A Game-Theoretic Approach to Respond to Attacker Lateral Movement

Mohammad A. Noureddine[1,3], Ahmed Fawaz[2,3], William H. Sanders[2,3], and Tamer Başar[2,3]

[1] Department of Computer Science
[2] Department of Electrical and Computer Engineering
[3] Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
{nouredd2, afawaz2, whs, basar1}@illinois.edu

Abstract. In the wake of an increasing number in targeted and complex attacks on enterprise networks, there is a growing need for timely, efficient and strategic network response. Intrusion detection systems provide network administrators with a plethora of monitoring information, but that information must often be processed manually to enable decisions on response actions and thwart attacks. This gap between detection time and response time, which may be months long, may allow attackers to move freely in the network and achieve their goals. In this paper, we present a game-theoretic approach for automatic network response to an attacker that is moving laterally in an enterprise network. To do so, we first model the system as a network services graph and use monitoring information to label the graph with possible attacker lateral movement communications. We then build a defense-based zero-sum game in which we aim to prevent the attacker from reaching a sensitive node in the network. Solving the matrix game for saddle-point strategies provides us with an effective way to select appropriate response actions. We use simulations to show that our engine can efficiently delay an attacker that is moving laterally in the network from reaching the sensitive target, thus giving network administrators enough time to analyze the monitoring data and deploy effective actions to neutralize any impending threats.

1 Introduction

In the wake of the increasing number of targeted and complex network attacks, namely Advanced Persistent Threats (APTs), organizations need to build more resilient systems. *Resiliency* is a system's ability to maintain an acceptable level of operation in light of abnormal, and possibly malicious, activities. The key feature of resilient systems is their ability to react quickly and effectively to different types of activities. There has been an ever-increasing amount of work on detecting network intrusions; Intrusion Detection Systems (IDSs) are widely deployed as the first layer of defense against malicious opponents [10]. However, once alarms have been raised, it may take a network administrator anywhere from weeks to months to effectively analyze and respond to them. This delay

creates a gap between the intrusion detection time and the intrusion response time, thus allowing attackers a sometimes large time gap in which they can move freely around the network and inflict higher levels of damage.

An important phase of the life cycle of an APT is lateral movement, in which attackers attempt to move laterally through the network, escalating their privileges and gaining deeper access to different zones or subnets [2]. As today's networks are segregated by levels of sensitivity, lateral movement is a crucial part of any successful targeted attack. An attacker's lateral movement is typically characterized by a set of causally related chains of communications between hosts and components in the network. This creates a challenge for detection mechanisms since attacker lateral movement is usually indistinguishable from administrator tasks. It is up to the network administrator to decide whether a suspicious chain of communication is malicious or benign. This gap between the detection of a suspicious chain and the administrator's decision and response allows attackers to move deeper into the network and thus inflict more damage. It is therefore essential to design response modules that can quickly respond to suspicious communication chains, giving network administrators enough time to make appropriate decisions.

Intrusion Response Systems (IRSs) combine intrusion detection with network response. They aim to reduce the dangerous time gap between detection time and response time. Static rule-based IRSs choose response actions by matching detected attack steps with a set of rules. Adaptive IRSs attempt to dynamically improve their performance using success/failure evaluation of their previous response actions, as well as IDS confidence metrics [23, 21]. However, faced with the sophisticated nature of APTs, IRSs are still unable to prevent network attacks effectively. Rule-based systems can be easily overcome by adaptive attackers. Adaptive systems are still not mature enough to catch up with the increased complexity of APTs.

In this paper, we present a game-theoretic network response engine that takes effective actions in response to an attacker that is moving laterally in an enterprise network. The engine receives monitoring information from IDSs in the form of a network services graph, which is a graph data structure representing vulnerable services running between hosts, augmented with a labeling function that highlights services that are likely to have been compromised. We formulate the decision-making problem as a defense-based zero-sum matrix game that the engine analyzes to select appropriate response actions by solving for saddle-point strategies. Given the response engine's knowledge of the network and the location of sensitive components (e.g., database servers), its goal is to keep the suspicious actors as far away from the sensitive components as possible. The engine is not guaranteed to neutralize threats, if any, but can provide network administrators with enough time to analyze suspicious movement and take appropriate neutralization actions. The decision engine will make use of the monitoring information to decide which nodes' disconnection from the network would slow down the attacker's movements and allow administrators to take neutralizing actions.

An important feature of our approach is that, unlike most IRSs, it makes very few pre-game assumptions about the attacker's strategy; we only place a bound on the number of actions that an attacker can make within a time period, thus allowing us to model the problem as a zero-sum game. By not assuming an attacker model beforehand, our engine can avoid cases in which the attacker deviates from the model and uses its knowledge to trick the engine and cancel the effectiveness of its actions. We show that our engine is effectively able to increase the number of attack steps needed by an attacker to compromise a sensitive part of the network by at least 50%. Additionally, in most cases, the engine was able to deny the attacker access to the sensitive nodes for the entire period of the simulation.

The rest of this paper is organized as follows. We describe the motivation behind our work in Section 2. We then present an overview of our approach and threat model in Section 3. Section 4 formally presents the response engine and the algorithms we use. We discuss implementation and results in Section 5. We review past literature in Section 6, which is followed by presentation of challenges and future directions in Section 7. We conclude in Section 8.

2 Motivation

The life cycle of an APT consists of the following steps [7, 2, 11]. The first is intelligence gathering and reconnaissance, which is followed by the establishment of an entry point into the target system. Subsequently, the attacker establishes a connection to one or more command and control (C&C) servers, and uses these connections to control the remainder of the operation. Following C&C establishment is *lateral movement*, wherein the attacker gathers user credential and authentication information and moves laterally in the network in order to reach a designated target. The last step includes performance of specific actions on the targets, such as data exfiltration or even physical damage [13].

Lateral movement allows attackers to achieve persistence in the target network and gain higher privileges by using different tools and techniques [2]. In a number of recent security breaches, the examination of network logs has shown that attackers were able to persist and move laterally in the victim network, staying undetected for long periods of time. For example, in the attack against the Saudi Arabian Oil Company, the attackers were able to spread the malware to infect 30,000 personal machines on the company's network through the use of available file-sharing services [8]. In the Ukraine power grid breach, attackers used stolen credentials to move laterally through the network and gain access to Supervisory Control and Data Acquisition (SCADA) dispatch workstations and servers. The attackers had enough privileges to cause more damage to the public power grid infrastructure [14]. Furthermore, through the use of USB sticks and exploitation of zero-day vulnerabilities in the Windows operating system, the Stuxnet malware was able to move between different workstations in an Iranian nuclear facility until it reached the target centrifuge controllers [13].

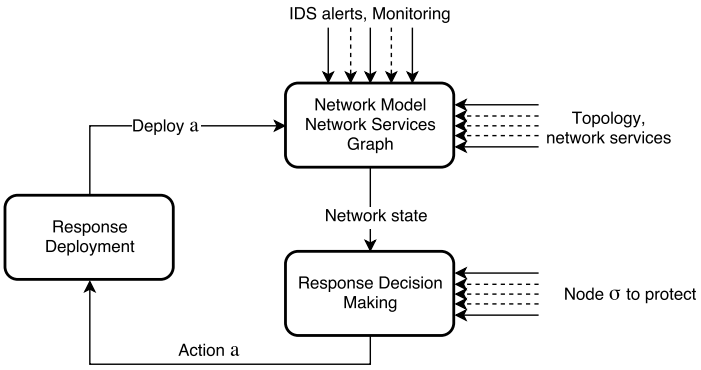


Fig. 1: Our defender model. The defense module uses IDS alerts and monitoring data along with observed attacker steps to build a network model. Trying to protect a sensitive node σ, it builds a zero-sum game and solves for the saddle-point strategies in order to select an appropriate response action a. The *Response Deployment* module is then responsible for the implementation of a in the network.

Early detection of lateral movement is an essential step towards thwarting APTs. However, without timely response, attackers can use the time gap between detection and administrator response to exfiltrate large amounts of data or inflict severe damage to the victim's infrastructure. It took network administrators two weeks to effectively neutralize threats and restore full operation to the Saudi Arabian Oil Company's network [8]. Furthermore, attackers attempt to hide their lateral movement through the use of legal network services such as file sharing (mainly Windows SMB), remote desktop tools, secure shell (SSH) and administrator utilities (such as the Windows Management Instrumentation) [2]. This stealthy approach makes it harder for network administrators to decide whether the traffic they are observing is malicious lateral movement or benign user or administrative traffic.

In this work, we present a game-theoretic approach for autonomous network response to potentially malicious lateral movement. The response actions taken by our engine aim to protect sensitive network infrastructure by keeping the attacker away from it for as long as possible, thus giving network administrators enough time to assess the observed alerts and take effective corrective actions to neutralize the threats.

3 Overview

We assume, in our framework, the presence of network level IDSs (such as Snort [20] and Bro [1]) that can provide the response engine with the necessary monitoring information. The response engine maintains the state of the network in the form of a network services graph, a graph data structure that represents the active services between nodes in the network. It then uses IDS

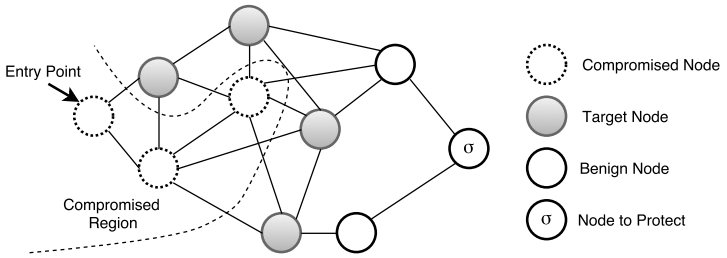


Fig. 2: An illustration of our game model. The attacker has compromised 3 nodes in the network, and has four potential targets to compromise next. The defender, seeing the three compromised nodes, has to decide where the attacker is going to move next and disconnect services from the node, thus slowing down the attack.

information to define a labeling function over the graph that marks suspicious nodes and communications used for a possible compromise. Using the labels, the engine observes chains of communications between likely compromised nodes. Such chains are considered suspicious and require the engine to take immediate response actions. The engine considers all suspicious chains as hostile; its goal is to prevent any attackers from reaching specified sensitive nodes in the network, typically database servers or physical controllers.

From the observed states, the response engine can identify compromised nodes and possible target nodes for the attacker. It will take response actions that disconnect services from target nodes so that it prevents the attacker from reaching the sensitive node. This step can provide the network administrators with enough time to assess the IDS alerts and take appropriate actions to neutralize any threats. Figures 1 and 2 illustrate high-level diagrams of our response engine and a sample observed network state with 10 nodes, respectively.

Our threat model allows for the presence of a sophisticated attacker that has already established an entry point in an enterprise network, typically using spear phishing and social engineering, and aims to move laterally deeper into the network. Starting from a compromised node, the attacker identifies a set of possible target nodes for the next move. We assume that the attacker compromises one node at a time in order to avoid detection. We argue that this assumption is reasonable since attackers typically want to use legitimate administrator tools to hide their lateral movement activities [24]. Therefore, unlike computer worms that propagate widely and rapidly [26], lateral movement tends to be targeted, slow and careful. We will explore more sophisticated types of attackers with multi-move abilities in our future work.

Figure 2 illustrates an example network services graph with ten nodes, where an attacker has established a point of entry and already compromised three nodes. We highlight the target nodes that the attacker can choose to compromise next. We assume no prior knowledge of the strategy by which the attacker will choose the next node to compromise. Building our response engine on the assumption of like-minded attackers would lead to a false sense of security, since

attackers with different motives would be able to overcome the responses of our engine, or possibly use them to their own advantage. Therefore, we formulate a defense-based game that attempts to protect a sensitive node in the network, regardless of the goals that the attacker is trying to achieve.

4 The response engine

In this section, we formally introduce our response decision-making problem and its formulation as a zero-sum game. We provide formal definitions for the network state, attack and response actions, and attack and response strategies, and then present how we build and solve the matrix game. We formulate the response engine's decision-making process as a complete information zero-sum game, in which the players are the engine and a potentially malicious attacker. We assume that both players take actions simultaneously, i.e., no player observes the action of the other before making its own move. In what follows, without loss of generality, we use the term attacker to refer to a suspicious chain of lateral movement communications. The response engine treats all communication chains as malicious and takes response actions accordingly. We use the terms *defender* and *response engine* interchangeably.

4.1 Definitions

Definition 1 (Network services graph). *A network services graph (NSG) is an undirected graph $G =< V, E >$ where V is the set of physical or logical nodes (workstations, printers, virtual machines, etc.) in the network and $E = V \times V$ is a set of edges.*

An edge $e = (v_1, v_2) \in E$ represents the existence of an active network service, such as file sharing, SSH, or remote desktop connectivity, between nodes v_1 and v_2 in the network.
For any $v \in V$, we define a $\texttt{neighborhood}(v)$ as the set

$$\texttt{neighborhood}(v) = \{u \in V \mid \exists (u, v) \in E\} \tag{1}$$

Definition 2 (Alert labeling function). *Given an NSG $G =< V, E >$, we define an Alert Labeling Function (ALF) as a labeling function ℓ over the nodes V and edges E of G such that*

$$\textit{For } v \in V,\ \ell(v) = \begin{cases} \texttt{True} & \textit{iff} \quad v \textit{ is deemed compromised,} \\ \texttt{False} & \textit{otherwise.} \end{cases} \tag{2}$$

$$\textit{For } e = (u, v) \in E,\ \ell(e) = \begin{cases} \texttt{True} & \textit{iff} \quad \ell(u) = \texttt{True} \wedge \ell(v) = \texttt{True}, \\ \texttt{False} & \textit{otherwise.} \end{cases} \tag{3}$$

A *suspicious chain* is then a sequence of nodes $\{v_1, v_2, \ldots, v_k\}$ such that

$$\begin{cases} v_1, v_2, \ldots, v_k \in V, \\ (v_i, v_{i+1}) \in E \quad \forall i \in \{1, \ldots, k-1\}, \text{ and} \\ \ell(v_i) = \texttt{True} \quad \forall i \in \{1, \ldots, k\} \end{cases}$$

We assume that an ALF is obtained from monitoring information provided by IDSs such as Snort [20] and Bro [1]. A suspicious chain can be either a malicious attacker moving laterally in the network, or a benign legal administrative task. The goal of our response engine is to slow the spread of the chain and keep it away from the sensitive infrastructure of the network, thus giving network administrators enough time to assess whether the chain is suspicious or not, and take appropriate corrective actions when needed.

Definition 3 (Network state). *We define the state of the network as a tuple* $s = (G_s =< V_s, E_s >, \ell_s)$ *where* G_s *is an NSG and* ℓ_s *is its corresponding ALF. We use* $\mathcal{S}$ *to refer to the set of all possible network states.*

For a given network state s, we define the set of vulnerable nodes $\mathcal{V}_s$ as

$$\mathcal{V}_s = \left\{ u \mid \left(u \in \bigcup_{v \in V_s \wedge \ell_s(v) = \texttt{True}} \texttt{neighborhood}(v) \right) \wedge \ell_s(u) = \texttt{False} \right\} \quad (4)$$

Definition 4 (Attack action). *Given a network state* $s \in S$*, an attack action* a_e *is a function over the ALFs, in which a player uses the service provided by edge* $e = (v, v')$ *such that* $\ell_s(v) =$ *True and* $v' \in \mathcal{V}_s$*, in order to compromise node* v'*. Formally we write*

$$a_e(\ell_s) = \ell' \text{ such that } \ell'(v') = \textit{True} \wedge \ell'(e) = \textit{True} \quad (5)$$

For a network state s, the set of possible attack actions $\mathcal{A}_s$ is defined as

$$\mathcal{A}_s = \{a_e \mid e = (u, v) \in E_s \wedge \ell_s(u) = \texttt{True} \wedge v \in \mathcal{V}_s\} \quad (6)$$

Definition 5 (Response action). *Given a network state* s*, a response action* d_v *is a function over the NSG edges, in which a player selects a node* $v \in \mathcal{V}_s$*, and disconnects available services on all edges* $e = (u, v) \in E_s$ *such that* $\ell_s(u) =$ *True. Formally, we write*

$$d_v(E_s) = E' \text{ such that } E' = E_s \setminus \{(u, v) \in E_s \mid \ell_s(u) = \textit{True}\} \quad (7)$$

For a network state s, we define the set of all possible response actions $\mathcal{D}_s$ as

$$\mathcal{D}_s = \{d_v \mid v \in \mathcal{V}_s\} \quad (8)$$

Definition 6 (Response strategy). *Given a network state* s *with a set of response actions* $\mathcal{D}_s$*, a strategy* $\mathbf{p}_r : \mathcal{D}_s \longrightarrow [0, 1]^{|\mathcal{D}_s|}$ *where* $\sum_{d_v \in \mathcal{D}_s} \mathbf{p}_r(d_v) = 1$ *is a probability distribution over the space of available response actions.*

1: **for each** time epoch $t_0 < t_1 < t_2 < \ldots$ **do**
2: (1) Obtain network state $s = (G_s, \ell_s)$.
3: (2) Compute the sets of possible attack and response actions $\mathcal{A}_s$ and $\mathcal{D}_s$
4: (3) Compute the payoff matrix $M_s = \texttt{BUILD\_GAME}(\mathcal{A}_s, \mathcal{D}_s, \texttt{threshold}, \sigma)$
5: (4) Compute the equilibrium response strategy $\hat{\mathbf{p}}_r$
6: (6) Sample response action $d_v \in \mathcal{D}_s$ from $\hat{\mathbf{p}}_r$
7: **end for**

Fig. 3: The steps taken by our response engine at each time epoch. The engine first obtains the state of the network from the available monitors, and uses it to compute the sets of possible attack and response actions $\mathcal{A}_s$ and $\mathcal{D}_s$. It then builds the zero-sum game matrix M_s using Algorithm 1, and solves for the equilibrium response strategy $\hat{p}_s$. It finally samples a response action d_v from $\hat{p}_s$ that it deploys in the network.

A response strategy $\mathbf{p}_r$ is a *pure response strategy* iff

$$\exists\ d_v \in \mathcal{D}_s \text{ such that } \mathbf{p}_r(d_v) = 1 \wedge (\forall d_{v'} \neq d_v,\ \mathbf{p}_r(d_{v'}) = 0) \tag{9}$$

A response strategy that is not pure is a *mixed response strategy.* Given a network state s, after solving a zero sum game, the response engine samples its response action according to the computed response strategy.

Definition 7 (Attack strategy). *Given a network state s and a set of attack actions $\mathcal{A}_s$, an attack strategy $\mathbf{p}_a : \mathcal{A}_s \longrightarrow [0,1]^{|\mathcal{A}|}$ where $\sum_{a_e \in \mathcal{A}_s} \mathbf{p}_a(a_e) = 1$ is a probability distribution over the space of available attack actions $\mathcal{A}_s$.*

Definition 8 (Network next state). *Given a network state s, a response action $d_v \in \mathcal{D}_s$ for $v \in V_s$, and an attack action $a_e \in \mathcal{A}_s$ for $e = (u, w) \in E_s$, using Equations (5) and (7), we define the network next state (nns) as a function $\mathcal{S} \times \mathcal{D}_s \times \mathcal{A}_s \longrightarrow \mathcal{S}$ where*

$$nns(s, d_v, a_e) = s' \text{ where } \begin{cases} (G_{s'} =< V_s, d_v(E_s) >, \ell_s) & \text{iff } v = w, \\ (G_{s'} =< V_s, d_v(E_s) >, a_e(\ell_s)) & \text{otherwise} \end{cases} \tag{10}$$

4.2 Formulation as a zero-sum game

The goal of our response engine is to keep an attacker, if any, as far away from a network's sensitive node (database server, SCADA controller, etc.) as possible. In the following, we assume that the engine is configured to keep the attacker at least `threshold` nodes away from a database server σ containing sensitive company data. The choices of `threshold` and σ are determined by the network administrators prior to the launch of the response engine.

Figure 3 shows the steps taken by our response engine at each time epoch $t_0 < t_1 < t_2 < \ldots < t$. In every step, the defender constructs a zero-sum defense-based matrix game and solves it for the saddle-point response strategy

from which it samples an action to deploy. Assume that in a network state s, the response engine chooses to deploy action $d_v \in \mathcal{D}_s$ for $v \in V_s$, and the attacker chooses to deploy action $a_e \in \mathcal{A}_s$ for $e = (u, w) \in E_s$. In other words, the defender disconnects services from node v in the network while the attacker compromises node w starting from the already compromised node u. If $v = w$, then the attacker's efforts were in vain and the response engine was able to guess correctly where the attacker would move next. However, when $v \neq w$, the attacker would have successfully compromised the node w. Note that this is not necessarily a loss, since by disconnecting services from certain nodes on the path, the response engine might be redirecting the attacker away from the target server σ. Furthermore, by carefully selecting nodes to disconnect, the engine can redirect the attacker into parts of the network where the attacker can no longer reach the target server σ, and thus cannot win the game. The attacker wins the game when it is able to reach a node within one hop of target server σ. The game ends when (1) the attacker reaches σ; (2) either player runs out of moves to play; or (3) the attacker can no longer reach σ.

Let $\texttt{sp}(u, \sigma)$ be the length of the shortest path (in number of edges) in G_s from node u to the target server σ. We define the payoffs for the defender in terms of how far the compromised nodes are from the target server σ. A positive payoff indicates that the attacker is more than `threshold` edges away from σ. A negative payoff indicates that the attacker is getting closer to σ, an undesirable situation for our engine. Therefore, we define the payoff for the defender when the attacker compromises node w as $\texttt{sp}(w, \sigma) - \texttt{threshold}$. If $\texttt{sp}(w, \sigma) > \texttt{threshold}$ then the attacker is at least $\texttt{sp}(w, \sigma) - \texttt{threshold}$ edges away from the defender's predefined dangerous zone. Otherwise, attacker is $\texttt{threshold} - \texttt{sp}(w, \sigma)$ edges into the defender's dangerous zone. Moreover, when the defender disconnects a node w that the attacker wanted to compromise, two cases might arise. First, if $\texttt{sp}(w, \sigma) = \infty$, i.e., w cannot reach σ, then it is desirable for the defender to lead the attacker into w, and thus the engine assigns d_w a payoff of 0 so that it wouldn't consider disconnecting w. Otherwise, when $\texttt{sp}(w, \sigma) < \infty$, by disconnecting the services of w, the defender would have canceled the effect of the attacker's action, and thus considers it a win with payoff $\texttt{sp}(w, \sigma) < \infty$.

Algorithm 1 illustrates how our response engine builds the zero-sum matrix game. For each network state s, the algorithm takes as input the set of response actions $\mathcal{D}_s$, the set of attack actions $\mathcal{A}_s$, the defender's `threshold`, and the target server to protect σ. The algorithm then proceeds by iterating over all possible combinations of attack and response actions and computes the defender's payoffs according to Equation (11). It then returns the computed game payoff matrix M_s with dimensions $|\mathcal{D}_s| \times |\mathcal{A}_s|$.

Formally, for player actions $d_v \in \mathcal{D}_s$ and $a_e \in \mathcal{A}_s$ where $v \in V_s$ and $e = (u, w) \in E_s$, we define the response engine's utility as

$$u_d(d_v, a_e) = \begin{cases} 0 & \text{iff } v = w \wedge \texttt{sp}(w, \sigma) = \infty \\ \texttt{sp}(w, \sigma) & \text{iff } v = w \wedge \texttt{sp}(w, \sigma) < \infty \\ \texttt{sp}(w, \sigma) - \texttt{threshold} & \text{iff } v \neq w \end{cases} \tag{11}$$

Algorithm 1 Algorithm $M_s = \texttt{BUILD\_GAME}\ (\mathcal{D}_s, \mathcal{A}_s, \texttt{threshold}, \sigma)$

Inputs: $\mathcal{D}_s, \mathcal{A}_s$, `threshold`, σ
Outputs: Zero-sum game payoff matrix M_s
for each response action $d_v \in \mathcal{D}_s$ **do**
  for each attack action $a_e \in \mathcal{A}_s$ **do**
    let $e \leftarrow (u, w)$
    if $v = w$ **then**
      if $\texttt{sp}(w, \sigma) = \infty$ **then**
        $M_s(v, w) \leftarrow 0$
      else
        $M_s(v, w) \leftarrow \texttt{sp}(w, \sigma)$
      end if
    else
      $M_s(v, w) \leftarrow \texttt{sp}(w, \sigma) - \texttt{threshold}$
    end if
  end for
end for

Since the game is zero-sum, the utility of the attacker is $u_a(a_e, d_v) = -u_d(d_v, a_e)$.

For a response strategy $\mathbf{p}_r$ over $\mathcal{D}_s$ and an attack strategy $\mathbf{p}_a$ over $\mathcal{A}_s$, the response engine's expected utility is defined as

$$U_d(\mathbf{p}_r, \mathbf{p}_a) = \sum_{d_v \in \mathcal{D}_s} \sum_{a_e \in \mathcal{A}_s} \mathbf{p}_r(d_v) u_d(d_v, a_e) \mathbf{p}_a(a_e) \tag{12}$$

Similarly, the attacker's expected payoff is $U_a(\mathbf{p}_a, \mathbf{p}_r) = -U_d(\mathbf{p}_r, \mathbf{p}_a)$.

In step 4 of Figure 3, the response engine computes the saddle-point response strategy $\hat{\mathbf{p}}_r$ from which it samples the response action to deploy. $\hat{\mathbf{p}}_r$ is the best response strategy that the engine could adopt for the worst-case attacker. Formally, for saddle-point strategies $\hat{\mathbf{p}}_r$ and $\hat{\mathbf{p}}_a$,

$$\begin{aligned} &U_d(\hat{\mathbf{p}}_r, \hat{\mathbf{p}}_a) \geq U_d(\mathbf{p}_r, \hat{\mathbf{p}}_a) \text{ for all } \mathbf{p}_r, \text{ and} \\ &U_a(\hat{\mathbf{p}}_a, \hat{\mathbf{p}}_r) \leq U_a(\mathbf{p}_a, \hat{\mathbf{p}}_r) \text{ for all } \mathbf{p}_a \end{aligned} \tag{13}$$

Finally, the engine chooses an action $d_v \in \mathcal{D}_s$ according to the distribution $\hat{\mathbf{p}}_r$ and deploys it in the network. In this paper, we assume that response actions are deployed instantaneously and successfully at all times; response action deployment challenges are beyond the scope of this paper.

5 Implementation and results

We implemented a custom Python simulator in order to evaluate the performance of our proposed response engine. We use Python iGraph [9] to represent NSGs, and implement ALFs as features on the graphs' vertices. Since the payoffs for the response engine's actions are highly dependent on the structure of the NSG, we use three different graph topology generation algorithms to generate the

initial graphs. The Waxman [25] and Albert-Barabási [3] algorithms are widely used to model interconnected networks, especially for the evaluation of different routing approaches. In addition, we generate random geometric graphs, as they are widely used for modeling social networks as well as studying the spread of epidemics and computer worms [17, 19]. Because of the lack of publicly available data sets capturing lateral movement, we assume that the Waxman and Albert-Barabási models provide us with an appropriate representation of the structural characteristics of interconnected networks.

We use the geometric graph models in order to evaluate the performance of our engine in highly connected networks. We pick the initial attacker point of entry in the graph ω and the location of the database server σ such that $sp(\omega, \sigma) = d$, where d is the diameter of the computed graph. This is a reasonable assumption, since in APTs, attackers usually gain initial access to the target network by targeting employees with limited technical knowledge (such as customer service representatives) through social engineering campaigns, and then escalate their privileges while moving laterally in the network.

We implement our response engine as a direct translation of Figure 3 and Algorithm 1, and we use the Gambit [16] Python game theory API in order to solve for the saddle-point strategies at each step of the simulation. We use the NumPy [12] Python API to sample response and attack actions from the computed saddle-point distributions. As stated earlier, we assume that attack and response actions are instantaneous and always successful, and thus implement the actions and their effects on the network as described in the network next-state function in Equation (10).

We evaluate the performance of our response engine by computing the average percentage increase in the number of attack steps (i.e., compromises) needed by an adversary to reach the target server σ. We compute the average increase with respect to the shortest path that the attacker could have adopted in the absence of the response engine. Formally, let k be the number of attack steps needed to reach σ and d be the diameter of the NSG; then, the percentage increase in attack steps is $\frac{k-d}{d} \times 100$. If the attacker is unable to reach the target server, we set the number of attack steps k to the maximum allowed number of rounds of play in the simulation, which is 40 in our simulations.

In addition, we report on the average attacker distance from the server σ as well as the minimum distance that the attacker was able to reach. As discussed earlier, we measure the distance in terms of the number of attack steps needed to compromise the server. A minimum distance of 1 means that the attacker was able to successfully reach σ. We also report and compare the average achieved payoff for the defender while playing the game. We ran our simulations on a Macbook Pro laptop running OSX El Capitan, with 2.2 GHz Intel Core i7 processors and 16 GB of RAM. We start by describing our results for various defender `threshold` values for an NSG with 100 nodes, and then fix the `threshold` value and vary the number of nodes in the NSG. Finally, we report on performance metrics in terms of the time needed to perform the computation for various NSG sizes.

Table 1: Characteristics of generated NSGs (averages)

NSG Generator	$\|V\|$	$\|E\|$	Diameter	Max Degree
Barabási	100	294	4	50.2
Waxman	100	336.6	4.9	13.7
Geometric	100	1059.8	5.2	34.5

5.1 Evaluation of `threshold` values

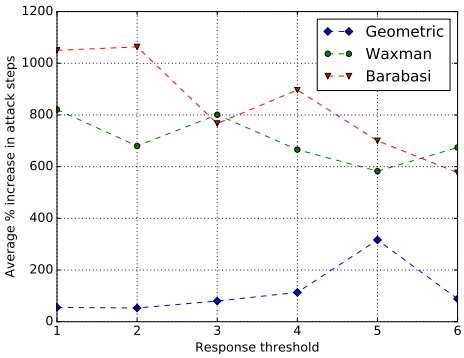
We start by evaluating the performance of our response engine for various values of the threshold above which we would like to keep the attacker away from the sensitive node σ. We used each graph generation algorithm to generate 10 random NSGs, simulated the game for `threshold` $\in \{1, 2, 3, 4, 5, 6\}$, and then computed the average values of the metrics over the ten runs.

Table 1 shows the structural characteristics in terms of the number of vertices, average number of edges, diameter, and maximum degree of the graphs generated by each algorithm. All of the graphs we generated are connected, with the geometric graphs showing the largest levels of edge connectivity, giving attackers more space to move in the network. The Waxman and Barabási generators have lower levels of edge connectivity, making them more representative of network services topologies than the geometric graphs are.

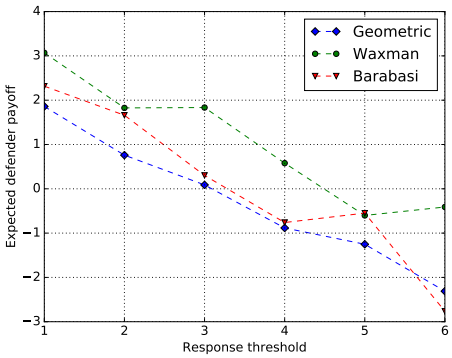
Figure 4a shows the average percentage increase in attacker steps needed to reach the target (or reach the simulation limit) for the various values of `threshold`. The results show that in all cases, our engine was able to increase the number of steps needed by the attacker by at least 50%. Considering only the Waxman and Barabási graphs, the engine was able to increase the number of steps needed by the attacker by at least 600%. This is a promising result that shows the effectiveness of our engine, especially in enterprise networks. Further, the results show that smaller values for `threshold` achieve a greater average increase in attacker steps. This is further confirmed by the average defender payoff curves shown in Figure 4b, in which smaller values of `threshold` achieve greater payoffs. In fact, this result is a direct implication of our definition of the payoff matrix values in Equation (11). The smaller the values of `threshold`, the more the engine has room to take actions that have a high payoff, and the more effective its strategies are in keeping the attacker away from the server.

Figures 4c and 4d show the average distance between the attacker and the server, and the minimum distance reached by the attacker, respectively. For the Waxman and Barabási graphs, the results show that our engine keeps the attacker, on average, at a distance close to the graph's diameter, thus keeping the attacker from penetrating deeper into the network. For both types of graphs, Figure 4d confirms that the attacker was unable to reach the target server (average minimum distance ≥ 1).

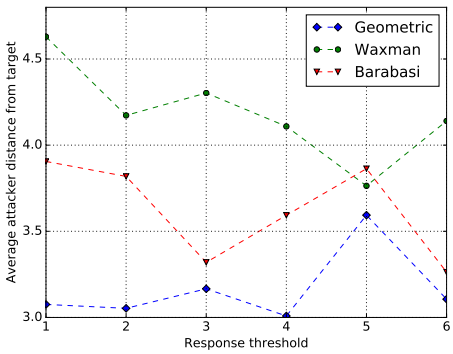
In the case of the geometric graphs, Figure 4d shows that the attacker was almost always able to reach the target server. We attribute this attacker success to the high edge connectivity in the geometric graphs. Although our engine is able to delay attackers, because of the high connectivity of the graph, they may



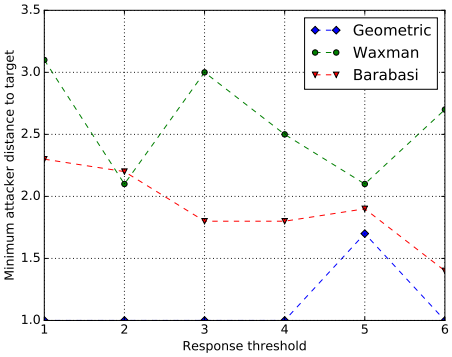
(a) Average % increase in attack steps



(b) Average defender payoff



(c) Average attacker distance from σ



(d) Attacker's minimum distance from σ

Fig. 4: Performance evaluation of our response engine with varying threshold values. Figure 4a shows that our engine was able to increase the number of compromises needed by the attacker by at least 55%. Figure 4b illustrates that the zero-sum game's payoff for the defender decreases almost linearly as the `threshold` increases. Figure 4c shows that the average attacker's distance from σ is very close to the NSG's diameter, while Figure 4d shows that, with the exception of the geometric NSG, our engine was able to keep that attacker from reaching the target data server σ. It was able, however, in the geometric NSG case, to increase the number of compromises needed to reach σ by at least 55%.

find alternative ways to reach the server. Nevertheless, our response engine was always able to cause at least a 50% increase in the number of attack steps needed to reach the server.

In summary, the results show that our response engine is able to effectively delay, and on average prevent, an attacker that is moving laterally in the network from reaching the target database server. It was effectively able to increase the number of attack steps needed by the adversary by at least 600% for the graphs that are representative of real-world network topologies. In addition, even when the graphs were highly connected, our engine was still able to increase the attacker's required amount of attack steps by at least 50%.

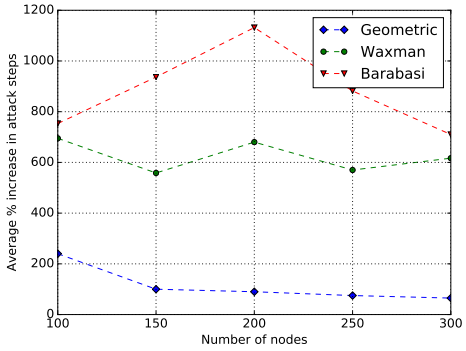
5.2 Scalability

Next, we measured the scalability of our response engine as the network grew in size. We varied the number of nodes in the network from 100 to 300 in steps of 50 and measured the average percentage increase in attack steps as well as the attacker's average distance from the target σ. Figure 5 shows our results for averages measured over five random NSGs generated by each of the NSG generation algorithms. We set the defender's `threshold` values to those that achieved a maximum average increase in attack steps as shown in Figure 4a, which are 5 for geometric NSGs, 2 for Barabási NSGs, and 3 for Waxman NSGs.

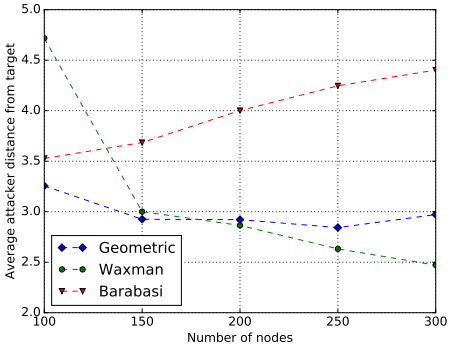
As shown in Figure 5a, our response engine can scale well as the size of the network increases, providing average percentage increases in attack steps between 550% and 700% for Waxman NSGs, 750% and 1150% for Barabási NSGs, and 50% and 220% for geometric NSGs. These results show that as the number of nodes, and thus the number of connection edges, increases in the network, our engine is able to maintain high-performance levels and delay possible attackers, even when they have more room to evade the engine's responses and move laterally in the network. This is further confirmed by the results shown in Figures 5b and 5c. For the Waxman and Barabási NSGs, the response engine is always capable of keeping the attacker at an average distance from the target server equal to the diameter of the graph. For the geometric NSGs, the attacker is always capable of getting close to and reaching the target server, regardless of the diameter of the graph. Our engine, however, is always capable of increasing the number of attack steps required by at least 50%, even for larger networks.

5.3 Computational performance

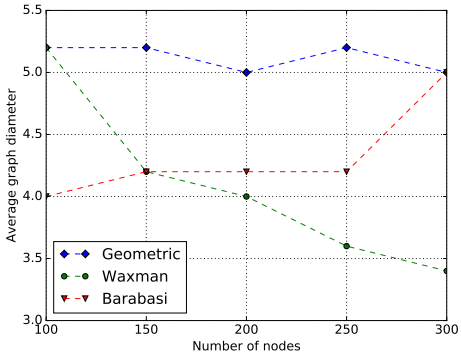
Finally, we evaluated the computational performance of our game engine as the scale of the network increased from 100 to 300 nodes. We used the same values for `threshold` as in the previous subsection, and measured the average time to solve for the saddle-point strategies as well as the average size of the matrix game generated during the simulation. Since all of the payoff matrices we generated are square, we report on the number of rows in the matrix games. The rows correspond to the number of available attack or response actions for the players (i.e., for a state s, we report on $|\mathcal{A}_s| = |\mathcal{D}_s|$). Our engine makes use



(a) Average % increase in attack steps

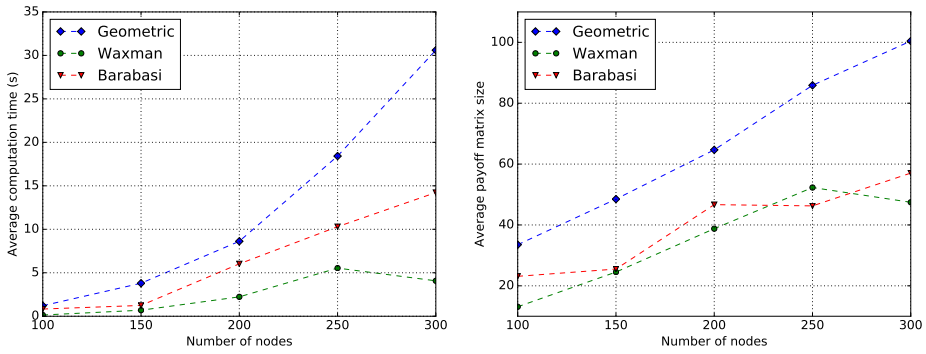


(b) Average attacker distance from σ



(c) Average graph diameter

Fig. 5: Performance evaluation of our response engine with increasing number of nodes in the network. Figure 5a shows that our engine maintains high levels of performance even when the network grows larger. The engine is also capable of keeping the attacker at an average distance close to the graph's diameter in the cases of the Waxman and Barabási NSGs, as shown in Figures 5b and 5c.



(a) Average time (s) to solve matrix game (b) Average size of the matrix game

Fig. 6: Computational performance evaluation of the engine for larger networks. Our response engine scales well with the increase in the size of the network.

of the `ExternalLogitSolver` solver from the Gambit software framework [16] to solve for the saddle-point strategies at each step of the simulation. In computing our metrics, we averaged the computation time and matrix size over 10 random graphs from each algorithm, and we limited the number of steps in the simulation (i.e., the number of game turns) to 10.

Figure 6b shows that for all NSG-generation algorithms, the size of the payoff matrices for the generated zero-sum game increases almost linearly with the increase in the size of the nodes in the network. In other words, the average number of available actions for each player increases linearly with the size of the network. Consequently, Figure 6a shows that the computational time needed to obtain the saddle-point strategies scales very efficiently with the increase in the size of the network; the engine was able to solve 50×50 matrix games in 15 seconds, on the average. The short time is a promising result compared to the time needed by an administrator to analyze the observed alerts and deploy strategic response actions.

In summary, our results clearly show the merits of our game engine in slowing down the advance of an attacker that is moving laterally within an enterprise network, and its ability to protect a sensitive database server effectively from compromise. For all of the NSG-generation algorithms, our engine was able to increase the number of attack steps needed by an attacker to reach the sensitive server by at least 50%, with the value increasing to 600% for the Waxman and Barabási NSG-generation algorithms. The results also show that our engine is able to maintain proper performance as networks grow in size. Further, the computational resources required for obtaining the saddle-point strategies increased linearly with the number of the nodes in the network.

6 Related work

Several researchers have tackled the problem of selecting cyber actions as a response to intrusions. The space can be divided into three parts; automated response through rule-based methods, cost-sensitive methods, and security games.

In rule-based intrusion response, each kind of intrusion alert is tagged with a suitable response. The static nature of rule-based intrusion response makes it predictable and limits its ability to adapt to different attacker strategies. Researchers have extended rule-based intrusion response systems to become cost-sensitive; cost models range from manual assessment of costs to use of dependency graphs on the system components to compute a response action's cost. In all of those cases, the process of selecting a response minimizes the cost of response actions over a set of predefined actions that are considered suitable for tackling a perceived threat. Stakhanova surveyed this class of systems in [26]. While cost-sensitive intrusion response systems minimize the cost of responding, they are still predictable by attackers, and a large effort is required in order to construct the cost models.

Bloem *et. al.* [6] tackled the problem of intrusion response as a resource allocation problem. Their goal was to manage the administrator's time, a critical and limited resource, by alternating between the administrator and an imperfect automated intrusion response system. The problem is modeled as a nonzero-sum game between automated responses and administrator responses, in which an attacker gain (utility) function is required. Obtaining such functions, however, is hard in practice, as attacker incentives are not known. The problem of finding attacker-centric metrics was tackled by ADAPT [22]. The ADAPT developers attempted to find a taxonomy of attack metrics that require knowledge of the cost of an attack and the benefit from the attack. ADAPT has created a framework for computing the metrics needed to set up games; however, assigning values to the parameters is still more of an art than a science.

Use of security games improved the state of IRSs, as they enabled modeling of the interaction between the attacker and defender, are less predictable, and can learn from previous attacker behavior [15, 5]. In [4], the authors model the security game as a two-player game between an attacker and a defender; the attacker has two actions (to attack or not attack), and the defender has two actions (to monitor or not monitor). The authors consider the interaction as a repeated game and find an equilibrium strategy. Nguyen *et. al.* [18] used fictitious play to address the issue of hidden payoff matrices. While this game setup is important on a high level and can be useful as a design guideline for IDSs, it does not help in low-level online response selection during a cyber attack.

To address the issue of high level abstraction in network security games, Zonouz [28] designed the Response and Recovery Engine (RRE), an online response engine modeled as a Stackelberg game between an attacker and a defender. Similar to work by Zhu and Başar [27], the authors model the system with an attack response tree (ART); the tree is then used to construct a competitive Markov decision process to find an optimal response. The state of the decision process is a vector of the probabilities of compromise of all the compo-

nents in the system. The authors compute the minimax equilibrium to find an optimal response. The strategy is evaluated for both finite and infinite horizons. Scalability issues are tackled using finite lookahead. The game, however, has several limitations: (1) the model is sensitive to the assigned costs; (2) the model required *a priori* information on attacks and monitoring (conditional probabilities) which is not available; and (3) the system uses a hard-to-design ART to construct the game.

7 Discussion and future work

The goals of our response engine are to provide networked systems with the ability to maintain acceptable levels of operation in the presence of potentially malicious actors in the network, and to give administrators enough time to analyze security alerts and neutralize any threats, if present. Our results show that the engine is able to delay, and often prevent, an attacker from reaching a sensitive database server in an enterprise network. However, the response actions that our engine deploys can have negative impacts on the system's provided services and overall performance. For example, disconnecting certain nodes as part of our engine's response to an attacker can compromise other nodes' ability to reach the database service. This can have severe impacts on the system's resiliency, especially if it is part of a service provider's infrastructure. In the future, we plan to augment our engine with response action cost metrics that reflect their impact on the network's performance and resiliency. We plan to add support for a resiliency budget that the engine should always meet when making response action decisions. In addition, we will investigate deployment challenges for the response actions. We envision that with the adoption of Software Defined Networks (SDNs), the deployment of such actions will become easier. Our engine can be implemented as part of the SDN controller, and can make use of an SDN control protocols to deploy its response actions.

In the context of APTs, attackers are often well-skilled, stealthy, and highly adaptive actors that can adapt to the changes in the network, including the response actions deployed by our engine. We will investigate more sophisticated models of attackers, specifically ones that can compromise more than one node in each attack step, and can adapt in response to our engine's deployed actions. In addition, knowledge of the attacker's strategies and goals would provide our response engine with the ability to make more informed strategic decisions about which response actions to deploy. Therefore, we plan to investigate online learning techniques that our engine can employ in order to predict, with high accuracy, an attacker's strategies and goals. However, the main challenge that we face in our framework's design and implementation is the lack of publicly available datasets that contain traces of attackers' lateral movements in large-scale enterprise networks. In addition to simulations, we will investigate alternative methods with which we can evaluate our response engine and the learning techniques that we devise. Such methods can include implementation in a real-world, large-scale testbed.

8 Conclusion

Detection of and timely response to network intrusions go hand-in-hand when secure and resilient systems are being built. Without timely response, IDSs are of little value in the face of APTs; the time delay between the sounding of IDS alarms and the manual response by network administrators allows attackers to move freely in the network.We have presented an efficient and scalable game-theoretic response engine that responds to an attacker's lateral movement in an enterprise network, and effectively protects a sensitive network node from compromise. Our response engine observes the network state as a network services graph that captures the different services running between the nodes in the network, augmented with a labeling function that captures the IDS alerts concerning suspicious lateral movements. It then selects an appropriate response action by solving for the saddle-point strategies of a defense-based zero-sum game, in which payoffs correspond to the differences between the shortest path from the attacker to a sensitive target node, and an acceptable engine safety distance threshold. We have implemented our response engine in a custom simulator and evaluated it for three different network graph generation algorithms. The results have shown that our engine is able to effectively delay, and often stop, an attacker from reaching a sensitive node in the network. The engine scales well with the size of the network, maintaining proper operation and efficiently managing computational resources. Our results show that the response engine constitutes a significant first step towards building secure and resilient systems that can detect, respond to, and eventually recover from malicious actors.

Acknowledgement

This work was supported in part by the Office of Naval Research (ONR) MURI grant N00014-16-1-2710. The authors would like to thank Jenny Applequist for her editorial comments.

References

1. The Bro network security monitor. `https://www.bro.org/`. 2014.
2. Lateral movement: How do threat actors move deeper into your network. Technical report, Trend Micro, 2003.
3. R. Albert and A. Barabási. Statistical mechanics of complex networks. *Rev. Mod. Phys.*, 74:47–97, Jan 2002.
4. T. Alpcan and T. Başar. A game theoretic approach to decision and analysis in network intrusion detection. In *Proceedings of the 42nd IEEE Conference on Decision and Control*, volume 3, pages 2595–2600, Dec 2003.
5. T. Alpcan and T. Başar. *Network Security: A Decision and Game-Theoretic Approach*. Cambridge University Press, New York, NY, USA, 2010.
6. M. Bloem, T. Alpcan, and T. Başar. Intrusion response as a resource allocation problem. In *Proceedings of the 45th IEEE Conference on Decision and Control*, pages 6283–6288, Dec. 2006.

7. R. Brewer. Advanced persistent threats: Minimizing the damage. *Network Security*, 2014(4):5–9, 2014.
8. C. Bronk and E. Tikk-Rangas. Hack or attack? Shamoon and the evolution of cyber conflict. Feb. 2013. Available at ssrn: `http://ssrn.com/abstract=2270860`.
9. G. Csardi and T. Nepusz. The iGraph software package for complex network research. *InterJournal, Complex Systems*, 1695(5):1–9, 2006.
10. R. Di Pietro and L. V. Mancini. *Intrusion Detection Systems*. Springer, 2008.
11. E. M. Hutchins, M. J. Cloppert, and R. M. Amin. Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains. *Leading Issues in Information Warfare & Security Research*, 1:80, 2011.
12. E. Jones, T. Oliphant, and P. Peterson. SciPy: Open source scientific tools for Python, 2001–. [Online; accessed 2016-06-16]. `http://www.scipy.org/`.
13. R. Langner. Stuxnet: Dissecting a cyberwarfare weapon. *IEEE Security & Privacy*, 9(3):49–51, May 2011.
14. R. M. Lee, M. J. Assante, and T. Conway. Analysis of the cyber attack on the Ukrainian power grid. SANS Industrial Control Systems. 2016.
15. M. H. Manshaei, Q. Zhu, T. Alpcan, T. Başar, and J. Hubaux. Game theory meets network security and privacy. *ACM Comput. Surv.*, 45(3):25:1–25:39, July 2013.
16. R. D. McKelvey, A. M. McLennan, and T. L. Turocy. Gambit: Software tools for game theory. Technical report, Version 15.1.0, 2016.
17. M. Nekovee. Worm epidemics in wireless ad hoc networks. *New Journal of Physics*, 9(6):189, 2007.
18. K. C. Nguyen, T. Alpcan, and T. Başar. Fictitious play with time-invariant frequency update for network security. In *Proceedings of the IEEE International Conference on Control Applications*, pages 65–70, Sept. 2010.
19. M. Penrose. *Random geometric graphs*, volume 5. Oxford University Press Oxford, 2003.
20. M. Roesch. Snort: Lightweight intrusion detection for networks. In *Proceedings of USENIX LISA*, volume 99, pages 229–238, 1999.
21. A. Shameli-Sendi, N. Ezzati-Jivan, M. Jabbarifar, and M. Dagenais. Intrusion response systems: Survey and taxonomy. *Int. J. Comput. Sci. Netw. Secur*, 12(1):1–14, 2012.
22. C. B. Simmons, S. G. Shiva, H. Singh Bedi, and V. Shandilya. ADAPT: A game inspired attack-defense and performance metric taxonomy. In *Proceedings of the 28th IFIP TC 11 International Conference*, SEC 2013, pages 344–365. Springer Berlin Heidelberg, 2013.
23. N. Stakhanova, S. Basu, and J. Wong. A taxonomy of intrusion response systems. *International Journal of Information and Computer Security*, 1(1-2):169–184, 2007.
24. Trend Micro. Understanding targeted attacks: Six components of targeted attacks, November 2015. [Online, Accessed: 06-05-2016]. `http://www.trendmicro.com/vinfo/us/security/news/cyber-attacks/targeted-attacks-six-components`.
25. B. M. Waxman. Routing of multipoint connections. *IEEE Journal on Selected Areas in Communications*, 6(9):1617–1622, Dec. 1988.
26. N. Weaver, V. Paxson, S. Staniford, and R. Cunningham. A taxonomy of computer worms. In *Proceedings of the 2003 ACM Workshop on Rapid Malcode*, pages 11–18, New York, NY, USA, 2003. ACM.
27. Q. Zhu and T. Başar. Dynamic policy-based IDS configuration. In *Proceedings of the 48th IEEE Conference on Decision and Control*, pages 8600–8605, Dec. 2009.
28. S. A. Zonouz, H. Khurana, W. H. Sanders, and T. M. Yardley. RRE: A game-theoretic intrusion response and recovery engine. *IEEE Transactions on Parallel and Distributed Systems*, 25(2):395–406, Feb. 2014.