Chapter 6
Towards High-Resolution Multi-Stage Security Games

Aron Laszka, Xenofon Koutsoukos, and Yevgeniy Vorobeychik

Abstract In recent years, we have seen a large number of cyber-incidents, which demonstrated how difficult it is to prevent cyber-breaches when facing determined and sophisticated attackers. In light of this, it is clear that defenders need to look beyond the first lines of defense and invest not only into prevention, but also into limiting the impact of cyber-breaches. Thus, an effective cyber-defense must combine *proactive defense*, which aims to block anticipated attacks, with *reactive defense*, which responds to and mitigates perceived attacks (e.g., isolating and shutting down compromised components). However, planning defensive actions in anticipation of and in response to strategic attacks is a challenging problem. Prior work has introduced a number of game-theoretic security models for planning defensive actions, such as Stackelberg security games, but these models do not address the overarching problem of proactive and reactive defenses in sufficient detail. To bridge this gap, we introduce a modeling approach for building high-resolution multi-stage security games. We describe several approaches for modeling proactive and reactive defenses, consider key modeling choices and challenges, and discuss finding optimal defense policies. With our study, we aim to lay conceptual foundations for developing realistic models of cyber-security that researchers and practitioners can use for effective cyber-defense.

Aron Laszka
Department of Computer Science, University of Houston, Houston, TX, USA e-mail: alaszka@houston.edu

Xenofon Koutsoukos
Department of Electrical Engineering and Computer Science, Vanderbilt University, Nashville, TN, USA e-mail: xenofon.koutsoukos@vanderbilt.edu

Yevgeniy Vorobeychik
Department of Computer Science and Engineering, Washington University, St. Louis, MO, USA e-mail: yvorobeychik@wustl.edu

6.1 Introduction

Traditionally, security research has focused on preventing attackers from breaching the security of a system or network. While researchers are making considerable advances in this direction, attack techniques are also evolving, which makes providing security an uphill battle. Indeed, attaining perfect security remains virtually impossible for practical systems. The number of reported cyber-incidents is increasing steadily, and the total cost of malicious cyber-activities to the U.S. economy has recently been estimated to be between $57 and $109 billion [9]. For instance, according to a 2017 industry report, 67% of companies with critical infrastructure suffered at least one attack in the preceding 12 months; in particular, 91% of power companies have experienced an attack [17].

In light of this, it is clear that defenders cannot focus only on the first lines of defense, and they must look beyond the prevention of cyber-breaches. Besides preventing breaches, defenders can also alleviate cyber-security risks by reducing the expected impact of successful attacks. In practice, there are a number of actions that defenders may take to limit the impact of a breach, such as quickly isolating and shutting down compromised hosts or reconfiguring the uncompromised ones. While these actions cannot prevent an attack, they can mitigate it before it could cause significant damage. We will refer to such actions collectively as *reactive defense* approaches to emphasize that these actions are taken in response to perceived (or suspected) cyber-attacks.[1]

An effective cyber-defense must combine this reactive approach with *proactive defense* actions. Proactive defense includes actions taken in anticipation of an attack, such as finding and patching software vulnerabilities before an adversary could exploit them. Optimal cyber-defense must consider the whole spectrum of available proactive and reactive actions, and it must implement them in a combination that minimizes cyber-security risks. However, real defenders typically have a finite budget, which limits the amount of resources, effort, and time available to them for implementing cyber-defenses. Consequently, they need to carefully plan what proactive actions to implement in anticipation of attacks and what reactive actions to implement under various attack scenarios in order to minimize cyber-risks subject to their budget constraints.

A key factor in this planning problem is the strategic nature of cyber-security. The most threatening, sophisticated attacks are very often strategic in the sense that adversaries tailor their malicious actions to the defenders' plans. In light of this, defenses must also be planned strategically: On the one hand, defenders must anticipate attacks and plan their actions accordingly, assuming that adversaries will adapt. On the other hand, defenders have to react to observed attacks to mitigate them (e.g., isolate and re-install compromised hosts), assuming that adversaries have mounted strategic attacks and are ready for strategic escalation.

[1] Note that we use the term "reactive defense" to refer to actions taken in response to perceived or suspected attacks. This is different from planning defenses in response to risks, which is sometimes referred to using similar terms (e.g., responsive or reactive security).

Such strategic interactions between defenders and attackers are modeled most naturally using game theory. Indeed, a number of game-theoretic models have been proposed for studying the defense of networked systems [38, 32]. However, prior work has not addressed the overarching problem of proactive and reactive defenses in sufficient detail. Firstly, a number of research efforts have studied high-level models of cyber-security, but these papers often consider very abstract notions of security investments (e.g., allocation of abstract defensive resources to targets [30]). Further, these models are typically based on two-stage security games, which consider only proactive actions, but not reactive ones. Secondly, a number of research efforts have studied the optimal implementation of particular actions in detail, some even considering continuous conflicts and reactive approaches (e.g., resetting potentially compromised computational resources [51]). However, these models typically include only one particular type of action, and it is often unclear—especially for practitioners—how to combine different types of actions most effectively.

To bridge this gap, we discuss how to build realistic, high-resolution multi-stage security games for networked systems, which can form a conceptual foundation for the optimal implementation of proactive and reactive defenses. We first consider the most widely used class of security models, called Stackelberg security games, and argue that these are not well suited for studying reactive defenses. We then discuss stochastic games, which provide a general mathematical framework for modeling multi-stage interactions. Based on this framework, we introduce our approach for modeling the proactive and reactive defense of networked systems against strategic attacks, focusing on key modeling choices and challenges. Then, we describe canonical types of proactive (redundancy, diversity, isolation, hardening, and detection) and reactive (islanding, resetting, and reconfiguration) defenses, again focusing on key modeling choices and challenges. Finally, we consider the problem of finding optimal defense strategies in our model, which is generally a computationally hard problem, and discuss reinforcement learning as a promising solution approach.

The remainder of this chapter is organized as follows. In Section 6.2, we describe two-stage Stackelberg game model of security. In Section 6.3, we discuss stochastic games for modeling multi-stage interactions in security. In Section 6.4, we introduce our modeling approach for building realistic multi-stage models of security. In Sections 6.5 and 6.6, we describe canonical approaches for proactive and reactive defenses, respectively, and we discuss how to model them. In Section 6.7, we consider how to solve realistic multi-stage security games and find optimal defense strategies. In Section 6.8, we provide concluding remarks.

6.2 Stackelberg Game Models of Security

A very natural game theoretic model of security, which has received considerable attention in recent years, is known as *Stackelberg games* [30, 50]. A Stackelberg game involves two stages: in the first stage, the defender chooses a defensive posture (such as which vulnerabilities to patch, or how to configure the firewall), and in the

second stage, the attacker chooses the best attack. The crucial feature of this model is that the attacker is assumed to observe the defensive decision; while in reality this is a very strong assumption, it is also a sound starting point for analysis, as it makes a worst-case assumption about the information available to the attacker, in the spirit of Kerckhoffs's principle [28].

Formally, let D and A denote the sets of actions available to the defender and attacker, respectively, with $d \in D$ and $a \in A$ referring to a particular defense / attack action. To allow for the possibility that the defender randomizes, we let S be the *strategy* set of the defender. Thus, the defender may *commit* to an action, in which case $S = D$, or may be able to commit to a probability distribution over D, in which case $S = \Delta(D)$, where $\Delta(D)$ is the set of all probability distributions over D. Next, we define the utility functions $u_D(s,a)$ and $u_A(s,a)$ for the defender and attacker, respectively, where $s \in S$ is the defender's strategy.

Suppose that the defender commits to a strategy $s \in S$. The attacker's *best response* to s is $\phi(s) \in \operatorname{argmax}_{a \in A} u_A(s,a)$. Correspondingly, the defender then aims to find a strategy s which maximizes its payoff *given* the attacker's best response function $\phi(s)$. In particular, a pair of defender and attacker strategies $(s^*, \phi^*(s))$ is a *Stackelberg equilibrium* if $\phi^*(s)$ is an attacker's best response for each s, and $s^* \in \operatorname{argmax}_{s \in S} u_D(s, \phi^*(s))$. This equilibrium concept raises a subtle but important issue of tie-breaking for the attacker. A common way to resolve it is to consider a *Strong Stackelberg equilibrium (SSE)* in which the attacker breaks ties in the defender's favor [50].

Stackelberg game models of security that we described above are clearly simplistic: in these models, the world has only two stages, with the defender making the first decision, followed by the attacker. In real security settings, the game involves many such stages. For example, after the attacker chooses an attack, once the attack has been observed, the defender can deploy additional mitigations, such as updating anti-virus software, patching vulnerabilities which have been exploited, and rebuilding the compromised machines. The attacker, in turn, can subsequently react to such measures, for example, by exploiting another vulnerability, and so on. A common and very general framework for capturing such multi-stage interactions is through the formalism of *stochastic games*, which we describe next. However, as we subsequently point out, stochastic games are *too* general, and fail to capture much structure exhibited in realistic problems. Consequently, we suggest moving to less general, high-resolution models of multi-stage interactions, which allow us to make progress towards applying game theoretic tools to realistic security scenarios.

6.3 Stochastic Games in Security

A stochastic game is a very general mathematical framework for modeling multi-stage interactions. In the context of security, a two-player stochastic game has a finite set of states X, finite sets of actions for the defender D and attacker A, a transition function $P^{da}_{xx'} = \Pr\{x' | x, d, a\}$, and immediate reward functions $u_D(d,a;x)$ and

$u_A(d,a;x)$ for the defender and attacker, respectively [16, 53]. The game proceeds in discrete time steps $t = \{0,1,2,\ldots\}$, where the state at time t, denoted by x_t, is determined stochastically according to the transition function, given the previous state x_{t-1} as well as the previous actions d_{t-1} and a_{t-1} that were taken by the players. The state x_t along with the actions d_t and a_t taken in that state then determine the players' immediate rewards. Let T be the time horizon of the game; it is finite if the game has a finite horizon, and infinite otherwise. Let the history of states and player actions through time T be $h = \{x_0,d_0,a_0,\ldots,x_T,d_T,a_T\}$ Then, we define the realized utility of a player $i \in \{D,A\}$ (attacker or defender) to be

$$\tilde{U}_i(h) = \sum_{t=0}^{T} \gamma^t \cdot u_i(d_t,a_t;x_t),$$

where $\gamma \in [0,1]$ is the discount factor, which weighs distance rewards exponentially less than current.[2] Since history is stochastic, we can define *expected* utility of player $i \in \{D,A\}$ starting in state x as

$$U_i(x) = \mathbb{E}_h[\tilde{U}_i(h) \,|\, x_0 = x].$$

An important and well-known result in (discounted) stochastic games is that there always exists an equilibrium in which player strategies depend only on current state and, in finite horizon games, time. Specifically, let a policy π_i of a player i determine the action this player takes at each time step of the game. In an infinite-horizon stochastic games, there is an equilibrium pair of policies (π_D,π_A) such that π_i depends only on state x; in finite-horizon stochastic games, such policies would also depend on the time step t.

There are two variations of stochastic games which are particularly relevant to multi-stage interactions in security. One, which is a special case of the stochastic game formalism above, involves alternating moves by the defender and attacker, in which the defender moves first. The significance of this model is that it is a natural extension of the standard two-stage Stackelberg game: indeed, a Stackelberg game model is just such a game with a horizon $T = 1$ (so that there are only two time steps, 0 and 1). Clearly, this extension captures in a very general way the intuition that we started with: the game between a defender and an attacker extends beyond two steps, with a defender reacting to an observed attack, the attacker subsequently reacting to the defender, and so on. A simple way to encode such an iterative encounter in the stochastic game formalism is as follows. Let state x encode which player's turn it is to move; we can do this by adding a binary state variable $x_m \in \{0,1\}$, which deterministically flips in each step. We can let $x_m = 0$ when it's the defender's turn to move, and $x_m = 1$ when the attacker moves. Additionally, let us extend the action sets of both players to allow them to depend on state. Thus, the defender's set of actions is $D(x)$ and the attacker's set is $A(x)$; this change has no effect on

[2] We chose the discounted version of the stochastic game here as we view it as the best model of security interactions, where players are sensitive to time. For example, other things being equal, an attacker would rather obtain intellectual property data sooner than later.

the theoretical properties of equilibria in stochastic games that we had noted above. Thus, whenever $x_m = 0$, $A(x) = \emptyset$ and, conversely, when $x_m = 1$, $D(x) = \emptyset$.

Another variation, which is actually (and somewhat surprisingly) qualitatively different from the conventional stochastic games, is the notion of *Stochastic Stackelberg games (SSGs)* [53, 52]. The definition of SSGs is nearly identical to stochastic games, with one crucial difference: first, the defender commits to a policy π_d, and then the attacker best responds with its own policy $\pi_a(\pi_d)$—note that now the attacker's policy can be different depending on which policy the defender commits to! It turns out that this difference makes SSGs dramatically more challenging to analyze and solve, in general [53, 52]. For example, it is no longer the case that we can restrict attention to policies for both players which only depend on current state, *even when the horizon is infinite* [53].

At this point, we have described several very general formalisms which allow us to capture multi-stage interactions in security. The major concern with these, however, is that they are *too* general: indeed, stochastic games are difficult to solve even when the state space is relatively small. In practice, the number of variables which determine state can be substantial, and even the representation of a stochastic game described above becomes intractable. Clearly, in order for us to significantly advance the art in considering interesting multi-stage interactions, we need to consider lower-level structure. In the remainder of this chapter, we propose and illustrate the idea of *high-fidelity* multi-stage games, that is, games in which we make use of much more specialized, high-fidelity models of the domain. While this necessarily loses generality, we argue that such modeling is necessary to reveal important structure in multi-stage games which can enable us to solve more realistic problems.

6.4 Towards Realistic Multi-Stage Game Models

We now discuss modeling approaches and assumptions for high-fidelity multi-stage games for studying the defense of networked systems. While our discussion will consider networked systems in general, we will use *cyber-physical systems* (CPS) as a running example to illustrate the practical applicability of our model. Defending CPS from cyber-physical attacks is an issue that is both pressing and challenging. As CPS are becoming more prevalent (e.g., smart grid, Industrial Internet of Things), the importance of ensuring that they are resilient to cyber-attacks is growing rapidly. For example, cyber-attacks against critical infrastructures, such as smart water-distribution and transportation networks, pose a serious threat to public health and safety. Indeed, real-world attacks have demonstrated that cyber-attacks may penetrate CPS and cause significant physical damage [37, 44, 56, 2, 49]. On the other hand, defending a complex and large-scale CPS, such as smart critical infrastructure, is extremely challenging. These systems often face a variety of threats, contain low-power and legacy components, have large attack surfaces, and have a number of undiscovered software vulnerabilities in their sizable codebases. In light of this, defending CPS is an ideal application example for security game models.

6.4.1 System Model

We first introduce a basic model of networked systems, which will provide a basis for the discussion of game-theoretic models. In general, we can model a networked system as a graph (C, E), where C is a set of components and E is a set of links between the components. Depending on the granularity of the model, a component may correspond to a subnetwork, a host, a running process, or just a software module. A link models a communication channel between two components, which can be either physical (e.g., wired link) or logical (e.g., VPN). A link may be either unidirectional or bidirectional, which we can model using either directed arcs (i.e., $E \subseteq C \times C$) or undirected edges (i.e., $E \subseteq \binom{C}{2}$). A key factor in network security is that links are not only used to transmit information, control signals, etc., but they may also be exploited by an attacker to escalate an attack by compromising the neighbors of an already compromised component.

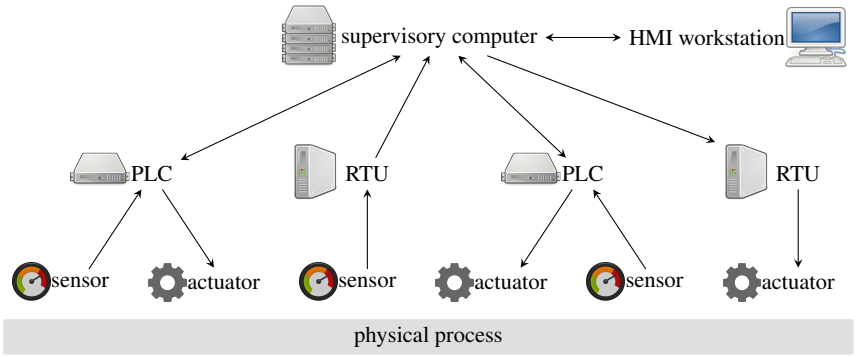

Fig. 6.1 Example cyber-physical system. Labeled icons represent components; arrows represent links through which sensor data and control signals can flow.

CPS Example To illustrate how we can model a networked system as a graph, we now present a high-level model of networked cyber-physical systems, focusing on the cyber parts of the systems. Figure 6.1 shows an example of a networked cyber-physical system, consisting of a variety of physical devices. We let the components C model such physical devices, which we divide into four component types:

- *sensor*: components that measure the state of physical processes (e.g., water-pressure sensors, induction-loop sensors for measuring traffic);
- *actuator*: components that directly affect physical processes (e.g., valves, pumps, circuit breakers);
- *processing*: components that process and store data and control signals (e.g., PLCs, RTUs, supervisory computers);
- *interface*: components that interact with human users (e.g., HMI workstations) or other systems, which are not part of the model.

The links E model communication links between these devices, which are used to transmit sensor data and control signals. The observability and controllability of the physical processes within the CPS depend not only on the functionality of the individual components, but also on the structure of the graph (C,E). Depending on this structure, an attacker may be able cause physical damage or loss by compromising a subset of the components, and tampering with sensor data or control signals.

6.4.2 Game-Theoretic Model

We next discuss how to build a stochastic security game based on the above model of networked systems. We first consider the players' action sets D and A, and then their utility functions u_i.

We assume alternating moves by the defender and the attacker, the defender moving first. In each time step, a player may take multiple actions. Slightly abusing notation, we let D and A denote the sets of actions available to the defender and attacker, respectively. A policy π_i determines the subset of actions to be taken in each time step.

Table 6.1 Defense Actions

Type	Name	Idea	Section
Proactive	Redundancy	deploying redundant components	6.5.1
	Diversity	implementing components using a diverse set of hardware and software	6.5.2
	Isolation	removing links between components	6.5.3
	Hardening	making components (or implementation types) more resilient to attacks	6.5.4
	Detection	deploying intrusion detection systems	6.5.5
Reactive	Islanding	removing links between components	6.6.1
	Resetting	resetting components into known secure states	6.6.2
	Reconfiguration	changing the configuration of components	6.6.3

We divide the defenders' actions D into two disjoint sets of actions (see Table 6.1):

- *Proactive defense actions* D_P: Proactive actions are taken in anticipation of attacks (e.g., deploying an intrusion detection system). In our model, we assume that the defender can take these actions only in time step $t = 0$, which represents everything that happens before the attacker may mount its attack. We discuss proactive actions in more detail in Section 6.5.
- *Reactive defense actions* D_R: Reactive actions are taken in response to an observed attack (e.g., shutting down and re-installing a compromised host). In our

model, we assume that the defender can take these actions only in time steps $t > 0$. We discuss reactive actions in more detail in Section 6.6.

Meanwhile, an attacker tries to compromise or impair the components of the system by attacking them.[3] We let $C_t^C \subseteq C$ and $C_t^I \subseteq C$ denote the sets of components that are compromised or impaired by the attacker at the end of time step t. Each attack—of which the attacker may mount multiple in a time step—targets a specific subset $K \subseteq C$ of components using a specific attack method (e.g., code injection attack or DDoS attack). The set of attack actions A corresponds to the possible combinations of targeted components and attack methods. Attacks are non-deterministic in the sense that they do not necessarily succeed in compromising or impairing all the targeted components K. For example, an attack might require finding a software vulnerability in a certain implementation or guessing a password, and the attacker might fail to do so. In general, the success probability of an attack is an increasing function of the set of components that have already been compromised or impaired.[4] Firstly, the attacker might exploit the implicit trust relations between components that are connected by links E to easily compromise the neighbors of an already compromised component. Secondly, the impairment of components may result in cascading failures, which makes the impairment of other components easier.

The attacker also incurs a cost for mounting its attacks. The cost of mounting an attack depends on both the set of targeted components K and the method of attack. For attack methods that are easily replicated for a large number of components (e.g., once a software vulnerability is found, the attacker may easily compromise a large number of hosts), the cost can be modeled as a submodular function of K, capturing the diminishing marginal cost of attacking an additional component.

In general, the attacker's goal is to cause damage or gain some benefit by compromising or impairing the components of the system, while the defender's goal is to minimize its losses due to successful cyber-attacks. These goals are captured using the players' utility functions u_i, which they aim to maximize through their action choices. In principle, we can express the defender's utility as the baseline utility provided by an operational system minus the losses caused by the attacks and the costs of implementing defensive actions. Note that since this baseline utility does not depend on the players' actions, it may be omitted without affecting the best-response or equilibrium strategies. Similarly, we can express the attacker's utility as the attacker's gain from compromising or impairing the components minus the costs of mounting its attacks. The form of the loss and gain functions depends on the specifics of the modeled system. However, in most systems, we can express loss and gain as functions of the compromised and impaired components C_t^C and C_t^I; hence, we may express a player's utility u_i for time step t as a function $u_i(d_t, a_t; C_t^C, C_t^I)$. In a simple model, we may also assume that the defender's loss and the attacker's gain

[3] For ease of presentation, we only consider attacks against components, but it would be straightforward to extend our modeling approach to also consider attacks against links.

[4] In practice, the probability may decrease since the defender may notice a large-scale attack and deploy countermeasures in response. In our model, this effect is captured explicitly through the defender's reactive actions.

are always equal. Note that even under this assumption, the game is not necessarily zero sum since the players also incur costs for their actions.

Finally, while it is impossible to provide generic loss/gain functions that are applicable to all system, we can provide modeling guidelines. For attacks against confidentiality, loss/gain may be expressed as a submodular function of the set of compromised components C_t^C since the information gained from compromising more and more components exhibits a diminishing return due to the possible overlap between the information contained by a set of components.[5] On the other hand, for attacks against integrity, loss/gain may be expressed as a supermodular function of the set of compromised components C_t^C since when information is stored redundantly on multiple components, the attack remains undetected only if all of these components are compromised. Similarly, system functionality that is provided redundantly by multiple components can be tampered with (or disabled) only by compromising the majority of the components (or impairing all of them).

CPS Example In many cyber-physical systems, loss can be measured in terms of physical impact. For example, a cyber-attack against a smart transportation network may cause disastrous traffic congestion [35, 55].[6] Such attacks have been made possible by the evolution of traffic control from standalone hardware devices into complex networked systems, which has exposed traffic control to attacks through wireless interfaces or even remote attacks through the Internet. As demonstrated by the 2006 incident in Los Angeles, tampering with traffic control can cause significant losses through congestion [23].

To formulate a multi-stage security game for smart transportation networks, we may model the physical part of the system using an established traffic model (e.g., Daganzo's well-known cell-transmission model [10, 11]), while we can model the cyber part of the system using the following components C:

- *interface*: human-machine interface components, which traffic operators can use to control traffic lights in the transportation network;
- *processing*: devices that process and forward traffic control signals;
- *actuator*: traffic lights with software-based controllers.

An attacker may try compromise these components, e.g., by connecting to traffic lights through local-area wireless networks and exploiting software vulnerabilities. Indeed, studies have found that many traffic control devices that are deployed in practice have unpatched known software vulnerabilities [21, 55]. Once an attacker has compromised some set of the components C^C, it can alter the schedules of traffic lights, thereby causing disastrous traffic congestion. We can quantify the impact of such an attack as the total increase in travel time experienced by all the drivers. Assuming a malicious attacker who is interested in maximizing the defender's loss,

[5] Defender's may turn this around by using, e.g., secret sharing schemes, which lead to a supermodular loss/gain functions for confidentiality. This possibility is considered explicitly among the defender's proactive actions; here, we consider a baseline case without such schemes.

[6] In practice, due to hardware-based failsafes, compromising a traffic signal does not allow an attacker to set the signal into an unsafe configuration that could immediately lead to traffic accidents [21].

we can measure both the defender's loss and the attacker's gain as the impact of the attack.

6.4.3 Imperfect and Incomplete Information

A key aspect of security games is that generally the players do not possess perfect and complete information. Firstly, the players might not know what actions their opponents have taken and, hence, which components are compromised, which means that they possess *imperfect information*. While imperfect information can affect both players, there is often an asymmetry between the players, which may put the defender at a grave disadvantage. On the one hand, the attacker knows which components it has attacked and—in most cases—which components it has compromised. On the other hand, the defender may not immediately learn of compromises. Indeed, a recent study has found that on average, it takes 191 days to detect a data breach [45]. Lack of perfect information can prevent the defender from reacting and implementing countermeasures in time to mitigate an attack, which enables the attacker to operate covertly in the compromised system, causing damage and extracting information. In practice, attackers often seek to remain covert for as long as possible in order to cause more damage or extract more information over a longer period of time. For example, sophisticated spyware (e.g., used in state-sponsored cyber-espionage campaigns) often remain covert for extended periods of time [26]. Even malware that causes physical damage in a cyber-physical system may remain covert for months, as demonstrated by the Stuxnet worm [27].

To some extent, the attacker might also suffer from imperfect information. While we typically assume—following Kerckhoffs's principle—that the attacker can learn the defender's strategy, this strategy may be a probability distribution over possible actions, and the attacker does not learn the specific action if it chosen truly randomly. Further, the attacker might also not be able to directly observe which components it has compromised. For example, the defender might secure a host (e.g., shutdown and re-install) that is not connected to the Internet, which the attacker has compromised earlier using a worm. In such a scenario, the attacker will not learn immediately that the component is no longer compromised (or if it ever were). In light of this, the players' policies cannot be defined as functions of the state x. Rather, each players' policy needs to be defined as a function of their observations.

In addition to imperfect information, the players may also suffer from incomplete information, i.e., not knowing the exact action sets, state transition functions, or utility functions. Firstly, the defender might not know what actions are available to an attacker (e.g., specific attack techniques and exploits) or how likely these actions are to successfully compromise or impair components. Further, the defender might also not know the attackers' objectives and what resources they have (e.g., script kiddies or nation-state sponsored attackers) [13, 14]. Secondly, the attacker might not have complete knowledge of the target system. Even though we follow Kerckhoffs's principle and assume that the attacker will be able determine the de-

sign of the system, the defender might still be able to deceive the attacker [47]. For example, the defender might deploy honeypots in the system in order to waste the attacker's effort and observe its behavior [43].

6.5 Proactive Defense

Proactive defense includes actions taken by a defender in anticipation of attacks. Here, we discuss various approaches for the proactive defense of networked systems in more detail, focusing on how to incorporate them into our game-theoretic model. Recall from our previous discussion that these actions are taken in time step $t = 0$ (i.e., before the attacker's first move).

6.5.1 Redundancy

Redundancy means deploying additional components in the system, which are not necessary for providing required system functionality or performance [5]. When facing denial-of-service attacks, which impair components, the benefits of redundancy are clear: in case of an attack, the redundant components may be used instead of the ones that are unavailable due to the attack. As long as a sufficient set of components are still available, the system might suffer from decreased performance, user experience, etc., but retains its functionality.

In practice, redundancy may be implemented, for example, by deploying additional physical hosts or storing redundant copies of information. In a fine-grained model, where components correspond to software modules or services, redundancy can be implemented even for security mechanisms. For example, multi-factor authentication methods grant a user access to a system only after verifying the user's identity using multiple authentication methods [12]. In a cyber-physical system, redundancy can be implemented by, e.g., deploying multiple sensors for monitoring the same physical process [1], or deploying multiple controllers and letting actuators act based on the median control value provided by these controllers.

While the benefits of redundancy are obvious in the case of denial-of-service attacks, they are much less straightforward in the case of integrity attacks that compromise and tamper with components. Since defenders—and the systems under their control—may not know which components have been compromised, when redundant components provide contradictory information, they face the challenging problem of deciding which components to trust. Further, simple redundancy might even increase risks when it comes to confidentiality. Without redundancy, the attacker would need to compromise a particular component to gain a particular piece of information. However, with redundancy, it needs to compromise one out of many redundant components, which may give the attacker more opportunities to succeed.

Consequently, to protect confidentiality, redundancy may need to combined with, e.g., secret sharing schemes [25, 46].

We can model redundancy by allowing the defender to choose the set of deployed components C from a family $\mathscr{C}$ of feasible sets. This family $\mathscr{C}$ consists of all the sets that are sufficient for providing required system functionality and performance. In a simple model, we may assume that a base deployment C_{base} is given, and the defender can choose only supersets $C \supseteq C_{base}$ (i.e., $\mathscr{C} = \{C \,|\, C \supseteq C_{base}\}$). By deploying additional components, the defender incurs some cost. In the case of hardware, this is the cost of purchasing, installing, and operating devices, which may be an additive or submodular function of the set of additional devices (i.e., fixed cost or diminishing marginal cost model). In the case of services and software modules, this may be development cost or the computational/communication cost of running an additional software components.

6.5.2 Diversity

Deploying redundant components may be a futile effort if all of the components are implemented and configured in the same way since an attacker might be able to compromise all of them with relatively little effort using a common software or configuration vulnerability. A defender can prevent this by implementing the components using a *diverse set of hardware and software*, for example, by running redundant web servers on different operating systems. Diversity reduces the impact of any common vulnerability since only the components that are implemented using the vulnerable software or hardware will be susceptible to the same exploit. Indeed, diversity has been recognized as an effective approach for improving network security, and prior work has studied the optimal assignment of implementation types to components [42]. On a larger, societal scale, monoculture (i.e., lack of diversity in software solutions) has been identified as a contributor to systemic cyber-risks [6, 18].

Similar to redundancy, diversity must be used carefully since it may increase risks in some cases. If an attacker needs to compromise a certain set of components to inflict damage, then diversity increases resilience since the probability of finding a vulnerability in multiple implementations is generally lower than finding one in a single implementation. However, if the attacker needs to compromise only one out of many components, then diversity decreases resilience since the more implementation types, the higher the probability that at least one of them has a vulnerability.

We can model diversity by letting the defender assign an implementation type to each component. More specifically, for each component $c \in C$, we can assume that a set of feasible implementations I_c is given, and the defender can select a particular implementation i_c. In practice, the defender typically incurs some cost for introducing a new implementation type into the system. For example, introducing a new software may require purchasing licenses and training for personnel. Consequently,

the cost of diversity depends on the set of all the implementation types $\bigcup_{c \in C} \{i_c\}$ that are used in the system.

6.5.3 Isolation

While links serve a useful purpose by providing connectivity between components, they also enable an attacker to escalate its attack by compromising the neighbors of a compromised component. A defender can prevent escalation and limit the impact of compromises by *isolating* components (or sets of components) from each other [48]. In practice, techniques for isolation range from sandboxes for software components to firewalls between networks. To minimize security risks, isolation may be implemented on a physical level by introducing an "air gap" (i.e., physical separation) between components. In the context of cyber-physical systems, "air gap" is typically used to protect safety-critical control systems [15].

We can model isolation by allowing the defender to remove links from the network. Equivalently, we may allow the defender to choose the set of links E to retain, under the constraint that this set of links must be chosen from a family $\mathscr{E}$ of feasible sets. This family $\mathscr{E}$ consists of all the sets that are sufficient for providing connectivity that is necessary for the required system functionality and performance. We may define the family of feasible sets using graph-theoretic notions; for example, we may require the set of links E to form a strongly connected graph of components C.

By severing useful links between components, the defender incurs various costs. For example, decreased connectivity may result in lower performance or functionality as well as increased usability and operational costs (e.g., information that could have been sent automatically on a link might have to be transferred manually using removable drives). Consequently, a defender must carefully choose which components to isolate from each other. Dividing a networked system into isolated parts is a challenging graph-theoretic problem, which has been studied in prior work, e.g., as a computationally-hard graph partitioning problem [4].

6.5.4 Hardening

Hardening includes techniques for protecting components from being compromised by an attacker. These techniques may be applied at either the hardware level (e.g., using tamper-resistant hardware to prevent attacks based on physical access) or at the software level (e.g., thorough testing for software vulnerabilities). Typically, hardware-level techniques are applied to individual components (i.e., protection for particular devices), while software-level techniques are applied to a set of components that are implemented using the same software (i.e., eliminating common vulnerabilities). Defenders may employ a variety of approaches for eliminating

software vulnerabilities, ranging from following secure-coding principles to hiring outside experts for penetration testing or crowdsourcing vulnerability discovery through bug-bounty programs [58, 36].

For hardware-level protection, we can model hardening by allowing the defender to choose how much to spend on improving the security of each component. For software-level protection, the defender needs to choose how much to spend on improving certain implementation types $i \in I$, where $I = \cup_{c \in C} I_c$ is the set of all implementation types in the system. In both cases, hardening either decreases the probability that an attack succeeds against the hardened components, or it increases the cost of launching a successful attack against the hardened components. Optimal security investments have been thoroughly studied in the economics of security literature [3, 22].

6.5.5 Detection

With respect to information, the defender is at a grave disadvantage compared to the attacker. Since the defender has imperfect information regarding which components have been attacked or compromised, it can only guess which actions to take to mitigate a potential attack most effectively. To decrease this information gap, defenders can deploy *intrusion detection* systems. An intrusion detection system (IDS) monitors a system or network and raises an alarm when it encounters malicious activity, which can then be investigated by system operators. In practice, IDS come in a wide variety. A host-based IDS is deployed on and monitors a particular host (e.g., running processes), while a network-based IDS monitors network traffic. A signature-based IDS searches for known attacks, while an anomaly-based IDS looks for deviation from normal operation. A variety of intrusion detection systems have also been proposed for cyber-physical systems [40]

However, practical intrusion detection systems are imperfect. On the one hand, they may fail to detect an actual attack, which is called a false-negative error. On the other hand, they may raise an alarm when they encounter suspicious but non-malicious activity, which is called a false-positive error. Both of these are errors should be minimized since false negatives prevent the defender from mitigating attacks, while false positives waste the limited amount of time and effort available for investigations. However, there is generally a trade-off between the two errors: decreasing the rate of false positives results in an increased rate of false negatives, and vice versa. Therefore, defender must carefully configure each IDS to minimize losses due to attacks and the costs of investigations at the same time. Finding optimal configurations for intrusion detection systems is a challenging problem by itself [31, 19, 20].

We can model detection by allowing the defender to place intrusion detection systems on components or links. We let $S^C \subseteq C$ denote the set of components with detectors, which model host-based IDS, and let $S^E \subseteq E$ denote the set of links with detectors, which model network-based IDS. For each IDS $s \in S^C \cup S^E$, the defender

must choose a trade-off between false-negative and false-positive errors by configuring the detector. We can represent the attainable combinations using a trade-off function $F_s : \mathbb{R}_+ \times A \to [0,1]$, where $F_s(f_s, a)$ is the estimated probability that attack a is undetected when the false-positive error rate of the detector is f_s. In each timestep, the defender's beliefs are updated based on which detectors have raise a true alarm, while the defender incurs cost for all the false alarms raised $\sum_{s \in S^C \cup S^E} f_s$.

6.6 Reactive Defense

Reactive defense includes actions taken by a defender in response to observed attacks. Here, we discuss approaches for the reactive defense of networked systems in more detail, focusing on how to incorporate them into our game-theoretic model. In contrast to proactive defense, these actions are taken in time steps $t > 0$ (i.e., after an attack may have been launched).

6.6.1 Islanding

Isolation can be an effective approach for limiting the impact of successful attacks (Section 6.5.3); however, it requires severing links proactively, which results in permanent usability and performance degradation, even when the defender has not observed an attack. Here, we consider *islanding*, which can be thought of as a reactive variant of isolation that severs links only after detecting an attack. More specifically, islanding means severing links to components (or to a set of components) that the defender suspects to be compromised by an attacker.

Islanding clearly has some advantages over isolation, but it can also be favorable compared to simply shutting down and resetting (e.g., re-installing) components that are suspected to be compromised. Since the defender possesses imperfect information regarding which components have been compromised, implementing any reactive defense is risky in the sense that the actions might not only be costly but also unnecessary. Shutting down and resetting a component is a drastic measure that may result in significant downtime. In a cyber-physical system, such as a power plant, where components have to sense and control physical processes in real time, downtime can be prohibitively expensive. On the other hand, islanding allows the defender to prevent the escalation of a suspected attack without shutting down the potentially compromised components. If these islanded components can provide some level of functionality (e.g., an islanded component in a cyber-physical system may still be able to control a physical process), then islanding can be a less risky option for the defender.

We can model islanding similar to isolation, by allowing the defender to choose which links are active in each time step. More formally, in each time step t, the defender may choose a set of active links $E_t \subseteq E$. Then, the attacker will only be

able use links E_t to escalate its attack in time step t, while the defender incurs cost due to the performance and usability loss from the unavailability of links $E \setminus E_t$.

6.6.2 Resetting

Even though islanding can contain a security breach by preventing the attacker from escalating the attack to compromise other components, it cannot eliminate the breach and secure the system. We now consider actions that return compromised components into their normal, uncompromised state, to which we refer as *resetting*. For components that model physical hosts, resetting typically involves shutting down and re-installing the hosts, while for software components, re-launching running processes may be enough to bring them into a secure state as long as they have not effected any permanent changes to, e.g., configuration files.

By resetting a component, the defender incurs cost due to the effort and time required to reset the component, as well as the cost of the component being unavailable while it is being reset. Since the defender does not have perfect information, it does not know when to reset a component: resetting a component that has not been compromised results in unnecessary expenses, while not resetting a compromised one may result in increased losses due to the prolonged impact of the attack. Consequently, deciding when to reset a potentially compromised component is a challenging problem. This problem has been studied extensively by prior work using the FlipIt model [51, 33, 7, 34, 57], and optimal resetting schedules have been proposed under various conditions. However, integrating these results into a multi-stage game where a variety of actions are available to the defender is an open problem, especially considering the structural properties of networked systems.

We can model resetting by allowing the defender to select which components $R_t \subseteq C$ to reset in each time step t. Selected components R_t are removed from the sets of compromised and impaired components C_t^C and C_t^I, respectively, but they also become (or remain) unavailable for a certain number of time steps, which models downtime due to resetting. Further, the defender may incur two types of costs. Firstly, it incurs the direct cost of resetting the components, which can be modeled as an additive function of R_t. Secondly, it incurs the cost of lost performance or functionality due to the downtime of the selected components, which may depend on the deployment of the system. For example, if there are redundant components available, there might not be any performance or functionality loss.

6.6.3 Reconfiguration

In addition to islanding and resetting potentially compromised components, the defender may also mitigate attacks and limit their impact by changing the behavior of uncompromised components. In particular, the defender can reconfigure compo-

nents that are still available and under its control in order to reduce the losses arising from an attack. For example, in a cyber-physical system, a controller may be reconfigured when some sensors or actuators are compromised or impaired, so that the new control maintains system stability and prevents system failure in spite of the attack [8].

We can model reconfiguration actions by letting the defender select in every time step a configuration for each available component. Formally, in each time step t, the defender selects for each component $c \in C \setminus C_t^I$ a configuration $F_{c,t}$. However, the configurations are applied only to uncompromised components $(C \setminus C_t^I) \setminus C_t^C$ (note that the defender does not necessarily know which components are compromised and which are under its control). Reconfiguring a component c may have some cost, such as the effort exerted to effect the change or the loss due to temporary outage while reconfiguring components, which the defender incurs only if it actually changes the configuration, i.e., if $F_{c,t} \neq F_{c,t-1}$. Further, the selected configurations may also have an impact on the performance and functionality of the system (e.g., in a cyber-physical system, a more stable controller may be less efficient), which affects the defender's utility.

6.7 Solving Multi-Stage Security Games

Our goal is to find an optimal defense policy π_D, which proscribes what defensive actions to take in each time step based on the observed state. Unfortunately, this problem is computationally challenging due to the sizes of the action and state spaces. Firstly, in each time step, the defender has to choose from a set of actions whose cardinality is an exponential function of the size of the graph (C,E) that models the system. For example, consider isolation and islanding actions, which are chosen from the set of all subsets of links E. Since the number of all subsets is $2^{|E|}$, the size of the defender's action space can be astronomical even for graphs of modest size.[7] Similarly, the number of possible states is also an exponential function of the size of the graph (C,E). For instance, the set of compromised components C^C is a subset of the set of components C; hence, there may be up to $2^{|C|}$ different sets of compromised components. Further, the set of possible observations for the defender, which may include various alerts generated by detectors, could be even larger.

Considering the sizes of the action and state spaces, it is challenging not only to find an optimal policy, but even just to represent one. A straightforward policy representation would specify what actions to take for each possible state, for example, in the form of a list. Clearly, the size of this list would be prohibitively large for any practical system. Therefore, there is a need for devising a *compact representation* of proactive and reactive action policies.

Even restricted to some compact representation, finding an optimal policy may be computationally hard. Indeed, prior work has shown that a number of subprob-

[7] Some of these subsets may not be feasible, but in general the number of feasible subsets may grow exponentially.

lems (e.g., finding optimal actions of a certain type in a given state) are NP-hard. For example, finding optimal configurations for intrusion detection systems may be an NP-hard problem when facing strategic attacks [31, 19]. In light of this, we must consider *efficient algorithms* that can find near-optimal actions. To devise such algorithms, we can take advantage of the structure of our problem. In other words, instead of resorting to generic meta-heuristics, we can tailor our algorithms to the rich structure of security states and defensive actions.

We can combine these algorithms with reinforcement learning approaches for finding near-optimal policies [41]. Many reinforcement learning algorithms, such as Q-learning [54], work by learning the values of the possible states (e.g., a state in which more components are compromised may be worth less to the defender than a state with fewer compromised components). Once these values have been learned, the best action in a certain state can be chosen based on which action results in the highest expected value for the following state, considering the probabilities of the various state transitions for a particular action.

Since exhaustively searching for the best action would not be feasible in our model, we propose to use an actor-critic method [29], which represents both the state values and the policy explicitly. Considering the complexity of the state and action spaces, there is a need to represent the state values and the policy efficiently, which we may do using (deep) neural networks. This model-free approach can be combined with efficient, model-specific algorithms for finding a near-optimal action in a particular state to support the exploration part of reinforcement learning.

However, considerable challenges remain in the application of reinforcement learning to multi-stage security games. Firstly, the actor-critic method can be used directly to find a near-optimal policy for one player, which constitutes an approximate best response, against a given policy of the opponent. Solving the game and finding a strategic cyber-defense policy, however, requires finding an equilibrium pair of defender and attacker policies. To find a mixed-strategy equilibrium, we can apply a double-oracle approach, which starts with a restricted set of strategies (i.e., policies), and then iteratively computes a mixed equilibrium over the restricted set and extends the set with best-response strategies against this equilibrium [39]. The application of a double-oracle approach may lead to further computational problems since computing many best-response strategies (i.e., running reinforcement learning many times to find policies) can be computationally expensive.

Another challenge arises from the fact that the players in our game have neither complete nor perfect information. Consequently, reinforcement learning has to find near-optimal policies for partially observable Markov decision processes. To achieve good results for partially observable processes, we can extend the actor-critic method with an internal state, for example, using recurrent neural networks [24].

6.8 Conclusion

To protect sensitive networked systems, defenders need to deploy complex cyber-defense solutions, which combine a variety of proactive and reactive techniques to minimize cyber-risks. Devising complex defense solutions for practical systems is a daunting task, which must be supported by strong theoretical models and efficient tools. To address this need, we introduced a modeling framework for high-resolution multi-stage security games for networked systems. We discussed a number of canonical proactive and reactive defense approaches, focusing on modeling choices and challenges. Finally, we considered the computational problem of finding optimal defense policies.

There remain several open problems in the area of high-resolution multi-stage security games. While we have laid foundations for theoretical models, incorporating a spectrum of practical defense methods into this framework requires further modeling work. Then, models need to be rigorously evaluated using data regarding past cyber-breaches as well as the architecture, performance, and functionality of a wide range of practical networked systems. Once these models have been established, the gap between theory and practice must be bridged by providing software tools for practitioners that facilitate the application of models to practical systems. Finally, finding optimal defense policies poses a very challenging computational problem. We have outlined approaches for addressing this problem, but developing efficient practical algorithms and tailoring reinforcement learning methods to multi-stage security games remain open problems.

References

1. Abbas, W., Laszka, A., Koutsoukos, X.: Resilient wireless sensor networks for cyber-physical systems. In: S. Zeadally, N. Jabeur (eds.) Cyber-Physical System Design with Sensor Networking Technologies, pp. 239–267. The Institution of Engineering and Technology (2016)
2. Abrams, M., Weiss, J.: Malicious control system cyber security attack case study – Maroochy Water Services, Australia. `http://csrc.nist.gov/groups/SMA/fisma/ics/documents/Maroochy-Water-Services-Case-Study_report.pdf` (2008)
3. Anderson, R., Moore, T.: The economics of information security. Science **314**(5799), 610–613 (2006)
4. Aspnes, J., Chang, K., Yampolskiy, A.: Inoculation strategies for victims of viruses and the sum-of-squares partition problem. Journal of Computer and System Sciences **72**(6), 1077–1093 (2006)
5. Avizienis, A., Laprie, J.C., Randell, B., Landwehr, C.: Basic concepts and taxonomy of dependable and secure computing. IEEE transactions on dependable and secure computing **1**(1), 11–33 (2004)
6. Birman, K.P., Schneider, F.B.: The monoculture risk put into context. IEEE Security & Privacy **7**(1), 14–17 (2009)
7. Bowers, K.D., Van Dijk, M., Griffin, R., Juels, A., Oprea, A., Rivest, R.L., Triandopoulos, N.: Defending against the unknown enemy: Applying FlipIt to system security. In: Proceedings of the 3rd Conference on Decision and Game Theory for Security (GameSec), pp. 248–263. Springer (2012)

8. Cómbita, L.F., Giraldo, J., Cárdenas, A.A., Quijano, N.: Response and reconfiguration of cyber-physical control systems: A survey. In: Proceedings of the 2nd Colombian Conference on Automatic Control (CCAC), pp. 1–6. IEEE (2015)
9. Council of Economic Advisers: The cost of malicious cyber activity to the u.s. economy. Tech. rep., Executive Office of the President (2018)
10. Daganzo, C.F.: The cell transmission model: A dynamic representation of highway traffic consistent with the hydrodynamic theory. Transportation Research Part B: Methodological **28**(4), 269–287 (1994)
11. Daganzo, C.F.: The cell transmission model, part II: Network traffic. Transportation Research Part B: Methodological **29**(2), 79–93 (1995)
12. Dasgupta, D., Roy, A., Nag, A.: Multi-factor authentication. In: Advances in User Authentication, pp. 185–233. Springer (2017)
13. Dritsoula, L., Loiseau, P., Musacchio, J.: Computing the nash equilibria of intruder classification games. In: International Conference on Decision and Game Theory for Security, pp. 78–97. Springer (2012)
14. Dritsoula, L., Loiseau, P., Musacchio, J.: A game-theoretic analysis of adversarial classification. IEEE Transactions on Information Forensics and Security **12**(12), 3094–3109 (2017)
15. Farwell, J.P., Rohozinski, R.: Stuxnet and the future of cyber war. Survival **53**(1), 23–40 (2011)
16. Filar, J., Vrieze, K.: Competitive Markov Decision Processes. Springer (1997)
17. GE Digital: The impact of cyber attacks on critical infrastructure. Tech. rep., General Electric (2017)
18. Geer, D., Bace, R., Gutmann, P., Metzger, P., Pfleeger, C., Querterman, J., Scheier, B.: CyberInsecurity: The cost of monopoly–how the dominance of Microsoft's products poses a risk to security. Tech. rep., Computer and Communications Industry Association (2003)
19. Ghafouri, A., Abbas, W., Laszka, A., Vorobeychik, Y., Koutsoukos, X.: Optimal thresholds for anomaly-based intrusion detection in dynamical environments. In: Proceedings of the 7th Conference on Decision and Game Theory for Security (GameSec), pp. 415–434 (2016)
20. Ghafouri, A., Laszka, A., Dubey, A., Koutsoukos, X.: Optimal detection of faulty traffic sensors used in route planning. In: Proceedings of the 2nd International Workshop on Science of Smart City Operations and Platforms Engineering (SCOPE), pp. 1–6 (2017)
21. Ghena, B., Beyer, W., Hillaker, A., Pevarnek, J., Halderman, J.A.: Green lights forever: Analyzing the security of traffic infrastructure. In: Proceedings of the 8th USENIX Workshop on Offensive Technologies (WOOT), vol. 14, pp. 1–10 (2014)
22. Gordon, L.A., Loeb, M.P.: The economics of information security investment. ACM Transactions on Information and System Security **5**(4), 438–457 (2002)
23. Grad, S.: Engineers who hacked into LA traffic signal computer, jamming streets, sentenced. Los Angeles Times (2009)
24. Hausknecht, M., Stone, P.: Deep recurrent Q-learning for partially observable mdps. In: 2015 AAAI Fall Symposium Series (2015)
25. Karnin, E., Greene, J., Hellman, M.: On secret sharing systems. IEEE Transactions on Information Theory **29**(1), 35–41 (1983)
26. Kaspersky Labs' Global Research & Analysis Team: Gauss: Abnormal distribution. `https://securelist.com/analysis/36620/gauss-abnormal-distribution/` (2012)
27. Kelley, M.B.: The Stuxnet attack on Iran's nuclear plant was 'far more dangerous' than previously thought. *Business Insider*, `http://www.businessinsider.com/stuxnet-was-far-more-dangerous-than-previous-thought-2013-11` (2013)
28. Kerckhoffs, A.: La cryptographie militaire. Journal des Sciences Militaires **IX**, 5–83 (1883)
29. Konda, V.R., Tsitsiklis, J.N.: Actor-citic agorithms. In: Proceedings of the 12th International Conference on Neural Information Processing Systems (NIPS), pp. 1008–1014. MIT Press (1999)

30. Korzhyk, D., Yin, Z., Kiekintveld, C., Conitzer, V., Tambe, M.: Stackelberg vs. nash in security games: An extended investigation of interchangeability, equivalence, and uniqueness. Journal of Artificial Intelligence Research **41**, 297–327 (2011)
31. Laszka, A., Abbas, W., Sastry, S.S., Vorobeychik, Y., Koutsoukos, X.: Optimal thresholds for intrusion detection systems. In: Proceedings of the 3rd Annual Symposium and Bootcamp on the Science of Security (HotSoS), pp. 72–81 (2016)
32. Laszka, A., Felegyhazi, M., Buttyan, L.: A survey of interdependent information security games. ACM Computing Surveys **47**(2), 23:1–23:38 (2014)
33. Laszka, A., Horvath, G., Felegyhazi, M., Buttyan, L.: FlipThem: Modeling targeted attacks with FlipIt for multiple resources. In: Proceedings of the 5th Conference on Decision and Game Theory for Security (GameSec), pp. 175–194 (2014)
34. Laszka, A., Johnson, B., Grossklags, J.: Mitigating covert compromises: A game-theoretic model of targeted and non-targeted covert attacks. In: Proceedings of the 9th Conference on Web and Internet Economics (WINE), pp. 319–332 (2013)
35. Laszka, A., Potteiger, B., Vorobeychik, Y., Amin, S., Koutsoukos, X.: Vulnerability of transportation networks to traffic-signal tampering. In: Proceedings of the 7th International Conference on Cyber-Physical Systems (ICCPS), p. 16. IEEE Press (2016)
36. Laszka, A., Zhao, M., Grossklags, J.: Banishing misaligned incentives for validating reports in bug-bounty platforms. In: Proceedings of the 21st European Symposium on Research in Computer Security (ESORICS), pp. 161–178 (2016)
37. Lee, R.M., Assante, M.J., Conway, T.: Analysis of the cyber attack on the Ukrainian power grid: Defense use case. Tech. rep., Electricity Information Sharing and Analysis Center (E-ISAC) (2016)
38. Manshaei, M.H., Zhu, Q., Alpcan, T., Bacşar, T., Hubaux, J.P.: Game theory meets network security and privacy. ACM Computing Surveys (CSUR) **45**(3), 25 (2013)
39. Mcmahan, H.B., Gordon, G.J., Blum, A.: Planning in the presence of cost functions controlled by an adversary. In: International Conference on Machine Learning, pp. 536–543 (2003)
40. Mitchell, R., Chen, I.R.: A survey of intrusion detection techniques for cyber-physical systems. ACM Computing Surveys **46**(4), 55 (2014)
41. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., et al.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529 (2015)
42. O'Donnell, A.J., Sethu, H.: On achieving software diversity for improved network security using distributed coloring algorithms. In: Proceedings of the 11th ACM Conference on Computer and Communications Security (CCS), pp. 121–131. ACM (2004)
43. Píbil, R., Lisỳ, V., Kiekintveld, C., Bošanskỳ, B., Pěchouček, M.: Game theoretic model of strategic honeypot selection in computer networks. In: Proceedings of the 3rd Conference on Decision and Game Theory for Security (GameSec), pp. 201–220. Springer (2012)
44. Polityuk, P.: Ukraine investigates suspected cyber attack on Kiev power grid. Reuters, `http://www.reuters.com/article/us-ukraine-crisis-cyber-attacks-idUSKBN1491ZF` (2016)
45. Ponemon Institute: 2017 cost of data breach study. Tech. rep., IBM (2017)
46. Rabin, T., Ben-Or, M.: Verifiable secret sharing and multiparty protocols with honest majority. In: Proceedings of the 21st Annual ACM Symposium on Theory of Computing (STOC), pp. 73–85. ACM (1989)
47. Schlenker, A., Thakoor, O., Xu, H., Tambe, M., Vayanos, P., Fang, F., Tran-Thanh, L., Vorobeychik, Y.: Deceiving cyber adversaries: A game theoretic approach. In: Proceedings of the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS) (2018)
48. Shu, R., Wang, P., Gorski III, S.A., Andow, B., Nadkarni, A., Deshotels, L., Gionta, J., Enck, W., Gu, X.: A study of security isolation techniques. ACM Computing Surveys (CSUR) **49**(3), 50 (2016)
49. Slay, J., Miller, M.: Lessons learned from the Maroochy water breach. In: E. Goetz, S. Shenoi (eds.) Critical Infrastructure Protection, pp. 73–82. Springer (2008)

50. Tambe, M. (ed.): Security and Game Theory: Algorithms, Deployed Systems, Lessons Learned. Cambridge University Press (2011)
51. Van Dijk, M., Juels, A., Oprea, A., Rivest, R.L.: FlipIt: The game of "stealthy takeover". Journal of Cryptology **26**(4), 655–713 (2013)
52. Vorobeychik, Y., An, B., Tambe, M., Singh, S.: Computing solutions in infinite-horizon discounted adversarial patrolling games. In: International Conference on Automated Planning and Scheduling (2014)
53. Vorobeychik, Y., Singh, S.: Computing stackelberg equilibria in discounted stochastic games. In: National Conference on Artificial Intelligence (2012)
54. Watkins, C.J., Dayan, P.: Q-learning. Machine Learning **8**(3-4), 279–292 (1992)
55. Zetter, K.: Hackers can mess with traffic lights to jam roads and reroute cars. WIRED, https://www.wired.com/2014/04/traffic-lights-hacking/ (2014)
56. Zetter, K.: Inside the cunning, unprecedented hack of Ukraine's power grid. WIRED, https://www.wired.com/2016/03/inside-cunning-unprecedented-hack-ukraines-power-grid/ (2016)
57. Zhang, M., Zheng, Z., Shroff, N.B.: A game theoretic model for defending against stealthy attacks with limited resources. In: Proceedings of the 6th Conference on Decision and Game Theory for Security (GameSec), pp. 93–112. Springer (2015)
58. Zhao, M., Grossklags, J., Liu, P.: An empirical study of web vulnerability discovery ecosystems. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS), pp. 1105–1117. ACM (2015)