise profiles, we expect an RL-based agent to perform near-optimal in the *No Noise*, *No Cred Alerts*, and *No Host Alerts* conditions, as accurate information is maximized for the defender to learn an effective see-then-re-image response policy. By contrast, with *High* noise we would expect the RL-based agent to learn a policy that leans heavily on *rebuild all*, as the noise makes a pinpoint response (i.e., *reimage* and/or *revoke credentials*) highly inaccurate. With regards to *win conditions*, this refers to the percentage of the network the attacker needs to compromise for it to win the episode. In the case of *No Penalty/110%*, the attacker has an unachievable goal and so the episode does not end with a win or a loss; however, the defender still accrues reward based on the blue reward function (Section 3.2.3).

5.2 Tabular Q-Learning Variations

Table 2d lists ten named Tabular Q-Learning defender variations, distinguished by the combination of host and credential state space representations and loss avoidance strategy. For example, the *Targeted* variation uses the vector state space representation for hosts and credentials (Section 4.1) and no loss avoidance strategy. In contrast, the *Host-Count (**h**)* variation uses the count state space for hosts, percent state space for credentials, and the High-Avoidance loss strategy. For simplicity's sake, each variation has the same state space representation for hosts (i.e., for *owned* and *online*) as it does for credentials (i.e., for *recent* and *active*). For the last variation even though the state space does not include any information about credentials the defender can still choose to *revoke credentials*, looking only at credentials after it has made that choice. With these variations we seek to answer the following:

Question 2. *Is there a single Tabular Q-Learning variation that* always *outperforms the others?*

We anticipate there will be no singular best variation; the utility of loss avoidance and state space representation may be out weighed by the noise parameters and win conditions of the environment. In particular, we expect that the abstract, loss avoidant models perform best in high-noise environments and the more specific models perform best in low-noise environments.

5.3 Transfer Learning

Lastly, we ask how effective the learned policies are in unseen environments; i.e., how well the policies *transfer* to new environments:

Question 3. *Which Tabular Q-Learning variations are effective when trained and evaluated in different environments?*

We do not exhaustively test all scenarios for transfer learning. Instead, we train each Tabular Q-Learning variation in the *No*, *Low*, and *High* noise settings on a fixed network (*Chain-4*) with fixed win condition (*No Penalty*). We then take each trained learner and evaluate it on a set of 12 other scenarios consisting of each of the same three noise conditions applied to three specific networks (*Flat-7*, *Chain-10*, and *Structured*).

6 RESULTS AND ANALYSIS

We create 90 different scenarios by iterating through each combination of noise profile from Table 2a (6 total), topology in Table 2b (5 total), and attacker win condition in Table 2c (3 total). Testing 5 baseline defenders and 10 Tabular Q-Learning variations (Table 2d) in the 90 scenarios gives us a total of 1350 trials. Each trial is evaluated with 100 game episodes consisting of 500 iterations each; at the end of the episode, we record the total reward, assigning the trial a score by averaging the reward over all 100 episodes. For transfer learning, we run 360 trials, consisting of each of the 10 Tabular

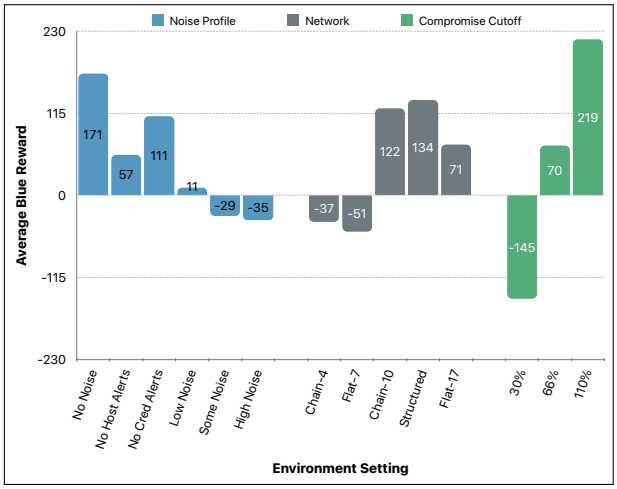

Figure 1: Average blue reward by scenario setting. A larger number indicates better defender performance, and ranges from -1000 to 500.

Q-Learning variations trained in 3 noise profiles and evaluated in 12 environments (with attacker win condition held constant).

6.1 Aggregate Analysis

This section walks through the aggregate results over each trial (Table 3), analyzing environmental variations, learner performance against baselines, and then learner performance against each other.

6.1.1 Environmental Analysis. Figure 1 shows the average blue reward for each scenario, which, since it is averaged over each defender, gives an approximation of the relative difficulty of each scenario setting. Considering noise, the *No* noise scenarios have the highest average reward and *High* noise the lowest, indicating the former's ease for the agents and the latter's difficulty. For network topologies, *Chain-4* and *Flat-7* both are negative on average – indicating hard to defend – whereas *Chain-10* and *Structured* have positive and much higher average rewards. *Flat-17* appears harder to defend than *Chain-10* and *Structured*, likely due to the ease of lateral movement. Average reward for attacker win conditions indicate that as the threshold increases so does the reward.

Summary. These results exemplify that defensive effectiveness strongly depends on network dynamics.

6.1.2 Learners Against Baselines. Table 3 provides aggregate results to compare each defender. Column 1 shows the average blue reward for each defender across *all* trials. Comparing the baselines (Rows 1-5) to the learners (Rows 6-15) we can see that the latter tend to accrue a much higher average reward than the former. These results also show that, on average, the *Host-Count* (Rows 10-12) and *Percentile* (Rows 13-15) variations perform better than the *Targeted* (Rows 6-8) variations and the *Count* (Row 9) variation.

However, while the learners average higher than the baselines, they are not *always* better than them, and moreover there is no *single best* Tabular Q-Learning variation (Question 2). Column 2 shows the percentage of scenarios where each defender scored the highest reward – i.e., was *best*: Rows 1 and 2 show that in 8% and 18% of the scenarios *Cred Hunter* and *Zapper* scored the highest,

Table 3: Results and analysis for each defender, with Column 1 showing the average blue reward across all trials for each defender; a larger number indicates better performance, with a range of -1000 to 500. Column 2 shows the percentage of scenarios where that defender scored the highest reward, with Column 3 the percentage where that defender scored within 20 points of the best strategy. Column 4 shows the percentage of scenarios where the defender was above all of the baselines, with Column 5 the percentage where it was above the *Blue-RAND* strategy. For Columns 2-5 the values range from 0 to 100, with a higher number indicating better efficacy. Column 6 is the percentage of scenarios where a negative result fell within a 95% confidence interval of the average, with a lower number indicating better efficacy.

Row ↓	Column → / Defender Strategy	1 Avg.	2 Best	3 Near Best	4 Above Base	5 Above Rand.	6 Loss Chance
1	Cred Hunter	-84	8	30	37	64	50
2	Zapper	-23	18	40	42	74	40
3	Blue Random	-75	0	0	7	100	45
4	Threshold	-284	0	3	12	38	72
5	None	-625	0	0	0	6	100
6	Targeted	73	2	12	12	47	29
7	Host-Targeted	111	1	26	20	71	30
8	Host-Targeted (**h**)	135	1	31	22	73	30
9	Count	150	2	64	40	86	44
10	Host-Count	205	8	67	45	91	44
11	Host-Count (a)	229	10	67	52	93	39
12	Host-Count (**h**)	247	14	70	50	100	30
13	Percentile	232	15	60	48	96	37
14	Percentile (**h**)	229	8	52	42	95	17
15	Percentile, No Creds (**h**)	200	7	38	38	86	7

for a total of *26%* of scenarios where one of the baselines was the strongest. By contrast, the learner that was most frequently the highest scorer among all strategies – *Percentile* (Row 13) – was only the best performer in 15% of all scenarios. Column 3 shows the percentage of scenarios where each agent was *near*-best; i.e., within 20 (~5% of maximum) blue reward points of the best. The *Count*-based variations (Rows 9-12) achieve near the best score in ~67% of the scenarios, leaving a large gap – ~33% – where the learners are not even *near* the best performing defense strategy. Indeed, Column 4 shows the percentage of scenarios each learner was at or above *all* baselines, with the *strongest* learner (*Host-Count (a)*) coming in above the baselines in only 52% of the scenarios. Taken as a whole, while the learners might offer higher average rewards, no one learner *always* outperforms the baselines.

Question 1 instead asks if Tabular Q-Learning can produce an *effective* defender. Column 5 shows the percentage of scenarios where each agent performed the same as or better than *Blue-RAND*. As a baseline metric, *Cred Hunter* (Row 1) and *Zapper* (Row 2) only outperform *Blue-RAND* in 64% and 74% of the scenarios. By contrast, five out of the seven *Count* (Rows 9-12) and *Percentile* (Rows 13-15) variations outperform *Blue-RAND* in 90% or more of the scenarios, with *Host-Count (**h**)* the same as or better in *all* of the scenarios.

Column 6 encodes a metric designed to approximate the likelihood each agent will consistently *lose* in a scenario, showing the percentage of scenarios where a negative score was within a 95% confidence interval of that agent's mean score, with a lower value indicating less likelihood of losing[6]. Within the baselines (Rows 1-5), *Zapper* has the lowest loss likelihood, with a loss being within

[6]Note that for the entire sample size we only consider the cases where a negative score is possible – specifically when the win condition is *30* or *66*.

a 95% confidence interval in 40% of its trials. The *Targeted* variations (Rows 6-8) all have a lower loss likelihood, coming in at ~30%. *Host-Count (**h**)* (Row 12) has the same loss likelihood, but the other *Count* variations (Rows 9-11) all have scores in the ~40% range. The *Percentile* variations have the lowest loss likelihood scores, with *Percentile (**h**)* (Row 14) coming in at 17%, and *Percentile, No Creds (**h**)* (Row 15) at 7%, indicating that they are unlikely to lose.

Summary. Depending on success criteria or desired metric, there are potential Tabular Q-Learning variations that can support ACD. For example, the *Count*-based and *Percentile*-based variations both perform near best relative to the baseline. Secondary decision criteria such as minimizing losing outcomes could favor one of the *Percentile*-based variations without major lost in reward score.

6.1.3 Learners Against Each Other. Between the three learner classes the *Targeted* (Rows 6-8) variations perform the worst and the *Count* variations (Rows 9-12) mostly perform best. The *Percentile* (Rows 13-15) variations perform slightly behind *Count*. This is likely explained by the size of each variation's state space: the *Targeted* variations have the largest state space, and it is unlikely the relevant states are properly explored. By contrast, the *Percentile* variations have the smallest state space, but are most abstract and may lose accuracy-impacting details through that abstraction.

Comparing models that use different host and credential state spaces shows the same pattern: *Host-Targeted* (Row 7) almost uniformly outperforms *Targeted* (Row 6) and *Host-Count* (Row 10) almost uniformly outperforms *Count* (Row 9). In both cases the former uses a more abstract credential representation, with its higher performance likely due to the smaller state space it needs to explore. By contrast, *Percentile (**h**)* (Row 14) mostly outperforms *Percentile, No Creds (**h**)* (Row 15) even with a larger state space, likely due to the latter having a state space that is *too* abstract.

These results also show the efficacy of the implemented loss avoidance strategies. *Host-Targeted (**h**)* (Row 8) almost uniformly outperforms *Host-Targeted* (Row 7), even in cases where losses are not significant. Same for *Host-Count (**h**)* (Row 12), which not only outperforms *Host-Count* (Row 10), but is often the strongest solution. However, *Percentile (**h**)* underperforms against *Percentile* (Row 13), though does score better in loss chance (Column 6).

Summary. These results indicate there is an ideal level of abstraction for state space representation that is compatible with loss avoidance strategies. While none of the variations present as singularly best, *Host-Count (**h**)* (Row 12) and *Percentile (**h**)* (Row 14) are strongest across most of the metrics in Table 3.

6.2 Transfer Learning

Table 4 shows the results for the transfer learning experiments expressed as the percentage difference between the learning agent versus the *Blue-RAND* defender's performance in the same scenario. As examples, Row 1, Column 1 shows that the *Targeted* variation trained in the *Chain-4* topology, *No* noise, and the *No Penalty* win condition scored **361%** better than *Blue-RAND* when evaluated in the same scenario, whereas Row 13, Column 5 shows that the *Host-Count* variation trained in the *Chain-4* topology, *No* noise, and the *No Penalty* win condition scored **the same** as *Blue-RAND* when evaluated in the *Chain-10* topology and the same noise profile and win condition. Note that the bold results show cases where the training and evaluation scenario were the same; these values show a pattern where all agents – aside from *Targeted* – do better than *Blue-RAND*, with the magnitude decreasing as noise increases.

Table 4: Results from transfer learning experiments reported as the percentage change versus the *Blue-RAND* strategy; a score of 0 means the same as *Blue-RAND*, while a negative score denotes worse performance and a positive one a better performance. Values in bold show performance when the training and evaluation scenarios are the same. Rows 31-33 show the average performance across learner variations for each of the noise categories, relative to had the evaluation noise been the same as the training noise.

		Column →	1	2	3	4	5	6	7
Row ↓	Defender	Train Noise	Eval Noise (Same Network)			Eval Network (Same Noise)			Avg.
			None	Low	High	Flat-7	Chn.-10	Strct.	
1	Targeted	None	**361**	1	-43	-15	-46	-48	-41
2		Low	350	**30**	-46	-41	-50	-53	-10
3		High	-56	-37	**-38**	-57	-53	-58	-47
4	Host-Targeted	None	**359**	29	-21	-4	-20	-23	-24
5		Low	280	**211**	-22	-21	-32	-33	-1
6		High	34	69	**76**	-34	-43	-48	-13
7	Host-Targeted (**h**)	None	**358**	38	-19	-3	-21	-23	-23
8		Low	345	**207**	-24	-18	-32	-34	4
9		High	13	67	**76**	-35	-43	-49	-15
10	Count	None	**350**	79	4	119	2	-3	17
11		Low	279	**268**	144	105	-13	-7	58
12		High	65	172	**201**	47	-11	-22	33
13	Host-Count	None	**351**	67	3	105	0	-3	15
14		Low	224	**270**	144	90	3	-1	55
15		High	275	210	**200**	56	-3	-11	58
16	Host-Count (a)	None	**361**	100	9	106	11	15	23
17		Low	314	**281**	149	109	2	2	71
18		High	64	167	**193**	52	-5	-12	34
19	Host-Count (**h**)	None	**361**	102	6	105	12	12	25
20		Low	344	**269**	162	99	1	-1	72
21		High	73	179	**204**	57	-3	-12	38
22	Percentile	None	**361**	95	6	256	262	244	129
23		Low	276	**266**	157	197	188	214	183
24		High	-5	142	**210**	38	187	194	101
25	Percentile (**h**)	None	**360**	98	11	256	262	243	106
26		Low	148	**238**	171	178	156	183	148
27		High	-57	119	**211**	-29	128	176	38
28	Percentile No Creds (**h**)	None	**361**	154	18	336	262	244	167
29		Low	360	**229**	157	162	237	272	213
30		High	43	127	**169**	86	169	184	83
31	Relative Average	None	0	-70	-91	-	-	-	-
32		Low	-19	0	-40	-	-	-	-
33		High	-88	-63	0	-	-	-	-

Column 7 shows the average results for each variation across all evaluated scenarios. Rows 1-9 demonstrate that the *Targeted* variations struggle the most with transfer learning. This is not particularly surprising, as these variations all learn policies that are tightly coupled with the topology they are trained on, so varying the evaluation topology results in using a learned policy that is effectively useless. The *Count* variations (Rows 10-21) do better than *Targeted* and outperform *Blue-RAND* on average, but their performance when training and evaluation are in different scenarios is significantly less than when training and evaluation are the same. The *Percentile* variations (Rows 22-30) perform best, taking a performance hit when transferring to different evaluation scenarios, but still well outperforming *Blue-RAND*.

Columns 4-6 can help explain the rationale for these differences. These columns show performance compared to *Blue-RAND* when the noise profile is the same but the evaluation topology is different.

Matching Column 7, the *Targeted* variations (Rows 1-9) all perform worse than *Blue-RAND* when evaluated on a different topology: because *Targeted* learns policies relevant to the training topology, by varying the evaluation topology the learned policies become useless. The *Count* variations (Rows 10-21) have better, but mixed results: for *Flat-7* they still outperform *Blue-RAND*, but for *Chain-10* and *Structured* they perform roughly the same or worse. This is due to the *Count* variations being coupled with the specific *size* of the training topology: when the evaluation topology has a different number of hosts (i.e., *Chain-10* with 10 and *Structured* with 13), their learned policies are ineffective as they do not have any training data for states above 4 hosts. In contrast to *Targeted* and *Count*, the state space for *Percentile* (Rows 22-30) is the same regardless of topology, and so these policies transfer the best.

The last piece to consider is noise (Columns 1-3) – these columns show the difference versus *Blue-RAND* when the topology is the same but the evaluation noise profile is different than the training noise profile. Rows 31-33 aggregate these results, showing the average – across all learner variations – performance difference between each train and evaluation noise pair, relative to how the variation would have scored if it had been trained and evaluated under the same noise profile and topology. These values suggest that training in the *No* or *High* noise settings and being evaluated in a different noise setting results in a significant performance loss. When the noise condition in evaluation does not match the noise condition in training, the performance drops more than half. Training under the *Low* noise profile and being evaluated under a different profile also takes a performance hit, but not nearly as bad as the other two; if trained under *Low* noise and evaluated in *No* noise the agent would only take ~20% performance hit, and under *High* noise a ~40% hit.

Summary. These results suggest two conclusions for real-world deployment consideration. First, an abstract state representation is more robust to transfer learning situations and should be preferred when deploying to unknown environments. Second, training with slightly more noise than is estimated can hedge performance when noise parameters are unknown.

7 DISCUSSION

Our results show that Tabular Q-Learning is not an *optimal* solution for ACD because no singular configuration can always outperform the baselines in all environments. However, Tabular Q-Learning can be an *effective* solution. This limitation can be partially explained by the MDP formalism. Our *Targeted*-based variations require a significant number of training episodes to overcome the *curse of dimensionality* [29] to solve the MDP, while the more abstract *Count* and *Percentile* variations lose specificity and struggle to learn an optimal policy. Moreover, the choice of MDP as a formalism is not ideal: the defensive agent receives noisy observations which are not a true representation of the network's state. Instead, a *Partially Observable Markov Decision Process* (POMDP) better captures this dynamic [19, 23], allowing the defender to reason over belief states by mapping the observation to potential hidden states. Tabular Q-Learning can provide approximate solutions for small POMDPs, but its efficacy drops as the problem grows [18].

Nonetheless, our results do suggest that an agent using Tabular Q-Learning can provide for an *effective* defender: most of the Tabular Q-Learning variations outperformed the baselines as measured by average reward, proximity to the best-solution, comparison to a random policy, and likelihood of producing a loss-capable policy. We conclude the variations using an abstract state space – i.e., those that were *Count*- and *Percentile*-based – performed strongest; the *Percentile*-based variations in particular also were able to maintain performance when evaluated in unseen environments/scenarios, showing strength for transfer learning. That the abstract variations outperformed the variations using specific state spaces – i.e., the *Targeted*-based variations – is particularly noteworthy as prior work has only considered the latter representation. Moreover, the abstract variations are promising candidates for real-world deployment as defenders can customize the abstraction based on specific network properties and defensive needs.

We believe that these results are important for helping to provide a *foundation* for understanding the research needs within ACD: our detailed and extensively documented analysis helps establish reproducible results for the broader research community to compare to as well as provides data to inform real-world adoption of ACD technologies. While our Tabular Q-Learning variations themselves do not provide for optimal defenders, by presenting our state abstraction ideas and showing their efficacy within Tabular Q-Learning, we help identify promising areas of future research within ACD.

8 FUTURE WORK

We plan to extend this work through four key areas:

Simulator Enhancements. We plan on extending our CyberBattleSim implementation to re-add defensive actions that were originally included in the initial release (i.e., modifying firewall rules, services, and vulnerabilities), as well as a *deception* action, which has been explored in prior work [35] but only from the attacker's perspective. Other extensions include a more detailed noise model, alternative reward structures, fine-grained detection information, and more network topologies.

Implementing State-of-the-Art RL. We plan to use our simulation and testing infrastructure with a more capable RL algorithm – e.g., PPO – to better compare to prior work. As part of this, we also plan to implement, vary, and better quantify the impact of state abstraction with a more capable algorithm. We believe that many of the same patterns will hold true but with much improved results.

Transfer Learning. We consider the problem of transfer learning to be paramount to implementing a real-world autonomous cyber defense system: because training in a production environment is difficult, almost any RL-based autonomous cyber defender will need to train in either a simulated or emulated environment. Regardless of the implementation, there will undoubtedly be differences between the evaluation and training environment which the RL algorithm will need to be able to overcome.

Adversarial Robustness. We concur with [22] in that future autonomous cyber defense systems will need to be robust to adversarial attacks, and similarly to [8] will need to defend against previously-unseen attacker strategies. Our hope is that using an abstract state representation can add robustness for these cases, but more testing is needed to understand trade-offs.

REFERENCES

[1] 2021. Cyber Autonomy Gym for Experimentation Challenge 1. https://github.com/cage-challenge/cage-challenge-1. Created by Maxwell Standen, David Bowman, Son Hoang, Toby Richer, Martin Lucas, Richard Van Tassel.

[2] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. 2002. Finite-time analysis of the multiarmed bandit problem. *Machine learning* 47, 2 (2002), 235–256.

[3] Callum Baillie, Maxwell Standen, Jonathon Schwartz, Michael Docking, David Bowman, and Junae Kim. 2020. Cyborg: An autonomous cyber operations research gym. *arXiv preprint arXiv:2002.10667* (2020).

[4] Deborah Bodeau, Richard Graubart, William Heinbockel, and Ellen Laderman. 2015. *Cyber resiliency engineering aid-the updated cyber resiliency engineering framework and guidance on applying cyber resiliency techniques*. Technical Report. MITRE CORP BEDFORD MA BEDFORD United States.

[5] Curtis Carver, JM Hill, John R Surdu, and Udo W Pooch. 2000. A methodology for using intelligent agents to provide automated intrusion response. In *Proceedings of the IEEE Systems, Man, and Cybernetics Information Assurance and Security Workshop, West Point, NY*. 110–116.

[6] Richard Elderman, Leon JJ Pater, Albert S Thie, Madalina M Drugan, and Marco A Wiering. 2017. Adversarial Reinforcement Learning in a Cyber Security Simulation.. In *ICAART (2)*. 559–566.

[7] Jerzy Filar and Koos Vrieze. 1997. *Competitive Markov decision processes*. Springer-Verlag.

[8] Myles Foley, Chris Hicks, Kate Highnam, and Vasilios Mavroudis. 2022. Autonomous Network Defence using Reinforcement Learning. In *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*. 1252–1254.

[9] Aurélien Garivier and Eric Moulines. 2008. On upper-confidence bound policies for non-stationary bandit problems. *arXiv preprint arXiv:0805.3415* (2008).

[10] Laura Greige and Peter Chin. 2021. Deep Reinforcement Learning for FlipIt Security Game. In *International Conference on Complex Networks and Their Applications*. Springer, 831–843.

[11] Kim Hammar and Rolf Stadler. 2020. Finding effective security strategies through reinforcement learning and self-play. In *2020 16th International Conference on Network and Service Management (CNSM)*. IEEE, 1–9.

[12] Yi Han, Benjamin IP Rubinstein, Tamas Abraham, Tansu Alpcan, Olivier De Vel, Sarah Erfani, David Hubczenko, Christopher Leckie, and Paul Montague. 2018. Reinforcement learning for autonomous defence in software-defined networking. In *International Conference on Decision and Game Theory for Security*. Springer, 145–165.

[13] Dan Horgan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado Van Hasselt, and David Silver. 2018. Distributed prioritized experience replay. *arXiv preprint arXiv:1803.00933* (2018).

[14] IACD. 2020. IACD Playbooks and Workflows. https://www.iacdautomate.org/intro-to-playbooks-and-workflows. [Accessed 2022].

[15] Incident Response Consortium. 2022. Playbook Policy Engine. https://www.incidentresponse.org/playbooks/. [Accessed 2022].

[16] Li Li, Raed Fayad, and Adrian Taylor. 2021. CyGIL: A Cyber Gym for Training Autonomous Agents over Emulated Network Systems. *arXiv preprint arXiv:2109.03331* (2021).

[17] Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Ken Goldberg, Joseph Gonzalez, Michael Jordan, and Ion Stoica. 2018. RLlib: Abstractions for distributed reinforcement learning. In *International Conference on Machine Learning*. PMLR, 3053–3062.

[18] Michael L Littman, Anthony R Cassandra, and Leslie Pack Kaelbling. 1995. Learning policies for partially observable environments: Scaling up. In *Machine Learning Proceedings 1995*. Elsevier, 362–370.

[19] Ashley SM McAbee, Murali Tummala, and John Mceachen. 2021. The use of partially observable Markov decision processes to optimally implement moving target defense. HICSS.

[20] Microsoft Defender Research Team. 2021. CyberBattleSim. *Created by Christian Seifert, Michael Betser, William Blum, James Bono, Kate Farris, Emily Goren, Justin Grana, Kristian Holsheimer, Brandon Marken, Joshua Neil, Nicole Nichols, Jugal Parikh, Haoran Wei* (2021).

[21] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*. PMLR, 1928–1937.

[22] Andres Molina-Markham, Cory Miniter, Becky Powell, and Ahmad Ridley. 2021. Network environment design for autonomous cyberdefense. *arXiv preprint arXiv:2103.07583* (2021).

[23] Scott Musman, Lashon Booker, Andy Applebaum, and Brian Edmonds. 2019. Steps toward a principled approach to automating cyber responses. In *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications*, Vol. 11006. International Society for Optics and Photonics, 110061E.

[24] Lisa Oakley and Alina Oprea. 2019. QFlip: An Adaptive Reinforcement Learning Strategy for the FlipIt Security Game. In *International Conference on Decision and Game Theory for Security*. Springer, 364–384.

[25] Guillermo Owen. 1995. *Game theory*. Academic Press.

[26] Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. 2017. Curiosity-driven exploration by self-supervised prediction. In *International conference on machine learning*. PMLR, 2778–2787.

[27] R. J. Williams. 1987. Reinforcement-learning connectionist systems.

[28] Ahmad Ridley. 2018. Machine learning for autonomous cyber defense. *The Next Wave* 22, 1 (2018), 7–14.

[29] John Rust. 1997. Using randomization to break the curse of dimensionality. *Econometrica: Journal of the Econometric Society* (1997), 487–516.

[30] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).

[31] David Silver and Joel Veness. 2010. Monte-Carlo planning in large POMDPs. *Advances in neural information processing systems* 23 (2010).

[32] Rock Stevens, Daniel Votipka, Josiah Dykstra, Fernando Tomlinson, Erin Quartararo, Colin Ahern, and Michelle L Mazurek. 2022. How Ready is Your Ready? Assessing the Usability of Incident Response Playbook Frameworks. In *CHI Conference on Human Factors in Computing Systems*. 1–18.

[33] Khuong Tran, Ashlesha Akella, Maxwell Standen, Junae Kim, David Bowman, Toby Richer, and Chin-Teng Lin. 2021. Deep hierarchical reinforcement agents for automated penetration testing. *arXiv preprint arXiv:2109.06449* (2021).

[34] Marten Van Dijk, Ari Juels, Alina Oprea, and Ronald L Rivest. 2013. FlipIt: The game of "stealthy takeover". *Journal of Cryptology* 26, 4 (2013), 655–713.

[35] Erich Walter, Kimberly Ferguson-Walter, and Ahmad Ridley. 2021. Incorporating Deception into CyberBattleSim for Autonomous Defense. *arXiv preprint arXiv:2108.13980* (2021).

[36] Wenhao Wang, Dingyuanhao Sun, Feng Jiang, Xingguo Chen, and Cheng Zhu. 2022. Research and Challenges of Reinforcement Learning in Cyber Defense Decision-Making for Intranet Security. *Algorithms* 15, 4 (2022), 134.

[37] Christopher JCH Watkins and Peter Dayan. 1992. Q-learning. *Machine learning* 8, 3 (1992), 279–292.

[38] Christopher John Cornish Hellaby Watkins. 1989. Learning from delayed rewards. (1989).

[39] Saman A Zonouz, Himanshu Khurana, William H Sanders, and Timothy M Yardley. 2013. RRE: A game-theoretic intrusion response and recovery engine. *IEEE Transactions on Parallel and Distributed Systems* 25, 2 (2013), 395–406.

A EXPERIMENTAL PARAMETERS

Table 5 lists parameters used for the experiments in Section 5. These include the specific timesteps for actions like *reimage* and *revoke credentials*, the number of episodes for training, and specific parameters for the Tabular Q-Learning variations (e.g., γ).

Table 5: Simulation parameters held constant across each experiment.

Variable	Value	Variable	Value
Reimage Timesteps	10	Revoke Timesteps	5
Exploration Iterations	75	Normal Iterations	500
Exploration Episodes	700	ϵ-Greedy Episodes	250
Exploitation Episodes	50	Test Episodes	100
γ	0.015	l_r	0.2
Loss Penalty	-1000		