Finding Needles in a Moving Haystack: Prioritizing Alerts with Adversarial Reinforcement Learning

Liang Tong,[1] **Aron Laszka,**[2] **Chao Yan,**[3] **Ning Zhang,**[1] **Yevgeniy Vorobeychik**[1]
[1]Washington University in St. Louis, {liangtong, zhang.ning, yvorobeychik}@wustl.edu
[2]University of Houston, alaszka@uh.edu
[3]Vanderbilt University, chao.yan@vanderbilt.edu

Abstract

Detection of malicious behavior is a fundamental problem in security. One of the major challenges in using detection systems in practice is in dealing with an overwhelming number of alerts that are triggered by normal behavior (the so-called false positives), obscuring alerts resulting from actual malicious activities. We introduce a novel approach for computing a policy for prioritizing alerts using adversarial reinforcement learning. Our approach assumes that the attacker knows the full state of the detection system and the defender's alert prioritization policy, and will dynamically choose an optimal attack. The first step of our approach is to capture the interaction between the defender and attacker in a game theoretic model. To tackle the computational complexity of solving this game to obtain a dynamic stochastic alert prioritization policy, we propose an adversarial reinforcement learning framework. In this framework, we use neural reinforcement learning to compute best response policies for both the defender and the adversary to an arbitrary stochastic policy of the other. We then use these in a double-oracle framework to obtain an approximate equilibrium of the game, which in turn yields a robust stochastic policy for the defender. We use case studies in network intrusion and fraud detection to demonstrate that our approach is effective in creating robust alert prioritization policies.[1]

1 Introduction

One of the core problems in security is *detection* of malicious behavior, such as malicious software and network traffic. There is a vast literature on different detection approaches, ranging from signature-based to machine-learning based (Buczak and Guven 2016; Milenkoski et al. 2015). Under the pressure of practical considerations such as liability and accountability, these systems are often configured to produce a large amount of alerts in order to be sufficiently sensitive to capture most attacks. As a consequence, cybersecurity professionals are routinely inundated with alerts, and must sift through these overwhelmingly uninteresting logs to identify alerts that should be prioritized for closer inspection. A considerable literature has therefore emerged attempting to reduce the number of false alerts without significantly affecting the ability to detect malicious behavior (Hubballi and Suryanarayanan 2014; Salah, Maciá-Fernández, and Díaz-Verdejo 2013; Ho et al. 2017). Most of these attempt to add meta-reasoning on top of detection systems to capture broader system state, combining related alerts, escalating priority based on correlated observations, or using alert correlation to dismiss false alarms. Nevertheless, despite significant advances, there are still vastly more alerts than time to investigate them. With this state of affairs, alert prioritization approaches have emerged, but rely predominantly on predefined heuristics, such as sorting alerts by suspiciousness score or by potential associated risk (Vasilomanolakis et al. 2015).

We propose a novel model and principled computational approach for robust alert prioritization. We model the problem of robust alert prioritization as a game in which the defender chooses a stochastic policy for selecting alerts as a function of observable state, while the attacker chooses which attacks to execute with full knowledge of the system state. Our computational approach first uses neural reinforcement learning to compute approximately optimal policies for either player in response to a fixed stochastic policy of their counterpart. It then uses these (approximate) best response *oracles* as a part of a double-oracle framework. A key technical challenge in this approach is the combinatorial set of defender and attacker actions in each state. We address this challenge by alternatively representing the associated policies as neural networks with continuous outputs, and folding this into an actor-critic framework. While this costs us finite-time theoretical convergence guarantees (since policy space is no longer finite), it enables a practical and highly effective implementation. Furthermore, our case studies in network intrusion and fraud detection using real-world data demonstrate the effectiveness of our approach compared to state-of-the-art alternatives.

Related Work Reinforcement learning using neural networks for value and policy approximation has led to significant recent successes across several applications, most notably game-playing (Mnih et al. 2015; Silver et al. 2016; 2018), and a number of advances in recent years have made such approaches broadly applicable and highly effec-

[1]Code and models are available under an open source license from https://github.com/shinington/AlertPrioritization. Due to space limitation, some content is deferred to an extended version available at https://arxiv.org/abs/1906.08805.

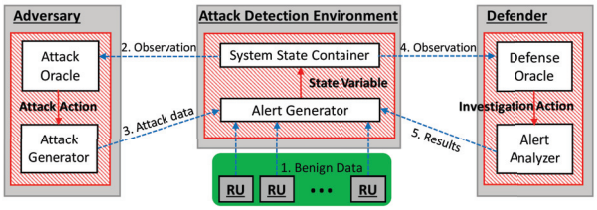

Figure 1: System model. The *Attack Oracle* computes the attacker's policy for executing attacks, which is implemented by the *Attack Generator* and then triggers alerts observed by the *Attack Detection Environment*. The *Defense Oracle* computes the defender's alert prioritization policy, which is implemented by the *Alert Analyzer*.

tive (Van Hasselt, Guez, and Silver 2016; Fortunato et al. 2017). These approaches have been extended to consider problems with continuous actions by Lillicrap et al. (2015), which is most pertinent to our setting. Our work can be viewed as an application of multi-agent reinforcement learning (Littman 1994; Hu and Wellman 2003). While the associated literature is broad, the approaches typically consider unstructured state and action spaces, and would not scale to our problem. Our work can also be viewed as an example of a security game (Sinha et al. 2018), but has a special structure that makes it distinct from other such games. More closely related is recent work on game-theoretic alert prioritization (Laszka et al. 2017; Schlenker et al. 2017; Yan et al. 2018); the key difference is that these consider the game as one-shot, whereas a critical feature of our model is that policies by both the defender and attacker can condition on observed system state. In addition, our work follows prior efforts in cybersecurity in attempting to reduce the number of false positive alerts, for example, through alert correlation (Salah, Maciá-Fernández, and Díaz-Verdejo 2013; Vasilomanolakis et al. 2015); our work is complementary in that it deals with whatever false positives remain after such approaches have been applied.

While our model is novel, our technical approach builds on recent efforts that integrate reinforcement learning with the double-oracle method (Lanctot et al. 2017; Wang et al. 2019). Lanctot et al. consider a more general problem of solving normal-form games, but do not explicitly leverage any special structure such as structured states and actions. Wang et al. use deep Q-learning to take advantage of structured states, but their game involves small action sets. In contrast, one of our principal technical challenges is that the action and state spaces of both players are combinatorial.

2 System Model

2.1 Overview

As displayed in Figure 1, our system is partitioned into four major components: a group of *regular users* (RU), an *adversary* (also called attacker), a *defender*, and an *attack detection environment* (ADE).

The regular users (RU) are the authorized users of a system. In contrast, the adversary is a sophisticated actor who attacks the target computer system. The attack detection environment (ADE) models the combination of the software artifact that is responsible for monitoring the system (e.g., network traffic, files, emails) and raising alerts for observed suspicious behavior, as well as relevant system state. System state includes attacks that have been executed (unknown to the defender), and alerts that have been investigated (known to both the attacker and defender). Crucially, the alerts triggered in the ADE may correspond either to behavior of the normal users RU, or to malicious behavior (attacks) by the adversary. We divide time into a series of discrete time periods. The defender is limited in how many alerts it can investigate in each time period and must select a small subset of alerts for investigation, while the adversary is limited in how many attacks it executes in each time period.

Next, we describe our model of the alert detection environment, our threat model, and our defender model.

2.2 Attack Detection Environment (ADE) Model

Our model of ADE is composed of two parts: an *alert generator* such as an intrusion detection system and *system state*.

An alert generator produces a sequence of alerts in each time period. We aggregate alerts based on a finite predefined set of types T.

At the end of each time period the system generates a collection of alert *counts* for each alert type $t \in T$. We assume that normal or benign behavior generates alerts according to a known distribution $\mathcal{F}$, where $\mathcal{F}_t(n)$ is the marginal probability that n alerts of type t are generated. We also refer to this as the distribution of *false alarms*, since if the defender were omniscient, they would never trigger such alerts. In practise, the distribution $\mathcal{F}$ can be learned by using past logs of *all* alerts over some time period. Since the vast majority of alerts in real systems are in fact false positives, any unidentified true positives in the logs will have a negligible impact.

We use three matrices to represent the state of ADE at time period k. The first represents the counts of alerts not yet investigated, grouped by type. Formally, we denote this structure by $\mathbf{N}^{(k)} = \{N_t^{(k)}\}_{t \in T}$, where $N_t^{(k)}$ is the number of alerts of type $t \in T$ that were raised but have not been investigated by the defender. This is observed by *both* the defender and the attacker. The second describes which attacks have been executed by the adversary; formally, $\mathbf{M}^{(k)} = \{M_a^{(k)}\}_{a \in A}$, where $M_a^{(k)}$ is a binary indicator where $M_a^{(k)} = 1$ iff the attack a was executed, and A is a finite set which represents possible attack actions. This matrix is only observed by the attacker. Finally, we represent which alerts are raised specifically due to each attack. Formally, $\mathbf{S}^{(k)} = \{S_{a,t}^{(k)}\}_{a \in A, t \in T}$, where $S_{a,t}^{(k)}$ represents the number of alerts of type $t \in T$ raised due to attack a. This is also only observed by the attacker.

2.3 Threat Model

Adversary's Knowledge. We consider a strong attacker who is capable of observing the current state of the ADE. Additionally, the attacker knows the randomized *policy* used

by the defender for choosing which alerts to inspect, and inspection decisions in previous rounds, but not the inspection decision in the current round (which happens after the attack).

Adversary's Capabilities. In each time period, the adversary can execute multiple actions a from a set of possible (representative) actions A. Each attack action $a \in A$ stochastically triggers alerts according to the probability distribution P, where $P_{a,t}(n)$ is the marginal probability that action a generates n alerts of type t. These probabilities can be learned by replaying known attack actions through actual detectors (as we do in the experiments in Section 5). Let E_a be the cost of executing an attack $a \in A$. The main limitation to the attacker capabilities is a budget constraint D that limits how many, and which combination of, attacks can be executed. Specifically, any attack decision $\boldsymbol{\alpha}_{-1}^{(k)}$ with $\alpha_{-1,a}^{(k)}$ the indicator that the attack a is executed by the attacker in a given time period k, must abide by the following constraint:

$$\sum_{a \in A} \alpha_{-1,a}^{(k)} E_a \leq D. \tag{1}$$

For our purposes, it is useful to represent the attacker with two modules: *Attack Oracle* and *Attack Generator*, as seen in Figure 1. The attack oracle runs a *policy*, which maps the observed state of the ADE to attacks that are executed. In each time period, after observing ADE state, the attack oracle chooses attack actions, which are then executed by the attack generator, triggering alerts and thereby modifying the state of the ADE.

Adversary's Goals. The adversary aims to successfully execute attacks: if an attack triggers a collection of alerts, but none of these are chosen by the defender to be inspected in the current round, the attack succeeds. Different attacks, however, entail different consequences and, therefore, different rewards to the attacker (and loss to the defender). As a result, the adversary will ultimately need to balance rewards to be gained from successful attacks and the likelihood of being detected. We formalize this in Section 3.

2.4 Defender Model

Defender's Knowledge. Unlike the adversary, the defender can only partially observe the state of the ADE. In particular, the defender only observes $\mathbf{N}^{(k)}$, the numbers of remaining uninvestigated alerts, grouped by alert type (since clearly the defender cannot directly observe actual attacks). In addition, we assume that the defender knows the attack budget and costs of (representative) attacks. In our experiments, we study the impact of relaxing this assumption (see Section 5), and provide practical guidance on this issue.

Defender's Capabilities. The defender chooses subsets of alerts in $\mathbf{N}^{(k)}$ to investigate in each time period k, constrained by the defender's budget B. Formally, let C_t be the investigation cost of an alert of type t, and let $\alpha_{+1,t}^{(k)}$ be the number of alerts of type t chosen to be investigated by the defender in period k. Then the budget constraint takes the following mathematical form:

$$\sum_{t \in T} C_t \alpha_{+1,t}^{(k)} \leq B. \tag{2}$$

An additional constraint imposed by the problem definition is that the defender can only investigate existing alerts:

$$\forall t \in T : \alpha_{+1,t}^{(k)} \leq N_t^{(k)}. \tag{3}$$

Just as with the adversary, it is useful to represent the defender as consisting of two modules: *Defense Oracle* and *Alert Analyzer*, as shown in Figure 1. The defense oracle runs a *policy*, which maps *partially observed* state of the ADE to the choice of a subset of alerts to be investigated. In each time period, after observing the set of as yet uninvestigated alerts, the defense oracle chooses which alerts to investigate, and this policy is then implemented by the alert analyzer, which thereby modifies ADE state (marking the selected alerts as having been investigated).

Defender's Goals. The goal of the defender is to guard a computer system or network by detecting attacks through alert inspection. To achieve its goal, the defender develops an investigation policy to allocate its limited budget to investigation activities in order to minimize consequences of successful attacks. We define this in the next section.

3 Game-Theoretic Model of Robust Alert Prioritization

We now turn to the proposed approach for robust alert prioritization. We model the interaction between the defender and attacker as a *zero-sum* game, which allows us to define and subsequently compute robust stochastic inspection policies for the defender. In this section, we formally describe the game model. We then present the computational approach for solving it in Section 4.

3.1 Strategies

The game has two players: the defender (denoted by $v = +1$) and the adversary (denoted by $v = -1$). Each player's strategies are policies, that is, mappings from an observed ADE state to the probability distribution over actions to take in that state. In a given state, the defender chooses a subset of alerts to investigate; thus, the defender's set of possible actions is the set of all alert subsets that satisfy the constraints (2) and (3). The attacker's choices in a given state correspond to subsets of actions A to take. Consequently, the set of adversary's actions is the set of all subsets of attacks satisfying constraint (1). Note that the combinatorial nature of both players' action spaces and of the state space makes even *representing deterministic policies* non-trivial; we will provide further details on this issue in Section 4. Moreover, we will consider stochastic policies. An equivalent way to represent stochastic policies is as probability distributions over deterministic policies, which map observed state to a *particular* action (subset of alerts for the defender, subset of attacks for the adversary). Henceforth, we call deterministic policies of the players their *pure strategies* and stochastic policies are termed *mixed strategies*, following standard terminology in game theory.

Let $\boldsymbol{\pi}_{-1}$ denote the attacker's policy, which maps the fully observed state of ADE, $\boldsymbol{O}_{-1}^{(k)} = \langle \mathbf{N}^{(k)}, \mathbf{M}^{(k)}, \mathbf{S}^{(k)} \rangle$, to a subset of attacks. Let $\boldsymbol{\alpha}_{-1}^{(k)} = \boldsymbol{\pi}_{-1}(\boldsymbol{O}_{-1}^{(k)})$, where

$\boldsymbol{\alpha}_{-1}^{(k)} = \{\alpha_{-1,a}^{(k)}\}_{a \in A}$ are (for the moment) binary indicators with $\alpha_{-1,a}^{(k)} = 1$ iff an action $a \in A$ is chosen by the attacker. In other words, the vector $\boldsymbol{\alpha}_{-1}^{(k)}$ represents the choice of actions made by the adversary. Similarly, $\boldsymbol{\pi}_{+1}$ denotes the defender's policy, which maps the portion of ADE state $\boldsymbol{O}_{+1}^{(k)} = \mathbf{N}^{(k)}$ observed by the defender to the number of alerts of each type to investigate. Analogous to the attacker, $\boldsymbol{\alpha}_{+1}^{(k)} = \boldsymbol{\pi}_{+1}(\boldsymbol{O}_{+1}^{(k)})$, where $\boldsymbol{\alpha}_{+1}^{(k)} = \{\alpha_{+1,t}^{(k)}\}_{t \in T}$ are the counts of alerts chosen to be investigated for each type t. Note that all alerts of type t are equivalent by definition; consequently, it makes no difference to the defender which of these are chosen, and we can therefore choose the fraction $\frac{\alpha_{+1,t}^{(k)}}{N_t^{(k)}}$ of alerts of type t uniformly at random. This will be significant when we consider policy representation in Section 4.

Let $\boldsymbol{\Pi}_v$ be player v's set of pure strategies, where each pure strategy $\boldsymbol{\pi}_v \in \boldsymbol{\Pi}_v$ is a policy as defined above. A mixed strategy of player v is then a probability distribution $\boldsymbol{\sigma}_v = \{\sigma_v(\boldsymbol{\pi}_v)\}_{\boldsymbol{\pi}_v \in \boldsymbol{\Pi}_v}$ over the player's pure strategies $\boldsymbol{\Pi}_v$ where $\sigma_v(\boldsymbol{\pi}_v)$ is the probability that player v uses policy $\boldsymbol{\pi}_v$. Since a mixed strategy $\boldsymbol{\sigma}_v$ is a distribution over a finite set of pure strategies (if we assume that alert counts are bounded), it satisfies $0 \leq \sigma_v(\boldsymbol{\pi}_v) \leq 1$ and $\sum_{\boldsymbol{\pi}_v \in \boldsymbol{\Pi}_v} \sigma_v(\boldsymbol{\pi}_v) = 1$. Let $\boldsymbol{\Sigma}_v$ denote the set of all mixed strategies of player v.

3.2 Utilities

For any strategy profile of the two players, $(\boldsymbol{\pi}_v, \boldsymbol{\pi}_{-v})$, we denote the utility of each player v by $U_v(\boldsymbol{\pi}_v, \boldsymbol{\pi}_{-v})$. Since our game is *zero-sum*, $\sum_{v \in \{+1,-1\}} U_v(\boldsymbol{\pi}_v, \boldsymbol{\pi}_{-v}) = 0$. When player v chooses pure strategy $\boldsymbol{\pi}_v \in \boldsymbol{\Pi}_v$ and its opponent $-v$ plays mixed strategy $\boldsymbol{\sigma}_{-v} \in \boldsymbol{\Sigma}_{-v}$, then the expected utility of v is

$$U_v(\boldsymbol{\pi}_v, \boldsymbol{\sigma}_{-v}) = \sum_{\boldsymbol{\pi}_{-v} \in \boldsymbol{\Pi}_{-v}} \sigma_{-v}(\boldsymbol{\pi}_{-v}) U_v(\boldsymbol{\pi}_v, \boldsymbol{\pi}_{-v}). \quad (4)$$

Similarly, the expected utility of player v when it chooses the mixed strategy $\boldsymbol{\sigma}_v \in \boldsymbol{\Sigma}_v$ and its opponent plays the mixed strategy $\boldsymbol{\sigma}_{-v} \in \boldsymbol{\Sigma}_{-v}$ is

$$U_v(\boldsymbol{\sigma}_v, \boldsymbol{\sigma}_{-v}) = \sum_{\boldsymbol{\pi}_v \in \boldsymbol{\Pi}_v} \sigma_v(\boldsymbol{\pi}_v) U_v(\boldsymbol{\pi}_v, \boldsymbol{\sigma}_{-v}). \quad (5)$$

Next, we describe how to compute the utility of player v, $U_v(\boldsymbol{\pi}_v, \boldsymbol{\pi}_{-v})$, when its policy is $\boldsymbol{\pi}_v$ and the opponent's policy $\boldsymbol{\pi}_{-v}$ are given.

Consider arbitrary pure strategies of both players, $\boldsymbol{\pi}_{+1}$ and $\boldsymbol{\pi}_{-1}$. The game begins with an initial system state $\langle \boldsymbol{N}^{(0)}, \boldsymbol{M}^{(0)}, \boldsymbol{S}^{(0)} \rangle = \langle \mathbf{0}, \mathbf{0}, \mathbf{0} \rangle$. The system state is then updated in each time period k as follows:

1. *Alert investigation*. The defender first investigates a subset of alerts produced thus far. Specifically, the defender chooses the number of alerts of each type to investigate $\{\alpha_{+1,t}^{(k)}\}_{t \in T}$ according to its policy $\boldsymbol{\pi}_{+1}(\boldsymbol{O}_{+1}^{(k)})$ given current observed state $\boldsymbol{O}_{+1}^{(k)}$. For each attack $a \in A$, let $\widetilde{M}_a^{(k)}$ be an indicator of whether attack a has been executed by the beginning of time period k, but has not been investigated. If $M_a^{(k)} = 0$, we have $\widetilde{M}_a^{(k)} = 0$ as no attack $a \in A$ has been executed. If $M_a^{(k)} = 1$, then $\widetilde{M}_a^{(k)} = 1$ with probability

$$p_a^{(k)} = \prod_{t \in T} \left\{ \frac{C(N_t^{(k)} - S_{a,t}^{(k)}, \alpha_{+1,t}^{(k)})}{C(N_t^{(k)}, \alpha_{+1,t}^{(k)})} \right\}, \quad (6)$$

   where $C(n, r)$ is the number of possible combinations of r objects from a set of n objects. $p_a^{(k)}$ is then the probability that attack a is not detected by the defender.
2. *Attack generation*. The adversary produces attacks by executing actions according to its policy $\{\alpha_{-1,a}^{(k)}\}_{a \in A} = \boldsymbol{\pi}_{-1}(\boldsymbol{O}_{-1}^{(k)})$ given the fully observed ADE state $\boldsymbol{O}_{-1}^{(k)}$. Then $M_a^{(k+1)} = \alpha_{-1,a}^{(k)}$ for each $a \in A$.
3. *Triggering alerts*. Each attack $a \in A$ can trigger alerts as follows. For each attack $a \in A$ and alert type $t \in T$, if $M_a^{(k+1)} = 1$, then $S_{a,t}^{(k+1)} = n$ with probability $P_{a,t}(n)$ for $n \geq 0$. This probability can be estimated, for example, by feeding inputs which include representative attacks into an attack detector and observing relative frequencies of alerts that are triggered. In addition, false alerts are generated according to the distribution $\mathcal{F}_t$, which we can estimate from data of normal behavior and associated alert counts. Let $f_t^{(k)}$ be the number of false alerts of type $t \in T$ that have been generated. Then the total number of alerts of type t in the next time period $k+1$ is $N_t^{(k+1)} = f_t^{(k)} + S_{a,t}^{(k+1)}$.

In order to define the reward received by the defender in time period k, we make the following assumption: *if any of the alerts raised by an attack is chosen to be inspected, then the attack is detected; otherwise, the attack is not detected.* Let L_a be the loss incurred by the defender when an attack $a \in A$ is not detected. Then the reward of the defender obtained in time period k is

$$R_{+1}^{(k)} = -\sum_{a \in A} L_a \cdot \widetilde{M}_a^{(k)}. \quad (7)$$

For an arbitrary pure strategy profile of the defender and adversary, $(\boldsymbol{\pi}_{+1}, \boldsymbol{\pi}_{-1})$, the defender's utility from the game is the expected total discounted sum of the reward accrued in each time period:

$$U_{+1}(\boldsymbol{\pi}_{+1}, \boldsymbol{\pi}_{-1}) = \mathbb{E}\left[\sum_{k=0}^{\infty} \tau^k \cdot R_{+1}^{(k)}\right], \quad (8)$$

where $\tau \in (0, 1)$ is a temporal discount factor which implies that future rewards are less important than current rewards. That is, imminent losses are more important to the defender than potential future losses. The adversary's utility is then $U_{-1}(\boldsymbol{\pi}_{+1}, \boldsymbol{\pi}_{-1}) = -U_{+1}(\boldsymbol{\pi}_{+1}, \boldsymbol{\pi}_{-1})$.

3.3 Solution Concept

Our goal of finding robust alert investigation policies amounts to computing a *mixed-strategy Nash equilibrium*

(MSNE) of our game by the well-known equivalence between MSNE, maximin, and minimax solutions in zero-sum games (Korzhyk et al. 2011). A mixed-strategy profile $(\boldsymbol{\sigma}_v^*, \boldsymbol{\sigma}_{-v}^*)$ of the two players is an MSNE if it satisfies the following condition for all $v \in \{+1, -1\}$

$$U_v(\boldsymbol{\sigma}_v^*, \boldsymbol{\sigma}_{-v}^*) \geq U_v(\boldsymbol{\sigma}_v, \boldsymbol{\sigma}_{-v}^*) \ \ \forall \boldsymbol{\sigma}_v \in \boldsymbol{\Sigma}_v. \tag{9}$$

That is, each player v chooses a stochastic policy $\boldsymbol{\sigma}_v^*$ that is the *best response* (is optimal for v) when its opponent chooses $\boldsymbol{\sigma}_{-v}^*$.

4 Computing Robust Alert Prioritization Policies

4.1 Solution Overview

For given sets of policies, $\boldsymbol{\Pi}_{+1}$ and $\boldsymbol{\Pi}_{-1}$, a standard approach to computing the MSNE of a zero-sum game is to solve a linear program of the following form:

$$\begin{array}{ll} \max & U_v^* \\ \text{s.t.} & U_v(\boldsymbol{\sigma}_v, \boldsymbol{\pi}_{-v}) \geq U_v^*, \ \forall \boldsymbol{\pi}_{-v} \in \boldsymbol{\Pi}_{-v} \\ & \sum_{\boldsymbol{\pi}_v \in \boldsymbol{\Pi}_v} \sigma_v(\boldsymbol{\pi}_v) = 1 \\ & \sigma_v(\boldsymbol{\pi}_v) \geq 0, \qquad \forall \boldsymbol{\pi}_v \in \boldsymbol{\Pi}_v \end{array} \tag{10}$$

where in our case the optimal solution $\boldsymbol{\sigma}_{+1}^*$ yields the robust alert prioritization policy for the defender. However, our problem entails two principal technical challenges: 1) the space of policies for both players is intractably large, and 2) it is even intractable to explicitly represent individual policies, since they map a combinatorial set of states to a combinatorial set of actions for both players.

To address these challenges, we propose an adversarial reinforcement learning approach which builds on several recent efforts (Lanctot et al. 2017; Wang et al. 2019) that combine a *double oracle* framework (McMahan, Gordon, and Blum 2003) with neural reinforcement learning. We start with an arbitrary small collection of policies for both players, $(\boldsymbol{\Pi}_{+1}, \boldsymbol{\Pi}_{-1})$, and solve the linear program (10), obtaining provisional equilibrium mixed strategies $(\boldsymbol{\sigma}_{+1}, \boldsymbol{\sigma}_{-1})$ of the restricted game. Next, we query the attack oracle to compute the adversary's best response $\boldsymbol{\pi}_{-1}(\boldsymbol{\sigma}_{+1})$ to the defender's equilibrium mixed strategy $\boldsymbol{\sigma}_{+1}$, and, similarly, query the defense oracle to compute the defender's best response $\boldsymbol{\pi}_{+1}(\boldsymbol{\sigma}_{-1})$ to the adversary's equilibrium mixed strategy $\boldsymbol{\sigma}_{-1}$. The best response policies are then added to the policy sets $(\boldsymbol{\Pi}_{+1}, \boldsymbol{\Pi}_{-1})$ of the players, and we then resolve the linear program and repeat the process. The process stops when neither player's best response policy yields appreciable improvement in utility compared to the provisional equilibrium mixed strategy (we provide further details in Section IV.B of the extended version). Note that even though the double-oracle approach converges in finite time for finite games, this is a vacuous guarantee in our case, where policy spaces are enormous. Moreover, as we describe below, our policy representation in fact induces an infinite space of policies. However, in our experiments the procedure converged in fewer than 15 iterations (see Figure 12 in the extended version).

The main question that remains is how to compute or approximate the best response oracles for both players. To this end, we use reinforcement learning techniques with policies represented using neural networks. Below, we explain our neural reinforcement learning methods, including the specific way in which we represent policies, in further detail.

4.2 Approximate Best Response Oracles with Neural Reinforcement Learning

We now turn to our approach to compute $\boldsymbol{\pi}'_v$, the (approximately) optimal response of player v when its opponent uses a mixed strategy $\boldsymbol{\sigma}'_{-v}$ such that

$$\boldsymbol{\pi}'_v = \arg\max_{\boldsymbol{\pi}_v} U_v(\boldsymbol{\pi}_v, \boldsymbol{\sigma}'_{-v}). \tag{11}$$

This problem poses a major technical challenge, since the spaces of possible policies for both the defender and the attacker are quite large. To address this, we propose using the reinforcement learning (RL) paradigm. However, the use of RL poses two further challenges in our setting. First, for a given state, each player's set of possible actions is combinatorial: the attacker is choosing subsets of attacks, whereas the defender is choosing subsets of alerts. Consequently, we cannot use common methods such as *Q-learning* (as done by Wang et al.), which requires explicitly representing the action-value function $Q(x, a)$ for every possible action a, even if we approximate this function over states x using, e.g., a neural network, as is common in deep RL. We can address this issue by appealing to *actor-critic* methods for RL, where the policy is represented as a parametric function $\boldsymbol{\pi}_{v;\theta}$ with parameters θ. However, this brings up the second challenge: actor-critic approaches learn policies using gradient-based methods, which require that the actions are continuous. In our case, however, the actions are discrete.

One solution is to learn the action-value function $Q(x, a)$ over a vector-based representation of actions, such as using a binary vector to indicate which attacks are used. The problem with this approach, however, is that the resulting policy $\boldsymbol{\pi}_v \in \arg\max_{a \in A} Q(x, a)$ is hard to compute in real time, since it involves a combinatorial optimization problem in its own right. We therefore opt for a much more scalable solution that uses the actor-critic paradigm with an alternative representation of the adversary and defender policies, which admits gradient-based learning.

First, consider the adversary. Recall that the adversary's policy maps a state to a subset of attack actions A, with the constraint on the total budget used by the chosen actions. Instead of returning a discrete subset of actions, we map the adversary's policy to a *probability distribution* over actions, overloading our prior notation so that $\alpha_{-1,a}^{(k)}$ now denotes the *probability* that action $a \in A$ is executed. Now the policy can be used with actor-critic methods, but it may violate the budget constraint. To address this final issue, we simply project the probability distribution into the feasible space at execution time by normalizing it using the total cost of the distribution, and then multiplying it by the budget constraint. Notice that in this process we have relaxed the attacker's budget constraint to hold only in *expectation*; however, this only makes the attacker stronger. An interesting side-effect of our transformation of the adversary's policy space is that the RL method will now effectively search in the space of

stochastic adversary policies. An associated benefit is that it leads to faster convergence of the double oracle approach.

Next, consider the defender. In this case, we can represent the policy as a mapping to fractions of the *total defense budget* allocated to each alert type t. In other words, for each alert type t, the policy will output the maximum fraction of the defense budget that will be used to inspect alerts of type t. This makes the mapping continuous, and also obviates the need to explicitly deal with the budget constraint.

The final nuance is that RL methods are typically designed for a fixed environment, whereas our setting is a game. However, note that since we are concerned only with each player's best response to the other's mixed strategy, we can embed the mixed strategy of the opponent as a part of the environment. Next, we describe our application of actor-critic methods to our problem, given the alternative representations of adversary and defender policies above.

The basic idea of the actor-critic method is that we can iteratively learn and improve a policy without enumerating actions by using two parallel processes that interact with each other: an actor network which develops a policy, and a critic network which evaluates the policy. Specifically, we propose *DDPG-MIX*, an actor-critic algorithm that operates in continuous action spaces and computes an approximate best response to an opponent who uses a stochastic policy. DDPG-MIX is an extension of the *Deep Deterministic Policy Gradient (DDPG)* approach proposed by Lillicrap et al. (2015) to our setting, and the full algorithm is described in the extended version (Algorithm 1).

4.3 Preprocessing

An important consideration in applying the above approaches is scalability of training. One way to significantly improve scalability is through preprocessing, and pruning alerts for which the (near-)optimal decision is obvious. We use the following pruning step to this end. Suppose that there is an alert type t which is generated by benign traffic with probability at most ϵ, where ϵ is very small (for example, $\epsilon = 0$, in which case alerts of type t *never* correspond to a false positive). In most realistic cases, it is nearly optimal to always inspect such alerts. Consequently, we prune all alerts with false positive rate below a small pre-defined ϵ (in our implementation, we set $\epsilon = 0$), and mark them for inspection, correspondingly reducing the available budget for inspecting other alerts.

5 Case Studies

In this section, we present case studies to investigate the robustness of our proposed approach for alert prioritization. We conduct our experiments in two applications: network intrusion detection, which employs a signature-based detection system, and learning-based fraud detection.

5.1 Experimental Methodology

We use the expected loss (negative utility) of the defender (equivalently, utility of the adversary) as the metric throughout our evaluation. Specifically, for a given defense policy, we evaluate the loss of the defender using several models of the adversary. First, we used the proposed *DDPG-MIX* algorithm, to compute the best response of the adversary, as anticipated by our approach. In addition, to evaluate the general robustness of our approach, we employed two alternative policies for the adversary: *Uniform*, a policy which uniformly distributes the adversary's budget over attack actions; and *Greedy*, a policy which allocates the budget to attacks in the order of expected adversary utility. Specifically, the *Greedy* adversary prioritizes the attack actions according to $L_a \cdot \min\{\frac{\tilde{D}}{c_a}, 1\}$, where $\tilde{D}$ is the available attack budget, adding actions in this priority order until the adversary's budget is exhausted. The implementation is detailed in Section V.A of the extended version.

We compare our approach for alert prioritization with several baselines. The first is *Uniform*, a policy which uniformly allocates the defender's budget over each alert type. Additionally, where feasible, we compare our approach with two baseline methods for game theoretic alert prioritization: *GAIN* (Laszka et al. 2017) and *RIO* (Yan et al. 2018). In both, neither player can observe the ADE state, and defender's policy is a probability distribution over alert type orderings, with *RIO* also optimizing inspection budgets for each alert type.

We first conduct our experiments by assuming that the defender knows the adversary's capabilities. Subsequently, we evaluate the robustness of our approach when the defender is uncertain about the adversary's capabilities. We also provide results on the computational cost of our approach in the extended version (Figure 12).

5.2 Case Study I: Network Intrusion Detection

Our first case study involves a signature-based network intrusion detection scenario, using Suricata[2], a state-of-the-art open source network intrusion detection system (NIDS), combined with the CICIDS2017 dataset (Sharafaldin, Habibi Lashkari, and Ghorbani 2018). The details of experimental setup are included in Section V.B of the extended version. The performance of the proposed approach is compared with two alternative policies for alert prioritization: *Uniform*, a policy which uniformly allocates the defender's budget over alert types, and *Suricata* priorities, where the defender exhausts the defense budget according to the built-in prioritization of the Suricata NIDS. Both *GAIN* and *RIO* fail to scale computationally to the size of our NIDS case study. Throughout, we refer to our proposed approach as *ARL*.

Results Figure 2 presents our evaluation of the robustness of alert prioritization approaches when the defender knows the adversary's capabilities, and the results suggest that our approach significantly outperforms the other baselines. Specifically, the proposed approach is 50% better than the *Uniform* policy, which in turn is significantly better than using Suricata priorities. There are a few reasons why deterministic priority-based approaches perform so poorly. First, determinism allows attackers to easily circumvent the policy by focusing on attacks that trigger alerts which are rarely inspected. Moreover, such naive deterministic policies also fail

[2]Available at https://suricata-ids.org/about/open-source/.

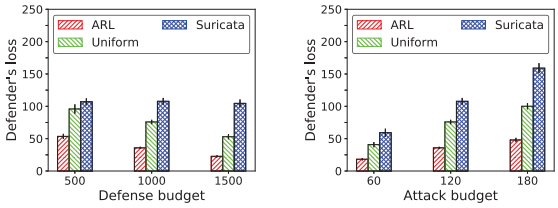

Figure 2: Network intrusion detection: loss of the defender when it knows the attack budget. Left: Defender's loss for different defense budgets, with attack budget fixed at 120. Right: Defender's loss for different attack budgets, with defense budget fixed at 1000.

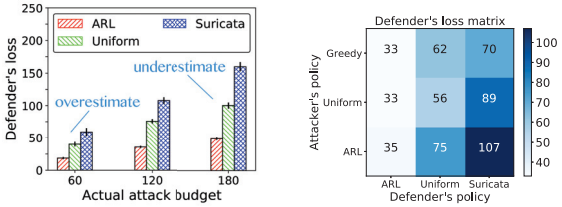

Figure 3: Network intrusion detection: loss of the defender when it is uncertain about the adversary's capabilities. The defense budget is fixed at 1000. Left: The defender is uncertain about the attack budget, and its estimate of the attack budget is 120. Right: The defender is uncertain about the attack policies but knows the attack budget being fixed at 120.

to exploit the empirical relationships between attacks and alerts they tend to trigger. In contrast, by learning a *policy* of alert inspection which maps arbitrary alert observations to a decision about which to inspect, we can make decisions at a significantly finer granularity.

Evaluating the alert prioritization methods when the defender is uncertain about the attack budget (Figure 3 (left)), we can observe that the proposed *ARL* approach still achieves the lowest defender loss both when the attack budget is underestimated and overestimated, and it is still far better than the baselines. In addition, Figure 3 (right) shows that although we assume a very strong adversary, our *ARL* approach significantly outperforms the other baselines even when the adversary is using a different attack policy.

5.3 Case Study II: Fraud Detection

Our second case study involves a learning-based fraud detector developped by using supervised learning on the fraud dataset[3]. The details of experimental setup are included in Section V.C of the extended version. We investigate the performance of the proposed approach by comparing with *Uniform*, *GAIN*, and *RIO* alert prioritization policies.

Results Figure 4 shows the results when the defender has full knowledge of the adversary's capabilities. We can observe that the proposed approach (*ARL*) outperforms other baselines in all settings, typically by at least 25%. The main

[3]Available at https://www.kaggle.com/mlg-ulb/creditcardfraud.

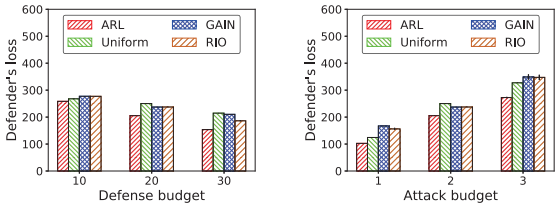

Figure 4: Fraud detection: loss of the defender when it knows the attack budget. Left: Defender's loss for different defense budgets, with attack budget fixed at 2. Right: Defender's loss for different attack budgets, with defense budget fixed at 20.

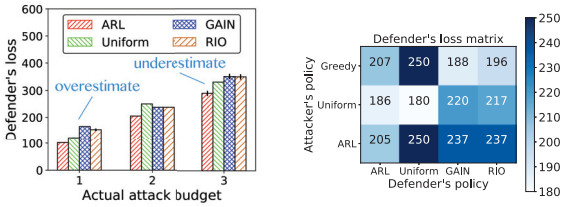

Figure 5: Fraud detection: loss of the defender when it is uncertain about the adversary's capabilities. The defense budget is fixed at 20. Left: The defender is uncertain about the attack budget, and its estimate of the attack budget is 2. Right: The defender is uncertain about the attack policies but knows the attack budget being fixed at 2.

reason for the advantage is similar to that in the NIDS setting: the ability to have a policy that is carefully optimized and conditional on state significantly increases its efficiency. Interestingly, the alternative game-theoretic alert prioritization approaches, *GAIN* and *RIO*, are in some cases worse than the uniformly random policy. The key reason is that they can be myopic in that they do not consider observed system state in the decision about which alerts to prioritize, and such context can be crucial.

Figure 5 (left) investigates the performance of our approach when the attack budget is uncertain. It can be seen that *ARL* remains the best approach to use, despite this uncertainty. Figure 5 (right) presents our evaluation of the robustness of *ARL* compared to other baselines when the attacker is using different policies (*Uniform* or *Greedy*) instead of the RL-based policy that is assumed by our approach. Here, the results are slightly more ambiguous than we observed in the NIDS domain: when the adversary is using the *Greedy* policy, *RIO* outperforms ARL by 7%. However, in this case, the adversary can gain a great deal by more carefully designing its policy. Thus, a rational adversary can cause *RIO* to degrade by nearly 21%, where *ARL* is quite robust to such adversaries.

6 Conclusion

We introduced a general model of alert prioritization, and proposed a novel double oracle and reinforcement learning based approach for finding approximately optimal prioriti-

zation policies efficiently. Our experimental results based on case studies with a signature-based network intrusion detection system and machine learning-based fraud detection demonstrate that these policies significantly outperform non-strategic approaches in nearly all cases, and prior strategic methods where these are feasible, even when the assumptions of our threat model are violated.

Acknowledgements

This work was partially supported by the National Science Foundation (grant IIS-1905558, CNS-1916926, and CNS-1837519), Army Research Office (grant W911NF1910241), Air Force Office of Scientific Research (grant FA95501810126), and NVIDIA.

References

Buczak, A. L., and Guven, E. 2016. A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Communications Surveys & Tutorials* 18(2):1153–1176.

Fortunato, M.; Azar, M. G.; Piot, B.; Menick, J.; Osband, I.; Graves, A.; Mnih, V.; Munos, R.; Hassabis, D.; Pietquin, O.; et al. 2017. Noisy networks for exploration. *arXiv preprint arXiv:1706.10295*.

Ho, G.; Sharma, A.; Javed, M.; Paxson, V.; and Wagner, D. 2017. Detecting credential spearphishing in enterprise settings. In *USENIX Security Symposium*, 469–485.

Hu, J., and Wellman, M. P. 2003. Nash Q-learning for general-sum stochastic games. *Journal of Machine Learning Research* 4:1039–1069.

Hubballi, N., and Suryanarayanan, V. 2014. False alarm minimization techniques in signature-based intrusion detection systems: A survey. *Computer Communications* 49:1–17.

Korzhyk, D.; Yin, Z.; Kiekintveld, C.; Conitzer, V.; and Tambe, M. 2011. Stackelberg vs. Nash in security games: An extended investigation of interchangeability, equivalence, and uniqueness. *Journal of Artificial Intelligence Research* 41:297–327.

Lanctot, M.; Zambaldi, V.; Gruslys, A.; Lazaridou, A.; Tuyls, K.; Pérolat, J.; Silver, D.; and Graepel, T. 2017. A unified game-theoretic approach to multiagent reinforcement learning. In *Neural Information Processing Systems*, 4193–4206.

Laszka, A.; Vorobeychik, Y.; Fabbri, D.; Yan, C.; and Malin, B. 2017. A game-theoretic approach for alert prioritization. In *AAAI Workshop on AI for Cyber Security*.

Lillicrap, T. P.; Hunt, J. J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; and Wierstra, D. 2015. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

Littman, M. L. 1994. Markov games as a framework for multi-agent reinforcement learning. In *International Conference on International Conference on Machine Learning*, 157–163.

McMahan, H. B.; Gordon, G. J.; and Blum, A. 2003. Planning in the presence of cost functions controlled by an adversary. In *International Conference on Machine Learning*, 536–543.

Milenkoski, A.; Vieira, M.; Kounev, S.; Avritzer, A.; and Payne, B. D. 2015. Evaluating computer intrusion detection systems: A survey of common practices. *ACM Computing Surveys* 48(1):12.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518(7540):529.

Salah, S.; Maciá-Fernández, G.; and Díaz-Verdejo, J. E. 2013. A model-based survey of alert correlation techniques. *Computer Networks* 57(5):1289–1317.

Schlenker, A.; Xu, H.; Guirguis, M.; Kiekintveld, C.; Sinha, A.; Tambe, M.; Sonya, S.; Balderas, D.; and Dunstatter, N. 2017. Don't bury your head in warnings: A game-theoretic approach for intelligent allocation of cyber-security alerts. In *International Joint Conference on Artificial Intelligence*, 381–387.

Sharafaldin, I.; Habibi Lashkari, A.; and Ghorbani, A. A. 2018. Toward generating a new intrusion detection dataset and intrusion traffic characterization. In *International Conference on Information Systems Security and Privacy*, 108–116.

Silver, D.; Huang, A.; Maddison, C.; ...; and Hassabis, D. 2016. Mastering the game of go with deep neural networks and tree search. *Nature* 529:484–489.

Silver, D.; Hubert, T.; Schrittwieser, J.; ...; and Hassabis, D. 2018. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science* 6419:1140–1144.

Sinha, A.; Fang, F.; An, B.; Kiekintveld, C.; and Tambe, M. 2018. Stackelberg security games: Looking beyond a decade of success. In *International Joint Conference on Artificial Intelligence*.

Van Hasselt, H.; Guez, A.; and Silver, D. 2016. Deep reinforcement learning with double Q-learning. In *AAAI Conference on Artificial Intelligence*.

Vasilomanolakis, E.; Karuppayah, S.; Mühlhäuser, M.; and Fischer, M. 2015. Taxonomy and survey of collaborative intrusion detection. *ACM Computing Surveys* 47(4):55.

Wang, Y.; Shi, Z. R.; Yu, L.; Wu, Y.; Singh, R.; Joppa, L.; and Fang, F. 2019. Deep reinforcement learning for green security games with real-time information. In *AAAI Conference on Artificial Intelligence*.

Yan, C.; Li, B.; Vorobeychik, Y.; Laszka, A.; Fabbri, D.; and Malin, B. 2018. Get your workload in order: Game theoretic prioritization of database auditing. In *IEEE International Conference on Data Engineering*, 1304–1307.