Towards a Game-theoretic Framework for Intelligent Cyber-security Alert Allocation*

Aaron Schlenker[1], Haifeng Xu[1], Chris Kiekintveld[2], Arunesh Sinha[3], Milind Tambe[1], Mina Guirguis[4], Solomon Sonya[5], Darryl Balderas[4], and Noah Dunstatter[4]**

[1] University of Southern California
[2] Texas State University
[3] University at Texas El Paso
[4] University of Michigan
[5] United States Air Force Academy

Abstract. In recent years, there have been a number of successful cyber attacks on enterprise networks by malicious actors. These attacks generate alerts which must be investigated by cyber analysts to determine if they are an attack. Unfortunately, there are magnitude more alerts than cyber analysts - a trend expected to continue into the future creating a need to find optimal assignments of the incoming alerts to analysts in the presence of a strategic adversary. We address this challenge with the four following contributions: (1) a *cyber allocation game* (CAG) model for the cyber network protection domain, (2) an NP-hardness proof for computing the optimal strategy for the defender, (3) techniques to find the optimal allocation of experts to alerts in CAG in the general case and key special cases, and (4) heuristics to achieve significant scale-up in CAGs with minimal loss in solution quality.

1 Introduction

Automated intrusion detection and prevention systems (IDPS) and security information and event management tools (SIEM) are important for computer network security. The alerts generated by these systems must be investigated by human cybersecurity analysts to assess whether they were generated by malicious activity, and if so, how to respond. Unfortunately, these automated systems are notorious for generating high rates of false positives [16]. Expert analysts are in short supply, so organizations face a key challenge in managing the enormous volume of alerts they receive using the limited time of analysts. Failing to solve this problem can render the entire system insecure, e.g., in

* A different version of this paper is accepted for publication in the IJCAI main track. This submission has the following new material: (1) We present an algorithm which gives a $\frac{1}{2}$ approximation of the defender's optimal utility. (2) We add a geometric interpretation of the main algorithm to provide more intuition to the reader. (3) More experimental results have been added to show scalability of our algorithms. (4) We have added discussion of future work to be completed to apply our model to the real world.

** [1]{aschlenk, haifengx, tambe}@usc.edu, [2]cdkiekintveld@utep.edu, [3]arunesh@umich.edu, [4]{msg, d_b118, nfd8}@txstate.edu, [5]solomon.sonya@usafa.edu

the 2013 attack on Target, IDPS raised alarms, but they were missed in the deluge of alerts [14].

There are many approaches for mitigating this problem by reducing the number of alerts. IDPS can be carefully configured, alert thresholds can be tuned, and the classification methods underlying the detections can be improved [15, 4, 11]. Other techniques include aggregating alerts [19], and visualizing alerts for faster analysis [13]. Even when using all of these techniques, there are still too many alerts for the analysts to investigate all of them in depth. Our work focuses on the remaining problem of assigning limited analysts to investigate alerts *after* automated pre-processing methods have been applied.

The typical approach to managing alerts is either ad-hoc or uses the obvious strategy of looking only at the alerts with the highest priority (e.g., risk). A recent work [7] used decision theory to optimize the scheduling of cyber-security analysts for screening alerts over multiple time periods. However, the heuristic approaches and [7] fail to account for the adversarial nature of the cyber security setting. An attacker who can guess or learn about a predictable alert management policy can exploit this knowledge to launch a successful attack. For example, if we had a policy that only inspects alerts from high valued assets for our organization, an attacker who can learn this will evade detection indefinitely by only conducting activities on lower valued assets.

To address this shortcoming of the previous method, our first contribution is a *Cyber-alert Allocation Game* (CAG), a game-theoretic model for optimizing the assignment of cyber alerts to a limited group of cyber analysts. Using game theory allows us to explicitly model the strategies an attacker with knowledge of the assignment policy could take to avoid detection. By following a randomized, unpredictable assignment strategy the defender can improve the effectiveness of alert assignments against strategic attackers. Our model considers the characteristics of the alerts (e.g., criticality of origin system), as well as the capabilities of the analysts in formulating the optimal policy for the defender. Our approach draws on the principles and modeling techniques of a large body of work that applies game theory to security problems [17]. However, CAG significantly differs from traditional security games [17, 9] due to the absence of an explicit set of targets, a large number of benign alerts and varying time requirements for inspections.[6]

Our second contribution in this paper is to show that finding the optimal strategy for a CAG is NP-hard, posing a major computational challenge. Third, we present an algorithm for finding optimal, implementable CAG policies. Fourth, we devise novel heuristics to solve large CAGs, and we provide empirical evaluation of our algorithms and model.

2 Motivating Domain

While many organizations face the challenge of cyber alert allocation, we highlight a scenario developed in consultation with experts at the United States Air Force (USAF). The Air Force Cyber defense unit (AFCYBER) is responsible investigating and resolv-

[6] A more thorough discussion of related work is provided in the IJCAI main track paper.

ing alerts generated by IDPS which prevent attacks on their cyber systems [2]. Prescreening of the alerts eliminates a large fraction of insignificant events, but thousands remain to be investigated. Any of these remaining alerts could indicate a malicious attack, but a large fraction are false positives.

Two primary features are used to prioritize the most critical alerts to investigate. First, each alert has a risk classification (e.g., high, medium, low) based on the type of event detected by the IDPS. Second, each alert has an origin location within the global network (e.g., a specific host, system); some locations (e.g., headquarters) are more critical to operations. The AFCYBER has a limited number of Incident Response Team (IRT) cyber analysts who investigate significant alerts after prescreening [1]. Each analyst has different areas of expertise, and may therefore be more effective and/or faster at investigating certain types of incidents. The USAF also must protect against an adaptive adversary who can observe strategies through beaconing and other techniques. The problem AFCYBER faces is an excellent example of our central analyst assignment problem in the real world.

3 Cyber-alert Allocation Games

We model the *Cyber-alert Allocation Game* (CAG) as a (zero-sum) Stackelberg game played between the defender (e.g., AFCYBER) and an adversary (e.g., hacker). The defender commits to a mixed strategy to assign alerts to cyber analysts. We make the worst-case assumption that the attacker moves with complete knowledge of the defender's strategy and plays a best-response attack strategy [10]. However, in a zero-sum game the optimal strategy for the defender is the same as the Nash equilibrium (i.e., when the attacker moves simultaneous) [18], so the order of the moves is *not* consequential in the model.

Systems and Alerts: The defender responds to alerts originating from a set of systems $k \in K$. A "system" in our model could represent any level of abstraction, ranging from a specific server to a complete network.IDPS for each system generate alerts of different types, $a \in A$. The alert types correspond to levels of severity (e.g., high, medium, and low), reflecting the likelihood of a malicious event. We represent the combination of the alert type and the origin system as an alert category, $c \in C$, where $c = (k, a)$. The alerts in a given category are not differentiable, so the defender must investigate all alerts within a category with the same probability. The total number of alerts for a given category c is denoted by N_c. We assume that the both the defender and attack know the typical value of N_c from historical averages (similar to [7]).

Attack Methodologies: Attackers can choose from many attack methodologies. These fall into high-level categories such as denial of service attacks, malware, web exploitation, or social engineering. We represent these broad classes of attacks as attack methods $m \in M$. For every attack method there is a corresponding probability distribution β_a^m which represents the probability that the IDPS generates an alert of type a for an attack method m. For example, if the attacker chooses $m = DoS$ the corresponding alert probabilities could be $\beta_{High}^{DoS} = .8$, $\beta_{Medium}^{DoS} = .15$ and $\beta_{Low}^{DoS} = .05$.

Cybersecurity Analysts: Cybersecurity analysts R are assigned to investigate alerts. The time required for an analyst to resolve an alert type a varies, and is represented

by T_a^r. Intuitively, T_a^r represents the portion of a time period that an analyst needs to resolve an alert of type a. A time period may be a shift, an hour or other fixed scheduling period. For example, if an analyst needs half a time period to resolve a, then $T_a^r = 0.5$. In our model: $T_a^r \leq 1, \forall\, a \in A$, i.e., an analyst can address multiple alerts within a time period. In addition to T_a^r, we allow modeling of the effectiveness of an analyst against an attack method, representing her expertise, via a parameter E_m^r.

Defender Strategies: A pure strategy P for the defender is a non-negative matrix of *integers* of size $|C| \times |R|$. Each c,r entry is the number of alerts in category c assigned to be investigated by cyber analyst r, denoted by $P_{c,r}$. The set of all pure strategies $\hat{P}$ is all allocations that satisfy the following constraints; C_a denotes all categories with the alert type a:

$$\sum_{a \in A} \sum_{c \in C_a} T_a^r P_{c,r} \leq 1 \qquad \forall r \in R \tag{1}$$

$$\sum_{r \in R} P_{c,r} \leq N_c \qquad \forall c \in C \tag{2}$$

$$P_{c,r} \text{ are integers} \tag{3}$$

Inequality (1) ensures that each analyst is assigned a valid number of alerts, while inequality 2 ensures we do not assign more alerts than the total in a category.

	r_1	r_2
c_1	0	2
c_2	1	1
c_3	0	0
c_4	1	0

(a) Pure Strategy

	r_1	r_2
c_1	0.5	2.5
c_2	0.8	0
c_3	0	0
c_4	0.2	0

(b) Marginal Strategy

Fig. 1. CAG Strategies for the defender.

Example CAG. Consider a CAG with two systems $K = \{k_1, k_2\}$, two alert levels $A = \{a_1, a_2\}$, and two analysts $r = \{r_1, r_2\}$. There are four alert categories $C = \{c_1, c_2, c_3, c_4\}$, where $c_1 = (k_1, a_1)$, $c_2 = (k_1, a_2)$, $c_3 = (k_2, a_1)$ and $c_4 = (k_2, a_2)$. For the alert categories we have $N_{c_1} = 3$, $N_{c_2} = 2$, $N_{c_3} = 0$, and $N_{c_4} = 1$. For r_1, assume $T_{a_1}^{r_1} = 1$ and $T_{a_2}^{r_1} = 0.5$; For r_2, assume $T_{a_1}^{r_2} = 0.4$ and $T_{a_2}^{r_2} = 0.2$. The analyst capacity constraint (Inequality (1)) for r_1 is instantiated as follows (the other columns are similar):

$$P_{c_1,r_1} + 0.5 \cdot P_{c_2,r_1} + P_{c_3,r_1} + 0.5 \cdot P_{c_4,r_1} \leq 1$$

For c_1 the alert capacity constraint (Inequality (2)) we have (the other rows are similar):

$$P_{c_1,r_1} + P_{c_1,r_2} \leq 3$$

An example of a pure strategy P is given in Figure 1(a). The dashed boxes in Figure 1(a) represent the set of variables in the analyst capacity constraints, i.e. constraints of type (1). We show an example marginal strategy in Figure 1(b). This drops constraint (3), but satisfies constraints (1) and (2).

We define a mixed strategy **q** over pure strategies $P \in \hat{P}$ ($\sum_{P \in \hat{P}} q_P = 1, 0 \leq q_P \leq 1$). From the mixed strategy we can calculate the marginal (expected) number of alerts of category c assigned to each analyst r, denoted by $n_{c,r} = \sum_P q_P P_{c,r}$. The *marginal* allocation is denoted by **n** with component $n_{c,r}$ representing the expected number of alerts in category c assigned to analyst r. The adversary plays a best response to the defender's marginal strategy **n** which amounts to choosing a system k to attack and an attack method m.

Utilities Since the alerts in a category are indistinguishable they are all investigated with the same probability $n_{c,r}/N_c$, which is the probability that an alert in category c is investigated by analyst r. The probability of detecting an attack of type m that results in an alert of type c is calculated as: $x_{c,m} = \sum_{r \in R} E^r_m n_{c,r}/N_c$. The payoffs for the defender depend on the system k that is attacked, the attack method m, and if the adversary is detected (or undetected) during investigation. This is denoted by $U^d_{\delta,c}$ and $U^u_{\delta,c}$, respectively, where c refers to the category (k, a) and δ is the defender. We formulate a CAG as a zero-sum game, hence the payoffs for the adversary (θ) are $U^d_{\theta,c} = -U^d_{\delta,c}$ and $U^u_{\theta,c} = -U^u_{\delta,c}$. If the adversary chooses k, m, and given β^m_a, the defender's utility is:

$$U_s = \sum_{a \in A} \beta^m_a [x_{c,m} * U^d_{\delta,c} + (1 - x_{c,m}) U^u_{\delta,c}] \tag{4}$$

4 Defender's Optimal Strategy

We start with a linear program, denoted as $MixedStrategyLP$, that computes the defender's optimal mixed strategy (as the maximin strategy):

$$\max_{\mathbf{n},\mathbf{v}} \quad v \tag{5}$$

$$s.t. \quad v \leq U_s \qquad \forall k, m \tag{6}$$

$$x_{c,m} = \textstyle\sum_{r \in R} E^r_m \frac{n_{c,r}}{N_c} \qquad \forall c, m \tag{7}$$

$$n_{c,r} = \sum_{P \in \hat{P}} q_P P_{c,r} \qquad \forall c, r \tag{8}$$

$$\sum_{P \in \hat{P}} q_P = 1, q_P \geq 0 \tag{9}$$

This LP requires *exponentially* many pure strategies $P \in \hat{P}$. The objective function in Equation 5 maximizes the defender's utility, v. Equation 6, which uses Equation 4, ensures the adversary selects a best response over all choices of $m \in M$ and $k \in K$. Equation 7 calculates the detection probabilities **x** from the marginal strategy **n**, which is computed by Equation 8. Equation 9 ensures the mixed strategy is valid.

Computing the maximin mixed strategy for the defender was shown to be NP-hard in the case of TSGs [5]. The computational hardness arises from the underlying team formation of applying a group of screening resources to screen incoming passengers. However, in CAGs we do not have teams of analysts, we only need to assign the alerts to individual analysts. Thus, one might hope that this could simplify the problem and admit a polynomial time algorithm. Unfortunately, this turns out not to be the case. Specifically, we show in Theorem 1 that the problem is still NP-hard, where the hardness arises from a different domain feature, i.e., the time values, T_a^r, for the analysts. All proofs can be found in the on-line appendix[7].

Theorem 1 *Computing the defender maximin strategy is weakly NP-hard when there is only one resource, and is strongly NP-hard with multiple resources.*

In some special cases, it is possible to compute the optimal marginal strategy in polynomial time. Specifically, if all T_a^r for a given analyst r are identical $\forall a \in A$, then the optimal marginal strategy can be found with an LP which is stated in Proposition 1. This result is discussed further in Section 5.

Proposition 1. *When $T_{a_i}^r = T_{a_j}^r \ \forall a_i, a_j \in A$ for each resource, then there is a polynomial time algorithm for computing the maximin strategy.*

Defender's Optimal Marginal Strategy

In the security games literature, two approaches are commonly used to handle scale-up: marginal strategies [10, 12] and column generation [8]. We adopt a marginal strategy based approach which finds the defender's marginal strategy **n** and does not need to explicitly enumerate the exponential number of pure strategies. The use of the marginal approach was motivated in part by results in [5] where it was shown column generation does not scale for TSGs, which is a related model to CAGs. We now introduce a relaxed version of LP (5)∼(9) in LP (10)∼(14). LP (10)∼(14) is similar to LP (5)∼(9) except that we replace equations (8) and (9) with equations (13) and (14) to model relaxed the relaxed marginal space. Recall that marginal strategies satisfy constraints (1)∼(2) (which lead to Equations 13 and 14) but drop constraint (3). The optimal marginal strategy **n** for the defender can then be found by solving the following $MarginalStrategyLP$ (MSLP):

$$\max_{\mathbf{n},\mathbf{v}} \quad v \tag{10}$$

$$v \leq U_s \qquad \forall k, m \tag{11}$$

$$x_{c,m} = \textstyle\sum_{r \in R} E_m^r \frac{n_{c,r}}{N_c} \qquad \forall c, m \tag{12}$$

$$\textstyle\sum_{a \in A} \sum_{c \in C_a} T_a^r n_{c,r} \leq 1 \qquad \forall r \tag{13}$$

$$\textstyle\sum_{r \in R} n_{c,r} \leq N_c, \quad n_{c,r} \geq 0 \qquad \forall r, c \tag{14}$$

[7] https://teamcore.usc.edu/papers/2017/ars17.Appendix.pdf

Though $MarginalStrategyLP$ computes the optimal marginal strategy $\mathbf{n}$, it may not correspond to any valid mixed strategy $\mathbf{q}$, i.e., there may not exist a corresponding mixed strategy $\mathbf{q}$ such that $\mathbf{n} = \sum_{P \in \hat{P}} q_P P$, $\sum_{p \in \hat{P}} q_P = 1$. Marginal strategies of this type are called *non-implementable*. However, when T_a^r have a particular structure, we can show the marginal strategy returned is the optimal for the defender. In these cases, we can efficiently compute the defender's optimal implementable marginal strategy using the MSLP.

Theorem 2 *For any feasible marginal strategy* $\mathbf{n}$ *to MSLP, there is a corresponding mixed strategy* $\mathbf{q}$ *that implements* $\mathbf{n}$ *whenever* $T_a^r = \frac{1}{w_a}$ *where* $w_a \in \mathbb{Z}^+$, $\forall r \in R, \forall a \in A$ *and* $N_c \geq \sum_{r \in R} \frac{1}{T_a^r}$, $\forall c \in C$ *for a given CAG.*

The intuition behind Theorem 2 is that when the $T_a^r = \frac{1}{w_a}$ and $w_a \in \mathbb{Z}^+$, the extreme points of the defender's strategy space become integer. This can be seen from the maximum number of alerts each resource is able to resolve. Whenever, $T_a^r = \frac{1}{w_a}$ the number of alerts of a given type a resource can solve will be w_a, which corresponds to an integer assignment. Hence, the defender's marginal strategy space is the same as the defender's pure strategy space when these conditions are true and the MSLP returns the optimal marginal strategy for the defender.

5 CAG Algorithmic Approach

The problem of non-implementability of marginals in security games has been studied in previous research [12, 5], but the non-implementability arose because of spatio-temporal resource constraints and constraints from combining resources into teams. For our problem, non-implementability arises from the presence of the T_a^r coefficients (we discuss an example later). In this section, we present an algorithm that takes the initial constraints on a CAG and converts them to ensure the implementability of the marginal strategy. To that end, [6] presents a useful approach, as they define a special condition on the constraints on the marginals called a *bihierarchy*. A bihierarchy captures a sufficient condition needed to guarantee the implementability of the defender's marginal strategy $\mathbf{n}$. Unfortunately, constraints on CAGs rarely satisfy the conditions for a bihierarchy and must be converted to achieve the bihierarchy condition.

Definitions and Notation The marginal assignments $\mathbf{n}$ for the defender form a $|C| \times |R|$ matrix. The assignment constraints on the defender's marginal strategy, namely Equations 13 and 14, are a summation of $n_{c,r}$ over a set $S \subset |C| \times |R|$ with an integral upper bound. For example, based on Equation 14, $\{\{c_1, r_1\}, \{c_1, r_2\}\}$ forms a constraint subset for the example CAG. The collection of all such S form a *constraint structure* H when all coefficients in the constraints are *unitary*, as they are in Equation 14.

A marginal strategy $\mathbf{n}$ is said to be *implementable* with respect to H if there exists a distribution (a.k.a., mixed strategy) $\mathbf{q}$ such that $\mathbf{n} = \sum_{P \in \hat{P}} q_P P$. A constraint structure H is said to be a *hierarchy* if, for any two constraint sets in H, we have that either one is a subset of the other or they are disjoint. More concretely, we have the following: $\forall S_1, S_2 \in H$, $S_1 \subset S_2$, $S_2 \subset S_1$ or $S_1 \cap S_2 = \emptyset$. H is said to be a *bihierarchy* if there exists hierarchies H_1 and H_2, such that $H = H_1 \cup H_2$ and $H_1 \cap H_2 = \emptyset$.

For any CAG, the row constraints $\sum_{r \in R} n_{c,r} \leq N_c$ form a hierarchy H_1. However, the column constraints, one for each resource $r \in R$, do not form a hierarchy: $\sum_{a \in A} \sum_{c \in C_a} T_a^r n_{c,r} \leq 1$. As mentioned earlier, the culprit lies in the T_a^r coefficients, as they can be non-unitary, and to achieve a hierarchy H_2 on the column constraints, and thus give us a bihierarchy, all T_a^r coefficients must be removed.

Constraint Conversion The T_a^r coefficients admits possibly non-implementable marginal strategies. For instance, in Figure 1(b) the marginal strategy is non-implementable, because it is impossible to get $n_{c_1,r_2} = 2.5$ by mixing pure assignments. This is because constraints (1) and (3), force the relevant pure strategy $P_{c_1,r_2} \leq \lfloor 1/0.4 \rfloor = 2$. We aim to convert the column constraints, namely: $\sum_{a \in A} \sum_{c \in C_a} T_a^r n_{c,r} \leq 1$ into a hierarchy by removing the T_a^r coefficients. The conversion can be completed by grouping together all $n_{c,r}$ which have the same T_a^r and introducing a new constraint on these sets of $n_{c,r}$. Specifically, each column constraint (equation 13) is replaced with $|A|$ constraints:

$$\sum_{c \in C_a} n_{c,r} \leq L_r^{C_a} \tag{15}$$

This conversion must be done for all analysts $r \in R$ for the column constraints to form a hierarchy H_2. $L_r^{C_a}$ gives an upper bound on the number of alerts of type a that an analyst can solve. The choices of $L_r^{C_a}$ must satisfy the original capacity constraint, namely: $\sum_{a \in A} T_a^r L_r^{C_a} \leq 1$ and $L_r^{C_a} \in \mathbb{Z}$.

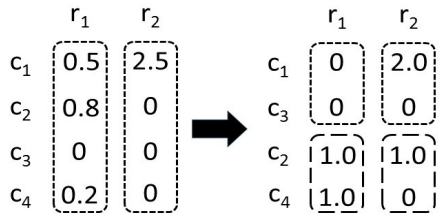


Fig. 2. Conversion of Column Constraints on CAG

Conversion Example We refer to the example CAG where the marginal strategy is given in Figure 2. We must convert the column constraints to a hierarchy. We highlight how this conversion is done for r_1 (as r_2 is converted in the same manner). Initially, for r_1 we have the following constraint:

$$T_{a_1}^{r_1} n_{c_1,r_1} + T_{a_2}^{r_1} n_{c_2,r_1} + T_{a_1}^{r_1} n_{c_3,r_1} + T_{a_2}^{r_1} n_{c_4,r_1} \leq 1$$

We remove the T_a^r coefficients by grouping together all $n_{c,r}$ which share T_a^r and introducing two new constraints like (15). This leads to two new constraints:

$$n_{c_1,r_1} + n_{c_3,r_1} \leq L_{r_1}^{C_{a_1}} \qquad\qquad n_{c_2,r_1} + n_{c_4,r_1} \leq L_{r_1}^{C_{a_2}}$$

These new constraints are shown for r_1 in Figure 2 on the right of the arrow. Next, we must set the $L_r^{C_a}$ variables. One possible combination is $H_2 = \{n_{c_1,r_1} + n_{c_3,r_1} \leq$

$0, n_{c_2,r_1} + n_{c_4,r_1} \leq 2\}$ (H_2 also includes constraints on r_2 which are not shown). This satisfies the original the original analyst capacity constraints as: $L_{r_1}^{C_{a_1}} + 0.5 \cdot L_{r_1}^{C_{a_2}} \leq 1$. However, there is another choice for $L_r^{C_a}$, $H_2 = \{n_{c_1,r_1} + n_{c_3,r_1} \leq 1, n_{c_2,r_1} + n_{c_4,r_1} \leq 0\}$. Given either of the two hierarchies H_2, we now have a bihierarchy. The original marginals shown in Figure 2 do not satisfy these new constraints; but solving the MSLP with these additional constraints in H_2 is guaranteed to give an implementable marginal.

Rounding T_a^r Values In the conversion process, we create a hierarchy H_2 on the column constraints by introducing $|A|$ $L_r^{C_a}$ values for each resource. The conversion process then allows for combinatorially many configurations of the $L_r^{C_a}$ values which satisfy the original capacity constraints for a resource, i.e. Constraint (13). To alleviate this search, an algorithm could take advantage of Theorem 2 and round each T_a^r to the nearest $\frac{1}{w_a}$ value which is greater than T_a^r where $w_a \in \mathbb{Z}^+$. The marginal strategy **n** returned for this modified CAG is then guaranteed to be implementable. However, as we show next this can lead to a $\frac{1}{2}$ loss for the defender in the worst case.

Counter Example Consider a CAG with one system $K = \{k_1\}$, two alert levels $A = \{a_1, a_2\}$, and one analyst $r = \{r_1\}$. There are two alert categories $C = \{c_1, c_2\}$, where $c_1 = (k_1, a_1)$ and $c_2 = (k_1, a_2)$. For the alert categories we have $N_{c_1} = 1$ and $N_{c_2} = 1$. For r_1, assume $T_{a_1}^{r_1} = 0.5 + \epsilon$ and $T_{a_2}^{r_1} = 0.5 - \epsilon$. If we round the T_a^r values up to the nearest $\frac{1}{w_a}$ we would have the following: $T_{a_1}^{r_1} = 1$ and $T_{a_2}^{r_1} = 0.5$.

Now we assume the adversary has one attack method m_1 with $E_{m_1}^{r_1} = 1$ and where $\beta_{a_1}^{m_1} = 0.5 + \epsilon$ and $\beta_{a_2}^{m_1} = 0.5 - \epsilon$. Assume $U_{\delta,c_1}^d = U_{\delta,c_2}^d$ and $U_{\delta,c_1}^u = U_{\delta,c_2}^u$ where $U_{\delta,c_1}^d > U_{\delta,c_1}^u \geq 0$. The adversary has one choice and hence, chooses to use m_1 to attack system k_1. The modified CAG would then assign the alert in c_1 to r_1 and receive a utility of $v' = (0.5 - \epsilon)U_\delta^u + (0.5 + \epsilon)U_\delta^d$. In the unmodified CAG, however, the defender would be able to assign both alerts to r_1 and therefore, achieve a utility $v^* = U_\delta^d$. In this case, the worst possible loss from the modification of the CAG happens when $U_\delta^u = 0$. We then get the following:

$$\frac{v'}{v^*} = \frac{(0.5 - \epsilon)U_\delta^u + (0.5 + \epsilon)U_\delta^d}{U_\delta^d} \leq \frac{(0.5 + \epsilon)U_\delta^d}{U_\delta^d}$$

This results in $v' = (0.5 + \epsilon)v^*$ in the worst case. Hence, rounding the T_a^r values means the defender can lose up to $\frac{1}{2}$ of the optimal utility. This amount of loss is not acceptable in cyber security domains which have highly sensitive targets and therefore, we must devise algorithms which provide better solutions that mitigate this loss.

Branch-and-Bound Search

So far, we have seen that a marginal strategy **n** for a CAG output from the MSLP may be non-implementable. Our goal is to ensure that the marginal strategy output by MSLP is implementable by adding new column constraints, i.e., by realizing a bihierarchy. The addition of new constraints as outlined above gives us a bihierarchy, but there are multiple ways to set the values of $L_r^{C_a}$ variables (as shown in the above example), creating a choice of what bihierarchy to create. Indeed, we may need to search through the combinatorially many ways to convert the constraints of CAG to a bihierarchy.

Previous work [5] proposed the MGA algorithm for creating bihierarchies, but MGA does not apply to CAGs as it does not deal with the non-unitary coefficients present in CAGs.

Here we propose a novel branch-and-bound search: out of the set of constraints that could be added to MSLP, find the best that would give the defender the optimal utility v^*. At the root node, we have the original constraints (13) and (14); running MSLP potentially yields a non-implementable marginal strategy $\mathbf{n}$. Then we branch from this root, where at each level in the tree, we add new constraints for an analyst r, and the children are expanded with the following rules:

1. Substitute $\sum_{a\in A}\sum_{c\in C_a} T_a^r n_{c,r} \leq 1$ with $|A|$ constraints: $\sum_{c\in C_a} n_{c,r} \leq L_r^{C_a}$ for all $a \in A$. The $|A|$ new constraints form a set $H_2(r)$. A branch is created for all combinations of $L_r^{C_a}$ which satisfy $\sum_{a\in A} T_a^r * L_r^{C_a} \leq 1$.
2. Solve the $MarginalStrategyLP$ at each node with the modified constraints.

Thus, at each level of the tree, we have substituted the capacity constraint of some analysts, and for these, we have constraints of type (15), but for others, we still have constraint (13). This set of constraints does not form a hierarchy H_2 as T_a^r coefficients are present in some analyst constraints. Still, at an intermediate node we have upper bound on the defender's utility v which is stated in Proposition 2, as each conversion from (13) to (15) introduces new constraints on the defender's strategy space.

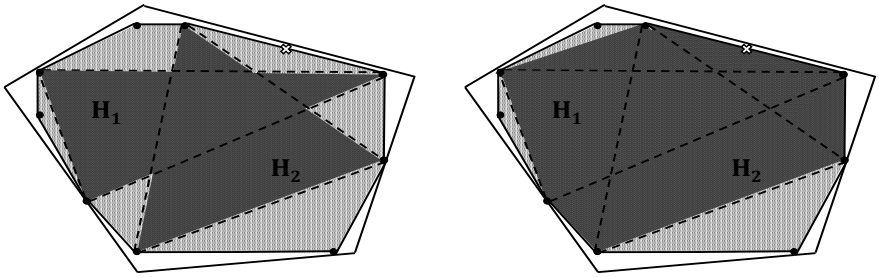
Proposition 2. *Each intermediate node in the tree gives an upper bound on the defender's utility v for all subsequent conversions for the remaining analyst capacity constraints.*

A leaf in the search tree has column constraints only of the form: $\sum_{a\in A} n_{c,r} \leq L_r^{C_a}$. Hence, they form a hierarchy H_2 as all $n_{c,r}$ have unitary coefficients and an integer upper bound. At a leaf, we can then solve the MSLP with the resulting bihierarchical constraints to find a lower bound on the defender's utility v. Combining this with Proposition 2 gives the components needed for a branch-and-bound search tree which returns the optimal bihierarchy for the defender.

Heuristic Search The full branch-and-bound procedure struggles with large CAG. To find good bihierarchies, we can take advantage of the optimal marginal strategy $\mathbf{n}^*$ returned from $MSLP$ at an intermediate node to reduce the amount of branching done. The intuition for this strategy, is that the optimal bihierarchy either contains, or is near, $\mathbf{n}^*$. For example, in the conversion done in Figure 2, we could set the $L_r^{C_a}$ values close to $\mathbf{n}$. We set $L_{r_2}^{C_{a_1}} = \lfloor \frac{1}{.4} \rfloor = 2$, while the leftover capacity for r_2 is used to set $L_{r_2}^{C_{a_2}} = 1$. $L_{r_2}^{C_{a_1}}$ could be set to another value, but our choice must stay close to $\mathbf{n}^*$.

For the heuristic search, we use the following rules to expand child nodes which is done by setting the $L_r^{C_a}$ for an analyst r: (1) $L_r^{C_a} = \lceil n_{C_a,r} \rceil$, (2) $L_r^{C_a} = \lfloor n_{C_a,r} \rfloor$ or (3) $L_r^{C_a} = \lfloor \frac{1-\sum_{a\in A} T_a^r * L_r^{C_a}}{T_a^r} \rfloor$, where $n_{C_a,r} = \sum_{c\in C_a} n_{c,r}$. The third rule is used whenever the $L_r^{C_a}$ value for an analyst r cannot be set to the roof or floor of $\mathbf{n}^*$, and is set to be the max value given the leftover analyst capacity. These choices are done in an attempt to capture the optimal marginal strategy $\mathbf{n}^*$. The set of all valid combinations of the $L_r^{C_a}$ values using the above rules which satisfy $\sum_{a\in A} T_a^r L_r^{C_a} \leq 1$ constitute the

search space at each intermediate node. The main point is that cuts down the amount of branching done at intermediate nodes from the original branch-and-bound search.



(a) Individual Bihierarchies (b) Convex Hull

Fig. 3. Geometric view of the defender's strategy space.

Convex Hull Extension The above searches return a set of good bihierarchies for obtaining a high value of v^* for the defender when solving MSLP, as each leaf contains a bihierarchy H_i. Each bihierarchy H_i contains a portion of the defender's mixed strategy space (due to new constraints). Thus, taking a convex hull over these bihierarchies increases the size of the defender's strategy space and hence, will only improve the defender's utility. In Figure 3 we show a geometric representation of the defender's strategy space. Individual points represent the defender's pure strategies and the region contained in the convex hull of these points is the defender's mixed strategy space, while the outer region represents the defender's relaxed marginal strategy space. Figure 3(a) shows how individual bihierarchies capture portions of the defender's mixed strategy space represented by the shaded regions enclosed by the dashed lines. Figure 3(b) shows that by taking the convex hull of the two bihierarchies H_1 and H_2 we can increase the size of the defender's strategy space without generating any new bihierarchies. Note, as each bihierarchy is implementable, the convex hull will also be implementable [5].

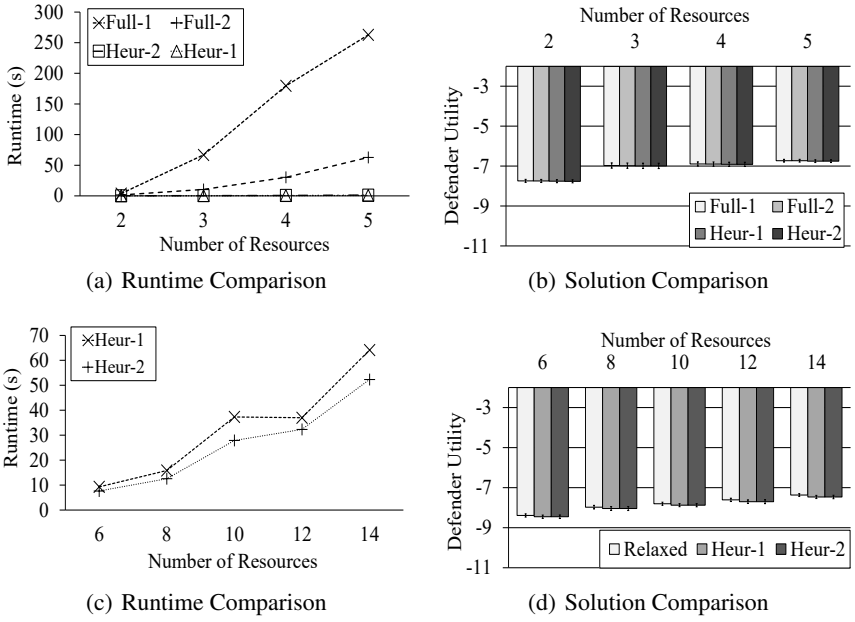
To take the convex hull, first notice each bihierarchy H_i is a set of linear constraints and can be written as $D_i n \leq b_i$ for matrix D_i and vector b_i. Hence, by definition $\mathbf{n}(H_i) = \{\mathbf{n} | D_i \mathbf{n} \leq b_i\}$. Using a result from [3] that represents the convex hull using linear constraints, we can write: $conv(\mathbf{n}(H_1), \ldots, \mathbf{n}(H_l)) = \{\mathbf{n} | \sum_i \mathbf{n}_i, D_i \mathbf{n}_i \leq \lambda_i b_i, \lambda_i \geq 0, \sum_i \lambda_i = 1\}$. This allows for the convex hull of the bihierarchies to be computed efficiently using an LP similar to MSLP.

In terms of the convex hull we have two options available: (1) Take the convex hull of all bihierarchies or (2) build the convex hull iteratively. In some cases, the set of bihierarchies available to the defender can be very large and hence, optimizing over all bihierarchies is not feasible. To alleviate this issue, the convex hull can be built iteratively. This is done by first sorting the bihierarchies by the defender utility v. Next, we take the convex hull of the top two bihierarchies which gives a utility $v^{'}$ to the de-

fender. We continue adding bihierarchies to the convex hull while the utility v' returned increases by at least some ϵ, and stop otherwise.

6 Evaluation

We evaluate the CAG model and solution algorithms with experiments inspired by the operations of the AFCYBER. The game payoffs are set to be zero-sum, i.e. $U^u_{\delta,c} = -U^u_{\theta,c}$, and the defender's payoffs are randomly generated with $U^u_{\delta,c}$ uniformly distributed in $[-1, -10]$. The rest of the game payoffs, $U^d_{\delta,c}$ and $U^d_{\theta,c}$, are set to be zero. For each experiment we average over 30 randomly generated game instances.



(a) Runtime Comparison (b) Solution Comparison

(c) Runtime Comparison (d) Solution Comparison

Fig. 4. Experimental Results for CAG instances.

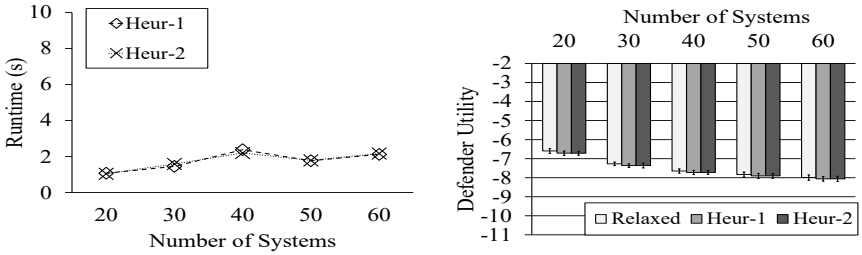
Full vs Heuristic Search Whether the heuristic approach of staying close to $\mathbf{n}^*$ would yield the right solution quality-speed tradeoff remains to be seen. To test this, we compare the performance of the full branch-and-bound search (Full) to the heuristic search (Heur). For this experiment we test four different variations of the algorithms. For the full search we test two variations: Full-1 which uses the full convex hull and Full-2 which uses the iterative convex hull. For the Heuristic search we test the same two variations, labeled as Heur-1 and Heur-2. For these instances we have 20 systems, 3 attack methods, and 3 alert types.

In Figure 4(a) we vary the number of resources on the x-axis and we show the runtime in seconds on the y-axis. As can be seen the runtime of the full search explodes

Resources	Full	Heuristic
2	152.56	9.53
3	1421.97	16.40
4	3567.70	23.23
5	6525.37	40.96

Table 1. Reduction in Nodes Searched

exponentially as the number of resources is increased. However, the average runtime of the heuristic approach is under 1 second in all cases and provides up to a 100x runtime improvement for 5 resources. The main reason for the speedup is due to the heuristic approach exploring a small fraction of the nodes explored by the full search, as low as 0.7% for 5 resources. To gain more insight into the source of the speed-up from the heuristic search we give the average number of nodes explored by each search method in Table 1. This table shows the exponential speed-up is due to the heuristic search exploring only a fraction of the total nodes explored by the Full-search, as low as 0.7% for 5 resources.



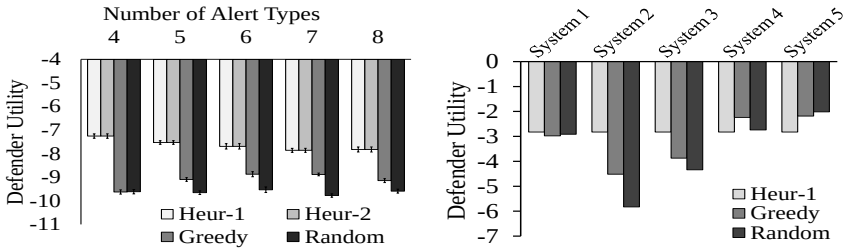
(a) Runtime Comparison (b) Solution Comparison

Fig. 5. Scaling Number of Systems

In Figure 4(b) we again vary the number of resources on the x-axis while the y-axis shows the defender's expected utility. This graph shows that all variations perform similarly, with the heuristic suffering less than 1% solution in defender utility compared to the full search for all game sizes. Still, these results show that our heuristic significantly improves runtime without sacrificing solution quality.

Solving large CAG Another important feature of real-world domains are the larger number of cybersecurity analysts available to investigate alerts and the number of systems to protect. Accordingly, our next experiment tests the scalability of our heuristic approach to large CAG instances. The parameters for the first experiment is 100 systems, 10 attack methods, and 3 alert levels.

In Figures 4(c) and 4(d) we show the runtime and solution quality results. In Figure 4(c) we vary the number of analysts on the x-axis and show the runtime in seconds on the y-axis. For example, Heur-1 takes an average of 40 seconds to solve a CAG with



(a) Runtime Comparison (b) Solution Comparison

Fig. 6. Allocation Approach Comparison.

10 analysts. This graph shows the heuristic runs in under a minute, even as we increase the analysts from 6 to 14. In Figure 4(d) we have the number of analysts on the x-axis and show the defender's expected utility on the y-axis. We compare the solution quality to the (potentially non-implementable) MSLP solution. For example, with 8 analysts the defender's expected utility is -7.98 for the relaxed method while Heur-1 gives -8.04. Therefore, this experiment shows that our heuristic approach scales to large CAG while achieving a utility close to the theoretical optimal value.

In our next experiment, we vary the number of systems that have to be protected. For this experiment the defender has 5 cyber experts to assign. Figure 5 shows the runtime and solution quality results. In Figure 5(a) we vary the number of systems on the x-axis and show the runtime in seconds on the y-axis. For instance, for 50 systems Heur-1 takes an average of 1.78 seconds to finish running. This graph shows Heur-1 and Heur-2 show no issues in scaling to a larger number of systems. In Figure 5(b) the x-axis shows the number of systems while the y-axis gives the defender's expected utility. We again compare the solution quality to the MSLP solution. In all cases, the heuristic approaches suffer only a small loss in defender expected utility compared to the MSLP value. As can be seen from this experiment, the heuristic approaches scale to CAG with a larger number of systems without sacrificing much in the way of solution quality.

Allocation Approach Our last experiment aim to show that our game theoretic approach for CAGs outperform approaches used in practice. In addition to our heuristic, we compare against a greedy approach which investigates the highest priority alerts from the most critical bases first and a random approach for the allocation. The parameters for this experiment are 20 systems, 5 attack methods, and 10 analysts. In Figure 6(a) we show the solution quality results. On the x-axis we vary the number of alert types and on the y-axis we show the defender's utility. For example, with 4 alert types the heuristics achieve a utility of -7.52 while the greedy and randomized allocations give -9.09 and -9.65, respectively. This difference is statistically significant ($p < 0.05$). In Figure 6(b), we show a solution comparison for a specific CAG instance. This graph gives intuition for why our approach performs so well. The greedy and random approaches tend to overprotect some systems (system 4) while leaving others without adequate protection (system 2).

7 Conclusion and Future Work

In this paper we address the pressing problem in cyber security operations of how to allocate cyber alerts to a limited number of analysts. We introduce the Cyber-alert Allocation Game (CAG) to analyze this problem and show computing optimal strategies for the defender is NP-hard. To solve CAG, we present a novel approach to address implementability issues in computing the defender's optimal marginal strategy. Finally, we give heuristics to solve large CAGs, and give empirical evaluation of the CAG model and solution algorithms.

Although this work is a crucial first step in applying game theory to real world cyber security settings, there remain significant challenges which need to be addressed in future work of which we highlight a few. Firstly, we assume the time to resolve an alert is known exactly, but in the real world there is uncertainty for how long it would take to resolve an alert. Second, the CAG model assumes that attacks shown up as known alert categories, but it is possible that in the real-world some attacks may show up as "unknown" categories. The question then is how to assign these alerts to analysts given we do not know which expert may have an expertise in dealing with this type of attack. Lastly, in CAG's there is not an overflow of alerts from one time period to the next. In the real-world, however, this could occur and resolving alerts in a timely manner would be crucial to limit the possible damage from an attack.

References

1. 688th Cyberspace Wing (2016), http://www.24af.af.mil/Units/688th-Cyberspace-Wing
2. 24th Air Force - AFCYBER (2017), http://www.24af.af.mil
3. Balas, E.: Disjunctive programming and a hierarchy of relaxations for discrete optimization problems. SIAM Journal on Algebraic Discrete Methods 6(3), 466–486 (1985)
4. Barbara, D., Jajodia, S.: Applications of data mining in computer security, vol. 6. Springer Science & Business Media (2002)
5. Brown, M., Sinha, A., Schlenker, A., Tambe, M.: One size does not fit all: A game-theoretic approach for dynamically and effectively screening for threats. In: AAAI conference on Artificial Intelligence (AAAI) (2016)
6. Budish, E., Che, Y.K., Kojima, F., Milgrom, P.: Designing random allocation mechanisms: Theory and applications. The American Economic Review 103(2), 585–623 (2013)
7. Ganesan, R., Jajodia, S., Shah, A., Cam, H.: Dynamic scheduling of cybersecurity analysts for minimizing risk using reinforcement learning. ACM Transactions on Intelligent Systems and Technology (TIST) 8(1), 4 (2016)
8. Jain, M., Kardes, E., Kiekintveld, C., Ordónez, F., Tambe, M.: Security games with arbitrary schedules: A branch and price approach. In: AAAI (2010)
9. Jain, M., Tsai, J., Pita, J., Kiekintveld, C., Rathi, S., Tambe, M., Ordóñez, F.: Software assistants for randomized patrol planning for the lax airport police and the federal air marshal service. Interfaces 40(4), 267–290 (2010)
10. Kiekintveld, C., Jain, M., Tsai, J., Pita, J., Ordóñez, F., Tambe, M.: Computing optimal randomized resource allocations for massive security games. AAMAS (2009)
11. Laszka, A., Lou, J., Vorobeychik, Y.: Multi-defender strategic filtering against spear-phishing attacks. In: AAAI (2016)

12. Letchford, J., Conitzer, V.: Solving security games on graphs via marginal probabilities. In: AAAI (2013)
13. Patton, R.M., Beaver, J.M., Steed, C.A., Potok, T.E., Treadwell, J.N.: Hierarchical clustering and visualization of aggregate cyber data. In: 2011 7th International Wireless Communications and Mobile Computing Conference. pp. 1287–1291. IEEE (2011)
14. Riley, M., Elgin, B., Lawrence, D., Matlock, C.: Missed alarms and 40 million stolen credit card numbers: How target blew it. http://www.zdnet.com/article/anatomy-of-the-target-data-breach-missed-opportunities-and-lessons-learned/ (2014), https://www.bloomberg.com/news/articles/2014-03-13/target-missed-warnings-in-epic-hack-of-credit-card-data, accessed: 2016-11-10
15. Sommer, R., Paxson, V.: Outside the closed world: On using machine learning for network intrusion detection. In: 2010 IEEE symposium on security and privacy. pp. 305–316. IEEE (2010)
16. Spathoulas, G., Katsikas, S.: Methods for post-processing of alerts in intrusion detection: A survey (2013)
17. Tambe, M.: Security and game theory: algorithms, deployed systems, lessons learned. Cambridge University Press (2011)
18. Yin, Z., Korzhyk, D., Kiekintveld, C., Conitzer, V., Tambe, M.: Stackelberg vs. nash in security games: Interchangeability, equivalence, and uniqueness. In: AAMAS. pp. 1139–1146. International Foundation for Autonomous Agents and Multiagent Systems (2010)
19. Zimmerman, C.: Ten strategies of a world-class cybersecurity operations center. MITRE corporate communications and public affairs. Appendices (2014)