Harnessing the Power of Deception in Attack Graph-Based Security Games

Stephanie Milani[1], Weiran Shen[1], Kevin S. Chan[2], Sridhar Venkatesan[3], Nandi O. Leslie[2], Charles Kamhoua[2], and Fei Fang[1]

[1] Carnegie Mellon University, 5000 Forbes Ave., Pittsburgh, PA 15213
smilani@andrew.cmu.edu,emersonswr@gmail.com,feif@cs.cmu.edu
[2] Army Research Laboratory, 2800 Powder Mill Road, Adelphi, MD 20783, USA
{kevin.s.chan.civ,nandi.o.leslie.ctr,charles.a.kamhoua.civ}@mail.mil
[3] Perspecta Labs Inc., 150 Mount Airy Road, Basking Ridge, NJ 07920, USA
svenkatesan@perspectalabs.com

Abstract. We study the use of deception in attack graph-based Stackelberg security games. In our setting, in addition to allocating defensive resources to protect important targets from attackers, the defender can strategically manipulate the attack graph through three main types of deceptive actions. We show that finding the optimal deception and defense strategy is at least NP-hard. We provide two techniques for efficiently solving this problem: a mixed-integer linear program for layered directed acyclic graphs (DAGs) and neural architecture search for general DAGs. We empirically demonstrate that using deception on attack graphs gives the defender a significant advantage, and the algorithms we develop scale gracefully to medium-sized problems.

Keywords: Deception · Attack graph · Security game.

1 Introduction

Security is a serious worldwide issue, involving defending important infrastructure [62], protecting endangered wildlife species [14, 25], securing computer networks [30, 41, 60], and more. Most security scenarios involve a defender who allocates limited security resources to protect targets from adversaries. To model and tackle these challenges, researchers have proposed game-theoretic approaches—especially those based on the Stackelberg security game (SSG) model. Many of these techniques have been successfully deployed in the real world [8, 50, 58, 59].

Attack graphs [1, 18, 31, 45, 53] are a commonly-used, versatile modeling tool for security challenges in different domains. They can model the abstract state dependency and transition relations of any vulnerable system [7, 19, 35, 44, 46, 49, 65], including a city's road system [33] and the topological structure of a company's computer network. Most existing work on attack graph-based SSGs assumes the attack graph is given and unchangeable. Another widely-used security technique is defensive deception [5, 48, 66], where the defender can alter

the appearance of targets or dynamically shift the attack surface. Commonly-used defensive deception mechanisms include honeypots [2, 6, 9, 16, 27, 37] and honeytokens [12] in cybersecurity, as well as camouflaging [63] and moving target defense [34]. Existing work on deception in security does not consider attack graphs or ignores the defender's ability to strategically allocate security resources to mitigate attacks. Despite its significance, the use of deception alongside the allocation of security resources on attack graph-based SSGs has not been studied.

To address this gap, we propose a variant of the SSG, an *Attack Graph Deception Game*, in which the defender can take deceptive and protective actions on an attack graph. In our novel game model, the defender uses deception to modify the graph structure; a less capable attacker may observe the modified graph structure during reconnaissance. The attacker then attacks targets by moving between nodes on the graph, while the defender attempts to thwart the attacker by protecting edges with her limited security resources. To model different attackers' skill levels, we consider the Bayesian setting, where the attacker has a type representing his ability to perceive the true graph structure.

We focus on directed acyclic attack graphs (DAGs) and are particularly interested in layered DAGs, which are commonly used to model networks [29, 38]. To solve layered DAGs, we propose a novel mixed-integer linear program (MILP)-based algorithm that builds upon the standard LP-based algorithm for solving Stackelberg games [17] and incorporates heuristic algorithms to significantly speed up computation. Our algorithm quickly finds the exact optimal solution for a special class of layered DAGs: bipartite DAGs. For general DAGs, we propose a neural architecture search (NAS)-based [23] algorithm, which uses a genetic algorithm to search for the modified attack graph and neural network optimization tools to find the remaining defense strategy. To our knowledge, this is the first use of NAS in SSGs. We conduct extensive experiments showing the scalability of our algorithms, and that using our algorithms and deception lead to significant increases in the defender's utility compared to baselines.

2 Game Model

An Attack Graph Deception Game is a two-player game on an attack graph. The defender chooses her strategy first; the attacker selects his strategy after surveillance. The attacker has a *type* $\theta \in \Theta$ drawn from a prior probability distribution. The defender knows the distribution, not the exact type. This type θ captures different skills and knowledge possessed by different attackers. For example, in cybersecurity, an attacker with superior reconnaissance skills has a greater chance of noticing honeypots and camouflage. In this section, we describe the attack graph, the players' strategy space, the players' payoff, and the solution concept. The notation that we use to describe our model is summarized in Table 1. We discuss the relaxation of some of our assumptions in Section 8.

Attack Graph. Attack graphs succinctly represent an attacker's possible attack paths. Of the many attack graph variants in the literature, we use a state-based Bayesian attack graph [21, 39, 55] $G = (N, E)$. The nodes N represent

states in which attackers can be. The edges E represent actions that attackers can take to transition between states. In the urban security, each node can represent an intersection of a road network. In cybersecurity, a node can represent the attacker's intrusion status, including the compromised computers and accessed databases. Following the common *monotonicity* assumption [7] that attackers will not relinquish previously attained capabilities, we assume the graph is a DAG. Furthermore, we assume that the players share knowledge of potential attacker actions.

We assume that at most one edge connects two nodes. For an attacker type θ and an edge $e = (n, n') \in E$, $q^\theta(e)$ is the intrinsic success probability for a type-θ attacker to reach state n' from state n through edge e. When an attacker reaches n, he receives reward $r(n) \geq 0$. *Targets* are the set of nodes $T \subset S$ where $r(n) > 0$. *Entry points* are the set of nodes with no incoming edges and $r(n) = 0$. We create auxiliary nodes as entry points for problems without natural ones.

Layered DAGs are a special case of DAGs where nodes are partitioned into l layers. Let n_i^j be the j-th node in layer i and $L_i = \{n_i^j \mid \forall j\}$ be the set of all nodes in layer i. In a layered DAG, E only contains edges that connect nodes in L_i to nodes in $L_{i+1}, \forall 1 \leq i \leq l-1$. Bipartite DAGs are special layered DAGs with $l = 2$. We are particularly interested in attack graphs that are layered DAGs because they are well-suited to model networks [29, 38], and we can take advantage of their structure to design efficient algorithms. For layered attack DAGs, we assume all nodes except those in the first layer may be targets.

Defender Strategy. In our model, the defender's action space consists of two action types: *deceptive* and *protective* actions. Deceptive actions aim to make attackers plan their attack with a misunderstanding of the game structure. Using her *deception budget* B^d, the defender can take three classes of deceptive actions: (i) hiding a real edge with a cost $c^h(e)$, (ii) adding a fake edge in a given set E_d with a cost $c^a(e)$, and (iii) modifying the perceived reward of a non-entry-point node with a cost $c^\delta(n)$ per unit of change. We assume the deceptive actions only change an attacker's *perception* of the attack graph (called the *induced* attack graph) and do not modify the true attack graph. Thus, a hidden edge is not truly removed; it is simply hidden from some attackers. An added fake edge is *virtual*: attackers cannot successfully move along that edge.

Each edge in the set E_d has perceived success probabilities $q^\theta(e)$ for each attacker type. These values are given as input. For the last action class, we assume a type-θ attacker's perceived reward of a node n is $r^\theta(n) = r(n) + \beta^\theta \Delta(n)$, where β^θ is the probability of observing the associated (perceived) change to the node reward $\Delta(n)$. The defender chooses the value of $\Delta(n)$ with a cost of $c^\delta(n)|\Delta(n)|$. In cybersecurity, a defender can hide edges by masking connections between physical machines or modifying the perceived routing table, add edges by faking network traffic or vulnerabilities, and change perceived rewards by manipulating outgoing traffic [3, 4] to make machines look like something else (e.g., an unimportant relay device). In physical security, she can hide and add edges by spreading misinformation about road closures or traffic [15], and change perceived rewards by signaling to attackers — either by spreading misinformation

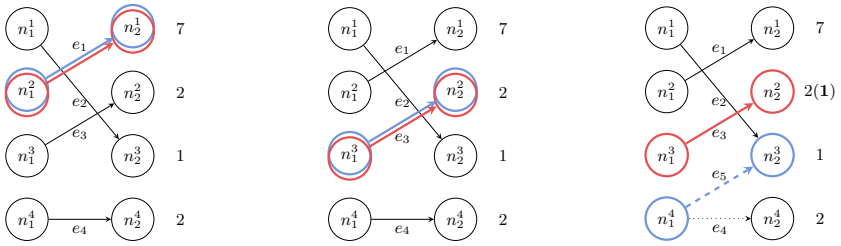
Notation	Definition
$\theta \in \Theta$	Attacker type from the set of attacker types
$G = (N, E)$	Attack graph, with set of a set of states N and set of edges E
$e = (n, n') \in E$	An edge in the set of edges
$q^\theta(e)$	Probability of type-θ attacker reaching n' from n using e.
$r(n)$	Attacker reward for reaching n
$T \subset N$	Set of targets where $r(n) > 0$
B^d	Defender's deception budget
B^a	Defender's effort budget
$c^h(e)$	Cost of hiding edge
$c^a(e)$	Cost of adding fake edge
$c^\delta(n)$	Cost per change to node reward
E_d	Set of fake edges that can be added
$\Delta(n)$	Perceived change to node reward
$r^\theta(n) = r(n) + \beta^\theta \Delta(n)$	Type-θ attacker's perceived reward of n
β^θ	Probability of observing $\Delta(n)$
$x(e)$	Interruption probability
C	Attacker's penalty if interrupted by defender
s_d	Defender's strategy
U_d	Defender's expected utility
BR^Θ	Best response(s) of the attacker type(s)

Table 1: Notation table.

or transforming targets to make them look like something else (e.g., through "uglification" [32]).

To deploy *protective* measures, the defender allocates effort $x(e)$ to edges with an effort budget B^a, increasing the chance of interrupting an attack. This effort can have different, domain-dependent meaning. $x(e)$ may represent the marginal probability of setting up a checkpoint on edge e [33] in urban security, or the time spent monitoring edge e in cybersecurity. We assume the probability that the defender will interrupt the attacker's movement on e is proportional to her allocated effort. Without loss of generality, we assume the coefficient to be 1. Thus, the interruption probability is $x(e)$. A defender's strategy includes a set of deceptive actions and a protective strategy x. We consider a deterministic deception strategy because the attacker observes the induced graph before choosing his strategy, so randomized deception does not benefit the defender [52].

Attacker Strategy. After observing the graph induced by the defender's deception strategy and the defender's protective strategy x, an attacker chooses a pure strategy. Following the literature [51], this strategy is an an attack path on his perceived attack graph. It is either null (the attacker chooses not to attack) or consists of a sequence of nodes or edges on the graph, starting from an entry point. To execute this strategy, the attacker starts from the entry point and moves along the edges as planned until he fails or successfully reaches the last node in the path. The attacker can fail to move along an edge for three reasons: (i) he fails due to the intrinsic possibility of failure (with probability $1 - q^\theta(e)$),



(a) Original graph. (b) When e_1 is protected. (c) With deception.

Fig. 1: Influence of defender strategy on attackers' best responses (blue for weak attacker, red for powerful attacker) on a bipartite DAG. The dashed line represents an *added* edge; the dotted line represents a *hidden* edge. The numbers next to a node show the true reward and the modified reward (parenthesized).

(ii) the defender interrupts his movement (with probability $q^\theta(e)(1-x(e))$), (iii) the edge is fake ($e \in E_d$). The game ends when the attacker's movement fails or he reaches the last node along the planned path. We assume the attacker is best responding based on his available information, including $q^\theta(e)$ and $r^\theta(n)$, and chooses a path that maximizes his total expected utility.

Player's Utility. When the attacker executes his attack strategy, he receives reward (or penalty) at each step. His total utility is the accumulated undiscounted reward (or penalty). Since the perceived reward and perceived success probability governs the attacker's response, we only discuss the attacker's perceived (expected) utility. If the attacker arrives at node n, he receives reward $r^\theta(n)$. If the defender interrupts the attack in one step of movement, the attacker pays a penalty C in that step, and the game ends. If the defender interrupts an attack before the attacker reaches any target, she gets 0. If the attacker successfully reaches a set of target nodes $T' \subset T$, the defender's utility is $-\sum_{n\in T'} r(n)$.

Solution Concept. We want to find the optimal defender strategy assuming that attackers best respond based on their induced attack graphs: $s_d^* = \arg\max_{s_d}\{U_d(s_d, BR^\Theta(s_d)\}$, where s_d is the defender's strategy containing both deceptive and protective actions, U_d is the defender's expected utility, and BR^Θ denotes the best response(s) of the attacker type(s) with defender-favoring tie-breaking. Our game model can be viewed as a two-stage problem: in stage one, the defender takes deceptive actions and stage two is a Bayesian Stackelberg game. We want to optimize the choice of deceptive actions in stage one and find a Strong Bayesian Stackelberg equilibrium in stage two.

For expository purpose, we consider two extreme attacker types: attackers who believe all and no deceptive actions, referred to as *weak* attackers $\mathtt{W}$ (w. p. γ) and *powerful* attackers $\mathtt{P}$ (w. p. $1-\gamma$), respectively. A type-$\mathtt{P}$ attacker will not be deceived, i.e., $q^\mathtt{P}(e) = 0$, $\forall e \in E_d$ and $\beta^\mathtt{P} = 0$. Our theoretical results and algorithms can be applied to the multiple attacker type setting.

Example Game Instance. To further elucidate our game model, we illustrate an example game instance on a bipartite DAG (Fig. 1) and show how

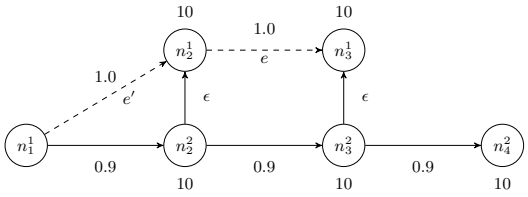


Fig. 2: Non-submodularity example. The edge labels show the success probabilities; the node labels show the rewards. ϵ is a sufficiently small positive number. Adding either edge e or e' will not affect the attacker's choice of the lower path; adding both will make the attacker choose the upper path.

the defender can use both deceptive and protective actions to improve her utility. We set the costs and deception budget so that the defender can at most take the following deceptive actions: adding edge e_5 from n_1^4 to n_2^3, hiding edge e_4, and changing the weak attacker's perceived reward of n_2^2 from 2 to 1. We set the effort budget B^a to 1 and the success probabilities to satisfy $q^\theta(e_2) < q^\theta(e_4) < q^\theta(e_3) < 1$ and $q^\theta(e) = 1$ for other edges. Here $\gamma = 0.5$, i.e., the defender encounters the two attacker types with equal probability.

Fig. 1a shows the best responses of both attackers BR^Θ on the graph with no defender actions taken. The defender's expected utility U_d is -7. Fig. 1b shows BR^Θ after the defender allocates all protective effort to e_1. Here $U_d = \sum_\theta -q^\theta(e_3)$. Fig 1c shows BR^Θ when the defender takes all possible deceptive actions and protects e_1. Here $U_d = -q^{\mathsf{P}}(e_3)$, as the weak attacker will fail due to the fake edge. This example shows that deception can benefit the defender.

3 Theoretical Analysis

In this section, we show that the defender's optimal utility as a function of added or hidden edges is not submodular and the problem of finding the optimal defender strategy is NP-hard.

Non-Submodularity Results. Let $f(D)$, where $D \subset E_d$, denote the defender's optimal expected utility when she adds a set of fake edges D. $f(D)$ is submodular if $\forall D \subset D', e \in E_d \setminus D'$, $f(D \cup \{e\}) - f(D) \geq f(D' \cup \{e\}) - f(D')$. Submodular functions enjoy desirable properties and often have efficient approximation algorithms [26]. However, as shown by the example in Fig. 2, $f(D)$ in our game model is not submodular[4].

In Fig. 2, the original graph contains all nodes and solid edges. Node n_1^1 is the entry point. We set $B^d = 7$, $c^\delta(n) = c^a(e) = 1, \forall e, n$, and $c^h(e) > 7, \forall e$, so the defender cannot hide any edges. We set $B^a = 0$ and $\gamma = 1$. Let $D = \emptyset$ and $D' = \{e'\}$. If the defender does not add an edge (the set of added edges is D) or adds only one dashed line (D' or $D \cup \{e\}$), then no matter how she

[4] For space, we omit the full proof for hiding edges. We follow a similar construction.

changes the node rewards, the attacker's best response is path $(n_1^1, n_2^2, n_3^2, n_4^2)$, resulting in $U_d = -24.39$. If the defender adds both edges e' and e, then she can change the reward of n_3^1 to 15 with the remaining budget 5. The attacker then chooses path (n_1^1, n_2^1, n_3^1) as he believes that it yields a higher utility (25) than the lower path. Here $U_d = 0$, as the fake edges will cause the attacker to fail. Thus, $f(D \cup \{e\}) - f(D) = 0$ and $f(D' \cup \{e\}) - f(D') = 24.39$, contradicting the definition of submodularity. The non-submodularity indicates that it may be hard to find efficient approximation algorithms for our problem.

Hardness Results. Our main result is that the problem is NP-hard when the defender can perform both protective and deceptive actions, even when restricted to layered DAGs.

Theorem 1. *The problem of finding the optimal deceptive actions is NP-hard even in layered DAGs.*

Proof. We prove the theorem by providing a polynomial time reduction from the knapsack problem (KP). In a typical KP, there are k items. Each item i is associated with a weight w_i and a value z_i. We want to find a subset of the items, such that $\sum_i z_i$ is maximized and $\sum_i w_i \leq W$, where W is given. Here we only consider cases where $w_i \in \mathbb{Z}, \forall i$, which are still NP-hard problems.

Given any KP instance with parameters w, z, W, we construct a problem instance with a layered DAG as follows. First, create a source node n_0 and k other *milestone* nodes $n_1, n_2, \ldots, n_{k+1}$. Second, for each $1 \leq i \leq k$, create two sets of nodes $\{u_i^j\}_{j=1}^{w_i}$ and $\{l_i^j\}_{j=1}^{W+1}$, respectively called upper and lower nodes. Third, for each upper node u_i^j, add edges (n_{i-1}, u_i^j) and (u_i^j, n_i). For each lower node l_i^j, add (n_{i-1}, l_i^j) and (l_i^j, n_i). Fourth, add nodes $\{l^j\}_{j=1}^{W+1}$ and edges $(n_k, l^j), (l^j, n_{k+1}), \forall j$. Last, set $r(u_i^j) = z_i$ for each upper node, $r(n_{k+1}) = M$, where M is sufficiently large: $M > \sum_{i=1}^k z_i$. For all other nodes n, set $r(n) = 0$.

Constructing this layered DAG with the above construction takes polynomial time. We consider the case where $\gamma = 1$ and $q^{\mathtt{W}}(e) = 1, \forall e$. We set $B^d = W$ and $B^a = 0$. Let $c^h(e) = 1, \forall e$, $c^a(e) > W, \forall e$, and $c^\delta(n), \forall n$ be sufficiently large, such that the defender can only hide edges. Since there are more than W ways go from one milestone node to the next, there is always a path from n_0 to n_{k+1}. Because the attacker selects the highest-valued path, the attacker will choose to go through an upper node to reach n_i from n_{i-1}, if able.

Let x_i be the binary variable indicating if the attacker's path from n_{i-1} to n_i contains an upper node. The attacker's utility is $M + \sum_{i=1}^k z_i x_i$, so the defender's goal is to hide edges to minimize $M + \sum_{i=1}^k z_i x_i$. If the defender wants to prevent the attacker from reaching the upper nodes, she must hide at least w_i edges at a cost of w_i. Let y_i indicate whether the defender hides these w_i edges. Since $x_i = 1 - y_i$, we can rewrite the defender's problem as an integer program that minimizes $M + \sum_{i=1}^k z_i(1 - y_i)$, subject to $\sum_{i=1}^k w_i y_i \leq W$ and $y_i \in \{0, 1\}$, which is equivalent to maximizing $\sum_{i=1}^k z_i y_i$, subject to $\sum_{i=1}^k w_i y_i \leq W$ and $y_i \in \{0, 1\}$. □

4 The MILP Approach for Layered DAGs

Here we provide an MILP-based algorithm for layered DAGs. We first describe our approach for bipartite DAGs and then extend it to general layered DAGs.

4.1 Bipartite DAG

We describe our MILP-based approach for bipartite DAGs. We begin with the following important observation.

Observation 1 *In the defender's optimal strategy, any edge added by the defender must be in the optimal path chosen by the weak attackers.*

Proof. Because the powerful attackers know that the edges are fake, only weak attackers will choose them. The statement trivially follows; otherwise, adding these edges would cause the attackers' optimal path to stay the same, and the defender would obtain the same expected utility. □

Here any attacker's action contains a single edge. According to Observation 1, the defender will add at most one edge in her optimal strategy: that edge must be the weak attacker's best response. Thus, we have $E_d = \{(n_1, n_2) \notin E \mid n_1 \in L_1, n_2 \in L_2\}$. Denote by e_0 the choice to not attack. Let $e_{\mathtt{W}} \in E \cup E_d \cup \{e_0\}$ and $e_{\mathtt{P}} \in E \cup \{e_0\}$ be the choices of the weak and powerful attacker, respectively. Let $x^m(e) \in \{0, 1\}$ indicate if $e \in E \cup E_d$ is in the perceived graph. Denote by $n_{end}(e)$ the endpoint of edge e. As with prior work [17], we must enumerate all possible action profiles $(e_{\mathtt{W}}, e_{\mathtt{P}})$ and for each profile solve a mathematical program:

$$
\begin{aligned}
\text{Maximize} \quad & U_d^{\mathtt{W}} + U_d^{\mathtt{P}} \\
\text{Subject to} \quad & U_d^{\mathtt{W}} = -\gamma \mathbb{1}[e_{\mathtt{W}} \in E] q^{\mathtt{W}}(e_{\mathtt{W}})(1 - x(e_{\mathtt{W}}))r(n_{end}(e_{\mathtt{W}})), \\
& U_d^{\mathtt{P}} = -(1-\gamma) \mathbb{1}[e_{\mathtt{P}} \in E] q^{\mathtt{P}}(e_{\mathtt{P}})(1 - x(e_{\mathtt{P}}))r(n_{end}(e_{\mathtt{P}})), \\
& U_a^{\mathtt{W}}(e) = q^{\mathtt{W}}(e)[(1 - x(e))r^{\mathtt{W}}(n_{end}(e)) - x(e)C], \forall e \in E \cup E_d, \\
& U_a^{\mathtt{W}}(e_0) = 0, \\
& U_a^{\mathtt{W}}(e_{\mathtt{W}}) \geq \max\{U_a^{\mathtt{W}}(e) - (1 - x^m(e))M, 0\}, \forall e \in E \cup E_d, \qquad (1) \\
& U_a^{\mathtt{P}}(e) = q^{\mathtt{P}}(e)[(1 - x(e))r^{\mathtt{P}}(n_{end}(e)) - x(e)C], \forall e \in E, \\
& U_a^{\mathtt{P}}(e_0) = 0, \\
& U_a^{\mathtt{P}}(e_{\mathtt{P}}) \geq \max\{U_a^{\mathtt{P}}(e), 0\}, \forall e \in E, \qquad (2) \\
& \sum_{e \in E \cup E_d} x^m(e) x(e) \leq r, \qquad (3) \\
& \sum_{n \in L_2} c^{\delta}(n) |r(n) - r^{\mathtt{W}}(n)| + \sum_{e \in E} c^h(e) \left[1 - x^m(e)\right] \\
& \qquad + c^a(e_{\mathtt{W}})(1 - \mathbb{1}[e_{\mathtt{W}} \in E]) \leq B^d, \qquad (4) \\
& 0 \leq x(e) \leq 1, \forall e \in E \cup E_d, \\
& x^m(e_{\mathtt{W}}) = 1, \text{ if } e_{\mathtt{W}} \neq e_0,
\end{aligned}
$$

$$
\begin{aligned}
&x^m(e) = 0, \forall e \in E_d \setminus \{e_{\mathtt{W}}\},\\
&x^m(e) \in \{0,1\}, \forall e \in E,\\
&l_k^+(n), l_k^-(n) \in \{0,1\}, \forall n \in L_2,\\
&0 \le m_k^+(e) \le l_k^+(n_{end}(e)), \forall e \in E \cup E_d,\\
&x(e) - [1 - l_k^+(n_{end}(e))] \le m_k^+(e) \le x(e), \forall e \in E \cup E_d,\\
&0 \le m_k^-(e) \le l_k^-(n_{end}(e)), \forall e \in E \cup E_d,\\
&x(e) - [1 - l_k^-(n_{end}(e))] \le m_k^-(e) \le x(e), \forall e \in E \cup E_d,
\end{aligned}
$$

where $U_a^{\mathtt{W}}$ and $U_a^{\mathtt{P}}$ are the utilities of the weak and powerful attacker, $U_d^{\mathtt{W}}$ and $U_d^{\mathtt{P}}$ are the defender's utilities from the weak and powerful attacker, and M is a sufficiently large positive number.

We safely remove $x^m(e)$ from constraint (3). To handle the absolute terms in constraint (4), we introduce two variables for each target $n \in L_2$: $r^+(n) = \max\{r^{\mathtt{W}}(n) - r(n), 0\}$ and $r^-(n) = \max\{r(n) - r^{\mathtt{W}}(n), 0\}$. Thus, we have: $r^{\mathtt{W}}(n) - r(n) = r^+(n) - r^-(n)$ and $|r^{\mathtt{W}}(n) - r(n)| = r^+(n) + r^-(n)$. For the quadratic term in constraints (1), we use similar techniques from previous work [57] to find approximate solutions. We first focus on solutions where $r^{\mathtt{W}}(n) - r(n)$ is a multiple of a basic step r_0 and use binary representations for $r^+(n)$ and $r^-(n)$. We introduce binary variables $l_k^+(n)$ and $l_k^-(n)$ and let $r^+(n) = r_0 \sum_k 2^k l_k^+(n)$ and $r^-(n) = r_0 \sum_k 2^k l_k^-(n)$. We introduce two new variables $m_k^+(e) = x(e) l_k^+(n_{end}(e))$ and $m_k^-(e) = x(e) l_k^-(n_{end}(e))$ to replace all possible quadratic terms:

$$
\begin{aligned}
&0 \le m_k^+(e) \le l_k^+(n_{end}(e)) \quad \text{and} \quad x(e) - [1 - l_k^+(n_{end}(e))] \le m_k^+(e) \le x(e),\\
&0 \le m_k^-(e) \le l_k^-(n_{end}(e)) \quad \text{and} \quad x(e) - [1 - l_k^-(n_{end}(e))] \le m_k^-(e) \le x(e).
\end{aligned}
$$

We improve the algorithm's scalability in two ways. First, we track the best solution U^* so far and add a constraint to each MILP: $U_d^{\mathtt{W}} + U_d^{\mathtt{P}} \ge U^*$. In addition to not affecting the final solution, this constraint speeds up the algorithm by rendering many MILPs infeasible. We can efficiently decide a MILP's feasibility. Second, we build a two-layer search tree: the first layer corresponds to the powerful attacker's choice; the second layer corresponds to the weak attacker's choice. We compute each node's upper and lower bounds and prune a branch if its upper bound is less than the lower bound of other branches. To compute the lower bound, we set $B^d = 0$, so the perceived graph of all attacker types is the same as the true attack graph. Because all attacker types choose the same attack, we view them as a single attacker.

We use two methods to compute the upper bound. In the first method, we set $B^d = \infty$ in the original game, so the defender can hide all edges. The weak attacker will not attack; thus, we need only consider the powerful attacker. In the second method, we relax the original MILP to yield a tighter bound. We remove the binary constraints, transforming the MILP into a linear program (LP). We relax the LP by randomly adding a fraction of the constraints described in Equations (1) and (2). Solving the new relaxed LP is much faster. Initially, we compute both global lower ($B^d = 0$) and upper ($B^d = \infty$) bounds. For each

leaf node, we compute a new upper bound using the second method described above, and update the lower bounds and upper bounds for other related nodes.

4.2 Layered DAG

Now we consider general layered DAGs, where an attacker's pure strategy is to choose a state from each layer to form a path. The number of possible pure strategies is $\prod_{i=1}^{l} |L_i|$, which can be exponential in the number of states. Using the multiple-LP method [17], we would need to solve exponentially many MILPs. Instead, we provide a different formulation by simulating backward induction.

For simplicity, we describe our formulation for weak attackers; handling powerful attackers is simpler. Let $x^m(e)$ indicate if e is in the weak attacker's perceived graph. For each node, we introduce variables $V^{\mathtt{W}}(n_i^j)$ and $V^{\mathtt{P}}(n_i^j)$, one for each attacker type, to represent the expected utilities of starting from n_i^j:

$$V^{\mathtt{W}}(n_i^j) = r^{\mathtt{W}}(n_i^j) + \max_{e:x^m(e)=1, n_{start}(e)=n_i^j} q^{\mathtt{W}}(e)[(1-x(e))V^{\mathtt{W}}(n_{end}(e)) + x(e)C],$$

where $n_{start}(e)$ is the start state of edge e. To handle the quadratic terms, we use their binary representations [57]. To handle the max operator, we introduce a binary variable $a^{\mathtt{W}}(e)$ indicating the attacker's choice if he starts from $n_{start}(e)$. We guarantee that only existing edges are chosen and, among all edges starting from the same state, at most one of them is selected: $a^{\mathtt{W}}(e) \le x^m(e)$ and $\sum_{x^m(e)=1, n_{start}(e)=n_i^j} a^{\mathtt{W}}(e) \le 1$. With $a^{\mathtt{W}}(e) = 1$, the defender's expected utility $U_d^{\mathtt{W}}(n_i^j)$ from weak attackers starting from n_i^j is:

$$U_d^{\mathtt{W}}(n_{start}(e)) = -r(n_{start}(e)) + q^{\mathtt{W}}(e)[(1-x(e))U_d^{\mathtt{W}}(n_{end}(e)) + x(e)C].$$

We apply similar techniques to obtain the defender's utility for powerful attackers. The defender's overall objective is $\gamma \sum_j U_d^{\mathtt{W}}(n_1^j) + (1-\gamma) \sum_j U_d^{\mathtt{P}}(n_1^j)$. We omit the complete MILPs for layered DAGs, as they can be obtained by modifying the MILPs for bipartite graphs using the above steps.

5 The NAS Approach for General DAGs

In this section, we present our neural architecture search (NAS) algorithm for general DAGs. We first describe how we find the attacker's best response. The rest of this section is devoted to solving the defender's problem.

Given the defender's strategy, let $V^\theta(n)$ be a type θ attacker's highest expected utility starting from node n, and E^θ be the set of edges in the perceived graph for that attacker type. From the definition of best response, we have:

$$V^\theta(n) = \max_{n':(n,n')\in E^\theta} \left\{ r^\theta(n) + q^\theta(n,n')[(1-x(n,n'))V^\theta(n') + x(n,n')C] \right\}.$$

We can equivalently view V^θ as a Bellman equation, so it can be solved with dynamic programming or, because we focus on DAGs, backward induction on the reversed topological order.

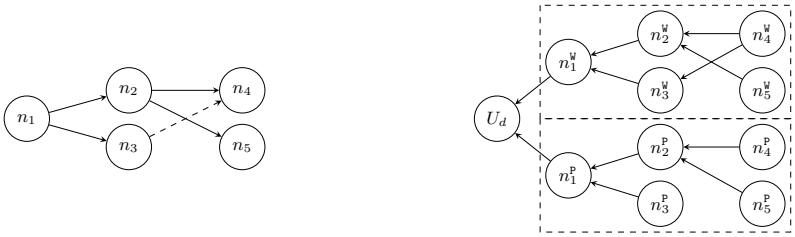


Fig. 3: The neural architecture search algorithm. The left figure is the DAG, where the dashed line is the added edge. The right figure is the corresponding neural network, where U_d is the defender's total utility: the sum of utilities from both types of attackers. The two dashed boxes correspond to the networks for the weak (top) and powerful (bottom) attackers.

We now describe our NAS approach for solving the defender's problem. We leverage the insight that the defender's actions can be divided into two categories: actions that change the attack graph structure (e.g., adding and hiding edges) and actions that change the parameter values of the graph (e.g., changing node values and protecting edges). Because the attack graph is a DAG, it naturally can be viewed as a feedforward neural network (NN). Thus, we redefine the problem as a NAS problem [23], in which we first use a search strategy to find a NN structure, then use machine learning techniques to optimize the parameter values. We use a genetic algorithm to propose architectures and a modified gradient-based technique to optimize the parameter values and evaluate the quality of each architecture. We cast the defender's utility as the objective function in the optimization problem of training a NN. To optimize the parameter values, we use the forward pass to simulate the attacker's decision-making process (backward induction). Thus, we reverse the DAG and start from the end nodes. We build a network for each attacker type based on their perceived graphs and obtain the defender's expected utility by aggregating the utilities obtained from different attacker types.

However, we still need to address the following issues. First, the rational attacker chooses a single, deterministic path from the set of possible paths. Thus, small changes in a state's value may not change the attacker's decision, and, by extension, the defender's utility. Therefore, the gradients of the defender's utility with respect to these variables are 0. Second, the defender has budgets B^d *and* B^a. To solve these problems, we borrow techniques from prior work [54]. For the first problem, we slightly relax the assumption of rational attackers and use the quantal response model [40]. Thus, the probability of choosing an edge (n, n') is:

$$\Pr^{\theta}\{n'|n\} = \begin{cases} \frac{\mathrm{e}^{\lambda V^{\theta}(n,n')}}{\sum_{m:(n,m)\in E^{\theta}} \mathrm{e}^{\lambda V^{\theta}(n,m)}} & \text{if } (n,n') \in E^{\theta}; \\ 0 & \text{otherwise,} \end{cases}$$

where E^{θ} is the set of edges in the perceived graph for a type θ attacker, and λ is a parameter that controls the rationality of the attackers. The term $V^{\theta}(n, n')$ is

defined as the expected utility of a type θ attacker moving through edge (n, n'): $V^\theta(n, n') = r^\theta(n) + q^\theta(n, n')[(1 - x(n, n'))V^\theta(n') - x(n, n')C]$, where

$$V^\theta(n) = \begin{cases} r^\theta(n) & \text{if } n \text{ has no outgoing edges;} \\ \sum_{m:(n,m)\in E^\theta} \Pr^\theta\{m|n\}V^\theta(n, m) & \text{otherwise.} \end{cases}$$

Thus, $U_d^\theta(n) = \sum_{m:(n,m)\in E} \Pr^\theta\{m|n\}U_d^\theta(m)$ captures the defender's utility from a type θ attacker. Instead of a regularization term, which cannot guarantee that the budget constraints are strictly satisfied, we use another method that "distributes" the budgets to different nodes and edges. We introduce a variable $z(e)$ for each edge $e \in E$ and two variables $y(n)$ and $d(n)$ for each target. Define:

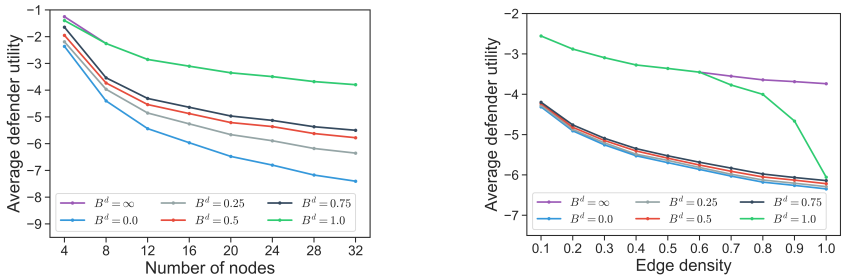
$$x(e) = \min\left\{1.0, \frac{B_r^d \mathrm{e}^{z(e)}}{\sum_{e'\in E} \mathrm{e}^{z(e')}}\right\} \text{ and } r^{\mathtt{W}}(n) - r(n) = \frac{B^a \mathrm{e}^{y(n)}}{c^\delta(n) \sum_n \mathrm{e}^{y(n)}} \frac{\mathrm{e}^{d(n)} - 1}{\mathrm{e}^{d(n)} + 1},$$

where B_r^d is the remaining deceptive budget after adding and hiding edges. The feasibility $0 \leq x(e) \leq 1$ and budget constraints are always satisfied for all possible combinations of $z(e)$, $y(n)$, and $d(n)$. The quantities $\Pr^\theta\{n'|n\}$, $x(e)$, and $r^{\mathtt{W}}(n)$ can be represented in all major deep learning packages using the sigmoid and softmax functions.

6 Experiments

We conduct experiments on both bipartite and general DAGs. We generate the graphs using the random Erdös-Renyi method [24]. To ensure the DAG property for general graphs, each node has a unique number; only edges (s, t) s.t. $s < t$ can be added. Edge density ρ denotes the probability that an edge is added to the original graph when constructed. We sample the edge success probabilities and the attacker's penalty C uniformly at random from the interval $(0, 1]$ for all experiments. We set $B^a = 1$. As hiding or adding an edge is more difficult in reality than changing node rewards, we set the costs of deceptive actions to $c^h(e) = c^a(e) = 1, \forall e$ and $c^\delta(n) = 0.1, \forall n$ (unless otherwise noted). We generate the target rewards by sampling uniformly at random from the interval $[5, 10]$. We generate 20 instances for each point in all plots. Because differential evolution [61], a population-based evolutionary algorithm (EA), works well on many problems, we use a modified *DE/rand/1/bin* variant [42] as a baseline (see Appendix for details). The EA uses `differential_evolution` from the `scipy` package [64]. We use Gurobi [28] to solve the MILPs and PyTorch [47] for NAS. We run all experiments on a machine with a Core i7 CPU at 4.2GHz.

With our experiments, we seek to answer the following questions: 1) What is the effect of different deception budgets on the defender's utility?, 2) What is the solution quality obtained by our algorithms?, 3) How well do our algorithms scale?, and 4) How do our algorithms allocate the deceptive budget?



(a) Defender utility vs. number of nodes

(b) Defender utility vs. edge density

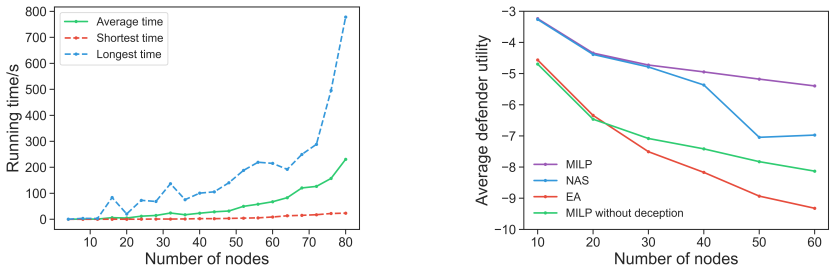
Fig. 4: Change in defender's utility versus graph size. The utility decreases as the deceptive budget decreases and as the graph size increases.

6.1 Bipartite DAGs

Effect of Different Deception Budgets. We investigate how different deceptive budgets and graph sizes affect the defender's utility U_d in a bipartite DAG setting with the following experiments. In the first experiment, we fix the edge density ρ to 0.5, vary the number of nodes N from 4 to 32 (from 2 to 16 on each side), and solve each game with a different B^d. In the second experiment, we fix $N = 16$, vary D in increments of 0.1, and solve each game with a different B^d. We only use the MILP algorithm for both experiments.

Fig. 4 shows that U_d decreases as the graph size increases and as B^d decreases. Fig. 4a depicts the first experiment's results. When $B^d = 1$, the defender achieves nearly the same U_d as $B^d = \infty$. Because the cost of adding or hiding an edge is 1, the defender can afford to manipulate the graph structure instead of only changing the perceived node rewards. This result shows that, in the bipartite DAG case, adding one edge can typically yield the best solution. Fig. 4b depicts the second experiment's results. When $B^d < 1$, there is a small utility improvement from an increased B^d; in contrast, U_d sharply increases when $B^d = 1$. When ρ increases, the performance when $B^d = 1$ quickly decreases. When ρ is large, few edges are available to add, so the effect of adding edges quickly diminishes.

Algorithm Performance. In our first experiment, we show the scalability of our improved MILP algorithm (we omit NAS since it is not guaranteed to produce the optimal solution). In the second experiment, we compare the solution quality of our MILP and NAS algorithms with the EA baseline. For the first experiment, we vary N from 4 to 80 in increments of 4. For the second experiment, we vary N from 10 to 60 in increments of 10. We set the maximum running time for the EA and NAS to 5 minutes, as the MILP algorithm terminates within 5 minutes for all instances. We set the population size to 60 for the EA and NAS's genetic algorithm part. We set the EA's recombination and mutation parameters to the `scipy` package's default values. For NAS, we set $\lambda = 5$ and use the Adam optimizer [36] with a learning rate of 0.2. We fix $\rho = 0.5$ for both experiments.



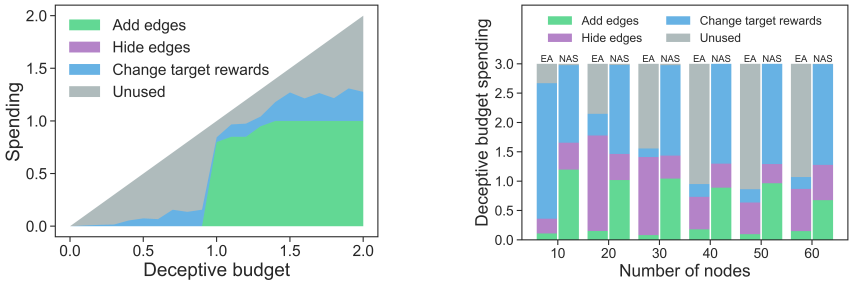
(a) Running time of MILP algorithm (b) Solution quality of all algorithms

Fig. 5: Scalability and performance on bipartite graphs

Fig. 5a depicts the results of the first experiment: our MILP algorithm scales well. The average running time for solving an instance with 80 nodes is less than 4 minutes; the longest individual running time is about 13 minutes. The standard multiple-MILP [17] algorithm would need to solve about $(40 \times 40 \times 0.5) \times (40 \times 40) = 1,280,000$ MILPs. With our heuristic algorithm, we only need to solve 4.6 MILPs on average.

Fig 5b shows the results of the second experiment. To examine the effect of the various algorithms on the defender's utility U_d, we conduct a one-way repeated measures ANOVA on all graph sizes. The results show that the algorithm used leads to a statistically significant difference in U_d ($F(3, 177) = 189.18, p = 5.66 \times 10^{-55}, \alpha = .05$). We conduct pairwise paired t-tests with the Bonferroni correction to determine which algorithms result in significant increases in U_d. We find that all algorithm pairs have a significant difference in U_d. In fact, NAS ($M = -3.0151, SD = 7.1396$), the MILP algorithm with no deception ($M = -6.8458, SD = 1.7518$), and the MILP algorithm with deception ($M = -2.2892, SD = 2.8603$) significantly outperform the EA ($M = -7.1396, SD = 3.3582$), $p = 7.8 \times 10^{-20}, .0012, 1.6 \times 10^{-24}$, respectively.

Budget Expenditure of the MILP Algorithm. We show how the MILP allocates B^d. In this experiment, each graph has $N = 16$ and $\rho = 0.5$. For each instance, we increase B^d from 0 to 2 and track how each budget is spent. Fig. 6a shows the results. The defender never hides an edge in any of the optimal solutions, suggesting that adding edges may be more useful than hiding edges. This suggestion is further strengthened when $B^d = 1$: we see a sudden increase in spending on adding edges. We also see that changing the node rewards may not be as useful as other deceptive actions: the unused budget generally occupies a large area. However, when $B^d > 1$, changing the node rewards is more frequently used than when $B^d \leq 1$. This result suggests that changing the node rewards is more useful when combined with adding edges, as it makes the fake path appear more valuable (and, thus, more attractive) to a weak attacker.



(a) Spending on bipartite DAGs (b) Spending on general DAGs

Fig. 6: Deceptive budget expenditure of the MILP (left), and EA and NAS (right)

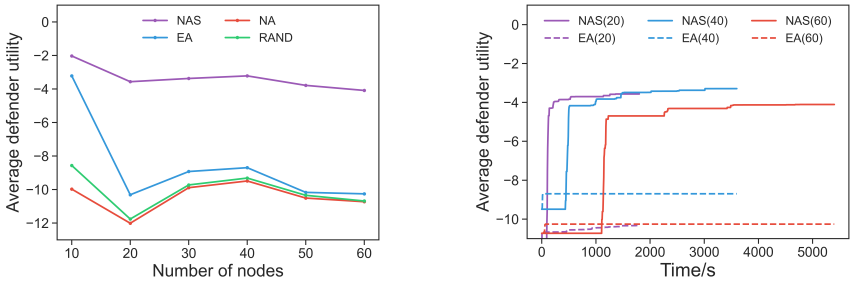
6.2 General DAGs

Algorithm Performance. We compare the performance of NAS and three baselines on general DAGs. We omit the MILP algorithm: it is not applicable. For the baselines, we use a random strategy (RAND), and a no-action strategy (NA), and EA. We fix $\rho = 0.5$ and vary N from 10 to 50 in increments of 10. We set $B^d = 3$ and sample $c^a(e)$ and $c^h(e)$ from the interval $[0.5, 1.5]$, such that $c^a(e_i) = c^a(e_j), \forall e_i, e_j$ and $c^h(e_i) = c^h(e_j), \forall e_i, e_j$. EA and NAS use the same parameters as before, but the maximum running time is $1.5 \times N$ minutes.

NAS leads to significantly higher defender utility U_d than all other algorithms (Fig. 7a). We perform a one-way repeated measures ANOVA on all graph sizes to examine different algorithms' effect on U_d. We find that the choice of algorithm produces statistically significant differences in U_d ($F(3, 357) = 1286.9, p = 1.7 \times 10^{-106}, \alpha = .05$). We conduct pairwise paired t-tests with the Bonferroni correction to determine which algorithms result in these significant increases. We find that all algorithm pairs have significant differences in U_d. Importantly, NAS ($M = -3.3453, SD = 6.3247$) leads to a significant increase in U_d compared to all algorithms: RAND ($M = -10.067, SD = 12.369$), NA ($M = -10.435, SD = 12.246$), and the EA ($M = -8.5974, SD = 15.220$), $p = 6.77 \times 10^{-47}, 7.83 \times 10^{-48}, 6.56 \times 10^{-38}$, respectively.

We are interested in how quickly NAS reaches solutions with high U_d. Fig. 7b shows how the NAS and EA solutions evolve. On average, NAS quickly (< 20 minutes) finds high-quality solutions and successfully refines the solution quality; in contrast, the EA struggles to improve the utility from its initial solution.

Budget Expenditure of Different Algorithms. We show how the EA and NAS allocate B^d. This experiment uses the same graphs as in Section 6.2. Fig. 6b shows how the EA and NAS spend B^d. On average, NAS allocates more of B^d to adding edges and changing the node rewards, indicating that the EA struggles to find solutions that involve adding edges and to determine the appropriate changes in the node rewards. We believe this is due to how the EA performs recombination and mutation.



(a) Average defender utility achieved vs. the size of the graph

(b) Performance over time. The graph's number of nodes is parenthesized.

Fig. 7: Performance of different algorithms on general graphs

7 Related Work

In addition to the related literature mentioned throughout the paper, we introduce and discuss some additional related works. Some SSG variants enable the defender to manipulate the game's payoff structure [13, 52, 57] or alter a system's observable features to influence the attacker's attack choice [43, 56]). However, these works do not capture the graphical structure of security problems. In contrast, in our work, the defender manipulates the attacker's perceived payoff structure, allocates defensive resources, and manipulates the attacker's perceived graphical structure of the game (including the actions that the attacker believes that he can take). Previous work that combines defensive deception and attack graph games typically only focuses on deploying honeypots or fake vulnerabilities in a network [10, 11, 19, 20, 22, 51]. Our model is more general in the sense that it can model these deception techniques by allowing the defender to manipulate the edges and perceived reward of targets. In addition, we consider the protective actions the defender may take to interrupt an attack. Another related work [29] uses a model of defensive deception to manipulate the attacker's belief; however, it abstracts away specific deceptive actions, and does not account for non-deterministic transitions between states, which are considered in our model.

8 Discussion and Conclusion

Our techniques can be applied to more general settings. For example, when the attacker's reward depends on the edge, not the node, we can replace the term $r(s_{end}(e))$ with $r(e)$ in the MILP and NAS algorithms. We can also easily relax the assumption that the probability of catching an attacker at edge e is $q^{\theta}(e)x(e)$. Our MILP algorithm works if all constraints are linear in the defender's protection effort; the NAS algorithm works for any differentiable function.

We introduced a novel variant of Stackelberg attack graph games, in which the defender can alter the perceived structure of the attack graph and the per-

ceived reward of the nodes in the graph, as well as allocate protective effort along the graph's edges. We proved the hardness of this problem and proposed two algorithms to solve special but important subcases of this game: a MILP algorithm with novel heuristics and a NAS algorithm in which the attack graph structure is the neural network architecture. We performed extensive experiments that show the effectiveness of deception and of our algorithms.

Acknowledgements. This research was sponsored by the U.S. Army Combat Capabilities Development Command Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-13-2-0045 (ARL Cyber Security CRA). The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Combat Capabilities Development Command Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

References

1. Abdallah, M., Naghizadeh, P., Hota, A.R., Cason, T., Bagchi, S., Sundaram, S.: Behavioral and game-theoretic security investments in interdependent systems modeled by attack graphs. IEEE Trans. Control Netw. Syst. (2020)
2. Achleitner, S., La Porta, T., McDaniel, P., Sugrim, S., Krishnamurthy, S.V., Chadha, R.: Cyber deception: Virtual networks to defend insider reconnaissance. In: Int. Workshop Managing Insider Secur. Threats (2016)
3. Albanese, M., Battista, E., Jajodia, S.: A deception based approach for defeating os and service fingerprinting. In: Conf. Commun. and Netw. Secur. (2015)
4. Albanese, M., Battista, E., Jajodia, S.: Deceiving attackers by creating a virtual attack surface. In: Cyber Deception (2016)
5. Almeshekah, M., Spafford, E.: Planning and integrating deception into computer security defenses. In: New Secur. Paradigms Workshop (2014)
6. Almeshekah, M., Spafford, E.: Cyber security deception. In: Cyber Deception (2016)
7. Ammann, P., Wijesekera, D., Kaushik, S.: Scalable, graph-based network vulnerability analysis. In: Conf. Comput. and Commun. Secur. (2002)
8. An, B., Ordóñez, F., Tambe, M., Shieh, E., Yang, R., Baldwin, C., et al.: A deployed quantal response-based patrol planning system for the us coast guard. Interfaces **43**(5) (2013)
9. Anwar, A.H., Kamhoua, C., Leslie, N.: A game-theoretic framework for dynamic cyber deception in internet of battlefield things. In: Int. Conf. Mobile and Ubiquitous Syst.: Comput., Netw. and Services (2019)
10. Anwar, A.H., Kamhoua, C., Leslie, N.: Honeypot allocation over attack graphs in cyber deception games. In: Int. Conf. Comput., Netw. and Commun. (2020)
11. Basak, A., Kamhoua, C., Venkatesan, S., Gutierrez, M., Anwar, A.H., Kiekintveld, C.: Identifying stealthy attackers in a game theoretic framework using deception. In: Conf. Dec. and Game Theory for Secur. (2019)
12. Bercovitch, M., Renford, M., Hasson, L., Shabtai, A., Rokach, L., Elovici, Y.: Honeygen: An automated honeytokens generator. In: IEEE ISI (2011)

13. Blocki, J., Christin, N., Datta, A., Procaccia, A.D., Sinha, A.: Audit games. In: Int. Joint Conf. Artif. Intell. (2013)
14. Bondi, E., Oh, H., Xu, H., Fang, F., Dilkina, B., Tambe, M.: Broken signals in security games: Coordinating patrollers and sensors in the real world. In: Int. Conf. Auton. Agents and Multi-Agent Syst. (2019)
15. Car and Driver: Artist shows google maps' control over our lives by creating a fake traffic jam (2020)
16. Cohen, F.: The use of deception techniques: Honeypots and decoys. Handbook Inf. Secur. **3**(1) (2006)
17. Conitzer, V., Sandholm, T.: Computing the optimal strategy to commit to. In: Conf. Electron. Commerce (2006)
18. Dong, C., Zhao, L.: Sensor network security defense strategy based on attack graph and improved binary pso. Saf. Sci. **117** (2019)
19. Durkota, K., Lisỳ, V., Bošanskỳ, B., Kiekintveld, C.: Approximate solutions for attack graph games with imperfect information. In: Conf. Dec. Game Theory for Secur. (2015)
20. Durkota, K., Lisỳ, V., Bošanskỳ, B., Kiekintveld, C.: Optimal network security hardening using attack graph games. In: Int. Joint Conf. Artif. Intell. (2015)
21. Durkota, K., Lisỳ, V., Bošanskỳ, B., Kiekintveld, C., Pěchouček, M.: Hardening networks against strategic attackers using attack graph games. Comput. & Secur. **87** (2019)
22. Durkota, K., Lisỳ, V., Kiekintveld, C., Bošanskỳ, B., Pěchouček, M.: Case studies of network defense with attack graph games. Intell. Syst. **31**(5) (2016)
23. Elsken, T., Metzen, J.H., Hutter, F.: Neural architecture search: A survey. J. Mach. Learn. Res. **20** (2019)
24. Erdős, P., Rényi, A.: On random graphs. Publ. Mathematicae Debrecen **6** (1959)
25. Fang, F., Nguyen, T.H., Pickles, R., Lam, W.Y., Clements, G.R., An, B., et al.: PAWS—a deployed game-theoretic application to combat poaching. AI Mag. **38**(1) (2017)
26. Feldman, M., Naor, J., Schwartz, R.: A unified continuous greedy algorithm for submodular maximization. In: Annu. Symp. Found. Comput. Sci. (2011)
27. Garg, N., Grosu, D.: Deception in honeynets: A game-theoretic analysis. In: Inf. Assurance and Secur. Workshop (2007)
28. Gurobi Optimization, LLC: Gurobi optimizer reference manual (2020)
29. Horák, K., Zhu, Q., Bošanskỳ, B.: Manipulating adversary's belief: A dynamic game approach to deception by design for proactive network security. In: Conf. Dec. and Game Theory for Secur. (2017)
30. IBM Security: Cost of a data breach report 2019 (2019), https://ibm.co/2CPsVnV
31. Ingols, K., Lippmann, R., Piwowarski, K.: Practical attack graph generation for network defense. In: Comput. Secur. Appl. Conf. (2006)
32. Instructables: How to make your bike look an ugly discouragement for thieves (2015), https://rb.gy/kb384b
33. Jain, M., Korzhyk, D., Vaněk, O., Conitzer, V., Pěchouček, M., Tambe, M.: A double oracle algorithm for zero-sum security games on graphs. In: Int. Conf. Auton. Agents and Multi-Agent Syst. (2011)
34. Jajodia, S., Ghosh, A.K., Swarup, V., Wang, C., Wang, X.S.: Moving target defense: creating asymmetric uncertainty for cyber threats, vol. 54. Springer Sci. & Bus. Media (2011)
35. Jajodia, S., Noel, S., O'berry, B.: Topological analysis of network attack vulnerability. In: Managing Cyber Threats (2005)

36. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
37. Kreibich, C., Crowcroft, J.: Honeycomb: Creating intrusion detection signatures using honeypots. Comput. Commun. Rev. **34**(1) (2004)
38. Kuipers, D., Fabro, M.: Control systems cyber security: Defense in depth strategies. Tech. rep., Idaho Nat. Labo. (2006)
39. Liu, Y., Man, H.: Network vulnerability assessment using bayesian networks. In: Data Mining, Intrusion Detection, Inf. Assurance, and Data Netw. Secur. (2005)
40. McKelvey, R.D., Palfrey, T.R.: Quantal response equilibria for normal form games. Games and Econ. Behav. **10**(1) (1995)
41. Mee, P., Schuermann, T.: How a cyber attack could cause the next financial crisis (2018), https://bit.ly/3f2lOFP
42. Mezura-Montes, E., Velázquez-Reyes, J., Coello Coello, C.A.: A comparative study of differential evolution variants for global optimization. In: Conf. Genetic and Evol. Comput. (2006)
43. Miah, M.S., Gutierrez, M., Veliz, O., Thakoor, O., Kiekintveld, C.: Concealing cyber-decoys using two-sided feature deception games. In: Int. Conf. Syst. Sci. (2020)
44. Nguyen, T.H., Wright, M., Wellman, M.P., Baveja, S.: Multi-stage attack graph security games: Heuristic strategies, with empirical game-theoretic analysis. Secur. and Commun. Netw. (2018)
45. Noel, S., Jajodia, S.: Managing attack graph complexity through visual hierarchical aggregation. In: Workshop Vis. and Data Mining for Comput. Secur. (2004)
46. Noel, S., Jajodia, S., O'Berry, B., Jacobs, M.: Efficient minimum-cost network hardening via exploit dependency graphs. In: Comput. Secur. Appl. Conf. (2003)
47. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., et al.: Pytorch: An imperative style, high-performance deep learning library. In: Adv. Neural Inf. Process. Syst. (2019)
48. Pawlick, J., Colbert, E., Zhu, Q.: A game-theoretic taxonomy and survey of defensive deception for cybersecurity and privacy. Comput. Surv. **52**(4) (2019)
49. Phillips, C., Swiler, L.P.: A graph-based system for network-vulnerability analysis. In: Workshop New Secur. Paradigms (1998)
50. Pita, J., Jain, M., Ordónez, F., Portway, C., Tambe, M., Western, C., et al.: Armor security for Los Angeles International Airport. In: AAAI Conf. on AI (2008)
51. Polad, H., Puzis, R., Shapira, B.: Attack graph obfuscation. In: Int. Conf. Cyber Secur., Cryptography, and Mach. Learn. (2017)
52. Schlenker, A., Thakoor, O., Xu, H., Tambe, M., Vayanos, P., Fang, F., et al.: Deceiving cyber adversaries: A game theoretic approach. In: Int. Conf. Auton. Agents and Multi-Agent Syst. (2018)
53. Schneier, B.: Attack trees. Dr. Dobb's Journal **24**(12) (1999)
54. Shen, W., Tang, P., Zuo, S.: Automated mechanism design via neural networks. In: Int. Conf. Auton. Agents and Multi-Agent Syst. (2019)
55. Sheyner, O., Haines, J., Jha, S., Lippmann, R., Wing, J.M.: Automated generation and analysis of attack graphs. In: IEEE Symp. Secur. and Privacy (2002)
56. Shi, Z.R., Procaccia, A.D., Chan, K.S., Venkatesan, S., Ben-Asher, N., Leslie, N.O., et al.: Learning and planning in feature deception games. arXiv preprint arXiv:1905.04833 (2019)
57. Shi, Z.R., Tang, Z., Tran-Thanh, L., Singh, R., Fang, F.: Designing the game to play: Optimizing payoff structure in security games. In: Int. Joint Conf. Artif. Intell. (2018)

58. Shieh, E., An, B., Yang, R., Tambe, M., Baldwin, C., DiRenzo, J., et al.: Protect: A deployed game theoretic system to protect the ports of the United States. In: Int. Conf. Auton. Agents and Multi-Agent Syst. (2012)
59. Shieh, E., An, B., Yang, R., Tambe, M., Baldwin, C., DiRenzo, J., et al.: Protect in the ports of Boston, New York and beyond: experiences in deploying Stackelberg security games with quantal response. In: Handbook Comput. Approaches to Counterterrorism (2013)
60. Stallings, W., Brown, L., Bauer, M.D., Bhattacharjee, A.K.: Computer Security: Principles and Practice (2012)
61. Storn, R., Price, K.: Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. J. Global Optim. **11**(4) (1997)
62. Tambe, M.: Security and Game Theory: Algorithms, Deployed Systems, Lessons Learned. Cambridge Univ. Press (2011)
63. Thakoor, O., Tambe, M., Vayanos, P., Xu, H., Kiekintveld, C., Fang, F.: Cyber camouflage games for strategic deception. In: Conf. Dec. and Game Theory for Secur. (2019)
64. Virtanen, P., Gommers, R., Oliphant, T.E., Haberland, M., Reddy, T., Cournapeau, D., et al.: Scipy 1.0: fundamental algorithms for scientific computing in python. Nature Methods **17**(3) (2020)
65. Wright, M., Wang, Y., Wellman, M.P.: Iterated deep reinforcement learning in games: History-aware training for improved stability. In: ACM Conf. Econ. and Comput. (2019)
66. Zhuang, J., Bier, V.M.: Reasons for secrecy and deception in homeland-security resource allocation. Risk Anal.: An Int. Journal **30**(12) (2010)

Appendix

We modify the DE [61] variant *DE/rand/1/bin* [42]. To initialize the population, we randomly choose the sequence of deceptive actions to consider. For each type, we determine the maximum number of components that can be altered (e.g., edges to add) for this individual. If allowed, we then modify the graph with randomly-selected modifications of that type. We also add a new termination condition based on the known optimal utility (0) for the defender if the defender has infinite protective and deceptive budgets. If any solution yields this utility, then we stop early and select it as the final solution. We also use a more compact solution representation: the full solution takes space $m_e(2|N|+1)+|N|+2|E|$, where m_e indicates the maximum number of edges that can be added to the graph given B^d. Each edge to be added takes space $2|N|$. To indicate that an edge is to be added, we take the arg max over the effort allocated in the first N and last N slots. The resulting indices i, j indicate the endpoints of the edge. If the summed effort at i, j is greater than a threshold, the edge is added. We further compact the representation when the defender cannot add or hide any edges without violating constraints by removing these parts of the solution, so each strategy uses space $|E|+|N|$.