A Game Theoretic Framework for Software Diversity for Network Security

Ahmed H. Anwar$^{1[0000-0001-8907-3043]}$, Nandi O. Leslie$^{1[]}$, Charles Kamhoua$^{1[0000-0003-2169-5975]}$, and Christopher Kiekintveld$^{2[0000-0003-0615-9584]}$

^{1} US Army Research Laboratory, MD 20783, USA
a.h.anwar@knights.ucf.edu,nandi.o.leslie.ctr@mail.mil,
charles.a.kamhoua.civ@mail.mil

^{2} University of Texas at El Paso, TX 79968, USA
cdkiekintveld@utep.edu

Abstract. Diversity plays a significant role in network security, and we propose a formal model to investigate and optimize the advantages of software diversity in network security. However, diversity is also costly, and network administrators encounter a tradeoff between network security and the cost to deploy and maintain a well-diversified network. We study this tradeoff in a two-player nonzero-sum game-theoretic model of software diversity. We find the Nash equilibrium of the game to give an optimal security strategy for the defender, and implement an algorithm for optimizing software diversity via embedding a graph-coloring approach based on the Nash equilibrium. We show that the opponent (i.e., adver- sary) spends more effort to compromise an optimally diversified network. We also analyze the complexity of the proposed algorithm and propose a complexity reduction approach to avoid exponential growth in runtime. We present numerical results that validate the effectiveness of the propo- sed software diversity approach.

Keywords: Software Diversity · Game Theory · Network Security.

1 Introduction

Diversity-based defenses to network security have recently emerged as a recognized approach to resilience. In particular, these types of defenses introduce uncertainty and probabilistic protection from attacks and provide a rich framework for diversifying program transformations [14,21,2]. In a few areas of security (e.g., moving target defense [2]) there has been a wide application of software diversity techniques. For example, methods have been proposed for giving the software a dynamically shifting attack surface for both binary executables and web applications [15]. However, there remain limitations with these defenses depending on the threat. For example, address space layout randomization is ineffective against buffer-overflow attacks, such as the de-randomization attack [23]. Graph theory, and particularly attack graphs, provides a unique formal approach to help quantify the efficiency and effectiveness of temporal and spatio-temporal diversity mechanisms. Moreover, security games on graphs [3,1,4] provide mathematical approaches that can offer insights into questions such as what is the tradeoff between network security and software diversity, and what must be diversified and when [14]. To address these questions, we introduce mathematical models that use game theory to examine the connection between the distribution of differing software configurations on a network and the resulting risk to network security against a motivated attacker.

Software diversity involves randomizing transformations that make program implementations diverge between each networked device or between each execution. These proactive defense strategies can increase the attackers' workload [14,16]. There is an analogy to biological ecosystems, where the resiliency of a population to disease or the invasion of a nonnative species depends heavily on biodiversity. Likewise, network resiliency can be increased (especially against novel threats) by using efficient strategies for increasing software diversity. Attack graphs are an important tool that models the network's topology and spatio-temporal vulnerabilities used to validate various defense approaches. Attack graphs can be generated in different ways to represent interactions between the host's vulnerabilities and its neighbors' vulnerabilities. In addition to using the attack graph for understanding how network topology and vulnerabilities impact the effectiveness of diversity-based defenses, a security game on an attack graph captures the connections between diversity, security, and reachability [21]. In this paper, a security game is formulated and played on an attack graph to study the effectiveness of diversity-based defenses. The developed game investigates the tradeoff faced by the defender between diversity cost and security level. Our main contributions in this paper are:

- We propose a general model suitable to study software diversity for the security of networked systems. Our model captures the set of vulnerabilities and the network topology through an attack graph.
- We formulate a novel game model to study the effect of diversity on network security under attack as a two-player nonzero-sum game.
- We present a complete algorithm to solve the game model and obtain the Nash equilibrium diversity strategy.
- We analyze the complexity of the proposed algorithm and introduce a complexity reduction approach that is shown to yield an almost exact reward for the defender in our numerical results.
- Finally, we present numerical results for the developed software diversity approach that show the effectiveness of the obtained diversity strategy at Nash equilibrium.

The rest of the paper is organized as follows. We discuss related work in section 2. In section 3, we present the system model, define the game model, and propose our algorithm for software diversity. Our numerical results is presented in section 4. Finally, we conclude our work and discuss future work in section 5.

2 Related Work

The scope of the problem we consider belongs to three interacting active research fields: network diversity, resilience, and game theory. Diversity has been a design objective to secure networks against various types of attacks including Zero-Day attacks [28]. It has been shown that the intuition behind the ability of diversity to increase the resiliency of systems and networks is effective [10]. Garcia et al. [10] show using a data-driven study that building a system with diverse operating systems is a useful technique to enhance its intrusion tolerance capabilities. Moreover, diversity-by-design has been used to increase a communication network's throughput [6,17]. For security, the authors in [5] proposed an automated approach to diversify network services to enhance the network's resilience against unknown threats and attacks. In their approach, they considered constraints associated with diversity. Software diversification techniques are also used to enhance network security by reducing the attack surface [27,26].

Graph coloring is a well-known problem in dynamic channel assignment to reduce interference between adjacent nodes [24,4]. However, the applications go beyond reducing network interference to include securing medical images [25,19]. Moreover, graph coloring has been used in several computer science applications such as data mining, clustering, image capturing, image segmentation, networking, etc. [11]. Game theory has been used directly to solve graph coloring problems; such research problems are named "coloring games". A coloring game is a two player non-cooperative game played on a finite graph using a set of colors in which players take turns to color the vertices of the graph such that no two adjacent vertices have the same color [13].

Game theory has been used extensively to study security problems and understand the strategic behavior of adversarial users and attackers [22,1,12]. In [1], a game-theoretic framework is developed to investigate jamming attacks on wireless networks and the defender mitigation strategy. Kiekintveld et al. [12] proposed a scalable technique to calculate security resource allocation optimal policy of practical problems like police patrolling schedule for subway and bus stations. A first step to quantify and measure diversity as a security metric appeared in [21] where a game model has been used to investigate the necessary conditions for network defender to diversify and avoid monoculture systems. However, diversity is understudied in the literature of security games. In this paper, we introduce a generalized nonzero-sum game model between the network defender and an attacker. Motivated by the aforementioned benefits of diversity, the defender player selects the best diversity policy in response to the attacker's strategies. The game is played over an attack graph that captures the network topology and the dependencies between the vulnerabilities in the network.

3 System Model

Consider a network of arbitrary size, $|\mathcal{N}|$, where $\mathcal{N}$ is the set of nodes of the network. The network topology is defined by its adjacency matrix. Let $\mathbf{H}$, denote the graph adjacency matrix, where any entry of the network $[\mathbf{H}_{u,v}] = 1$ if node v is connected to node u, and is equal to 0 otherwise, for every $u, v \in \mathcal{N}$.

We assume the network graph is represented using a directed graph denoted by $G(\mathcal{N}, \mathbf{H})$, where $[\mathbf{H}_{u,v}] = 1$, denotes an edge between node u and node v. This assumption fits a hierarchical network with a set of entry nodes that allow users to access the network. Such networks resemble networks with a chain of command. It also represents an interesting scenario where the depth of the network can be captured. Adversaries are interested in reaching targets that are practically installed in deeper layers of the network.

For each node $v \in \mathcal{N}$, there is a certain software type that is running on the node. A software type can abstract several properties, for example, it can model the operating system, honeypot type, or specific application. For simplicity, we assume the set of all used software types to be $\mathcal{S}$, where each node is assigned one software type $s \in \mathcal{S}$. Let NSW denote node-software matrix of size $|\mathcal{N}| \times |\mathcal{S}|$. For instance, the NSW matrix shown in equation (1) represents a network of 3 nodes and a set $\mathcal{S} = \{s_1, s_2\}$, where node 1 runs software type s_1 and the remaining two nodes run software type s_2.

$$NSW = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{bmatrix} \tag{1}$$

Each software type has one or more vulnerabilities. We let $\mathcal{V}$ be the set of all vulnerabilities. Again we use matrix representation to define the software to vulnerabilities relation. Let SWV be a $|\mathcal{S}| \times |\mathcal{V}|$ be a binary matrix, where each row is a vector associated with each software type that indicates which vulnerability is associated with that software. Specifically, any entry $SWV[i, j] = 1$ if and only if a software $s_i \in \mathcal{S}$ suffers vulnerability $v_j \in \mathcal{V}$, and $SWV[i, j] = 0$ otherwise.

Given NSW and SWV, the set of vulnerabilities that could be exploited by the attacker can be defined for each node. However, to target a node that node should satisfy two conditions. First, it should be *reachable* through a path. Secondly, that node should be *exploitable* through at least one vulnerability. Note that an attacker can *reach* any node if and only if there exists a path between network's entry node and that node subject that the attacker can also compromise all the nodes that belong to this path.

Let $\mathcal{P}_v$ be a set of nodes connecting the network entry node and any node $v \in \mathcal{N}$. Specifically, $\mathcal{P}_v = \{v_0, ..., v\}$, where v_0 is an entry node, and v denotes any node in the network, however v usually denotes the node being targeted by the attacker. Hence, the set of software implemented on each node affects the ability of an adversary to reach v as he is required to exploit all nodes that belong to the path $\mathcal{P}_v$.

We define a two-player nonzero-sum game between the network administrator as the defender and an adversary as the attacker. We consider the defender to be player 1 and the attacker to be player 2. We start by discussing the attacker problem and the possible attack strategies.

3.1 Attacker problem

The goal of the attacker is to compromise a subset of targeted nodes in the network using an attack toolbox. We assume that the attacker has a set of probes that allow him to compromise a set of vulnerabilities. More specifically, each probe in the toolbox can exploit a subset of vulnerabilities. Let $\mathcal{B}$ denote the set of all probes, i.e, the toolbox available to the attacker. The relation between each probe in $\mathcal{B}$ and the kind of vulnerabilities it exploits is characterized through a probe matrix.

The probe matrix denoted by $\mathbf{P}$ is a $|\mathcal{B}| \times |\mathcal{V}|$ binary matrix. Any entry $\mathbf{P}[i,j] = 1$ if the i^{th} probe is capable of exploiting the j^{th} vulnerability, for every $i \in \mathcal{B}$ and $j \in \mathcal{V}$, and $\mathbf{P}[i,j] = 0$ otherwise. For instance, the matrix in equation (2) represents two probes within the attacker action space and three vulnerabilities. If the attacker attacks the network using the first probe, she will only compromise the subset of reachable nodes with software that suffers vulnerability Vul^1. On the other hand, if the attacker attacked the network using the second probe, she will be able to compromise all reachable nodes with software type that suffers Vul^2 and Vul^3.

$$\mathbf{P} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix} \tag{2}$$

The attacker increases his payoff by maximizing the number of compromised nodes in the network. Therefore, the attacker chooses to use a collection of probes instead of using a single probe when attacking the network. We can readily define the attacker action space as the collection set of all elements in the probe set $\mathcal{B}$. Let the attacker action space be denoted by $\mathcal{A}_2$. Specifically, $\mathcal{A}_2 = \{0,1\}^{|\mathcal{B}|}$. Therefore, any attack action $a_2 \in \mathcal{A}_2$ is a binary vector of length $|\mathcal{B}|$, where $a_2(i) = 1$ when the i^{th} probe is used in the attack, and $a_2(i) = 0$ otherwise, for $i = 1, 2, ..., |\mathcal{B}|$. To avoid trivial game scenarios, we assume a cost associated with each probe which can represent (for example) the increased likelihood of detection. Let $C_a(a_2)$ be the cost for each probe. As we discuss in more detail later, the attacker faces an interesting trade-off as he wants to attack the network using a larger number of probes to compromise more nodes, while reducing his attack cost to avoid expensive attacks. Next, we discuss the defender problem before we fully characterize the players' payoff functions in more detail.

3.2 Defender problem

We focus on a defender who uses diversity to enhance network security. The defender action affects the node software matrix, NSW. Based on the available number of software types and how they are assigned to nodes in the network, the defender can increase the level of diversity in the network. The defender can potentially use all the available software types to achieve the maximum level of security through diversity. However, a highly diversified network is harder to operate and maintain. Therefore, the defender incurs a cost associated with the diversity size, $|\mathcal{S}|$. Let $C_d(a_1)$ denote the cost associated with the defense strategy, for any defender action $a_1 \in \mathcal{A}_1$, where $\mathcal{A}_1$ is the defender action space. The defender action space contains all the combinations of software types. The defense strategy a_1 selects a subset out of the software set, $\mathcal{S}$. For instance, a defense strategy $a_1 = \{s_1, s_2, s_3\}$ means that the defender is implementing 3 different software types to run on different nodes in the network.

Allocating the selected software types over different nodes is similar to the well-known graph coloring problem. Therefore, we adapt graph-coloring algorithms to implement strategies that ensure that neighboring nodes do not run the same software type whenever possible. Having a larger palette with more colors will directly enhance the effectiveness of the graph coloring algorithm. This in turn reduces the attacker reachability to a smaller set of nodes. In Algorithm 1, we leverage the graph coloring algorithm proposed in [9] to implement such an approach.

The defender trade-off is to minimize the size of the set of software types to be diversified, to reduce nodes' reachability while minimizing the cost associated with such a defense strategy. In other words, the defender aims to secure the maximum number of nodes using the minimum number of different software types. However, the attacker attempts to compromise the maximum number of nodes using the smallest number of probes. Next, we quantify the payoff functions for both players.

3.3 Payoff functions

The goal of the defender is to secure the network through securing as many nodes as possible using software diversity. Protecting nodes can be achieved through the careful distribution of different software types to neighboring nodes. The subset of secured nodes depends on their topological locations in the network and the vulnerabilities associated with the software type assigned to each of them as defined via the SWV matrix. Given the software vulnerability matrix, SWV, and node software matrix NSW, one can easily define a node vulnerability matrix, NV, that defines the subset of vulnerabilities associated with each node as follows,

$$NV = NSW \times SWV. \tag{3}$$

Recall that the graph is colored according to the action played by the defender, a_1, and hence NSW is defined. However, the attacker action, a_2, defines the set of exploitable vulnerabilities according to probe matrix $\mathbf{P}$. The attacker goal is to maximize the number of compromised nodes, which is denoted by K, and can be expressed as follows:

$$K = mean\left(\sum_{v\in\mathcal{N}}\left(\sum_{u\in\mathcal{P}_v} \mathbb{1}_{\{u\in\mathcal{E}(a_2)\}}\right)\right), \tag{4}$$

where $\mathbb{1}_{\{.\}}$ is an indicator function, which is equal to one when $\{u \in \mathcal{E}(a_2)\}$, where $\mathcal{E}(a_2)$ is the set nodes that can be exploited and compromised by the attacker. The set of exploitable nodes $\mathcal{E}(a_2)$ contains all nodes that are assigned a software type that has any of the vulnerabilities that can be compromised using the probes in a_2. Let $\mathcal{V}^{a_2}$ denote the set of vulnerabilities that the attacking probe(s) can exploit given the attacker action, a_2. Also, let $NV(u)$ be the set of vulnerabilities associated with the software type running on node u. Then, the set of exploitable nodes can be defined as, $\mathcal{E}(a_2) = \{u \in \mathcal{N} | NV(u) \cap \mathcal{V}^{a_2} \neq \Phi\}$. Therefore, K represents the average distances between exploitable nodes (i.e, subgraphs diameter).

The defender encounters a diversity cost $C_d(.)$ that depends on the number of software types (colors) used to color the network graph. For simplicity, we assume a fixed cost per color.

Therefore, the defender payoff function can be written as:

$$R_1(a_1, a_2) = -K - C_d(a_1), \tag{5}$$

and the attacker payoff function is written as:

$$R_2(a_1, a_2) = K - C_a(a_2). \tag{6}$$

The K term captures an interesting tradeoff for the defender. If the defender has a large budget and does not care about the defense cost, using a very large number of software types is still a double-edged sword. A higher number of software types (i.e, colors) allows for a better graph coloring outcome and hence limits the attacker's ability to reach a bigger community. However, since each software suffers a subset of vulnerability, this may increase the number of exploitable nodes in the graph. Therefore, using all the available software types to color the graph may not be in favor of the defender. It is worth noting that we do not assume that any of the software types are risk-free, otherwise, the problem is trivial and the defender better off using a complete monoculture of that secured software.

On the attacker side, using the maximum number of available probes is always in favor of the attacker if the cost per probe is zero. Thus, the game is designed to investigate the trade-off between the reward of exploiting nodes using available probes, and the cost associated with each subset of probes.

3.4 Game problem

We now formulate the game $\Gamma(P, \mathcal{A}, \mathcal{R})$, where:

- $\mathcal{P} = \{Defender, Attacker\}$ is the set of players.
- $\mathcal{A} = \{\mathcal{A}_1 \times \mathcal{A}_2\}$ is the game action space, which is the product of the action space of the defender and the action space of the defender as defined in the previous subsection.
- $\mathcal{R} = \{R_1, R_2\}$ denotes the game reward set.

As shown in equations (5) and (6), Γ is a nonzero-sum game with a finite number of pure actions for every player. Nonzero-sum reflects the fact that the attacker does not benefit from the cost paid by the defender, and vice versa. Let A and B denote payoff matrices for the defender and attacker, respectively. Both matrices are of size $|\mathcal{A}_1| \times |\mathcal{A}_2|$. The defender maximizes over the rows of A, and the attacker maximizes his reward over the columns of B. Moreover, let $\mathbf{x}$ be a vector of $|\mathcal{A}_1| \times 1$ and $\mathbf{y}$ be a vector of $|\mathcal{A}_2| \times 1$. A mixed strategy, $\mathbf{x}$, is a probability distribution over the action space, $|\mathcal{A}_1|$. Similarly, the attacker mixed strategy, $\mathbf{y}$, is a probability distribution over the action space, $|\mathcal{A}_2|$.

Theorem 1. *For the finite game Γ, there exists at least one point ($\mathbf{x}^*$,$\mathbf{y}^*$) of mixed equilibrium.*

Proof. The proof follows Nash's theory in [20] directly. The theory states that for every pair of payoff matrices A, B there is a nonzero number of mixed equilibria.

For any mixed strategy, $\mathbf{y}$, played by the attacker, the defender maximizes his expected reward by solving the following optimization problem to find his best response strategy $\mathbf{x}^*$,

$$\begin{aligned} \underset{\mathbf{x}}{\text{maximize}} \quad & \mathbf{x}^T A \mathbf{y} \\ \text{subject to} \quad & \sum_{i=1}^{|\mathcal{A}_1|} x(a_1^i) = 1, \\ & \mathbf{x} \geq 0. \end{aligned} \tag{7}$$

On the other side, for any mixed strategy, $\mathbf{x}$, played by the defender, the attacker finds the optimal attacking strategy $\mathbf{y}^*$ by solving the following optimization problem,

$$\begin{aligned} \underset{\mathbf{y}}{\text{maximize}} \quad & \mathbf{x}^T B \mathbf{y} \\ \text{subject to} \quad & \sum_{j=1}^{|\mathcal{A}_2|} y(a_2^j) = 1, \\ & \mathbf{y} \geq 0. \end{aligned} \tag{8}$$

It has been shown by Chen et al. [8] that for the general n-person nonzero-sum non-cooperative games, computing Nash equilibria is PPAD-complete. However, for the two-player case of a nonzero-sum game with a finite number of pure strategies as Γ, a necessary and sufficient condition for a point to be a point of equilibrium is that it is a solution of a single programming problem of a quadratic objective function and a set of linear constraints and the objective function has a global maximum of zero as shown in [18]. Based on the work in [18], MATLAB code has been developed in [7] that computes at least one point of Nash equilibrium using sequential quadratic programming based quasi-Newton technique which is used to solve the above optimization problems in (7) and (8). We apply the procedure shown in Algorithm 1 to obtain a software diversity strategy based on the formulated game.

Algorithm 1 Diversify

```
1: procedure DIVERSIFY($G, \mathcal{S}, \mathcal{B}, SWV, P, C_d, C_a$)          ▷ Input parameters
2:     System Initialization
3:     Define: $\mathcal{A}_1$, $\mathcal{A}_2$
4:     for $a_1 \in \mathcal{A}_1$ do
5:         Graph Color $(G, a_1)$                          ▷ Graph coloring algorithm
6:         Update $NSW$                                    ▷ Build node-software matrix
7:         Compute $NV = NSW \times SWV$                   ▷ node-vulnerability matrix
8:         for $a_2 \in \mathcal{A}_2$ do
9:             Compute $R_1(a_1, a_2) \rightarrow$ update $A$
10:            Compute $R_2(a_1, a_2) \rightarrow$ update $B$
11:    GameSolver$(A, B) \rightarrow \mathbf{x}^*, \mathbf{y}^*$       ▷ Mixed strategies equilibrium for (7),(8)
```

3.5 Game complexity

The computational time for solving the game programs depends on the dimensions of the action space $\mathcal{A}$, or the number of pure actions for each player. Unfortunately, the time grows exponentially in the number of strategies of both players.

For our game model, the number of pure actions does not grow with the number of nodes of the network. Instead, it grows with the number of software types available to the defender, and with the number of probes used by the attacker. However, this rate of growth is exponentially increasing with the number of software types (colors). For instance, if the number of available software $|\mathcal{S}|$, then the number of pure strategies for the defender, $|\mathcal{A}_1| = 2^{|\mathcal{S}|}$. Similarly, if the number of available probes to the attacker is $|\mathcal{B}|$, the number of pure attacker's pure actions is $|\mathcal{A}_2| = 2^{|\mathcal{B}|}$.

3.6 Complexity reduction:

Since the set of vulnerabilities associated with each software type is known to the defender, the defender can prioritize the use of available software types accordingly. More specifically, let $\mathbf{w}$ be weight vector of size $|S|$, such that,

$$\mathbf{w} = SWV \times e, \tag{9}$$

where e, is a column vector of all ones of size $|\mathcal{V}| \times 1$. Hence, $\mathbf{w}$ represents the weight of each software type $s \in \mathcal{S}$, in terms of the number of vulnerabilities it introduces into the network when used by the defender.

Therefore, the defender does not need to consider all the possible combinations of the available software types. Instead, the defender optimizes over the number of software types (i.e, number of colors) to implement, and sorts the software set according to their weights in ascending order. For instance, if the defender decided to use three software types, she can immediately pick the three colors with the smallest weights according to $\mathbf{w}$ as defined in (9).

This approach leads to a significant reduction in the complexity of the game as the size of the action space $|\mathcal{A}_1|$ will not grow exponentially with the number of the available software types $|\mathcal{S}|$, it will grow linearly, instead.

Moreover, in the case of a perfect information game, the defender is assumed to know the probe matrix $\mathbf{P}$ as defined in equation (2). Therefore, the defender may sort the attacker's probes according to their potential damage. Along the same lines as $\mathbf{w}$, let $\mathbf{d} = \mathbf{P} \times e$ denote the damage vector of size $|\mathcal{B}| \times 1$, the defender can sort the probes available to her opponent in descending order according to damage vector $\mathbf{d}$ assuming a worst-case scenario in which the attacker always uses the most powerful probe first. The attacker is now optimizing the number of probes to use when launching an attack. With this reduction, we can redefine the action space for both players as follows, $\bar{\mathcal{A}}_1 = \{1, 2, ..., |\mathcal{S}|\}$ and $\bar{\mathcal{A}}_2 = \{1, 2, ..., |\mathcal{B}|\}$.

Using these heuristics we can significantly enhance the runtime of Algorithm 1 and present a Fast-Diversify as shown in Algorithm 2.

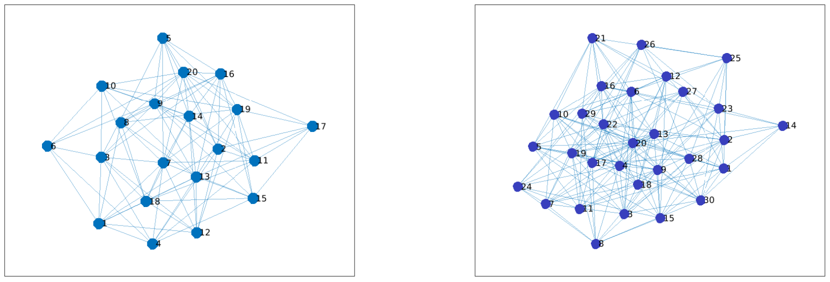
Algorithm 2 Fast-Diversify

```
 1: procedure DIVERSIFY($G, \mathcal{S}, \mathcal{B}, SWV, P, C_d, C_a$)        ▷ Input parameters
 2:     System Initialization
 3:     Compute $\mathbf{w}$, $\mathbf{d}$
 4:     Sort $\mathbf{w}$ "Ascend"
 5:     Sort $\mathbf{d}$ "Descend"
 6:     Define: $\bar{\mathcal{A}}_1$, $\bar{\mathcal{A}}_2$
 7:     for $a_1 \in \mathcal{A}_1$ do
 8:         Graph Color $(G, a_1)$                            ▷ Graph coloring algorithm
 9:         Update $NSW$                                    ▷ Build node-software matrix
10:         Compute $NV = NSW \times SWV$                   ▷ node-vulnerability matrix
11:         for $a_2 \in \mathcal{A}_2$ do
12:             Compute $R_1(a_1, a_2) \rightarrow$ update $A$
13:             Compute $R_2(a_1, a_2) \rightarrow$ update $B$
14:     GameSolver$(A, B) \rightarrow \mathbf{x}^*, \mathbf{y}^*$       ▷ Mixed strategies equilibrium for (7),(8)
```

In the following section, we present numerical results to validate the developed algorithms.

4 Numerical Results

We now present numerical results that validate the proposed game model. First, we consider a 20-node network that we generated such that any two nodes are connected directly with an edge with probability 0.5, as illustrated in Fig. 1a. We investigate the behavior of the players based on the Nash equilibrium strategies computed for the proposed game model with different values of the game model parameters.



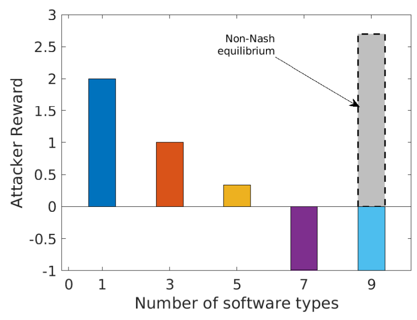
(a) A 20-node network topology. (b) A 30-node network topology

Fig. 1: The two generated network topologies with randomly generated edges.

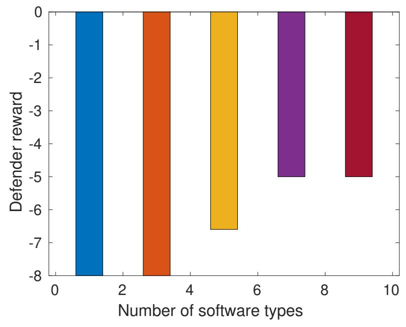
For the network topology shown in Fig. 1a, we plot the attacker's reward in Fig. 2a for different numbers of software types. It is clear that as the number of available software types increases, the defender can color the graph more efficiently, and hence software diversity will significantly reduce the attacker reward. However, increasing the number of available software types does not imply that the reward of the defender increases steadily since the defender plays his Nash equilibrium strategy. The Nash equilibrium strategy may lead the defender not to use all the available colors, since the use of a larger set of software types may introduce new vulnerabilities to the network. Therefore, in Fig. 2a, the attacker reward when the defender unilaterally deviated from his Nash strategy was higher than the attacker reward even when the defender used only a single software type (i.e., mono-culture case). In Fig. 2b, the defender reward is plotted at a different number of software types for the Nash equilibrium defense strategy from both sides. The defender reward is non-decreasing as the number of available software types increases.

To understand the effect of the cost parameter for both players, we plot the defender and attacker rewards at different cost values when the game played on a 30-node network, as shown in Fig. 1b.

As shown in Fig. 3a, the defender and the attacker rewards are plotted for different cost values per each software used by the defender at Nash equilibrium. The defender reward decreases as the cost increases since the defender tends to exclude more software types to avoid the defender cost C_d. On the other side, the attacker reward increases as the defender diversity capabilities are limited by the increasing cost while the cost per probe is fixed at 1. For the considered network, the vulnerability set contains 3 vulnerabilities, and the attacker has four probe types. To illustrate the role of the cost per probe, we plotted the reward of both players versus the cost per probe in Fig. 3b. At a low cost per probe, the defender reward is stably low as the attacker can afford the cost to use all his probes in the attack. As the cost per probe increases beyond 1, the defender reward starts to increase. However, since the defender cost is fixed at

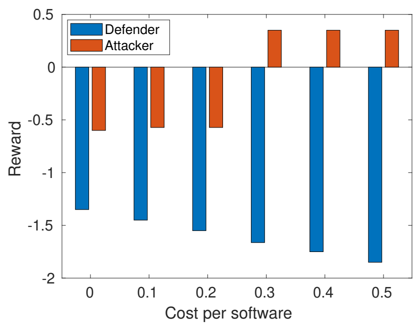


(a) Attacker reward.

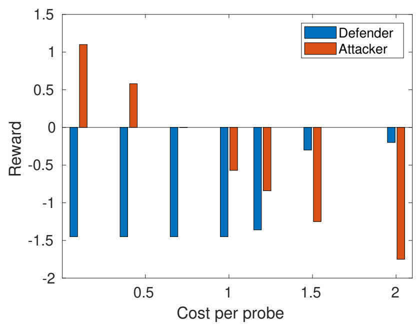


(b) Defender reward.

Fig. 2: Players' reward vs. the number of software types at Nash equilibrium.



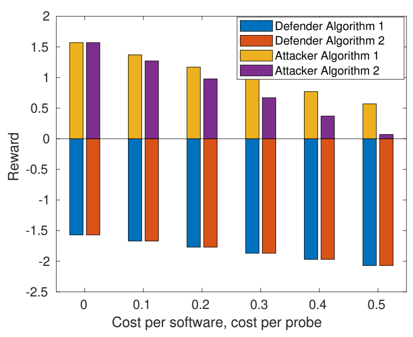
(a) Varying the cost per each software type, the cost per probe is 1



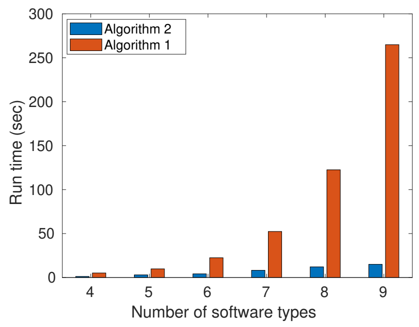
(b) Varying the cost per probe, the cost per software is 0.1.

Fig. 3: Comparing players' reward versus their action cost values

a low value of 0.1, the attacker is being punished more and hence the attacker cost starts to decrease significantly.



(a) Players reward at different cost values.



(b) Comparing the runtime at different numbers of software type.

Fig. 4: Comparison between the performance of the two proposed algorithms.

Finally, we compare the efficiency of the proposed algorithm versus the cost per software and cost per probe in Fig. 4a. We made the cost per software equal to the cost per probe for simplicity. As shown in Fig. 4a, the Fast-Diverisfy algorithm yields the exact reward for the defender and very comparable to the attacker as Algorithm 2 assumes a worst-case scenario for the attacker to reduce complexity. Moreover, in Fig. 4b we compare the runtime of both algorithms to show the significant reduction in complexity achieved via Algorithm 2.

5 Conclusion and Future work

We studied a software diversity approach for network security via a formulated game-theoretic model over an attack graph. In this context, we developed a novel game model to study the interactions between network defender and an adversary when software diversity is the main defensive strategy for the defender. We adapted a graph-coloring algorithm for computing Nash equilibrium diversifying strategy and developed a complexity reduction approach to obtain Nash equilibrium more efficiently making the proposed diversifying algorithm applicable in large-scale networks with a larger number of colors. Numerical results comp- uted using our model show both the benefits of software diversity as well as the detailed tradeoffs that are necessary for both attackers and defenders in this scenario. We also validate the computational effectiveness of our algorithms for practical applications. Our ongoing research is focused on extending the formulated game model to account for cases of incomplete information. In this case, the software vulnerability matrix and the probe matrix are unknown beforehand by the defender.

Acknowledgment

Research was sponsored by the Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-19-2-0150. 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 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 herein.

References

1. Anwar, A.H., Atia, G., Guirguis, M.: Game theoretic defense approach to wireless networks against stealthy decoy attacks. In: 2016 54th Annual Allerton Conference on Communication, Control, and Computing (Allerton). pp. 816–821. IEEE (2016)
2. Anwar, A.H., Atia, G., Guirguis, M.: It's time to migrate! a game-theoretic framework for protecting a multi-tenant cloud against collocation attacks. In: 2018 IEEE 11th International Conference on Cloud Computing (CLOUD). pp. 725–731. IEEE (2018)
3. Anwar, A.H., Kelly, J., Atia, G., Guirguis, M.: Stealthy edge decoy attacks against dynamic channel assignment in wireless networks. In: MILCOM 2015-2015 IEEE Military Communications Conference. pp. 671–676. IEEE (2015)
4. Anwar, A.H., Kelly, J., Atia, G., Guirguis, M.: Pinball attacks against dynamic channel assignment in wireless networks. Computer Communications **140**, 23–37 (2019)
5. Borbor, D., Wang, L., Jajodia, S., Singhal, A.: Diversifying network services under cost constraints for better resilience against unknown attacks. In: IFIP Annual Conference on Data and Applications Security and Privacy. pp. 295–312. Springer (2016)
6. Casini, E., De Gaudenzi, R., Herrero, O.D.R.: Contention resolution diversity slotted aloha (crdsa): An enhanced random access schemefor satellite access packet networks. IEEE Transactions on Wireless Communications **6**(4), 1408–1419 (2007)
7. Chatterjee, B.: An optimization formulation to compute nash equilibrium in finite games. In: 2009 Proceeding of International Conference on Methods and Models in Computer Science (ICM2CS). pp. 1–5. IEEE (2009)
8. Chen, X., Deng, X.: Settling the complexity of two-player nash equilibrium. In: 2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06). pp. 261–272. IEEE (2006)
9. Farzaneh, M.: Graph Coloring by Genetic Algorithm. `https://www.mathworks.com/matlabcentral/fileexchange/74118-graph-coloring-by-genetic-algorithm` (2020), [MATLAB Central File Exchange. Retrieved July 12, 2020.]
10. Garcia, M., Bessani, A., Gashi, I., Neves, N., Obelheiro, R.: Os diversity for intrusion tolerance: Myth or reality? In: 2011 IEEE/IFIP 41st International Conference on Dependable Systems & Networks (DSN). pp. 383–394. IEEE (2011)
11. Jensen, T.R., Toft, B.: Graph coloring problems, vol. 39. John Wiley & Sons (2011)
12. Kiekintveld, C., Jain, M., Tsai, J., Pita, J., Ordóñez, F., Tambe, M.: Computing optimal randomized resource allocations for massive security games. In: Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 1. pp. 689–696 (2009)

13. Kierstead, H.A.: Asymmetric graph coloring games. Journal of Graph Theory **48**(3), 169–185 (2005)
14. Larsen, P., Homescu, A., Brunthaler, S., Franz, M.: Sok: Automated software diversity. In: 2014 IEEE Symposium on Security and Privacy. pp. 276–291. IEEE (2014)
15. Le Goues, C., Forrest, S., Weimer, W.: Current challenges in automatic software repair. Software quality journal **21**(3), 421–443 (2013)
16. Le Goues, C., Nguyen-Tuong, A., Chen, H., Davidson, J.W., Forrest, S., Hiser, J.D., Knight, J.C., Van Gundy, M.: Moving target defenses in the helix self-regenerative architecture. In: Moving target defense II, pp. 117–149. Springer (2013)
17. Liva, G.: Graph-based analysis and optimization of contention resolution diversity slotted aloha. IEEE Transactions on Communications **59**(2), 477–487 (2010)
18. Mangasarian, O.L., Stone, H.: Two-person nonzero-sum games and quadratic programming. Journal of Mathematical Analysis and applications **9**(3), 348–355 (1964)
19. Moumen, A., Bouye, M., Sissaoui, H.: New secure partial encryption method for medical images using graph coloring problem. Nonlinear Dynamics **82**(3), 1475–1482 (2015)
20. Nash, J.F., et al.: Equilibrium points in n-person games. Proceedings of the national academy of sciences **36**(1), 48–49 (1950)
21. Neti, S., Somayaji, A., Locasto, M.E.: Software diversity: Security, entropy and game theory. In: HotSec (2012)
22. Roy, S., Ellis, C., Shiva, S., Dasgupta, D., Shandilya, V., Wu, Q.: A survey of game theory as applied to network security. In: 2010 43rd Hawaii International Conference on System Sciences. pp. 1–10. IEEE (2010)
23. Shacham, H., Page, M., Pfaff, B., Goh, E.J., Modadugu, N., Boneh, D.: On the effectiveness of address-space randomization. In: Proceedings of the 11th ACM conference on Computer and communications security. pp. 298–307 (2004)
24. Sohn, S.: Graph coloring algorithms and applications to the channel assignment problems. In: IT Convergence and Security 2012, pp. 363–370. Springer (2013)
25. Thiyagarajan, P., Aghila, G.: Reversible dynamic secure steganography for medical image using graph coloring. Health Policy and Technology **2**(3), 151–161 (2013)
26. Wang, S., Wang, P., Wu, D.: Composite software diversification. In: 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME). pp. 284–294. IEEE (2017)
27. Wartell, R., Mohan, V., Hamlen, K.W., Lin, Z.: Binary stirring: Self-randomizing instruction addresses of legacy x86 binary code. In: Proceedings of the 2012 ACM conference on Computer and communications security. pp. 157–168 (2012)
28. Zhang, M., Wang, L., Jajodia, S., Singhal, A., Albanese, M.: Network diversity: a security metric for evaluating the resilience of networks against zero-day attacks. IEEE Transactions on Information Forensics and Security **11**(5), 1071–1086 (2016)