Partially-Observable Security Games for Attack-Defence Analysis in Software Systems

Narges Khakpour[1] and David Parker[2]

[1] Newcastle University, Newcastle NE1 7RU, UK
narges.khakpour@newcastle.ac.uk
[2] University of Oxford, Oxford, OX1 2JD, UK
david.parker@cs.ox.ac.uk

Abstract. Given the presence of residual vulnerabilities in software systems, it is critical to apply suitable countermeasures in order to minimize the likelihood of an attack. In this paper we propose a formal approach, based on stochastic games, to threat analysis and synthesis of defence strategies for protecting systems with vulnerabilities. Crucially, we support analysis under partial observation, where some of the attacker's activities are unobservable or undetectable by the defender. We construct a one-sided partially observable security game and transform it into a perfect game, for which formal analysis is feasible. We prove that this transformation is sound for a sub-class of security games and a subset of objectives specified in the temporal logic rPATL. We implement our approach and evaluate it by applying it to a real-life example.

1 Introduction

In today's increasingly complex software systems, many vulnerabilities still remain in the system after being discovered due to factors such as the lack of a patch or knowledge to fix them, cost considerations, or organizational preferences for availability and usability over security [31]. Threat modelling is as a widely-used technique for specifying, discovering and analyzing security weaknesses in order to assess and manage the security of systems with vulnerabilities [18].

A wide range of models and languages has been proposed for threat modelling and analysis (see [18,22] for a survey). Attack graph-based [28] and attack tree-based [32] approaches are among the most common. While attack trees have received a lot of attention in the past, they are often constructed manually or semi-automatically, which is a challenge when developing automated online analysis techniques. In contrast, attack graphs can be generated fully automatically using formal approaches, such as model checking [28] or logic-based reasoning [27], making them a suitable candidate for automated analysis.

The opposing objectives and dependent strategies of attack and defence make them well-suited for *game-based* threat analysis, enabling us to model the interactions between the attacker and the defender to construct effective defences and assess threats. Game theory has been applied to analyze security in different research communities in recent years [2,13,14] (see [25] for a survey).

However, most of the research achievements in this area are based on the hypothesis that both players have full observation of the opponent's moves, which is unrealistic in practice. This is because it is not always possible to observe all the attacker's activities; many vulnerabilities and malicious activities cannot be detected due to factors such as the high overhead of monitoring, expensive security controls, or ineffective security mechanisms.

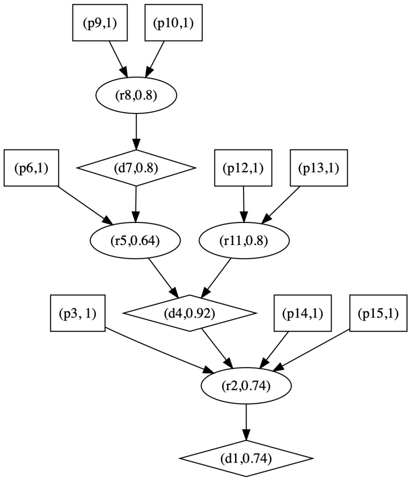
In this paper, we propose an approached based on *stochastic games*, under *partial observation*, for the analysis of attack-defence scenarios in systems with vulnerabilities and the automatic synthesis of defence strategies. We employ attack graphs for the automatic construction of attack scenarios, and build on stochastic modelling and verification with games [12] for security analysis, which provides formal assurance for systems operating in uncertain environments.

We define the semantics of attack graphs in terms of Symbolic Markov Decision Processes, which specify the attacker's behaviour. We specify defences as a set of rules, triggered as a reaction to an attack or as a preventive action to stop potential future attacks. A *fully observable* (also called *perfect*) stochastic two-player game is then constructed from the attacker's and defender's behaviour, in which each player has full observation of the other players' actions. This allows us to analyze attack-defence scenarios and synthesize defence strategies using existing formal verification methods and tools. We specify attacker or defender objectives in the temporal logic rPATL (Probabilistic ATL with Rewards) [12] and use the stochastic game verification tool PRISM-games [23].

To support reasoning under partial observation, we define a *partially observable one-sided security game* (PO-security game). This provides partial observation of the attacker's behaviour but, on the other hand, assumes that the attacker has full observation of the defender's actions. This type of game, where one player has full observation and the other has partial observation, is called a *one-sided partially observable game* [11]. Note that, if the defender wins in this one-sided game setting, they will also win in a setting where the attacker has only partial observation [10]. While the formal verification community has studied analysis under partial observation and presented theoretical results (e.g., [3, 8–10]), progress on developing practical methods for stochastic games under partial observation has been very limited, even for the simpler one-sided case [7].

To analyze a PO-security game, we transform it into a perfect game and then analyze the resulting model with PRISM-games. We prove that this transformation is sound for a subclass of security games called ODT (Observable Defence Triggers) security games and a subset of rPATL properties (called observable step-unbounded defence objectives). This means that a synthesized strategy for the defender in the resulting perfect security game is a valid strategy for the defender in the original PO-security game.

We implement a prototype tool to support the proposed approach and evaluate it through application to a real-life case study. Our experimental results demonstrate that transforming a PO-security game significantly enhances the performance of our analysis and can serve as a promising technique for reducing the state space when dealing with large attack graphs.



Node	Label
p3	plycent03_has_weak_cpu
p6	plycent02_access_ssh_plycent03
p9	has_access_to_plycent02
p10	vul_cve-2018-7566_exists_on_plycent02
p12	has_access_ssh_to_plycent03
p13	attacker_located_on_internet
p14	plycent03_offers_ssh
p15	vul_cve-2018-5390_exists_on_plycent03
d1	plycent03_is_down
d4	gained_ssh_access_to_plycent03
d7	has_root_exec_code_priv_on_plycent02
r2	perform_remote_DOS
r5	perform_multi-hop_access
r8	tamper_memory
r11	directly_access_to

Fig. 1: An Example Attack Graph

The contributions of this paper are: (i) constructing a security game from an attack graph and the defender's behaviour; (ii) introducing partially observable security games and an approach for their analysis; and (iii) applying it to a real-life case study. The paper is organized as follows. Section 2 presents preliminaries on stochastic games. Section 3 constructs a fully observable security game, and Section 4 discusses PO-security games. We present the evaluation in Section 5, discuss related work in Section 6, and conclude in Section 7. Formal proofs can be found in an extended version of this paper [20].

2 Preliminaries

2.1 Attack Graphs

An attack graph shows strategies that can be taken by an attacker to reach its *final goal*. It comprises three types of nodes: (a) *condition nodes*, depicted as rectangles, show the *primitive capabilities* that an attacker has, (b) *derived nodes*, represented by diamonds, are associated with *derived capabilities* gained through attack actions, and (c) *rule nodes*, represented by ellipses, show actions. Fig. 1 shows an attack graph where the label of a node is of the form (id, p): id is the node identifier and p is the probability of reaching that node. The table gives a description of each node identifier. When the prerequisites of a rule (i.e. its predecessor nodes) in the attack graph are satisfied, it is activated and leads to its consequence (i.e., its successor node) with a probability.

The attack graph in Fig. 1 describes an attack scenario where the attacker's goal is to take the system plycent03 down. The attacker is located on the internet and has direct access to the system plycent02, which has a vulnerability that can lead to memory tampering. They also have access to the SSH service offered by plycent03, which suffers from the vulnerability vul_cve-2018-5390 that can be exploited to perform a DoS attack. The attacker can directly access the SSH service on plycent03 (via the path r11 and r2), or indirectly via exploiting the vulnerability on plycent02 (via the path r8, r5, and r2). Once the attacker gains access to the SSH service of plycent03, they can exploit its vulnerability to perform a Denial of Service (DoS) attack.

From the system description and the attack goal, we generate an attack graph using MulVAL (Multihost, Multistage Vulnerability Analysis) [27], similar to [21]. MulVAL is a logic programming-based system to automatically build attack scenarios. The logical attack graph produced by MulVAL relies on the method proposed in [31] to calculate a cumulative security metric called AGP (Attack Graph-based Probabilistic Metric) that shows the likelihood of an attacker reaching a specific derived node in the attack graph from an entry point, considering the causality among attack steps. The probability of an attack depends on the attacker's resources, skills, attack success rate, etc. We use the scores provided by the Common Vulnerability Scoring System (CVSS)[3] and domain experts to initialize the scores for primitive and rule nodes. CVSS is a vulnerability database that provides different scores to measure the severity and exploitability of vulnerabilities.

2.2 Probabilistic Model Checking

Probabilistic model checking is an approach to formal verification and strategy synthesis for probabilistic systems, including for stochastic games [12, 29]. Let $\mathbb{P}_X$ be the set of all discrete probability distribution functions over the set X, $V = \langle v_1, \ldots, v_n \rangle$ be a vector of variables, $\mathcal{D}_{v_i}$ be the domain of a variable v_i, and $\mathcal{D}_V = \prod_{i=1}^{n} \mathcal{D}_{v_i}$. A *valuation* $\boldsymbol{\nu}$ of V is a tuple $\langle \boldsymbol{\nu}_1, \ldots, \boldsymbol{\nu}_n \rangle \in \mathcal{D}_V$, and we denote the i-th value of $\boldsymbol{\nu}$ by $\boldsymbol{\nu}(v_i)$ for $1 \leq i \leq n$.

Our modelling approach uses *Symbolic Markov Decision Processes* (SMDPs), a high-level description of a Markov Decision Process (MDP) in a similar form to the modelling language of the PRISM [24] and PRISM-games [23] tools.

Definition 1 (Symbolic Markov Decision Process). *A* Symbolic Markov Decision Process *(SMDP) is a tuple* $\mathrm{M} = \langle V, \boldsymbol{\nu}_0, \Sigma, \delta \rangle$ *where* $V = \langle v_1, \ldots, v_n \rangle$ *is a tuple of variables,* $\boldsymbol{\nu}_0 \in \mathcal{D}_V$ *gives the initial conditions on the variables,* Σ *is a finite non-empty set of actions, and* δ *is a finite set of probabilistic transitions* $\langle \phi, \sigma, \mathcal{P}_U \rangle$ *where* $\phi \subseteq \mathcal{D}_V$ *is a predicate on* V *which guards the transition,* $\sigma \in \Sigma$ *is an action and* $\mathcal{P}_U \in \mathbb{P}_U$ *is a discrete probability distribution over the set* U *of update functions of the form* $u : \mathcal{D}_V \mapsto \mathcal{D}_V$*, defined as a set of assignments.*

SMDP M starts in its initial state. A transition can fire if its guard is satisfied and, when fired, the variables are updated with the probabilities corresponding to its update functions. The semantics of M is an MDP $\overline{\mathrm{M}} = \langle V, S, s_0, \Sigma, \overline{\delta} \rangle$

[3] https://www.first.org/cvss/

where $S \subseteq \mathcal{D}_V$ is a set of states, $s_0 = \boldsymbol{\nu}_0$ is the initial state and $\overline{\delta} : S \times \Sigma \to \mathbb{P}_S$ is a (partial) transition function defined as follows:

$$\overline{\delta} = \{\langle\langle s, \sigma\rangle, f(U, s)\rangle \mid \exists \delta = \langle \phi, \sigma, \mathcal{P}_U\rangle \in \delta, s \models \phi\}$$
$$f(U, s) = \begin{cases} \langle t, p\rangle & u \in U, t = u(s), p = \mathcal{P}_U(u) \\ \text{undef} & \text{otherwise} \end{cases}$$

Definition 2 (Two-Player Stochastic Game). *Let $\overline{\mathrm{M}_1} = \langle V_1, S_1, s_{1,0}, \Sigma_1, \overline{\delta}_1\rangle$ and $\overline{\mathrm{M}_2} = \langle V_2, S_2, s_{2,0}, \Sigma_2, \overline{\delta}_2\rangle$ be two MDPs specifying two players' behaviour. A (turn-based) stochastic two-player game is a tuple $\mathcal{G} = \langle P, V, S, s_0, \Sigma, \mathsf{sch}, \Delta\rangle$ where $P = \{1, 2\}$ is the set of players, V is a vector of state variables over $V_1 \cup V_2 \cup \{t\}$, $S \subseteq \mathcal{D}_V$ is the set of states, s_0 is the initial state, $\Sigma = \Sigma_1 \cup \Sigma_2$ is the set of actions, $\mathsf{sch} : S \to P$ is a scheduler and $\Delta = \{\langle\langle \boldsymbol{\nu}, \sigma\rangle, \mathcal{P}_S\rangle\}$ is a (partial) transition function defined as follows.*

Let $\boldsymbol{\nu}_{\downarrow_{V_i}}$ be the projection of $\boldsymbol{\nu}$ onto the variables in V_i. Let $(\boldsymbol{\nu} \downarrow_{V_i})\backslash x$ denote a tuple obtained by setting the subtuple V_i in $\boldsymbol{\nu}$ to x, and $\overline{\delta}_i(x, \sigma)(x')$ represent the probability of a transition from state x to state x' with action σ according to $\overline{\delta}_i$. The probability of a transition from $\mathcal{P}_S$ in state $\boldsymbol{\nu}$ with the active player i to a state $\boldsymbol{\nu}'$ by performing action σ is:

$$\mathcal{P}_S(\boldsymbol{\nu}') = \begin{cases} p & \boldsymbol{\nu}_{\downarrow_{\{t\}}} = i \\ & \wedge\ \exists x.\ \overline{\delta}_i(\boldsymbol{\nu}_{\downarrow_{V_i}}, \sigma)(x) = p \wedge \boldsymbol{\nu}' = (((\boldsymbol{\nu} \downarrow_{V_i})\backslash x) \downarrow_{\{t\}})\backslash \mathsf{sch}(\boldsymbol{\nu}) \\ 0 & \text{otherwise} \end{cases}$$

Informally, the transition function in Definition 2 says that a state with active player i is updated to a new state $\boldsymbol{\nu}'$ in which the active player's state $\boldsymbol{\nu}_{\downarrow_{V_i}}$ is updated to x with probability p, the other player's local substate remains unmodified and the active player is updated according to the scheduler function. We represent the game by $\overline{\mathrm{M}_1} \mid\mid \overline{\mathrm{M}_2}$ and partition the game's states into sets S_1 and S_2, controlled by each of the two players. Furthermore, $\Sigma(s)$ denotes the actions that can be performed in state $s \in S$. We also annotate games with *reward structures* $\xi : \Sigma \to \mathbb{R}_{\geq 0}$, which can be used to model rewards or costs.

Given a game $\mathcal{G}$ and an objective ψ, the goal of strategy synthesis is to synthesize strategies for (coalitions of) players to be able to win the game. A strategy informally determines the optimal action that should be performed by the coalition in each state to achieve its objective. The objective ψ is specified using the temporal logic rPATL (probabilistic alternating-time temporal logic with rewards). This logic allows us to express quantitative properties of the games, e.g., to ensure that the probability of an event's occurrence meets some threshold (for example: "can the defender protect the network in such a way that the probability of the system being compromised by the attacker is less than 0.1?"). rPATL is a branching-time logic and contains state formulas ϕ and path formulas ψ. The syntax is given by the following grammar:

$$\phi ::= \top \mid \alpha \mid \neg\phi \mid \phi \wedge \phi \mid \langle\langle C\rangle\rangle \mathsf{P}_{\bowtie p}[\psi] \mid \langle\langle C\rangle\rangle \mathsf{R}^{\xi}_{\bowtie x}[\mathsf{F}\phi]$$
$$\psi ::= \mathsf{X}\,\phi \mid \phi_1 \mathsf{U}^{\leq k} \phi_2 \mid \phi_1 \mathsf{U} \phi_2$$

where α is an atomic proposition, C is a coalition of players, $\bowtie \in \{<, \leq, >, \geq\}$, $p \in [0,1]$, $x \in \mathbb{R}$, $k \in \mathbb{N}$ and ξ is a reward structure.

The operators X ("next"), $\mathsf{U}^{\leq k}$ ("bounded until"), U ("until") and F ("eventually") are the standard temporal operators. Informally, $\langle\langle C \rangle\rangle \mathsf{P}_{\bowtie p}[\psi]$ states that the coalition C has a strategy to ensure that the path formula ψ will be satisfied with a probability meeting the bound $\bowtie p$, regardless of the strategies of other players. Similarly, $\langle\langle C \rangle\rangle \mathsf{R}^{\xi}_{\bowtie x}[\mathsf{F}\phi]$ means that C has a strategy guaranteeing that the expected reward ξ accumulated until reaching a state where ϕ is true satisfies $\bowtie x$. We write $s \models \phi$ to indicate that state s of $\mathcal{G}$ satisfies the formula ϕ, and write $\mathcal{G} \models \phi$, if $s_0 \models \phi$. Further, $\mathsf{Sat}(\phi) = \{s \in S \mid s \models \phi\}$ denotes the set of states satisfying ϕ. We refer the reader to [12] for the full semantics of rPATL. In this paper, we use PRISM-games [23] to synthesize strategies for players according to objectives specified in rPATL.

3 Fully-Observable Security Games

We construct a perfect (fully observable) two-player stochastic security game between the attacker and defender, where the attacker's goal is to gain a specific capability, and the defender tries to prevent the attacker from doing so. In such a turn-based stochastic game, each player independently chooses an action in every round, and a player's action is fully observable by the opponent, i.e., the attacker and the defender have perfect observation of each other's moves.

Attacker Behaviour. We follow a similar approach to [30] to formalize an attack graph where the state of an attacker is represented using a set of Boolean variables, each corresponding to the attacker holding either a primitive capability (box node) or a derived capability (diamond node). Initially, the attacker only holds the primitive capabilities and has no derived capabilities. A transition corresponds to the attacker performing an action by applying a rule σ and, when it is triggered (i.e., the rule prerequisites hold and its consequence e does not hold), the attacker gains the new capability e with probability $p(\sigma)$.

Example 1. The attacker's behaviour in Fig. 1 contains four rule nodes, each of which is translated into a transition. For instance, the rule node $r2$ is translated to the transition $\langle \phi, \mathsf{remote\_DOS}, \mathcal{P}_U \rangle$ where:

$$\begin{array}{l} \phi = p3 \wedge d4 \wedge p14 \wedge p15 \wedge \neg d1 \\ \mathcal{P}_U(\{d1 \mapsto \top\}) = 0.74 \qquad \mathcal{P}_U(\emptyset) = 0.26. \end{array}$$

This states that, if the attacker has capabilities $\{p3, d4, p14, p15\}$ but not $d1$, they can acquire $d1$ with a probability of 0.74, or fail with a probability of 0.26.

Defender Behaviour. The choices of each player depend on the moves of the other, i.e., their capabilities and privileges may change as a consequence of their opponent's moves. For example, if an attacker gains full control over a system through a ransomware attack, the defender will lose access to that system. Similarly, if the defender modifies configurations to block service access on a victim, the attacker may adjust its strategy accordingly.

Therefore, an attacker's decision to target a system should account for potential defensive countermeasures, and vice versa. We define two types of defence: *reactive defence* and *proactive defence*. Reactive defences are typically responses to an ongoing attack, whereas proactive defences are preventive actions taken to thwart potential future malicious activities. An example of reactive defence includes updating permissions to block an attacker's access to a system upon detection of malicious activity by an intrusion detection system. An example of proactive defences is regular vulnerability scans on critical systems to detect and rectify potential weaknesses, proactively patching applications, and monitoring critical security controls. Defences performed by the defender respond to the attacker gaining new capabilities, with the aim of either revoking those capabilities or preventing the attacker from acquiring specific capabilities in the future.

We define *defence-triggering capabilities* as a set of attacker capabilities V_{AD} that the defender can observe and against which it can apply countermeasures. Therefore, the defender's state variables include the defence-triggering state variables of the attacker in addition to its own internal variables. Furthermore, the actions of the attacker and the defender should be distinct.

Definition 3 (Defender Behaviour). *Let* $\mathrm{AT} = \langle V_A, \boldsymbol{\nu}_{A,0}, \Sigma_A, \Delta_A \rangle$ *be the behaviour of the attacker, and* $V_{AD} \subseteq V_A$ *be the set of defence-triggering capabilities of the attacker. The behaviour of the defender is defined as* $\mathrm{DE} = \langle V_D, \boldsymbol{\nu}_{D,0}, \Sigma_D, \Delta_D \rangle$ *where* $V_{AD} \subseteq V_D$, $\boldsymbol{\nu}_{D,0} \implies \boldsymbol{\nu}_{A,0}$, *and* $\Sigma_D \cap \Sigma_A = \emptyset$.

Example 2. A reactive defence to the capability of executing code with root privilege by the attacker on the system plycent02 is to patch the vulnerability cve-2018-7566 (the rule node r8 in Fig. 1). This defence is specified as a transition $\langle \phi, \mathsf{patch\_vul{-}2018{-}7566}, \mathcal{P}_U \rangle$ where:

$$\begin{aligned}
&\phi = \mathsf{has\_root\_exec\_code\_on\_plycent02} \wedge \mathsf{vul\_vul{-}2018{-}7566\_exists\_on\_plycent02}\\
&u = \{\mathsf{has\_root\_exec\_code\_on\_plycent02} \mapsto \bot;\\
&\qquad \mathsf{vul\_vul{-}2018{-}7566\_exists\_on\_plycent02} \mapsto \bot\}\\
&\mathcal{P}_U(u) = 0.85\ ,\ \mathcal{P}_U(\emptyset) = 0.15
\end{aligned}$$

This states that, if the attacker can execute code with root privileges on plycent02, exploiting the vulnerability patch_vul−2018−7566, this defence rule eliminates the vulnerability and revokes its root privileges with probability 0.85. We assume that such defence probabilities are determined by a domain expert.

Security Game. Given the behaviours of the attacker and defender, specified as SMDPs, and a scheduler sch, we construct a security game $\mathcal{G} = \langle \{A, D\}, V, S_A \cup S_D, s_0, \Sigma_D \cup \Sigma_A, \mathsf{sch}, \Delta \rangle$ according to Definition 2, where $\{A, D\}$ is the set of players, S_A is the set of states controlled by the attacker, and S_D is the set of states controlled by the defender. In each state of this security game, each action $\sigma \in \Sigma$ (where $\Sigma = \Sigma_A \cup \Sigma_D$) yields at most one probabilistic transition.

The security game can further be associated with reward structures that assign rewards or costs to the states or transitions.

We use the probabilistic model checker PRISM-games to analyse various classes of properties for a single player or for both players, e.g., the likelihood or costs associated with achieving specific capabilities or goals by the attacker. Our primary objective is to prevent the attacker from reaching its targets, which can include intermediate or final objectives such as gaining control of a specific system to access sensitive information on another victim. PRISM-games can also synthesize a strategy $\Gamma : S \rightarrow \Sigma$ that determines the optimal action to be taken in each state by the player controlling it. We use strategy synthesis to develop defence strategies aimed at enhancing system protection.

Game Assumptions. We make the following assumptions in our security games. The game consists of one attacker and one defender. The attacker's goal is to obtain as many capabilities as possible, and the defender's goal is to prevent the attacker from acquiring more capabilities and to revoke their existing ones. If an attacker obtains a capability, they will hold it until it gets revoked by the opponent. An attacker tries an action if it can result in gaining new capabilities. The defender has the ability to check and observe the consequences of (observable) attacks immediately, i.e. we rely on intrusion exploitation systems and intrusion detection systems to detect and report (some of) the attacks immediately.

4 One-Sided Partially-Observable Security Games

In a perfect stochastic game, each round involves one player independently choosing an action that is fully observable by the opponent, i.e. each player has complete knowledge of the opponent's capabilities. Most security games also assume perfect observation [13, 14], meaning that players have full visibility of the game state and the opponent's actions. However, this assumption is not realistic in practice. Not all of the attacker's steps are observable by the defender, nor are all attacks easily detectable. Detecting some malicious activities and distinguishing them from normal operations can be very difficult or costly.

We instead assume that the attacker has full observation but the defender has partial observation, i.e., a *one-sided partially observable game* [11]. Note that if the defender wins in the one-sided setting, they will also win in a setting where the attacker has partial observation [10]. Similarly, if the attacker cannot win in this setting, they cannot win in a setting where the defender has full observation. We define a *partially observable stochastic two-player security game* as follows.

Definition 4 (One-Sided Partially-Observable Security Game). *Let* $P = \{A, D\}$*,* $i \in P$ *and* $\mathrm{M}_i = \langle V_i, \boldsymbol{\nu}_{i,0}, \Sigma_i, \delta_i \rangle$ *be two SMDPs representing the behaviour of the attacker and the defender. A (turn-based) stochastic two-player PO security game is a tuple* $\mathcal{G} = \langle P, V, S, \mathsf{Obs}, s_0, \Sigma, \mathsf{sch}, \Delta \rangle$ *where* V *is a vector defined over* $V_A \cup V_D \cup \{t\}$*,* $S \subseteq \mathcal{D}_V$ *is the set of states and* $\mathsf{Obs} : \mathcal{D}_V \rightarrow \mathcal{D}_{V_D}$ *is a function that defines the observation of the defender in a state. The initial state is* s_0 *such that* $s_0 \models \boldsymbol{\nu}_{A,0}$ *and* $\mathsf{Obs}(s_0) \models \boldsymbol{\nu}_{D,0}$*,* $\Sigma = \Sigma_A \cup \Sigma_D$ *and* $\mathsf{sch} : S \rightarrow P$ *is* *the scheduler. The transition $\Delta : S \times \Sigma \to \mathbb{P}_U$ is a (partial) transition function defined as follows, where $\mathsf{up}(\mathsf{u}, \sigma)$ is an update function:*

$$\Delta = \{\langle\langle s, \sigma\rangle, f(U, s, \sigma)\rangle | \exists\langle\phi, \sigma, \mathcal{P}_U\rangle \in \delta_A \wedge s \models \phi \vee \exists\langle\phi, \sigma, \mathcal{P}_U\rangle \in \delta_D \wedge \mathsf{Obs}(s) \models \phi\}$$
$$f(U, s, \sigma) = \begin{cases} \langle \mathsf{up}(u, \sigma)(s), p\rangle & u \in U, p = \mathcal{P}_U(u) \\ \text{undef} & \text{otherwise} \end{cases}$$

In this paper, we focus on a specific one-sided partially observable security game, defined as follows. We partition the actions of the attacker into two sets: *observable* and *unobservable* actions. The defender can observe only the observable actions of the attacker and their consequences. An attacker's action involves applying a rule to obtain a new derived capability; if the action is observable, the defender can also see the newly obtained capability. For instance, network traffic can be monitored to detect direct network access between hosts, whereas a DoS attack might go undetected if the host is not equipped with an appropriate IDS.

Therefore, we define the game state as $V = \langle v_{A,0}, \ldots, v_{A,n}, v_{D,0}, \ldots, v_{D,m}, t\rangle$, where $V_A = \langle v_{A,0}, \ldots, v_{A,n}\rangle$ represents the attacker's state, $V_D = \langle v_{D,0}, \ldots, v_{D,m}\rangle$ represents the defender's state, and t indicates the active player. The defender's observation in a game state $\boldsymbol{\nu}$ is defined as:

$$\mathsf{Obs}(\boldsymbol{\nu}) = \langle \boldsymbol{\nu}(v_{D,0}), \ldots, \boldsymbol{\nu}(v_{D,m}), t\rangle. \tag{1}$$

Note that V_{AD} is duplicated in V. Let $\overrightarrow{u}$ be a function that applies the updates of the variables in V_{AD} by u to the sub-states of both players. The function $\mathsf{up}(u, \sigma)$ applies the update of an unobservable action locally only to the attacker's sub-state. Otherwise, the updates should be applied to the both players' sub-states:

$$\mathsf{up}(u, \sigma) = \begin{cases} u & \sigma \notin \Sigma_O \cup \Sigma_D \\ \overrightarrow{u} & \text{otherwise} \end{cases} \tag{2}$$

where Σ_O gives the observable actions of the attacker.

Analyzing a Partially Observable Security Game. To analyze a partially observable (PO) security game, we transform it into a perfect security game and analyze the perfect game instead. If a derived capability is obtained through an observable action, it also becomes observable to the defender; otherwise, the update remains unobserved. If the defender knows the possible consequences of their observations, they can infer the attacker's next observable actions.

To achieve this, we need to identify the sets of observable prerequisites that can lead to an observable action being performed through a sequence of unobservable actions, ultimately resulting in the attacker achieving an observable derived capability. For instance, assume the defender cannot detect a memory tampering attack by the attacker (i.e., rule node r8 in Fig. 1 becomes unobservable). This means it will not be possible to detect whether the attacker can execute code with root privileges on `plycent02` (node d7). However, the defender knows that if the attacker holds the observable capabilities p9, p10 and p6, then it can ultimately perform the action in node r5 via performing r8 and then r5. Thus, the observable prerequisites of r5 become {p6,p9,p10}. There might be several sets of observable prerequisites that can lead to performing an observable action. For the algorithm to compute observable prerequisites, see [20].

We transform an attacker's behaviour into an abstract behaviour that hides the unobservable actions and only considers their impact on performing observable actions. Let $\mathsf{doPrequisites}(r)$ be a function that returns the set of all observable prerequisites of r. Each element dop of $\mathsf{doPrequisites}(r)$ represents the prerequisites of a specific path, and we associate a transition with each path. The guard of this transition is the conjunction of all observable capabilities in dop and the negation of r's consequence, i.e. this transition is triggered when all its observable prerequisites hold and r's consequence has not been achieved. This transition will set r's consequence with a probability that is the product of all the action probabilities along the path, i.e., this path can be taken if all the actions along it can be performed successfully.

Example 3. Let all the rule nodes in Fig.1 be unobservable, except for r2. The observable behaviour of the attacker is translated into two symbolic transitions, corresponding to the paths {r11, r2} and {r8, r5, r2}. The transition associated to {r11, r2} is $\langle p_{14} \wedge p_{15} \wedge p_{12} \wedge p_{13} \wedge p_3 \wedge \neg d_1, \texttt{path0}, \mathcal{P}_U \rangle$, where $U = \{\{d_1 \mapsto \top\}, \emptyset\}$, $\mathcal{P}_U(\{d_1 \mapsto \top\}) = 0.74 * 0.8 = 0.59$, and $\mathcal{P}_U(\emptyset) = 0.4$.

Definition 5 (Attacker's Observable Behaviour). *Let $\mathcal{R}$ be the set of the attacker actions, i.e., the set of rule nodes in the attack graph, and $\mathcal{C}$ and $\mathcal{H}$ be the set of observable primitive and derived capabilities, respectively. Let $\Sigma_O \subseteq \mathcal{R}$ be the set of observable actions and $p : \mathcal{R} \rightarrow [0, 1]$ be a function giving the probability of successfully applying action rules $r \in \mathcal{R}$. The observable behaviour of G is defined as* [AT] $= \langle V_O, \boldsymbol{\nu}_{O,0}, \Sigma_O, \delta_O \rangle$*, where V_O is a vector of variables defined on $\mathcal{C} \cup \mathcal{H}$. The initial conditions on the variables are given by: $\boldsymbol{\nu}_0 = \bigwedge_{\phi \in \mathcal{C}} \phi \wedge \bigwedge_{\phi \in \mathcal{H}} \neg\phi$. For each rule $r \in \Sigma_O$ and each of its distinct sets of observable prerequisites* $\mathsf{dop} \in \mathsf{doPrequisites}(r)$*, a transition $\langle \neg e \wedge \bigwedge_{d_i \in \mathsf{dop}} d_i, \mathsf{act}(\mathsf{dop}), \mathcal{P}_U \rangle \in \delta_O$ is defined, where* $\mathsf{act}(\mathsf{dop})$ *assigns an action label in Σ_O to* dop*, e corresponds to the successor derived node of r, $U = \{\emptyset, u\}$, $u = \{e \mapsto \top\}$, and:*

$$\mathcal{P}_U(u) = \prod_{r_i \in \mathsf{dop}} p(r_i), \ \mathcal{P}_U(\emptyset) = 1 - \mathcal{P}_U(u).$$

The reward structure for the attacker's actions is defined as $\xi'(r) = \sum_{r_i \in \mathsf{dop}} \xi(r_i)$.

We build and analyze a fully-observable security game, as given in Definition 5, instead of directly analyzing the partially observable security games constructed using the functions Obs and up defined in Equations 1 and 2. Let V'_{AD} be the attacker's copy of V_{AD} (i.e., the set of common variables between the attacker and the defender). The scheduler of the perfect game is defined as follows, where sch is the scheduler of the original PO security game:

$$\mathsf{sch}'(\boldsymbol{\nu}) = \mathsf{sch}(\boldsymbol{\nu}\downarrow_{(V_O \setminus V'_{AD}) \cup V_D \cup \{t\}}), \tag{3}$$

Table 1: Services and Vulnerabilities

Machine	Software/ Service	Vulnerability	Consequence
plyrhel01	http	cve-2018-1000120	Remote Code Execution
plycent01	Linux kernel	cve-2017-13215	Privilege Escalation
plycent02	postgress		Improper Authentication
plycent03	ssh	cve-2018-7566	Memory Tampering, Authentication Bypass
	http	cve-2018-1273	Remote Code Execution

where the scheduler of a state $\boldsymbol{\nu}$ is the value returned by the original scheduler sch for the corresponding state in $\boldsymbol{\nu}$ in the original game, essentially projecting onto the state variables of the attacker, the defender, and the variable t.

Soundness. An *ODT (Observable Defence-Triggering) security game* is a PO security game in which all the defence-triggering capabilities of the attacker are observable, i.e., $V_{AD} \subseteq V_O$. We prove that the transformation of an ODT security game, using Definition 4 and the functions Obs and up defined in Equation 1 and Equation, 2 to a perfect security game is sound for a subset of rPATL.

Definition 6. *A rPATL formula ϕ is an* observable step-unbounded defence objective*, if (i) it is defined over $V_O \cup V_D$ (i.e., it contains no unobservable variable), (ii) it contains no X or U^k operators, and (iii) in any sub-formula $\ll C \gg \mathsf{P}_{\bowtie\ p}[\psi]$, $C \subseteq \{D\}$.*

The results below state that an observable step-unbounded defence objective holds in an ODT security game defined in Definition 4, if and only if it is satisfied by a perfect game with the attacker behaviour obtained using Definition 5. Proofs of these results are in the extended version of this paper [20].

Theorem 1. *Let $\mathcal{G}$ be a one-sided partially observable security game and $[\mathcal{G}]$ be its transformation to a perfect game. For any observable step-unbounded defence objective ϕ, $\mathcal{G} \models \phi$ if and only if $[\mathcal{G}] \models \phi$.*

Theorem 2. *For an observable step-unbounded defence objective $\ll D \gg \mathsf{P}_{\bowtie q}[\psi]$ or $\ll D \gg \mathsf{R}_{\bowtie q}[\mathsf{F}\phi]$, PRISM-games synthesizes the same winning strategies for the defender in $\mathcal{G}$ and $[\mathcal{G}]$.*

5 Evaluation

Scenario. To evaluate our approach, we designed an attack scenario against a system shown in Fig. 2. Table 1 shows some of the services and software vulnerabilities present on each machine. In this scenario, the goal is to launch a DoS attack on the database server plycent02. The attacker is located on the internet and can access web pages served by the HTTP server hosted on plycent01. Dotted arrows show the sequence of service accesses made by the attacker to reach its final goal. The attack graph generated for this scenario using MulVAL comprises 50 nodes and 18 attack steps. We use the security metrics AGP provided by MulVAL, CVSS, and expert knowledge to determine the probability of attacker actions. The countermeasures considered in this attack scenario include stopping a service, limiting access to a service, or patching vulnerabilities.

We define two reward structures: (i) aCosts, which models the costs of an attack step for the attacker; and (ii) dCosts, which captures the cost of defence in addition to the damages an attacker can cause to the system by a successful attack step. To estimate the costs in aCosts, we considered several factors, including the time needed to perform the attack, ease of exploitation, and tools and requirements for performing the attack. To estimate the damage of an attack step (costs imposed on the defender), we use the impact score provided by the CVSS database, quantifying the outcome that an attacker can achieve as a result of exploiting the vulnerability, as well as business security preferences.

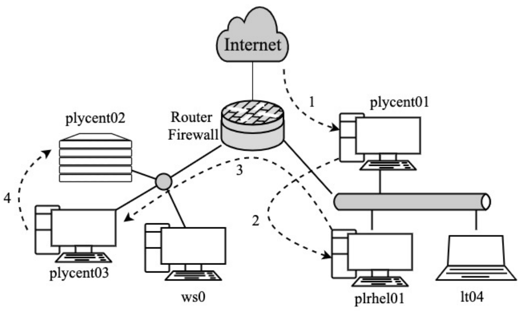


Fig. 2: The architecture and attack steps

We considered three cases of systems that can be developed with the architecture presented in Fig. 2: a tax system, an image hosting system, and a hotel booking system. Each has its own requirements and preferences in terms of service availability, confidentiality, and integrity of information. For the tax system, the integrity of information and the availability of the system during operation are of utmost importance, making the cost of taking any service down very high. Patching, on the other hand, is of minor importance. In the image hosting system, availability is important, but integrity and confidentiality are less important. In the hotel booking system, all three aspects—availability, confidentiality, and integrity—result in some loss of revenue and are therefore important. The cost of a defence varies from 0 to 500.

Results. We compute, with PRISM-games, the minimum or maximum probability of satisfying a property ψ that can be guaranteed by a coalition C ($\langle\langle C\rangle\rangle \mathsf{P}_{\mathsf{opt}}\ [\psi]$, $\mathsf{opt} \in \{min, max\}$). The maximum or minimum reward can be computed similarly. We analyzed several properties for each game:

(1) The minimum cost (defence cost and damages caused by the attacker) to defend against the attack and block it: $\langle\langle \mathsf{def}\rangle\rangle \mathsf{R}^{\mathsf{dCosts}}_{\mathsf{min}=?}\ [\mathsf{F\ attackerBlocked}]$.

(2) The maximum cost for the attacker to launch a DoS attack against the web server plycent02: $\langle\langle \mathsf{def}\rangle\rangle \mathsf{R}^{\mathsf{aCosts}}_{\mathsf{max}=?}\ [\mathsf{F\ DoS(plycent02)}]$.

(3) The maximum probability with which the defender successfully blocks the attack: $\langle\langle \mathsf{def}\rangle\rangle \mathsf{P}_{\mathsf{max}=?}\ [\mathsf{F\ attackerBlocked}]$.

(4) The minimum probability which which the attacker achieves its final goal: $\langle\langle \mathsf{att}\rangle\rangle \mathsf{P}_{\mathsf{min}=?}\ [\mathsf{F\ DoS(plycent02)}]$.

We carried out experiments with different schedulers. The attacker performs 14 different types of actions in total, and we considered three different schedulers that return control to the defender after an action of a specific type, e.g. when a suspicious log-in occurs or traffic is observed. Table 2 shows a partial analysis of the results. The first column shows the number of attacker action types after which control returns to the defender. For instance, a "2-actions" scheduler means that the attacker performs two actions, after which control is returned to the defender to apply a defense if necessary. The 'Model Size' column indicates the number of states and transitions of the security game for different systems.

The results show that, with an increase in the number of actions to which the defender can react, the size of model increases. This is expected, as more transitions from the defender will be included in the model and this will increase the state space. In addition, the minimum cost (both attacker damage and defence costs) imposed on the system decreases with an increase in the number of attacker actions to react to, e.g., the minimum cost to defend against the attack in the tax system changes from 732 units to 205 when the number of actions with a defence reaction changes from 2 to 6. This is because the defender has a better chance of applying an effective countermeasure if control is returned more often. Furthermore, the minimum cost for the attacker to reach its final goal increases slightly: with an increase in the defence actions, it might become more challenging for the attacker to succeed and it would need to try different scenarios. In a similar fashion, the probability that the defender succeeds in blocking the attack increases, e.g., from 0.31 with two action types to 0.68 for the case of 6 action types.

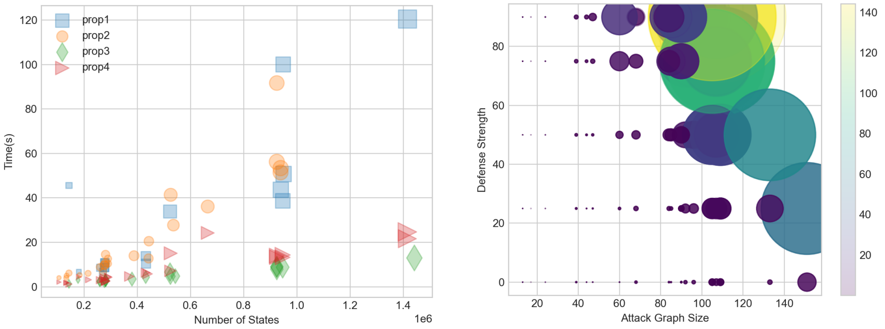
Table 2: Results for varying scenarios

Scheduler	**Model size**	**Property**	**Case**	**Result**
2-actions	17.6K/ 47K	(1)	Tax System	732
			Image Hosting	732
			Hotel Reservation	732
		(3)	-	0.31
4-actions	47K/ 121k	(1)	Tax System	277
			Image Hosting	266
			Hotel Reservation	318
		(3)	-	0.67
6-actions	97K/ 242K	(1)	Tax System	205
			Image Hosting	185
			Hotel Reservation	245
		(3)	-	0.68

Performance Analysis. To evaluate the performance of our approach, we conducted experiments on a real-life network system comprising 16 hosts offering different services. We scanned the hosts and identified 155 types of software vulnerabilities, all reported to the CVSS database during the period of 2017-2018. All experiments were run on a machine with a 2.9 GHz Intel Core i7 CPU and 16 GB RAM, running macOS Catalina. We conducted experiments with different sets of observations, vulnerabilities, defence strategies, and objectives on a real-life case study. To test the implementation of our prototype, we ran approximately 1241 different experiments. Sample model files are available.[4]

The size of the game model depends on the size of the attack graph as well as the defences. *Defence strength* indicates the percentage of possible defences applied in the system. As the size of the attack graph and the defence strength

[4] https://www.prismmodelchecker.org/files/sefm24/



(a) (b)

Fig. 3: (a) Model checking time for varying game sizes. (b) Games size for varying defence strength and attack graph size (circle colour/size indicates the number of states/transitions, in the ranges $(0, 1.4 * 10^6)$ and $(0, 14.5 * 10^6)$, respectively).

increase, the number of states and transitions increase accordingly, as illustrated in Figure 3(b). In this figure, colour represents the number of states, and the marker size denotes the number of transitions at each point. For larger attack graphs with high defence strength, we encountered the state explosion problem. Figure 3(a) shows the model checking time for four different properties based on the number of model states. The marker size represents the number of transitions at each point. The model checking time ranged from a few milliseconds for smaller models to approximately 120 seconds for a model with around 1.4×10^6 states, and generally increases with the size of the model.

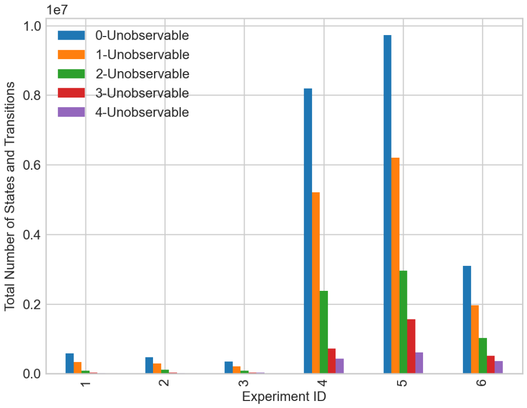


Fig. 4: Model Size vs Partial observation

The number of transitions in a security game under perfect and partial observation for six different experiments is presented in Fig. 4. The x-axis shows the experiment ID, and the y-axis the sum of the number of transitions and states for different observations. In these experiments, the number of attacker action types ranged from 9–16. We ran each experiment under five different observation conditions, including full observation (Observable), and with n unobservable actions, for $1 \leq n \leq 4$. In this figure, $n-$Unobservable means that n actions of the attacker have become unobservable. Actions that are challenging to definitively associate with malicious activity are considered unobservable.

In some experiments, while the number of state variables remained fixed (because the consequences of unobservable actions were partially observable, i.e., neither fully observable nor fully unobservable), the model size was reduced significantly. With an increase in the number of unobservable actions, the number of transitions decreased accordingly. For instance, in experiment 5, the model size for the case of full observation is approximately 9.7M, while it reduces to approximately 6.2M, 2.9M, 1.5M and 0.6M for 1 to 4 unobservable actions, respectively. Our experiments show that this transformation can be used as a state reduction technique to tackle the complexity of large-scale attack graphs. We can transform a perfect security game with a large state space into a smaller game, where some sub-scenarios that do not trigger any defence are abstracted away by being considered as a single step. This results in a more abstract game where the internal details of the attacker's behaviour are hidden, and only their interactions with the defender are taken into account.

6 Related Work

Games In Security. A recent survey by Hunt et al. [19] reviews research on attacker-defender games, but it focuses solely on protecting physical assets and does not address cybersecurity. The authors in [14] proposed a game-theoretic approach for a network hardening problem where the attacker plan is represented as an attack graph and the defender seeks an optimal solution to deploy honeypots into the network to deceive the attacker. A two-player strategic game based on an extension of attack trees was constructed in [4], where the defender wants to protect the system by buying countermeasures and the attacker tries to exploit the vulnerabilities and gain some profit. The game is solved by finding the Nash equilibria, with the goal of finding the most promising attack/defence actions. Chowdhary et al. [13] use attack graphs to build a zero-sum stochastic Markov game between the attacker and the defender to help the administrator apply countermeasures in a cloud network.

None of the above approaches support partial observations. The authors in [1] proposed a method for cyber-deception using game theory and reinforcement learning, where the defender has partial observations of the attacker's actions. In contrast to [1,13,14] that are application-specific, our approach can be applied to any type of attacks and defences. We believe the runtime for our method may be faster than for [14]: we analyze a system with 53 vulnerabilities in under 1 second, whereas [14] needs up to 1000 seconds to analyze a network with 8 types of vulnerabilities. But further investigation is needed to confirm this.

In [2], an extension of attack/defence trees with temporal ordering of actions is used to build a two-player stochastic game between the attacker and the defender. They use probabilistic model checking techniques to formally verify security properties of the attack-defence scenarios, and synthesize strategies for attackers or defenders using PRISM-games. A similar approach is taken in [15], but with some relaxed model assumptions and some additional optimizations. Gadyatskay et al. define a stochastic timed semantics for attack-defence trees in terms of a network of timed automata in [16]. They use the UPPAAL model checker to perform a quantitative analysis and find attacker parameters that minimize cost. In contrast to [2,15], we use attack graphs that can be generated fully automatically and we support partial observation, while automatic construction of attack trees is still a major challenge despite the recent efforts [33]. Our security game is boolean which makes it more scalable for analysis of realistic network systems, as confirmed by our experiments results.

Analysis under Partial Observation. Partially observable Markov decision processes (POMDPs) extend MDPs with partial observability and have many applications in fields such as AI, scheduling and planning. The formal verification community has studied POMDPs and presented both theoretical results, e.g. [9,10] and practical methods, e.g. [5,6,26], the latter being implemented in the popular model checkers PRISM [24] and Storm [17].

In [34], the problem of motion planning, that is often modeled as a POMDP, is reduced to a two-player stochastic game and solved using PRISM-games. The problem of controller synthesis for timed games under state-based partial observation has been studied in [8] and implemented in UPPAAL-TIGA [3]. Skandylas et al. [30] proposed an approach to synthesize countermeasures for component-based software systems, considering both the structure and the behaviour of the system. We can only reason about the system behaviour. However, in contrast to our work, they do not support partial observation.

7 Conclusions

In this paper, we propose a game-based approach for the analysis of attack graphs under partial observation. We construct a two-player security game from an attack graph and the defender's behaviour, and use PRISM-games to analyze the game and synthesize strategies for the players. To support defence under partial observation, we introduce one-sided partially observable security games and present an algorithm to transform them into perfect games, proving soundness for a subclass of security games and objectives. Our experiments show that the transformation algorithm is a promising state reduction technique to handle large-scale attack graphs. We plan to integrate our approach into a self-protection engine to support online defence, and also to extend our experiments to study the impact of different factors on the effectiveness of the defence.

Acknowledgements. This work was funded in part by the Swedish Knowledge Foundation (No. 20160186) and the UK Engineering and Physical Sciences Research Council (EP/X037274/1).

References

1. A. H. Anwar, C. A. Kamhoua, N. O. Leslie, and C. Kiekintveld. Honeypot allocation for cyber deception under uncertainty. *IEEE Trans. Netw. Serv. Manag.*, 19(3):3438–3452, 2022.
2. Z. Aslanyan, F. Nielson, and D. Parker. Quantitative verification and synthesis of attack-defence scenarios. In *Proc. CSF'16*, pages 105–119, 2016.
3. G. Behrmann, A. Cougnard, A. David, E. Fleury, K. G. Larsen, and D. Lime. UPPAAL TIGA user-manual. *Aalborg University*, 2007.
4. S. Bistarelli, M. Dall'Aglio, and P. Peretti. Strategic games on defense trees. In *Formal Aspects in Security and Trust*, pages 1–15, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.
5. A. Bork, S. Junges, J. Katoen, and T. Quatmann. Verification of indefinite-horizon POMDPs. In *ATVA*, volume 12302 of *Lecture Notes in Computer Science*, pages 288–304. Springer, 2020.
6. A. Bork, J. Katoen, and T. Quatmann. Under-approximating expected total rewards in POMDPs. In *TACAS (2)*, volume 13244 of *Lecture Notes in Computer Science*, pages 22–40. Springer, 2022.
7. S. Carr, N. Jansen, S. Bharadwaj, M. T. Spaan, and U. Topcu. Safe policies for factored partially observable stochastic games. In *Robotics: Science and System XVII*, 2021.
8. F. Cassez, A. David, K. G. Larsen, D. Lime, and J. Raskin. Timed control with observation based and stuttering invariant strategies. In *Proc. ATVA'07*, pages 192–206, 2007.
9. K. Chatterjee, M. Chmelik, and M. Tracol. What is decidable about partially observable Markov decision processes with omega-regular objectives. In *Proc. CSL'13*, pages 165–180, 2013.
10. K. Chatterjee and L. Doyen. Partial-observation stochastic games: How to win when belief fails. *ACM Trans. Comput. Log.*, 15(2):16:1–16:44, 2014.
11. K. Chatterjee, L. Doyen, and T. A. Henzinger. A survey of partial-observation stochastic parity games. *Formal Methods in System Design*, 43(2):268–284, 2013.
12. T. Chen, V. Forejt, M. Kwiatkowska, D. Parker, and A. Simaitis. Automatic verification of competitive stochastic systems. *Formal Methods in System Design*, 43(1):61–92, 2013.
13. A. Chowdhary, S. Sengupta, A. Alshamrani, D. Huang, and A. Sabur. Adaptive MTD security using Markov game modeling. In *International Conference on Computing, Networking and Communications, ICNC 2019, Honolulu, HI, USA, February 18-21, 2019*, pages 577–581, 2019.
14. K. Durkota, V. Lisý, B. Bosanský, and C. Kiekintveld. Optimal network security hardening using attack graph games. In *Proc. IJCAI'15*, pages 526–532, 2015.
15. J. Eisentraut and J. Kretinsky. Expected cost analysis of attack-defense trees. In *Proc. QEST'19*, volume 11785 of *LNCS*, pages 203–221. Springer, 2019.
16. O. Gadyatskaya, R. R. Hansen, K. G. Larsen, A. Legay, M. C. Olesen, and D. B. Poulsen. Modelling attack-defense trees using timed automata. In *Proc. FORMATS'16*, pages 35–50, 2016.
17. C. Hensel, S. Junges, J. Katoen, T. Quatmann, and M. Volk. The probabilistic model checker storm. *Int. J. Softw. Tools Technol. Transf.*, 24(4):589–610, 2022.
18. J. B. Hong, D. S. Kim, C.-J. Chung, and D. Huang. A survey on the usability and practical applications of graphical security models. *Computer Science Review*, 26:1 – 16, 2017.

19. K. Hunt and J. Zhuang. A review of attacker-defender games: Current state and paths forward. *Eur. J. Oper. Res.*, 313(2):401–417, 2024.
20. N. Khakpour and D. Parker. Partially-observable security games for attack-defence analysis in software systems. *CoRR*, abs/2211.01508, 2022.
21. N. Khakpour, C. Skandylas, G. S. Nariman, and D. Weyns. Towards secure architecture-based adaptations. In *Proceedings of the 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS@ICSE 2019, Montreal, QC, Canada, May 25-31, 2019*, pages 114–125, 2019.
22. B. Kordy, L. Piètre-Cambacédès, and P. Schweitzer. DAG-based attack and defense modeling: Don't miss the forest for the attack trees. *Computer Science Review*, 13, 03 2013.
23. M. Kwiatkowska, G. Norman, D. Parker, and G. Santos. PRISM-games 3.0: Stochastic game verification with concurrency, equilibria and time. In *Proc. 32nd International Conference on Computer Aided Verification (CAV'20)*, volume 12225 of *LNCS*, pages 475–487. Springer, 2020.
24. M. Z. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *CAV*, volume 6806 of *Lecture Notes in Computer Science*, pages 585–591. Springer, 2011.
25. M. H. Manshaei, Q. Zhu, T. Alpcan, T. Basar, and J. Hubaux. Game theory meets network security and privacy. *ACM Comput. Surv.*, 45(3):25:1–25:39, 2013.
26. G. Norman, D. Parker, and X. Zou. Verification and control of partially observable probabilistic systems. *Real-Time Systems*, 53(3):354–402, 2017.
27. X. Ou, S. Govindavajhala, and A. W. Appel. Mulval: A logic-based network security analyzer. In *Proceedings of the 14th Conference on USENIX Security Symposium - Volume 14*, SSYM'05, pages 8–8, Berkeley, CA, USA, 2005. USENIX Association.
28. O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. M. Wing. Automated generation and analysis of attack graphs. In *Proceedings 2002 IEEE Symposium on Security and Privacy*, pages 273–284. IEEE, 2002.
29. A. Simaitis. *Automatic Verification of Competitive Stochastic Systems.* PhD thesis, Department of Computer Science, University of Oxford, 2014.
30. C. Skandylas, N. Khakpour, and J. Cámara. Security countermeasure selection for component-based software-intensive systems. In *22nd IEEE International Conference on Software Quality, Reliability and Security, QRS 2022, Guangzhou, China, December 5-9, 2022*, pages 63–72. IEEE, 2022.
31. L. Wang, T. Islam, T. Long, A. Singhal, and S. Jajodia. An attack graph-based probabilistic security metric. In *Data and Applications Security XXII, 22nd Annual IFIP WG 11.3 Working Conference on Data and Applications Security, London, UK, July 13-16, 2008, Proceedings*, pages 283–296, 2008.
32. J. D. Weiss. A system security engineering process. In *Proceedings of the 14th National Computer Security Conference*, volume 249, pages 572–581, 1991.
33. W. Wideł, M. Audinot, B. Fila, and S. Pinchinat. Beyond 2014: Formal methods for attack tree–based security modeling. *ACM Comput. Surv.*, 52(4):75:1–75:36, Aug. 2019.
34. L. Winterer, S. Junges, R. Wimmer, N. Jansen, U. Topcu, J. Katoen, and B. Becker. Motion planning under partial observability using game-based abstraction. In *56th IEEE Annual Conference on Decision and Control, CDC 2017, Melbourne, Australia, December 12-15, 2017*, pages 2201–2208, 2017.