Teamwork Makes the Defense Work: Defense Resource Allocation with Composable Targets

Siyu Liu
Arizona State University
Tempe, United States
sliu274@asu.edu

Rida Bazzi
Arizona State University
Tempe, United States
bazzi@asu.edu

Fei Fang
Carnegie Mellon University
Pittsburgh, United States
feifang@cmu.edu

Tiffany Bao
Arizona State University
Tempe, United States
tbao@asu.edu

ABSTRACT

Despite the success of game-theoretic models in security resource allocations against adversaries, existing works have fallen short in addressing the critical challenge of team defense with composable targets. Composable targets, commonly seen in cybersecurity practices like vulnerability analysis, consist of heterogeneous tasks that can be processed by different defenders. The intrinsic heterogeneity and potential precedence constraints among tasks present a great challenge to devising optimal defender strategies.

In this paper, we propose a general-sum Stackelberg game model for team defense with composable targets. We develop SWING, a novel method that efficiently calculates optimal defense strategies by combining binary search, linear programming, and column generation. We prove that our algorithm calculates strong Stackelberg equilibrium (SSE), and that in practice, it is runtime-efficient at finding optimal strategies. To further enhance the applicability of SWING, we extend its capabilities to encompass defense tasks with precedence constraints. This is achieved by leveraging flexible job shop problem (FJSP) literature to devise a branch-and-bound-based method. Our empirical evaluations illustrate that this extension enhances runtime efficiency and substantially improves solution quality compared to baseline methods.

KEYWORDS

Stackelberg Security Game; Resource Allocation

ACM Reference Format:
Siyu Liu, Rida Bazzi, Fei Fang, and Tiffany Bao. 2025. Teamwork Makes the Defense Work: Defense Resource Allocation with Composable Targets. In *Proc. of the 24th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2025), Detroit, Michigan, USA, May 19 – 23, 2025*, IFAAMAS, 9 pages.

1 INTRODUCTION

Since its introduction in 2006, the Stackelberg Security Game (SSG) model [6] has been highly influential in security research and has been widely applied to real-world scenarios, including infrastructure protection [17], environmental crime prevention [7, 8, 10], and cybersecurity [23]. In these applications, defenders protect assets such as airport gates [17] or other specific locations at designated times [8].

While the SSG model has proven effective for many real-world challenges, the growing complexity of modern scenarios demands more sophisticated approaches to protect the targets. One prominent example is cybersecurity; as computing techniques become more complex and the scale of analysis grows, defending cyber targets has evolved into a highly intricate process composed of multiple tasks. For instance, in the CAG model [23], detecting an attack used to involve an analyst successfully interpreting a single cyber alert. However, modern cyberattacks have grown so sophisticated that detecting an attack now requires analyzing a series of alerts by a team of tier-1 analysts, synthesizing the information, and having higher-tier analysts assess whether an attack is present. As a result, targets have become *composable*, as they consist of multiple interdependent tasks. Furthermore, these tasks have *precedence constraints*, so completing these tasks must follow a specific order. A target remains unprotected unless all associated tasks are completed, with each analyst finishing their part within their resource constraints.

The composable targets property introduces significant complexity to the Stackelberg Security Game (SSG) model, making existing approaches inadequate. Solving these problems with composable targets remains NP-hard, and the solution space is substantially larger than in previous models. While cybersecurity games are known for their vast strategy space [24], the introduction of composable targets has caused the strategy space to increase exponentially in the number of targets and analysts. For example, given the CAG model's solution space as M (which is already exponential to the number of resources and targets), our solution space is $O\left(M^t \cdot m^{p!}\right)$ where t is the number of tasks for each target, m is the number of agents (i.e., resources in a generic SSG model), and p is the number of tasks assigned to each agent.

Moreover, a composable target complicates the marginal strategy feasibility problem and makes prior solutions not applicable. The bihierarchy [4] structure of the constraints does not hold in our problem. Other solutions, such as Blade [32], solve the marginal strategy feasibility problem by converting the problem to an LP and seeking a solution through relaxation. If no solution is found, they add a linear constraint to the equations to further restrict the search

Proc. of the 24th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2025), Y. Vorobeychik, S. Das, A. Nowé (eds.), May 19 – 23, 2025, Detroit, Michigan, USA. © 2025 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org).

space. However, the feasibility problem in our domain is a MILP rather than LP. As a result, no linear constraint is generated when no feasible solution is found, and the search space remains. Furthermore, because of the infeasible compact-form strategy, existing algorithms that leverage the monotonic structure in compact-form strategies, such as ORIGAMI, cannot be directly applied to our problem. The polyhedron structure of ORIGAMI ensures that any feasible compact form strategy has a corresponding feasible mixed normal form strategy. This structure does not exist in our problem.

This paper presents a novel approach, SWING (Stackelberg equilibrium With Implementable Normal-form stateGy), to addressing the *composable target team defense problem*. Our key insight is that the composable target team defense problem exhibits a *monotonic infeasibility* property (Theorem 4.4). Specifically, if no feasible defender's compact-form strategy exists at a given attacker's utility value, no feasible strategy will exist for any lower attacker utility. This property makes us leverage the binary search on compact-form strategy space to decide optimal compact-form strategies efficiently.

To convert the compact-form strategy into a normal-form strategy, we propose a column generation-based method, which formulates two distinct Mixed-Integer Linear Programming (MILP) problems. Each MILP identifies optimal columns for scenarios with and without precedence constraints. We further enhance algorithm efficiency for scenarios with precedence constraints using a branch-and-bound method combined with a preprocessing technique.

In our evaluation, we assess the performance of SWING for the composable target team defense problem both with and without precedence constraints. The results demonstrate that, compared to industry-level commercial generic MILP solvers Gurobi, SWING not only yields strategies with higher defender utility but also speeds up the solving process by a factor of two. The improvements in defender utility and time performance become more pronounced as the number of targets increases.

SWING has a significant impact on real-world cybersecurity practices. We applied it to a Fortune 500 technology company T[1] to allocate weekly high-risk vulnerability mitigation tasks to 38 analysts (See Section 6.3). SWING increased the average detection rate of high-risk vulnerabilities from 10% to 90.91%. Given that the impact of high-risk vulnerabilities can range from millions to billions of dollars in damages, increasing the mitigation rate for such vulnerabilities could save the company hundreds of thousands of dollars every week. Our analysis also revealed that improving the efficiency of existing employees is more cost-effective than increasing the workforce.

This paper makes four contributions: 1). We present the composable target team defense problem and develop a novel general-sum Stackelberg game model for the problem. 2) We introduce the monotonic infeasibility property, demonstrate that models with such a property can be solved with a binary search over compact-form strategies even though the strategies can be infeasible, and prove that our problem has this property. 3). We develop SWING, a novel solution for the compact-form strategy feasibility problem, and 4) 3). We demonstrate that SWING significantly enhances efficiency and solution quality compared to baseline methods and can be used in real-world scenarios. The code implementation of SWING is available at https://github.com/sefcom/SWING.

[1]For privacy reasons, we cannot disclose the company's name; we will refer to the company as T throughout this paper.

2 MOTIVATION AND RELATED WORK

Motivating Domain. Composable target team defense has become increasingly prevalent in cybersecurity. As networks and software systems expand, defending targets becomes more complex, requiring defenders to break down per-target defense into manageable tasks, typically handled by teams such as Security Operations Centers (SOCs) or security response teams [28]. This defense structure reflects the inherently asymmetrical nature of cybersecurity, where defenders must identify and remediate all potential vulnerabilities. At the same time, an attacker needs only to exploit a single one to launch a successful attack.

One composable target team defense example is vulnerability mitigation in large IT companies, such as Microsoft's Security Response Center [18] and Google's Vulnerability Reward Program [12]. These programs highlight the critical role of cybersecurity response teams in securing the company products by mitigating vulnerabilities.

We will use a Fortune 500 technique company T as a concrete example of vulnerability mitigation in IT companies. The process of mitigating vulnerabilities is a collaborative effort involving multiple steps, as illustrated in Figure 1. This process typically begins with the submission of a bug report, which can originate from internal code auditing or external sources such as bug bounty programs. Upon receiving the report, a security analyst investigates the issue and attributes the vulnerability to a specific component of the product. The analyst then collaborates with the relevant developer, guiding them through the patch development process. Once the patch is created, the security response team rigorously tests it to ensure its effectiveness. Finally, the patched product is sent to the appropriate department and released to the public.

Each task in this workflow must be completed in order before the vulnerability is fully mitigated. In the meantime, every unresolved vulnerability presents an opportunity for attackers to exploit, potentially causing catastrophic damage to users, such as service outages, data breaches, or even unauthorized control of systems. Given the limited number of security personnel and their limited working hours, many of the received bug reports remain unprocessed. Therefore, it is crucial to optimize the task assignment to ensure the best possible security outcomes, especially for high-risk vulnerabilities.

Related Work. Stackelberg Security Games (SSGs) have been successfully applied to real-world optimal resource allocation scenarios [9, 17, 20–22, 25, 26, 30], yet little has addressed team defense with composable targets. One of the closest existing works is the Cyber-alert Allocation Game (CAG) [23], which aims to optimize the assignment of cyber alerts to cyber analysts, analogous to our goal of assigning tasks to defense agents. However, their model does *not* support composable targets or precedence constraints among tasks. Therefore, the implementability issues inherited from our model differ from the CAG model. The existing model's solution cannot be sufficiently applied to the composable target team defense problem.

Figure 1: Vulnerability mitigation workflow in Company T.

3 MODEL FORMULATION

We model the composable target team defense problem as a general-sum two-player Stackelberg Security Game (SSG), where the defender (leader) first commits to defend a set of *targets*, and the attacker (follower) then selects one target to attack. We focus on one-target attacks, similar to prior works [1, 15, 23], because many real-world attacks, especially cyberattacks, are one-target [19].

Each target has four associated utilities. For target i, if the attacker attacks it and the defender covers (uncovers) it, then the attacker gets a utility of $U_a^{c,i}$ ($U_a^{u,i}$) while the defender receives a utility $U_d^{c,i}$ ($U_d^{u,i}$). We assume that $U_d^{c,i} > U_d^{u,i}$ and $U_a^{c,i} < U_a^{u,i}$, i.e., a defended target is beneficial to the defender yet undesirable to the attacker. We also assume both players are rational, similar to existing resource allocation in adversarial environment research [3, 23].

Let n denote the number of targets. We present the attacker's (mixed) strategy as a vector $\boldsymbol{g} \in \{0,1\}^n$ with $\sum_{i=1}^n \boldsymbol{g}_i = 1$, where $\boldsymbol{g}_i$ is the ith entry of $\boldsymbol{g}$. We then present the defender's strategy in a *compact form*[2] [11, 17, 31] as $\boldsymbol{p} = [0,1]^n$ where $\boldsymbol{p}_i$ is the probability that target i is covered. Given the players' strategy, their expected payoffs can be formed as follows:

$$\text{Defender:} \quad R_d(\boldsymbol{p}, \boldsymbol{g}) = \sum_{i=1}^n \boldsymbol{g}_i (\boldsymbol{p}_i U_d^{c,i} + (1 - \boldsymbol{p}_i) U_d^{u,i}) \tag{1}$$

$$\text{Attacker:} \quad R_a(\boldsymbol{p}, \boldsymbol{g}) = \sum_{i=1}^n \boldsymbol{g}_i (\boldsymbol{p}_i U_a^{c,i} + (1 - \boldsymbol{p}_i) U_a^{u,i}) \tag{2}$$

Regarding defender's normal form strategy, we first define a pure normal form strategy as a binary vector $\boldsymbol{a} = \{0,1\}^n$, with $\boldsymbol{a}_i = 1$ iff target i is covered. Let $\mathcal{A}$ be the set of pure strategies, and $|\mathcal{A}|$ be the size of the set. The mixed normal form strategy $\boldsymbol{x}$ is a $|\mathcal{A}|$-vector. Let A be an $n \times |\mathcal{A}|$ binary matrix denoting if a column pure strategy covers a row target. Therefore we have

$$\boldsymbol{p} = A\boldsymbol{x} \tag{3}$$

meaning that $\boldsymbol{p}_i$ is computed by summing up all the probabilities of the pure strategies with target i covered. In our game setting, covering a composable target entails completing a series of *tasks* [28] through collaborations within a team of *defense agents* (agents for short). We call a target i has a *task set* denoted as $\mathcal{T}_i$, and the set of all tasks is $\mathcal{T} = \cup_{i=1}^n \mathcal{T}_i$. Suppose the defense team has m agents. To model the resource budget, we define the following notations:

- C: Cost matrix, an $m \times |\mathcal{T}|$ matrix with each entry $C_{i,j}$ representing the cost for an agent i to complete a task t_j.
- $\boldsymbol{b}$: Budget constraint, an m-dimensional vector $\boldsymbol{b}$ where $\boldsymbol{b}_i$ is time budget for agent i.
- S: Assignment matrix, an $m \times |\mathcal{T}|$ binary matrix representing the task assignment to each agent. $S_{i,j} = 1$ iff task t_j is assigned to agent i.

We define an action $\boldsymbol{a}$ is *feasible* if an assignment matrix S exists that 1) it assigns all tasks to agents ($\sum_{i=1}^m S_{i,j} = 1, \forall t_j \in \cup_{\{k|\boldsymbol{a}_k=1\}} \mathcal{T}_k$), and 2) every agent's cost is within the budget constraint ($SC^T \le \boldsymbol{b}$). A strategy is feasible iff all associated actions are feasible. We use $\hat{\mathcal{A}}$ to denote the set of all the feasible defender actions.

[2]Previous work also names it marginal form or marginal strategy

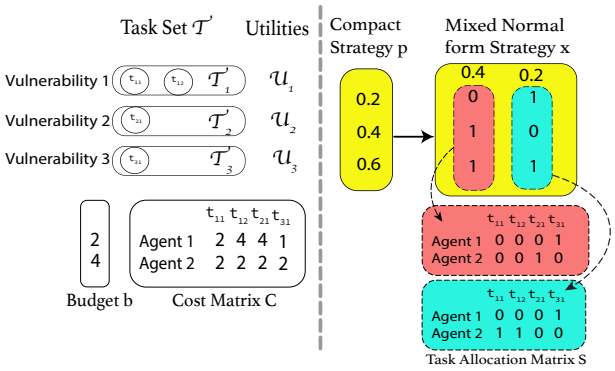


Figure 2: Problem example.

Each vulnerability is composed of different tasks with different precedence constraints. A vulnerability is successfully defended if all tasks are finished; otherwise, the target is not defended. In the meantime, each defense agent has a hard-limit budget and a cost for different types of tasks. The goal is to compute the defender's optimal and feasible normal-form strategies. Specifically, given n composable targets, m defense agents with their utility matrix $\boldsymbol{\mathcal{U}}$, task set $\mathcal{T}$, cost matrix C, and budget constraint $\boldsymbol{b}$, we will compute the defender's optimal and feasible normal form strategy $\boldsymbol{x}$, as well as the corresponding feasible assignment matrix S.

Example. Figure 2 demonstrates an example the composable target team defense problem. $\{t_{1,1}, t_{1,2}\}$, $\{t_{2,1}\}$ and $\{t_{3,1}\}$ are the task sets that belong to vulnerability 1, 2 and 3. There are two agents available, and because there are four tasks in total, the defender has a 2×4 cost matrix C with each entry representing the time for each agent to complete each task. Suppose $[0.2, 0.4, 0.6]^T$ is the compact strategy denoting the probability of defending each vulnerability. The mixed normal form strategy is that the defender has probability 0.4 to apply the pure strategy of defending vulnerability 2 and 3 and 0.2 to defend 1 and 3, since $[0.2, 0.4, 0.6]^T = 0.4 \cdot [0, 1, 1]^T + 0.2 \cdot [1, 0, 1]^T$. Defending vulnerabilities 2 and 3 requires completing all the tasks in these two vulnerabilities. Thus, we have a task allocation solution S for each pure strategy with positive probability in the mixed normal form strategy. For the pure strategy $[0, 1, 1]^T$, the task allocation is to allocation task $t_{2,1}$ to agent 2 and task $t_{3,1}$ to agent 1.

4 DEFENDER'S OPTIMAL STRATEGY

Composable target team defense brings unique challenges in computing optimal defender strategies under a large normal-form strategy space. The solution to our problem comprises two parts: computing the optimal compact form strategy and ensuring the computed optimal compact strategy is feasible.

We will compute the strong Stackelberg equilibrium (SSE) for optimal strategies because our model shares the commonality [17] that the attacker would choose strategy optimally for the defender when these strategies lead to the same maximum attacker payoffs.

Let ϕ and ψ be the defender's and attacker's payoffs, and M be a huge value. We formulate the composable target team defense problem as a MILP (P0) as follows:

$$\max \quad \phi \tag{4}$$

$$\boldsymbol{g}_i \in \{0,1\} \quad 1 \le i \le n \tag{5}$$

$$\textstyle\sum_{i=1}^{n} \boldsymbol{g}_i = 1 \tag{6}$$

$$\textstyle\sum_{\boldsymbol{a}\in\hat{\mathcal{A}}} \boldsymbol{a}_i \boldsymbol{x}_i = \boldsymbol{p}_i \quad 1 \le i \le n \tag{7}$$

$$\boldsymbol{x}_i \in [0,1] \quad 1 \le i \le |\hat{\mathcal{A}}| \tag{8}$$

$$\textstyle\sum_{i=1}^{|\hat{\mathcal{A}}|} \boldsymbol{x}_i \le 1 \tag{9}$$

$$\phi - (\boldsymbol{p}_i U_d^{c,i} + (1-\boldsymbol{p}_i) U_d^{u,i}) \le (1-\boldsymbol{g}_i)\cdot M \; 1 \le i \le n \tag{10}$$

$$0 \le \psi - (\boldsymbol{p}_i U_a^{c,i} + (1-\boldsymbol{p}_i) U_a^{u,i}) \le (1-\boldsymbol{g}_i)\cdot M \; 1 \le i \le n \tag{11}$$

$\hat{\mathcal{A}}$ in equation 9 is the set of all feasible defender's pure strategies. Equations 5, 6, and 11 ensure that the attacker selects a single target and gains the highest payoff, and Equation 10 ensures the defender gains the highest payoff. Determining the feasible pure strategy set is hard because for any pure strategy $\boldsymbol{a}$, solving integer programming is required to check whether a task assignment is within budget limits. And from all the feasible compact strategies, we need to find the optimal one that maximizes the defender's payoff. To achieve this, we propose to use binary search to find the optimal defender's payoff iteratively. We make the key observation that the attacker's payoff ψ is continuous and monotonically decreasing as the defender increases coverage in the compact form strategy. If the compact form strategy is an SSE, the defender's payoff will monotonically increase as the coverage increases. Using the SSE compact form strategy as the bridge, we can find that the defender's payoff monotonically increases as the attacker's payoff decreases. This property makes the binary search possible to find the optimal defender's payoff. Instead of doing a binary search on the defender's payoff like previous work [33], we do a binary search on the attacker's payoff because our defender's payoff is not continuous. As the key challenge of the algorithm is runtime efficiency and the performance bottleneck is the compact-to-normal form conversion, we need to reduce the number of enumerations to decrease the number of conversions by doing the binary search. The following paragraphs will provide a formal definition and proof of this property.

To achieve this goal, we first define the comparison of two compact-form strategies $\boldsymbol{p}$ and $\boldsymbol{p'}$, $\boldsymbol{p} \ge \boldsymbol{p'}$ iff $\boldsymbol{p}_i \ge \boldsymbol{p'}_i \; \forall i$, meaning that the coverage of any target i in $\boldsymbol{p}$ is greater or equal to that in $\boldsymbol{p'}$. Based on the definition, we have the following propositions:

Proposition 4.1. *The defender's SSE compact-form strategy $\boldsymbol{p}$ monotonically decreases as the attacker's payoff ψ increases.*

Proof. Existing work [17] has shown that the SSE compact-form strategy of a given ψ, denoted as $f(\overline{\psi})$, can be computed by:

$$\boldsymbol{p}_i = {}^{(\overline{\psi}-U_a^{u,i})}/_{(U_a^{c,i}-U_a^{u,i})} \quad \text{if } \overline{\psi} < U_a^{u,i} \tag{12}$$

$$\boldsymbol{p}_i = 0 \quad \text{otherwise} \tag{13}$$

Because $\overline{\psi} - U_a^{u,i} < 0$, $\boldsymbol{p}_i$ decreases as ψ increases. Therefore, the defender's SSE compact-form strategy $\boldsymbol{p}$ monotonically decreases as the attacker's payoff ψ increases. □

This proposition yields the theorem below:

Algorithm 1: The SWING Algorithm Framework

Input: target set $[n]$, utility matrix for every target, agent set $[m]$, time cost matrix C and budget constraint vector $\boldsymbol{b}$.
Output: A normal form feasible and optimal strategy $\boldsymbol{x}$.
Initialization: $left \leftarrow \max_i U_a^{c,i}$, $right \leftarrow \max_i U_a^{u,i}$.
while $right - left \ge \theta$ **do**
  $\overline{\psi} \leftarrow \frac{left+right}{2}$.
  Compute the compact form defender's SSE strategy $\boldsymbol{p} := f(\overline{\psi})$
  Solve program (P2) and let $\boldsymbol{x}^*$ be the solution.
  if $\mathbf{1}^T\boldsymbol{x}^* \le 1$ **then**
    $left \leftarrow \overline{\psi}$.
  else
    $right \leftarrow \overline{\psi}$.
  end if
end while
return $\boldsymbol{p}$ and $\boldsymbol{x}^*$

Theorem 4.2. *The attacker's payoff ψ decreases monotonically as the defender's payoff ϕ increases.*

Proof. Let $\boldsymbol{p} := f(\psi)$. Proposition 4.1 tells us ψ decreases, $\boldsymbol{p}$ increases. Let $\boldsymbol{g}$ be the attacker's strategy best response to $\boldsymbol{p}$ that aligns with the definition of SSE. Then when $\boldsymbol{p}$ increases, defender's payoff $R_d(\boldsymbol{p}, \boldsymbol{g})$ increases. □

A normal form strategy $\boldsymbol{x}$ that corresponds to $\boldsymbol{p}$ is feasible if it satisfies Equations 7 and 9. Let $\hat{A}$ be an $n \times |\hat{\mathcal{A}}|$ binary matrix with each column a feasible pure strategy $\boldsymbol{a} \in \hat{\mathcal{A}}$. Finding $\boldsymbol{x}$ satisfying $\hat{A}\boldsymbol{x} = \boldsymbol{p}$ is essentially the same as solving the following linear program (P1):

$$\min \mathbf{1}^T\boldsymbol{x} \quad s.t. \quad \hat{A}\boldsymbol{x} = \boldsymbol{p}, \boldsymbol{x} \ge \mathbf{0}, \tag{P1}$$

A feasible normal form strategy exists iff $\min \mathbf{1}^T\boldsymbol{x} \le 1$. Next, we relax the constraints in program (P1) and yield an inequality-constrained program (P1) with the change of $A\boldsymbol{x} = \boldsymbol{p}$:

$$\min \mathbf{1}^T\boldsymbol{x} \quad s.t. \quad \hat{A}\boldsymbol{x} \ge \boldsymbol{p}, \boldsymbol{x} \ge \mathbf{0} \tag{P2}$$

The solution for (P2) is at least as good as the one for (P1). Also, we have the following lemma:

Proposition 4.3. *Given $\boldsymbol{p}$, let $\boldsymbol{x}^*$ be the optimal solution of (P2) and $\delta := \mathbf{1}^T\boldsymbol{x}^*$. There exists a polynomial time computable function that can map $\boldsymbol{x}^*$ to $\boldsymbol{x'}^*$ where $A\boldsymbol{x'}^* = \boldsymbol{p}$ and $\mathbf{1}^T\boldsymbol{x'}^* = \delta$. (Proof in appendix)*

This proposition tells us that solving (P2) is equivalent to solving (P1). With Propositions 4.1 and 4.3, we have the following theorem:

Theorem 4.4. *For any attacker's payoff ψ, if ψ does not have a corresponding feasible normal-form strategy, then any attacker's payoffs less than ψ will not have a feasible normal-form strategy.*

Proof. Let $\boldsymbol{p} = f(\psi)$, according to Proposition 1, if $\psi > \psi'$, $f(\psi) < f(\psi')$. In other words, $\boldsymbol{p} < \boldsymbol{p'}$. If we solve program (P2) for both $\boldsymbol{p}$ and $\boldsymbol{p'}$ and let $\boldsymbol{x}^*$ and $\boldsymbol{x'}^*$ as their optimal solutions respectively. Obviously, $\mathbf{1}^T\boldsymbol{x}^* < \mathbf{1}^T\boldsymbol{x'}^*$ since all entries of $\hat{A}$ and $\boldsymbol{x}$ are non-negative. If $\mathbf{1}^T\boldsymbol{x} \le 1$, the normal form strategy is feasible; and if $\mathbf{1}^T\boldsymbol{x} > 1$, it is infeasible. Since $\mathbf{1}^T\boldsymbol{x}^* < \mathbf{1}^T\boldsymbol{x'}^*$, if $\boldsymbol{x}^*$ is infeasible, then $\boldsymbol{x'}^*$ is infeasible. Conversely, if $\boldsymbol{x'}^*$ if feasible, then $\boldsymbol{x}^*$ is feasible. □

These two theorems prove the lowest attacker's payoff with a feasible normal-form strategy will yield the defender's optimal strategy and that we can solve the model by binary search where we search the value of the attacker's payoff and compute the corresponding defender's strategy and payoff. Furthermore, the searching value only needs to range between $\max_i U_a^{c,i}$ and $\max_i U_a^{u,i}$ (the maximum attacker utilities over targets, with and without defense) because even though the defender covers all the targets, the attacker can still select the target with the highest attack utility under defense. As shown in Algorithm 1, in every iteration, we select the average value of the current searching range (Line 5) and check if the attacker's payoff has a corresponding normal-form strategy (Line 6 - Line 8). If so, we continue the search to the left half of the value range, otherwise we go for the right half. The algorithm terminates when the search range is marginally small.

5 NORMAL-FORM STRATEGY

In this section, we focus on Algorithm 1's Line 7 of converting compact form strategy to normal form strategies considering two scenarios: without and with schedule constraints.

5.1 Conversion without Schedule Constraints

We use column generation for the solution since the number of matrix $\hat{A}$ columns ($O(2^n)$) is exponentially larger than the number of rows (n). Column generation starts with a small initialized subset of columns and iteratively adds the next optimal and feasible column $\boldsymbol{a}^*$ that can help reduce the objective function of (P2). Let $\boldsymbol{y}$ be the dual variables of (P2). Finding the next optimal column is essentially to solve the following subproblem (P3):

$$\max \quad \sum_{k=1}^{n} \boldsymbol{y}_k \boldsymbol{a}_k \tag{14}$$

$$\sum_{i=1}^{m} S_{i,j} \geq \boldsymbol{a}_{\{k|t_j \in \mathcal{T}_k\}} \qquad 1 \leq k \leq n, \forall t_j \in \mathcal{T} \tag{15}$$

$$\sum_{i=1}^{m} S_{i,j} \leq 1 \qquad \forall t_j \in \mathcal{T} \tag{16}$$

$$\sum_{t_j \in \mathcal{T}} S_{i,j} \cdot C_{i,j} \leq \boldsymbol{b}_i \qquad 1 \leq i \leq m \tag{17}$$

Note that $S_{i,j}, \boldsymbol{a}_k \in \{0,1\} \forall i, j, k$. Equation 15 and 16 is to show that if target k is covered then all the tasks t_j that belong to task set $\mathcal{T}_k$ must be assigned to one and only one agent. And equation 17 is the feasibility check that the assignment matrix S cannot exceed the budgets. Then we have Algorithm 2, a column generation method to solve (P2).

Algorithm 2: Conversion without Schedule Constraints

1: **Input** : Marginal form strategy $\boldsymbol{p}$, matrix A.
2: **Output** : The $\boldsymbol{x}^*$ of (P2) and the corresponding task assignment matrix S.
3: **Initialization** : let A be an $n \times n$ identity matrix.
4: **while** True **do**
5: Let $\boldsymbol{y}$ be the dual variables of program (P2).
6: Let $\boldsymbol{a}^*$ be the optimal solution of program (P3) .
7: **if** $\boldsymbol{a}^*$ in A or $\boldsymbol{a}^*$ is dominated by any column in A **then**
8: *break*.
9: **end if**
10: $A \leftarrow [A \quad \boldsymbol{a}^*]$.
11: **end while**
12: $\boldsymbol{x}^* \leftarrow$ the solution of P2 with the current A.
13: **Return** $\boldsymbol{x}^*$.

5.2 Conversion with Schedule Constraints

When incorporating task ordering constraints within each target, column generation remains a viable approach to solving the program (P2). However, instead of using (P3) to generate a new column, we need a new program (P4) due to distinct feasibility requirements stemming from schedule constraints. In this formulation, let t_{ij} be the jth task in $\mathcal{T}_i$ and c_{ijk} be the processing time of t_{ij} for agent k. And $\boldsymbol{b}_k$ is the time budget agent k possesses. We define s_{ijk} as the starting time of task t_{ij} for agent k. v_{ijk} serves as an indicator for whether t_{ij} is processed by agent k. v_{ijk} is set to 1 if it is; and 0 otherwise. Introducing z_{ijhgk} to capture schedule of task t_{ij} and t_{hg} on agent k, $z_{ijhgk} = 0$ if t_{ij} is processed before t_{hg} on agent k and $z_{ijhgk} = 1$ otherwise. We formulate (P4) as follows:

$$\max \sum_{i=1}^{n} \boldsymbol{y}_i \boldsymbol{a}_i \tag{18}$$

$$s_{ijk} \leq v_{ijk} \cdot M \quad \forall i, j \tag{19}$$

$$s_{ijk} \geq s_{hgk} + c_{hgk} - (3 - z_{ijhgk} - v_{ijk} - v_{hgk}) \cdot M \forall i < h, \forall j, g, k \tag{20}$$

$$s_{hgk} \geq s_{ijk} + c_{ijk} - (z_{ijhgk} + 2 - v_{ijk} - v_{hgk}) \cdot M \forall i < h, \forall j, g, k \tag{21}$$

$$\sum_k s_{ijk} \geq \sum_k s_{ij-1k} + \sum_k v_{ij-1k} c_{ij-1k} \cdot M \tag{22}$$

$$\sum_k v_{ijk} \leq \sum_k v_{ij-1k} \quad \forall i, j \tag{23}$$

$$\sum_k v_{ijk} \geq \boldsymbol{a}_i \quad \forall i, j \tag{24}$$

$$\sum_k v_{ijk} \leq 1 \quad \forall i, j \tag{25}$$

$$s_{ijk} \geq 0 \quad \forall i, j, k \tag{26}$$

$$s_{ijk} + v_{ijk} \cdot c_{ijk} \leq b_k \quad \forall i, j, k \tag{27}$$

$$v_{ijk} \in \{0,1\} \quad \forall i, j, k \tag{28}$$

$$\boldsymbol{a}_i \in \{0,1\} \quad \forall i \tag{29}$$

If task t_{ij} is not assigned to agent k, constraint 19 sets the starting time of t_{ij} for agent k to 0. Constraints 20 and 21 ensure that an agent can only process one task at a time. And two tasks t_{ij} and t_{hg} can only be sequenced when both v_{ijk} and v_{hgk} both take 1; otherwise, they bear no relationship with one another on agent k. Constraint 22 guarantees that each task is performed in a specified order. Constraint 23 requires task t_{ij} must follow the completion of t_{ij-1}. Constraints 24 and 25 mirror Constraints15 and 16, respectively.

In Line 6 of Algorithm 2, replacing (P3) with (P4) transforms Algorithm 2 into a methodology addressing (P2) while accounting for task schedule constraints.

Solving program (P4). Directly solving (P4) is time-consuming due to the large number of variables inherent in the formulation. We create a branch and bound-based algorithm that approximates the solution for (P4). Solving (P4) entails two steps: a) identifying all feasible pure strategies $\boldsymbol{a}$, and b) finding the pure strategy with the optimal reward $\sum_{i=1}^{n} \boldsymbol{y}_i \boldsymbol{a}_i$ from feasible strategy set. The assessment of feasibility for any given pure strategy $\boldsymbol{a}$ can be reduced to the Flexible Job shop Scheduling Problem (FJSP) [2], where one has multiple jobs, each of which requires to complete a set of operations in a specific order. Each operation can be processed by a set of machines with varying processing power. There is a rich existing literature of algorithms proposed to solve FJSP [5, 16, 34], and these

Algorithm 3: Sovling (P4) with branch and bound

Input: The dual variables of (P1) $\boldsymbol{y}$, task set and their schedule and time cost $c_{ijk} \forall i, j, k$
Output: The optimal feasible pure strategy $\boldsymbol{a}$ that has the maximum reward.
$nodelist \leftarrow \{Root\}$, $v^* \leftarrow 0$.
while *nodelist* is not empty **do**
  Remove from *nodelist* the node with the best reward upper bound
  Let an undetermined variable $\boldsymbol{a}_i \leftarrow 1$ and we get a new node $node_1$
  Call *FJSP-heuristic* to determine whether it is feasible
  if Feasible **then**
    Not prune, add this new node to *nodelist*
  else
    Prune
  end if
  Let $\boldsymbol{a}_i \leftarrow 0$ and we get a new node $node_0$. Add $node_0$ to the *nodelist*
  for node $node_0$ and the feasible $node_1$ **do**
    $R_{ub} \leftarrow$ *Solve program (P3)*
    Call *Greedy-assign + FJSP-heuristic* to get a feasible solution $\hat{\boldsymbol{a}}$, and let R_{lb} be the reward of $\hat{\boldsymbol{a}}$
    if $R_{lb} > v^*$ **then**
      $v^* \leftarrow R_{lb}$
      Return $\hat{\boldsymbol{a}}$
    end if
    if $R_{lb} > R_{ub}(node_v)$ for $node_v \in nodelist$ **then**
      Remove $node_v$ from *nodelist*
    end if
    if $R_{ub} < R_{lb}(node_v)$ for $node_v \in nodelist$ **then**
      Remove current node ($node_0$ or $node_1$) from *nodelist*
    end if
  end for
end while
Return the leaf node $\boldsymbol{a}$ with highest R_{lb}.

well-established methods can be effectively leveraged to ascertain the feasibility of any provided pure strategy $\boldsymbol{a}$.

As Algorithm 3 shows, we introduce a branch and bound-based method to heuristically search for the optimal pure strategy $\boldsymbol{a}^*$ with the highest reward. We incrementally fix individual entries within $\boldsymbol{a}$ while constructing a binary tree structure. At each node, we compute lower and upper bounds to determine the viability of further exploration. Our unique problem context involves an initial assessment of feasibility using an FJSP algorithm prior to establishing lower and upper bounds, which makes us diverge from conventional branch and bound methods, Specifically, we employ the $FJSP-heuristic$ [34] due to its superior runtime efficiency. If a node is feasible, we proceed to compute its lower and upper bounds; otherwise, we prune the branch without further analysis. For the upper bound R_{ub} of a node, we utilize the maximum objective function value derived from solving (P3) without considering ordering constraints. We adopt a greedy approach to establish a lower bound, progressively allocating the highest reward target while verifying feasibility using the FJSP heuristic. The sum $\sum_{i=1}^{n} \boldsymbol{y}_i \boldsymbol{a}_i$ serves as the calculated lower bound for each node. To further accelerate the computation, we maintain a record of maximum reward value v^* during column generation. And we terminate the branch and bound search and return if there is a new column $\boldsymbol{a}$ with value $v > v^*$.

Another approach to improve efficiency is incorporating a warm-up session at the outset of the column generation process. This session establishes an initial "favorable" column, with a higher likelihood of persisting within the support set of the final solution. Each FJSP solution can be represented by a disjunctive graph G [2] where nodes signify operations and edges denote operation schedule. The critical path within graph G delineates the shortest time required for completing all jobs. Applying the FJSP heuristic to the pure strategy $\boldsymbol{a} = \mathbf{1}$, which encompasses all targets, yields an infeasible solution graph G with a minimal time surpassing the budgets. By extracting the critical path from G and progressively de-assigning the target with the maximum operation (task) count along this path, a new strategy $\boldsymbol{a}'$ is derived. This iterative process, bolstered by successive applications of FJSP-heuristic to $\boldsymbol{a}'$, persists until a feasible pure strategy is achieved. Subsequently, this attained strategy is incorporated into the initial column set.

6 EVALUATION

The experiments aim to answer the three following questions:

(1) How efficient and effective is SWING in computing the defender's optimal strategy under different scenarios?
(2) How effective is approach's design in improving the performance of solving the composable target team defense problem with precedence constraints?
(3) How effective is SWING in performing real-world cybersecurity defense?

We implemented SWING as a single-thread Python program. We use the commercial optimization software Gurobi to represent the generic LP/MILP solver compared against SWING. All experiments are run on a server with Intel(R) Xeon(R) CPU E5-2676 v3 at 2.40GHz.

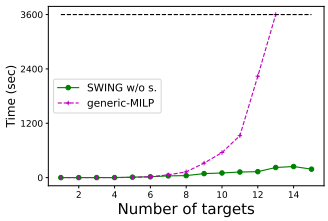
6.1 Algorithm Effectiveness and Efficiency

In this experiment, we compare the performance of SWING against two baseline approaches: (1) generic-MILP, which treats the composable target team defense problem as a generic Mixed Integer Linear Programming (MILP) problem solved by an MILP solver, and (2) random, which randomly selects targets to cover and assigns tasks to agents using a greedy allocation strategy.

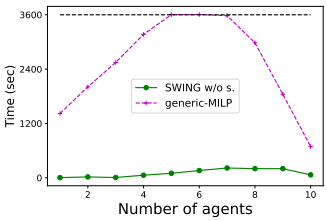
For each comparison, we evaluate two different scenarios: one without precedence constraints. We treat the number of targets and the number of agents as two independent variables in the experiment. For each scenario, we vary the value of one variable while keeping the other constant. When the parameters are not treated as variables, we set the number of targets to 12 and the number of agents to 10 in no-schedule scenarios and the number of targets to 6 and agents to 5 in with-schedule scenarios.

In each configuration, the experiment was repeated 20 times. Utility values were randomly generated between 1 and 10 for covered (uncovered) targets for the defender (attacker) and between -10 and -1 for the uncovered (covered) targets. All experiments are set with a one-hour threshold. We cut out the experiment and claim no solution if the approach is not completed within 1 hour.

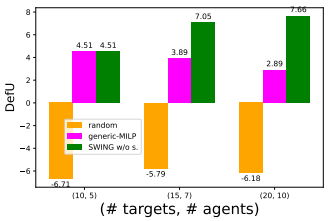
Figure 3a and Figure 4a present the time cost of each approach to solving games with varying numbers of targets for tasks without and with precedence constraints, respectively. Our algorithm exhibits significantly superior efficiency compared to generic-MILP. Specifically, SWING is 16 times faster than generic-MILP on average. As the number of targets increases, the computational cost of generic-MILP grows exponentially. It hits the 1-hour cutoff when the number of targets reaches 13 for tasks without precedence constraints. In contrast, SWING solves all instances within an average



(a) Time cost over different targets.

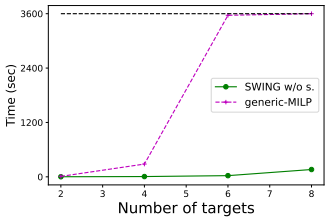


(b) Time cost over different agents.



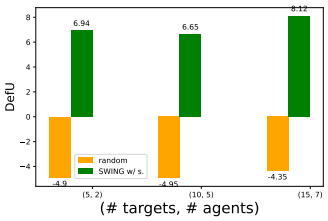
(c) Defender utility.

Figure 3: Time cost and defender utility for the games without precedence constraints.



(a) Time cost over different targets.

(b) Time cost over different agents.



(c) Defender utility.

Figure 4: Time cost and defender utility for the games with precedence constraints. The baseline approaches failed to generate any validate solutions within 1 hour.

time of 223 seconds. For tasks with precedence constraints, generic-MILP cannot solve any of the games within 1 hour, while SWING performs efficiently with dozens of targets and agents.

Figure 3b and Figure 4b show the time cost over different agent numbers. Generic-MILP reached a 1-hour cutoff in the intermediary range at 5 to 7 agents, while our algorithm follows a similar trend with an initial increase followed by a subsequent decrease. This phenomenon aligns with intuition, as an extreme scarcity or abundance of security resources yields either a diminutive feasible strategy or an infeasible set. In the most extreme instances, where security resources are either insufficient to cover none of the targets or sufficient to cover all, both algorithms exhibit prompt termination.

We also compare the defender's utility associated with the strategies generated by the three approaches, as shown in Figure 3c and Figure 4c. With relatively small target and agent numbers, SWING and generic-MILP found strategies with equal defender utilities. When the scale increases, generic-MILP is cut off under the 1-hour threshold and ends with sub-optimal solutions. In contrast, our algorithm consistently generates effective solutions within 1 hour. Our algorithm demonstrates progressively enhanced solution improvements relative to generic-MILP as the scale amplifies.

6.2 Ablation Study for Precedence Constraints

Recall that we developed a two-pronged approach to assess the implementability of a compact-form strategy and compute the corresponding normal-form mixed strategy for games with task precedence constraints. In addition to the branch-and-bound algorithm, we developed a warm-up process that identifies favorable initial columns. In this experiment, we conduct an ablation study to evaluate the effectiveness of both the base design and the additional warm-up process in solving the composable target team defense problem.

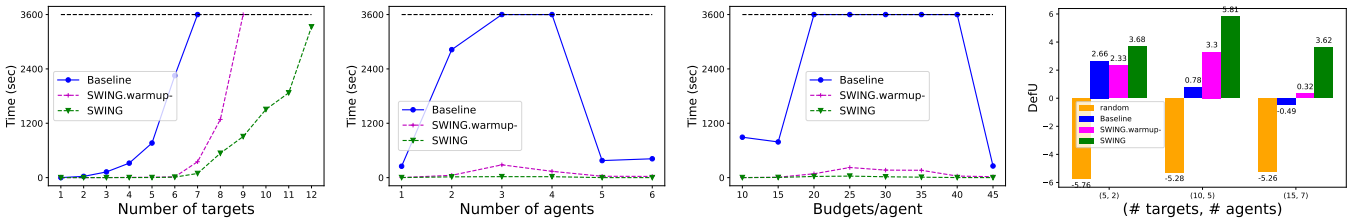
Specifically, we compare SWING with a variant, SWING.warmup-, which excludes the warm-up process, and benchmark both against a baseline that applies the no-precedence constraint solution(use MILP solver to solve program P4) to the precedence problem. The experiment design is identical to that of Section 6.1, where we enumerate the number of targets or agents while controlling other parameters.

Figure 5a shows the exponential growth in runtime for all approaches as the number of targets increases. While our algorithm uses branch-and-bound techniques to prune unnecessary branches early, its worst-case complexity remains $O(2^n)$. Even so, the baseline hits the 1-hour cutoff with the fewest targets. SWING.warmup- reaches the 1-hour cutoff at 9 targets, whereas SWING requires less than $\frac{1}{3}$ of that time at the same target count.

Figure 5b presents the time cost as the number of agents varies. All three approaches initially exhibit a rise in runtime, followed by a decline as more security resources are allocated. With 5 or 6 agents, the runtimes stabilize, highlighting the saturation point where additional agents no longer significantly impact time costs. The results also underscore the complexity introduced by precedence constraints, as the baseline shows significant time costs even for small games (6 targets and 3 agents).

Figure 5c demonstrates the time cost with varying budget sizes, holding 6 targets and 5 agents constant. As budgets increase from 20 to 40, the baseline exceeds the 1-hour threshold. In contrast, both SWING and SWING.warmup- peak at a budget of 25, with runtimes of 33 seconds and 223 seconds, respectively. SWING is more than 100 times faster than the baseline, emphasizing its efficiency in such scenarios. In smaller-scale instances, both SWING and SWING.warmup- terminate quickly, highlighting the performance gains from the branch-and-bound approach.

Finally, we compare defender utilities produced by the three approaches across different game setups, as shown in Figure 5d.

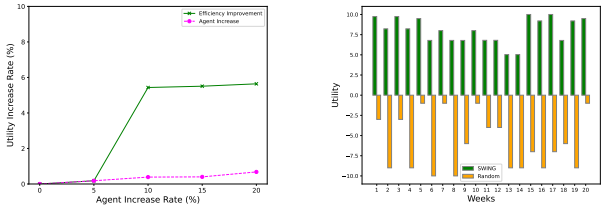


(a) Time cost over targets. (b) Time cost over agents. (c) Time cost over agent budgets. (d) Defender utility.

Figure 5: Time cost and computed defender utility for the baseline and two SWING variants running different game instances. SWING.warmup-: SWING without warm-up.

SWING consistently delivers higher defender utilities than both the baseline and SWING.warmup- across all instances.

6.3 Real-World Application: Company T



(a) Defender utility over weeks. (b) Invested utility increase.

Figure 6: Defenfender utility over weeks and the increase with different types of investiment.

We collaborated with a Fortune 500 technology company T and applied SWING to optimize their task assignment in vulnerability mitigation. Company T has a team of 38 employees dedicated to vulnerability mitigation, and their mitigation process follows the workflow illustrated in Figure 1. The company receives about 50 high-risk reports each week that require immediate resolution.

Since the total time required to analyze and mitigate all urgent vulnerabilities exceeds the available working hours of their employees, Company T must prioritize vulnerabilities. It currently employs a hybrid algorithm combining greedy and randomization. The assignment algorithm is based on the weekly working hours of each employee. It randomly selects a subset of urgent vulnerabilities so that all tasks associated with them can be completed within the week.

We applied SWING to 20 weeks of mitigation process where every week is treated as an independent game and compared our solution against the company's current task assignment strategy. Figure 6a presents the defender utility associated with SWING versus that with the current assignment method for each week. The results indicate that SWING consistently outperformed the existing strategy.

We also calculated the probability that an attacker would exploit an unmitigated vulnerability under both the current assignment strategy and SWING. We consider sophisticated attackers, such as Advanced Persistent Threat (APT) actors, who may have insider knowledge and can optimize their attack strategy based on the defender's strategy, based on the consideration that high-risk vulnerabilities are more likely to be used for APT attacks. SWING increased the average detection rate of high-risk vulnerabilities from 10% to 90.91%. Given that the impact of high-risk vulnerabilities can range from millions to billions of dollars in damages [13, 14, 27, 29], increasing the mitigation rate for such vulnerabilities could save the company hundreds of thousands of dollars every week. We also used our model to guide the company in making future resource investments. Specifically, we examined whether Company T should invest in hiring additional personnel or improving the efficiency of its existing workforce (e.g., through training or purchasing better tools). To address this, we retrospectively applied SWING to the 20-week data, simulating scenarios with different numbers of employees and reduced per-task time costs. Figure 6b illustrates the outcomes of these simulations. Our results indicate that improving efficiency contributes to more utility increases with the same increase rate of employees.

7 CONCLUSION

In this paper, we introduced the composable target team defense problem and presented SWING, a novel algorithm designed to efficiently generate optimal feasible defender strategies with budget constraints against adversaries. Our extensive testing demonstrates that SWING significantly outperforms existing methods, both in terms of efficiency and effectiveness, when applied to real-world-scale simulated instances. We also apply SWING in a real-world scenario for a Fortune 500 company. The study demonstrates SWING's ability to better allocate defender resources, reducing the likelihood of an attacker successfully exploiting vulnerabilities. It also suggests that SWING can guide future investments and provide the company insights on enhancing their security posture efficiently.

ACKNOWLEDGMENTS

For her generous and valuable assistance in conducting experiments, we give heartfelt thanks to Ati Priya Bajaj. This work is sponsored by and related to the Department of Navy award N00014-23-1-2563 issued by the Office of Naval Research and National Science Foundation under Award No. 2247954. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Office of Naval Research or the National Science Foundation.

REFERENCES

[1] Avrim Blum, Nika Haghtalab, and Ariel D Procaccia. 2014. Learning Optimal Commitment to Overcome Insecurity. In *Advances in Neural Information Processing Systems*, Vol. 27. Curran Associates, Inc. https://papers.nips.cc/paper/2014/hash/cc1aa436277138f61cda703991069eaf-Abstract.html

[2] Paolo Brandimarte. 1993. Routing and scheduling in a flexible job shop by tabu search. *Annals of Operations Research* 41, 3 (1993), 157–183. https://doi.org/10.1007/BF02023073

[3] Matthew Brown, Arunesh Sinha, Aaron Schlenker, and Milind Tambe. 2016. One size does not fit all: a game-theoretic approach for dynamically and effectively screening for Threats. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence* (Phoenix, Arizona, 2016-02-12) *(AAAI'16)*. AAAI Press, 425–431.

[4] Eric Budish, Yeon-Koo Che, Fuhito Kojima, and Paul Milgrom. [n.d.]. Designing Random Allocation Mechanisms: Theory and Applications. 103, 2 ([n. d.]), 585–623. https://doi.org/10.1257/aer.103.2.585

[5] Ronghua Chen, Bo Yang, Shi Li, and Shilong Wang. 2020. A self-learning genetic algorithm based on reinforcement learning for flexible job-shop scheduling problem. *Computers & Industrial Engineering* 149 (2020), 106778. https://doi.org/10.1016/j.cie.2020.106778

[6] Vincent Conitzer and Tuomas Sandholm. 2006. Computing the optimal strategy to commit to. In *Proceedings of the 7th ACM conference on Electronic commerce*. ACM, Ann Arbor Michigan USA, 82–90. https://doi.org/10.1145/1134707.1134717

[7] Fei Fang, Thanh Nguyen, Rob Pickles, Wai Lam, Gopalasamy Clements, Bo An, Amandeep Singh, Milind Tambe, and Andrew Lemieux. 2016. Deploying PAWS: Field Optimization of the Protection Assistant for Wildlife Security. *Proceedings of the AAAI Conference on Artificial Intelligence* 30, 2 (Feb. 2016), 3966–3973. https://doi.org/10.1609/aaai.v30i2.19070

[8] Fei Fang and Thanh H. Nguyen. 2016. Green security games: apply game theory to addressing green security challenges. *ACM SIGecom Exchanges* 15, 1 (Sept. 2016), 78–83. https://doi.org/10.1145/2994501.2994507

[9] Fei Fang, Thanh Hong Nguyen, Rob Pickles, Wai Y Lam, Gopalasamy R Clements, Bo An, Amandeep Singh, Milind Tambe, Andrew Lemieux, et al. 2016. Deploying PAWS: Field Optimization of the Protection Assistant for Wildlife Security.. In *AAAI*, Vol. 16. 3966–3973.

[10] Fei Fang, Peter Stone, and Milind Tambe. 2015. When Security Games Go Green: Designing Defender Strategies to Prevent Poaching and Illegal Fishing. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence*.

[11] Fei Fang, Peter Stone, and Milind Tambe. 2015. When security games go green: designing defender strategies to prevent poaching and illegal fishing. In *Proceedings of the 24th International Conference on Artificial Intelligence* (Buenos Aires, Argentina, 2015-07-25) *(IJCAI'15)*. AAAI Press, 2589–2595.

[12] Google. 2024. Google Bug Hunting Community. https://bughunters.google.com/

[13] IBM. 2024. IBM Cost of a Data Breach Report. https://www.ibm.com/reports/data-breach

[14] Information Week. 2024. CrowdStrike Outage Drained $5.4 Billion From Fortune 500: Report. https://www.informationweek.com/cyber-resilience/crowdstrike-outage-drained-5-4-billion-from-fortune-500-report

[15] Debarun Kar, Fei Fang, Francesco Delle Fave, Nicole Sintov, and Milind Tambe. 2015. " A Game of Thrones" When Human Behavior Models Compete in Repeated Stackelberg Security Games. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*. 1381–1390.

[16] Hamid Karimi, Seyed Habib A. Rahmati, and M. Zandieh. 2012. An efficient knowledge-based algorithm for the flexible job shop scheduling problem. *Knowledge-Based Systems* 36 (2012), 236–244. https://doi.org/10.1016/j.knosys.2012.04.001

[17] Christopher Kiekintveed, Manish Jain, Jason Tsai, James Pita, Fernando Ordónez, and Milind Tambe. 2009. Computing optimal randomized resource allocations for massive security games. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*. 689–696.

[18] Microsoft. 2024. Microsoft Security Response Center. https://www.microsoft.com/en-us/msrc

[19] Offsec Co. 2023. exploitdb: The Exploit Database. https://www.exploit-db.com/.

[20] James Pita, Manish Jain, Janusz Marecki, Fernando Ordóñez, Christopher Portway, Milind Tambe, Craig Western, Praveen Paruchuri, and Sarit Kraus. 2008. Deployed ARMOR protection: the application of a game theoretic model for security at the Los Angeles International Airport. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems: industrial track*. 125–132.

[21] James Pita, Milind Tambe, Chris Kiekintveld, Shane Cullen, and Erin Steigerwald. 2011. Guards: game theoretic security allocation on a national scale. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*. 37–44.

[22] Anibal Sanjab, Walid Saad, and Tamer Başar. 2017. Prospect Theory for Enhanced Cyber-Physical Security of Drone Delivery Systems: A Network Interdiction Game. https://doi.org/10.48550/arXiv.1702.04240

[23] Aaron Schlenker, Haifeng Xu, Mina Guirguis, Christopher Kiekintveld, Arunesh Sinha, Milind Tambe, Solomon Sonya, Darryl Balderas, and Noah Dunstatter. 2017. Don't Bury your Head in Warnings: A Game-Theoretic Approach for Intelligent Allocation of Cyber-security Alerts. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*. 381–387. https://doi.org/10.24963/ijcai.2017/54

[24] Arunesh Sinha, Fei Fang, Bo An, Christopher Kiekintveld, and Milind Tambe. 2018. Stackelberg Security Games: Looking Beyond a Decade of Success. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*. International Joint Conferences on Artificial Intelligence Organization, Stockholm, Sweden, 5494–5501. https://doi.org/10.24963/ijcai.2018/775

[25] Milind Tambe. 2011. *Security and game theory: algorithms, deployed systems, lessons learned*. Cambridge university press.

[26] Jason Tsai, Zhengyu Yin, Jun-young Kwak, David Kempe, Christopher Kiekintveld, and Milind Tambe. 2010. Urban security: Game-theoretic resource allocation in networked domains. In *Twenty-Fourth AAAI Conference on Artificial Intelligence*.

[27] USA Federal Trade Commission. 2024. Equifax Data Breach Settlement. https://www.ftc.gov/enforcement/refunds/equifax-data-breach-settlement

[28] Daniel Votipka, Rock Stevens, Elissa Redmiles, Jeremy Hu, and Michelle Mazurek. 2018. Hackers vs. testers: A comparison of software vulnerability discovery processes. In *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 374–391.

[29] Wired. 2022. The Untold Story of NotPetya, the Most Devastating Cyberattack in History. https://www.wired.com/story/notpetya-cyberattack-ukraine-russia-code-crashed-the-world

[30] Haifeng Xu, Long Tran-Thanh, and Nicholas R. Jennings. 2016. Playing Repeated Security Games with No Prior Knowledge. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems* (Richland, SC, 2016-05-09) *(AAMAS '16)*. International Foundation for Autonomous Agents and Multiagent Systems, 104–112.

[31] Rong Yang, Benjamin J Ford, Milind Tambe, and Andrew Lemieux. 2014. Adaptive resource allocation for wildlife protection against illegal poachers.. In *Aamas*. 453–460.

[32] Rong Yang, Albert Xin Jiang, and Milind Tambe. 2013. Scaling-up Security Games with Boundedly Rational Adversaries: A Cutting-plane Approach. In *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*. 404–410.

[33] Rong Yang, Fernando Ordonez, and Milind Tambe. 2012. Computing optimal strategy against quantal response in security games.. In *AAMAS*. 847–854.

[34] Mohsen Ziaee. 2014. A heuristic algorithm for solving flexible job shop scheduling problem. *The International Journal of Advanced Manufacturing Technology* 71, 1 (2014), 519–528. https://doi.org/10.1007/s00170-013-5510-z