A Formal Approach to Attack Graphs

Davide Catta[1,3*], Jean Leneutre[2], Vadim Malvone[2], Aniello Murano[1]

[1] Università degli studi di Napoli, Federico II, Naples, Italy.
[2] Télécom Paris, Institut Polytechnique de Paris, Palaiseau, France.
[3] LIPN, CNRS UMR 7030, Université Sorbonne Paris Nord,, Paris, France.

*Corresponding author(s). E-mail(s): catta@lipn.univ-paris13.fr;

Abstract

An attack graph is a concise portrayal of the various paths within an open system that enable an attacker to reach a prohibited state (such as gaining access to a restricted resource), despite the system's preventive measures. The assessment of system vulnerability involves examining the presence of such paths. In this work, we analyze attack graphs using a game-theoretic approach. Specifically, we introduce a well-suited game model that represents the dynamics between the system and the attacker, and propose an automata-based solution to demonstrate the absence of vulnerability.

Keywords: Attack Graphs, Game Theory, Automata Theoretic Approach, Imperfect Information

1 Introduction

The inherent complexity of modern systems comes at a cost: as they become more complex and intelligent, ensuring their security becomes increasingly challenging. When dealing with security, the motto "Better safe than sorry" holds true. This is because the cost of repairing a system flaw during maintenance is at least two orders of magnitude higher, compared to fixing it at an early design stage. Consequently, to develop a secure system, tools that can detect vulnerabilities and unexpected behaviors at the earliest stages of their life cycle are essential. *Formal methods* have been a success story in checking systems' reliability [1]. They allow for verifying whether a system is correct by formally checking if a mathematical model of it meets a formal representation of its desired behavior.

Recently, classic approaches such as model checking and automata-theoretic techniques, originally developed for monolithic systems [2, 3], have been significantly extended to handle *reactive* and *multi-agent systems* [4–8].These systems encapsulate the behavior of two or

more rational agents interacting in a cooperative or adversarial manner, aiming at a designed goal [9].

In system security checking, a malicious attack can be viewed as an attempt by an attacker to gain unauthorized access to resources or compromise the system's integrity. In this context, *attack graph* [10] is one of the most prominent attack models developed, receiving much attention in recent years. An attack graph represents a state where each node corresponds to an attacker at a specific network location, and edges represent state transitions, i.e., attack actions by the attacker. It is the system's duty to prevent unauthorized accesses from the attacker in each state of the graph.

To build attack graphs assumptions must made about the attackers profile (defined by their motivations, resources, technical abilities, knowledge about the system under attack ...). When considering critical systems with sensible assets, the worst-case scenario is usually considered, that is the strongest possible attacker profile. In term of game theory modeling, it means that the corresponding player will have perfect information on the game.

These attack graphs can then be leveraged to design new cybersecurity policies, in particular cybersecurity reaction policies when facing an ongoing attack. In this latter case, it is crucial to be able to take into account the uncertainty about the observation of an ongoing attack progress (it is not always possible to exactly know the state of the attacker in the system). In term of game theory modeling, it becomes interesting to consider a defender player with imperfect information on the game.

In this paper, we introduce a novel approach to reasoning about attack graphs using a game-theoretic approach and an automata-based solution to evaluate system reliability. We first establish a two-player turn-based reachability game between the system defender and the potential attacker, where the attacker moves along adjacent states within the attack graph and the defender takes countermeasures to inhibit attacks. We demonstrate how simple attack graphs can be reduced to such game models. We then construct a finite tree automaton that accepts all the walking trees that allow the defender to prevent the attack from reaching designated states, regardless of the attacker's behavior. By checking the non-emptiness of the automaton, we show the system's robustness, i.e., the absence of bad paths in the attack graph. Notably, the construction of the automaton and its emptiness check can be performed in linear time.

Subsequently, we consider a novel and interesting extension of attack-graphs in which edges convey numerical information. This information expresses the cost for a malicious user to make an attack. We provide a reduction between this latter class of attack graphs and the previously introduced game.

Finally, we introduce a notion of imperfect information in attack graphs. Some arcs of the attack graph will be in an equivalence relation, representing attacks that are indistinguishable by the defender. We demonstrate a model reduction between attack graphs with imperfect information and a particular form of two-player turn-based games with imperfect information, showing how such games can be solved using existing techniques [11].

The novelty and importance of employing an automata-theoretic approach lie in its ability to provide a rigorous and formal method for security verification[12–14]. Automata-theoretic techniques enable precise modeling of system behaviors and the interactions between defenders and attackers. This approach allows for the systematic exploration of all possible attack scenarios and defense strategies, ensuring comprehensive coverage of potential security

threats. Furthermore, the use of finite tree automata facilitates efficient computation, making it feasible to apply these techniques to real-world systems. By leveraging automata theory, we can achieve a higher level of assurance in the security and reliability of complex systems, addressing challenges that traditional methods may overlook.

Additional material with respect to prior publications. Part of the present work already appears in [15]. Unlike previous work, we focus on defender strategies instead of attacker strategies. Additionally, we introduce a quantitative version of the games presented earlier (see Section 4), and games with imperfect information (see Section 5).

Outline. In section 2, we present related works. In section 3 we formally introduce attack graphs and two-player turn-based games, showing a reduction from the former to the latter. We also show how to represent defender strategies via trees and a tree automaton accepting all such trees, which is used to prove whether the defender has a winning strategy. In section 4 we introduce attack graphs with quantitative information and show how to reduce this model to the same game model introduced in section 3. In section 5 we further extend our attack graph model by introducing imperfect information and show how such model can be reduced to a two-player game with imperfect information. Finally, section 6 concludes the paper by also presenting future directions.

2 Related work

Attack graphs generators

The practical study on attack graphs mainly refers to "non model-based" approaches, with few exceptions. The authors in [16] introduce A2g2v, a model checker that generates attack graphs and detects an attack sequence by means of a counterexample. The authors in [17] introduce a model checker for vulnerability analysis via attack graphs; it uses the verification tool SMV [18], so it can only show one attack (counterexample) at the time. Differently, [19] uses a modified version of the tool NuSMV [20] to represent all possible attacks. The authors in [21] introduce MulVAL, an attack graph generation and network security-analyzer tool based on logical programming; it reduces the bottleneck of the state-explosion problem by making use explicitly of the logical dependencies between attack goals and configuration information. Most of the existing attack-graph tools are brute-force forward-search based, which is a huge limitation in practice. Conversely, our automata-based approach allows checking convoluted security properties, including liveness and regular behaviors [22], useful to specify service guarantees against real malicious activity.

Analysis of attack graphs

Besides the problem of attack graph generation, a large body of works on attack graphs proposes methods to analyze them, as surveyed in [23] and [24]. These methods can be roughly divided into two groups: the risk assessment methods aiming at analyzing the attacker's behavior and the risk treatment methods aiming at deploying new security countermeasures. For instance, in the first group, [25] considers an hybrid attribute based attack graph (i.e an attack graph extended with continuous and discrete variables such as clocks), transforms

it into a timed automata, and checks reachability properties in the *Metric Interval Temporal Logic (MITL)*. The second type of methods targets a *security hardening* of the system by adopting an optimal security policy to improve security. Since repairing all vulnerabilities may be infeasible, these methods propose to remove some appropriate vulnerabilities (or deploy new security countermeasures) to minimize the impact of attack under a given defense cost threshold. Several algorithms to find optimal security hardening policies for attack graphs have been proposed: [26–28]. These works consider a static view of cybersecurity corresponding to a prevention approach. A more dynamic view or reaction approach is needed when facing attacks: given an action of the attacker, which countermeasure the defender must deploy in priority to minimize the risk on the system.

Attack graphs and game theory

To achieve previous goal, several existing works have proposed different game-theoretic solutions for finding an optimal defense policy based on attack graphs. Most of these approaches do not use formal verification to analyze the game, but rather try to solve them using analytic and optimization techniques. The goal is to find the Nash equilibrium in order to characterize the most promising attacker/defender's actions. The works in [29, 30] study the problem of hardening the security of a network by deploying honeypots to the network to deceive the attacker. They model the problem as a Stackelberg security game in which the attack scenario is represented using attack graphs. The authors in [31] tackle the problem of allocating limited security countermeasures to harden security based on attack scenarios modeled by Bayesian attack graphs using partially observable stochastic games. They provide heuristic strategies for players and employ a simulation-based methodology to evaluate them. The work in [32] proposes an approach to select an optimal corrective security portfolio given a probabilistic attack graph. They define a Bayesian Stackelberg game that they solve by converting it into Mixed-Integer Conic Programming (MICP) optimization problem.
Notice that, all the approaches mentioned above, are customized by the specific problem under exam; in other words every problem has its own approach. Conversely, the approach we propose in this paper, by means of tree automata, is general and can handle any problem by means of a unique tool. This offer an unified way of analysis for different settings.

Attack graphs and formal methods

The work in [33] shares some ideas with our approach. However, they use a timed-logic framework and timed games to express and evaluate network security properties, which result in an EXPTIME-complete procedure. In [34], the authors define an attack / defense stochastic game under partial observation based on an attack graph. This game is transformed into a perfect game and uses PRISM-games to analyze the perfect security game, and the properties to be checked are specified into a subset of the temporal logic rPATL (Probabilistic ATL with Rewards) [35]. This approach requires to consider a quantitative extension of attack graphs including a metric to express the probability that a specific node in the attack graph is reached during an attack. Such metric being difficult to define in an accurate manner, our work focuses on the symbolic part of attack graphs, leaving quantitative part for future work.

Formal methods and bounded resources

Resource consumption in games is a well established area of research. Energy games [36], are two player games played on multi-weighted graphs. In such graphs, the set of states is partitioned into Player 1 states and Player 2 states, and each transition consumes, or produces, a finite amount of resources (represented as a finite numerical vector of fixed length). Player 1 wins if, given an initial amount of resources b, he can play forever while keeping b above 0. In this case, Player 1 has a winning strategy. The problem of determining, given an energy game and an initial amount of resources b, whether there is a winning Player 1 strategy for the game, was shown to be decidable in 2EXPTIME in [37]. Games with resource constraints have also been studied in a logical context. Versions of ATL where models are the concurrent version of multi-weighted graphs, are also discussed in the literature. These logics model different approaches to resource use. In some of them [38], transitions only consume resources, while in others transitions consume and produce resources [39]. Resource production can be bounded or unbounded, *i.e.*, the amount of resources produced by transitions can have a max or not. To our knowledge, with the only exception of [40], unbounded production of resources lead to the undecidability of the model checking problem for such logics.

Formal methods and imperfect information

In the above mentioned approaches, it is assumed perfect information of the players/agents. However, in real-life scenarios it is common to have situations in which agents have to play without having all relevant information at hand. In computer science these situations occur for example when some variables of a system are internal/private and not visible to an external environment [41]. In game models, the imperfect information is usually modelled by setting an indistinguishability relation over the states of the game [41, 42]. In this case, during a play, it may happen that some players cannot tell precisely in which state they are, but rather they observe a set of states. Therefore these players cannot base their strategy on the exact current situation. This means that over indistinguishable sets they can only use the same move or, similarly, that some perfect information moves are not valid any more. This feature deeply impacts on the MC complexity. For example, ATL becomes undecidable in the context of imperfect information and memoryful strategies [43]. To overcome this problem, some works have either focused on an approximation to perfect information [44, 45] or developed new notions on strategies [46–51].

To conclude, as far as we are aware of, none of the papers dealing with attack graphs in the formal methods literature have used tree automata, in particular, not in the case of imperfect information.

3 Attack Graphs

Before introducing the concept of attack graph, let us fix some preliminary notions that will be used along the paper.

Attack	Location	Precondition	Postcondition	Counter measure
att_1	Web Server		$web\_server : root$	-
att_2	Server	$web\_server : root$	$server : root$	c_2
att_3	Workstation	$web\_server : root$	$password : 1234$	-
att_4	Database A	$server : root$	$databaseA : root$	c_4
att_5	Database B	$server : root \wedge$ $password : 1234$	$databaseB : root$	c_5

Table 1 Atomic attacks and countermeasures over the LAN depicted in Figure 1.

3.1 Preliminaries

A *directed graph* is given by a finite non-empty set V of states (or vertices) and a finite set $E \subseteq V \times V$ of edges. If $(v, v') \in E$ we say that v is a parent of v' and that v' is a child of v. A vertex v that has no children will be called a *leaf*. A *rooted* directed graph is a tuple $\mathcal{G} = \langle V, r, E \rangle$ such that $\langle V, E \rangle$ is a directed graph and r is a distinguished vertex called *root*. A *path* is a finite, non-empty sequence of vertices $\mathcal{P} = v_0, \ldots, v_n$ such that v_i is a parent of v_{i+1} for any $i \in \{0, \ldots, n-1\}$. The size (or length) of a path $|\mathcal{P}|$ is the number of its elements. We say that $\mathcal{P} = v_0, \ldots, v_n$ is a path from v to v' if $v_0 = v$ and $v_n = v'$. A rooted directed graph is *connected* if for any vertex v there is a path from the root r to v. A directed graph is *acyclic* (or monotonic) if for each path $\mathcal{P} = v_0, \ldots, v_n$ over G we have that $v_0 = v_n$ implies $0 = n$. A *directed tree* is a rooted directed graph that is acyclic, connected, and in which each non-root

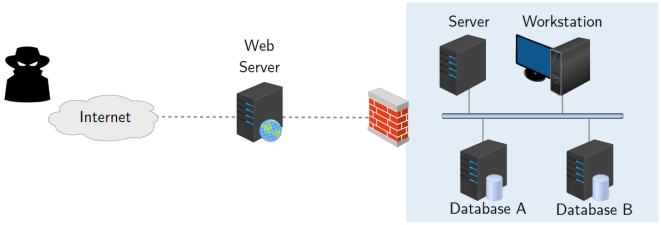

Fig. 1 An illustrating LAN architecture example.

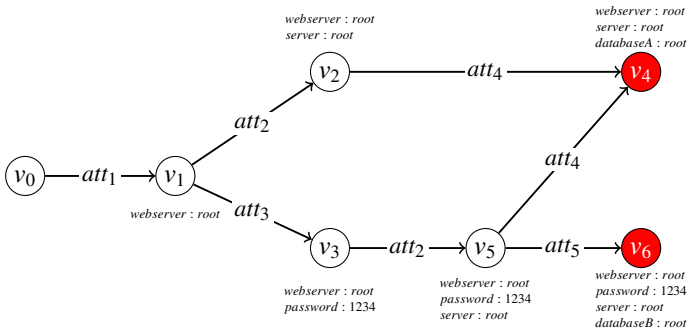

Fig. 2 Example of attack graph.

vertex has exactly one parent. Alternatively, one can define a directed tree as a rooted directed graph that is acyclic, connected, and in which there is exactly one path from the root to any vertex of the graph.

Now that we have fixed these preliminaries notions, we expose attack graphs. The term attack graph has been first introduced by Phillips and Swiler [52]. The general idea is to represent the possible attack paths in a system as a graph. This graph is generated given a description of the system architecture (topology, configurations of components, etc.) together with the list of existing vulnerabilities, the attacker's profile (his capability, passwords knowledge, privileges, etc.) and attack templates (attacker's atomic action, including preconditions and postconditions). An attack path in the graph corresponds to a sequence of atomic attacks. Several works have developed this approach, see e.g., [21, 53–56], and [57] for a survey.

There is no standardized definition of an attack graph: each of the previously cited works introduced its own attack graph model with its specificity, in particular regarding the semantics of nodes and edges (some works even use hypergraphs and not graphs to have a more concise representation of attack paths). However, all introduced models can be mapped into a *canonical attack graph* as introduced in [58]. It is a labelled oriented graph, where:

- each node represents both the state of the system (including existing vulnerabilities) and the state of the attacker including constants (attacker skills, financial resources, etc.) and variables (knowledge of the network topology, privilege level, obtained credentials, etc.);
- each edge represents an action of the attacker (a scan of the network, the execution of an exploit based on a given vulnerability, access to a device, etc.) that changes the state of the network or the states of the attacker; an edge is labelled with the name of the action (several edges of the attack graph may have the same label).

To reduce the complexity of analysis, we will consider in the rest of the paper *monotonic* attack graphs, i.e., acyclic graphs. It means that the attacker will never backtrack its attack steps which is a reasonable assumption in term of cybersecurity risk modeling.

Furthermore, an attack graph is said *complete* whenever the following condition holds: for every state q and for every atomic attack att, if the preconditions of the atomic attack hold in q, then there is an out-coming edge from q labelled with att.

We now give an example of an attack graph that corresponds to the architecture of the illustrating scenario depicted in Figure 1. Precisely, we consider an enterprise local area network (LAN) featuring a *Server*, a *Workstation*, and two databases *Database A* and *Database B*. The LAN also provides a *Web Server*. Accesses via Internet to the LAN are controlled by a firewall.

Table 1 gathers all possible atomic attacks an attacker can perform over the LAN. For instance, att_2 specifies that an attacker can exploit a vulnerability related to the *Server*: as a precondition the attacker needs to have root access to the *Web Server* and, as a postcondition, he will obtain root access to the *Server*.

An attack graph built from this set of atomic attacks and collecting possible attack paths is depicted in Figure 2. The attacker's initial state is a node in the attack graph. Let us suppose that the attacker is in state v_1 and wants to reach state v_4. To get to this target, he can perform the sequences of atomic attacks att_2, att_4 or att_3, att_2, att_4.

From the defender side, we consider that she is able to dynamically deploy a predefined set of countermeasures: for instance by reconfiguring the firewall filtering rules, or patching

some vulnerabilities, that is by removing one or several preconditions of an atomic attack. A given countermeasure c will prevent the attacker from longing a given attack att: deploying c is equivalent to removing all the edges in the attack graph labelled with att. In real situations, due to budget limitation or technical constraints, the set of available countermeasures may not cover all atomic attacks. In our previous example, as reported in the last column of Table 1, we suppose that the defender has at her disposal a countermeasure c_2 for attack att_2, c_4 for attack att_4, and c_5 for attack att_5, but no one for the attacks att_1 and att_3.

Along the paper we address attack graphs in the context of attack/response scenarios. We assume that:

1. The defender always knows the attack graph state reached by the attacker, i.e. the defender can detect an atomic attack launched by the attacker (using security supervision tools like the *Intrusion Detection System*). This is a very strong assumption, that will be relaxed in section sec:ImperfectInformation
2. At every moment, the attacker is in a unique state of the attack graph.
3. When the attacker launches an attack (if the preconditions are satisfied and the corresponding edge has not been removed by the defender), then the attack always succeeds (i.e. the attacker reaches the next state).
4. When the defender detects the attacker's state, she can react by deploying a unique countermeasure, whose effect is to remove all edges in the attack graph labelled with such a countermeasure.
5. When the defender deploys a new countermeasure, the attacker has the knowledge of its effect (i.e., the attacker knows which edges have been removed from the attack graph).

Some of these assumptions will be relaxed along the paper. In particular, assumption (1) will be discharged: has we have anticipated in the introduction, we will consider attack/response scenarios in which the defender has *imperfect information* about the position of the attacker in the attack graph.

In the LAN example, a possible attacker-defender interaction is the following: the attacker starts in state v_0, performs attack att_1 and reaches state v_1; then, the defender deploys countermeasure c_2, so the attacker cannot perform attack att_2 from v_1; then, the attacker performs attack att_3 from v_1 and reaches state v_3; finally, since the defender deploys countermeasure c_2, the attacker is stopped in v_3.

3.2 From Attack Graphs to Two-Player Games

We now give a formal definition of attack graph and two-player game. Then we show a model reduction among them.

Definition 1. *An attack graph is a tuple* $\mathcal{M} = \langle \mathsf{V}, \mathsf{v}_0, \mathsf{E}, \mathsf{L}, \mathsf{Tr} \rangle$*, where:*

- $\langle \mathsf{V}, v_0, \mathsf{E} \rangle$ *is a rooted directed graph that is acyclic and connected;*
- $\mathsf{L} : \mathsf{E} \to \mathbb{N}$ *is a function that labels the elements of* E*;*
- $\mathsf{Tr} \subseteq \mathsf{V}$ *is a set of target states.*

We formalize a two-player turn-based game as follows.

Definition 2. *A turn-based two-player game (*2TG*, for short) is a tuple* $\mathcal{G} = \langle \mathsf{S}, \mathsf{s}_0, \mathsf{R}, \mathsf{W} \rangle$ *where:*

- $\langle S, s_0, R\rangle$ *is a directed tree such that:*
  1. $S = S_A \cup S_D$ *and* $S_A \cap S_D = \emptyset$*. The set of states* S_A *is the set of state that are owned by the attacker, while states in* S_D *are owned by the defender. We impose that the root* s_0 *is an attacker state.*
  2. $R = R_A \cup R_D$, $R_A \subseteq S_A \times S_D$, $R_D \subseteq S_D \times S_A$ *and* $R_A \cap R_D = \emptyset$.
- $W \subseteq S_D$ *is the set of states that are winning for the attacker: while the attacker reach one of these states, using an edge in* R_A*, he wins the game.*

The size of a game $\mathcal{G}$ is the cardinality of S. Given a game $\mathcal{G}$, each player moves a token along the states via the relation R, starting from the initial state, with the attacker moving first. If the token is in an attacker's (resp., defender's) state, then he can move in a subset of states that belongs to the defender (resp., the attacker). More precisely, a *play* is a path of $\langle S, s_0, R\rangle$ whose first element is the root s_0 of the tree. In the following, we will use the Greek letters ρ and τ to denote plays.

A strategy for a game $\mathcal{G}$ is usually defined as a function. A function that specifies, at each moment of the game, which move a player must play according to the moves previously played (the history of the game). A strategy is *winning* when the player, who is following the strategy, wins, whatever the strategy of the opponent is. We choose another equivalent definition, motivated by our approach to solve games. We see a strategy as a tree in which each node is a state of the game, each path from the root of the tree to a given node is a play over the game, each play ending in one of the attacker's (the opponent) states s, has as many children as there are available R_A-reachable state from s and each play ending in one of the defender's (the proponent) state s' has at most one child that is R_D-reachable state from s'.

Definition 3. *A Defender strategy* σ *for a game* $\mathcal{G} = \langle S, s_0, R, W\rangle$ *is a finite tree whose root is* s_0*, whose branches are plays over* $\mathcal{G}$ *and that satisfy the following properties:*

1. *For each node* s *of* σ*: if* $s \in S_A$ *then* s *has as many children as there are nodes* $s' \in S_D$ *such that* $(s, s') \in R_A$*;*
2. *For each node* s *of* σ*: if* $s \in S_D$ *and* $s \in W$ *then* s *has no children, otherwise* s *has one child* $s' \in S_A$ *such that* $(s, s') \in R_D$.

A Defender strategy σ *is winning whenever no leaf of the strategy belongs to* W.

Let $\mathcal{M} = \langle V, v_0, E, L, \mathsf{Tr}\rangle$ be an attack graph, we denote by Act_d the set of actions of the defender in $\mathcal{M}$. If $v \in V$ we define $E(v) = \{(v, v') \in E\}$. If $e = (v, v') \in E$, $\pi_i(e)$ for $i \in \{1, 2\}$ denote the i-projection of e. If $E' \subseteq E$ is a set of edges $\Pi_i(E') = \{v \in V \mid v = \pi_i(e) \text{ for } e \in E'\}$. We let nil denote the empty list.

Now, we have all the ingredients to present our reduction.

In Algorithm 1 we devise a procedure to reduce an attack graph $\mathcal{M}$ to a two-player turn-based game $\mathcal{G}$. The algorithm proceeds as follows. For every state of $\mathcal{G}$ the procedure keeps track of the edges disabled by the defender along the path from the initial state to the current one. In detail, we initialize a token that determines the turn (line 2), a list to handle the edges disabled by the defender in the initial state (line 3), the set of states in $\mathcal{G}$ (line 4), and a queue to keep track of the states that have not yet been explored (line 5). Then, there is a loop (lines 6-23) that is divided in two different parts w.r.t. the token value. If token = 1, i.e., it is the turn of the attacker, then given the state (v, rm_v) from the queue (line 8), for each state v' in accordance with the adjacent states of v that are not disabled by the defender, we add a

Algorithm 1 From AG to Two-Player Game

```
1:  procedure ReduceToGame(M, Act_d)
2:      token = 1
3:      rm_{v_0} = nil
4:      S_A = {(v_0, rm_{v_0})}
5:      queue = [(v_0, rm_{v_0})]
6:      while queue ≠ ∅ do
7:          for i = 1 to size(queue) do
8:              (v, rm_v) = dequeue(queue)
9:              if token = 1 then
10:                 for v' ∈ Π_2(E(v) \ rm_v) do
11:                     enqueue(queue, (v', rm_v))
12:                     S_D = S_D ∪ {(v', rm_v)}
13:                     R_A = R_A ∪ {((v, rm_v, ), (v', rm_v))}
14:                     if v' ∈ Tr then
15:                         W = W ∪ {(v', rm_v)}
16:                     end if
17:                 end for
18:                 token = 2
19:             else
20:                 for a ∈ Act_d do
21:                     rm'_v = UPDATE(a, M)
22:                     enqueue(queue, (v, (rm_v ∪ rm'_v)))
23:                     S_A = S_A ∪ {(v, (rm_v ∪ rm'_v))}
24:                     R_D = R_D ∪ ((v, rm_v), (v, (rm_v ∪ rm'_v)))
25:                 end for
26:                 token = 1
27:             end if
28:         end for
29:     end while
30: end procedure
31: return ⟨S_A ∪ S_D, v_0, R_A ∪ R_D, W⟩
32: procedure Update(a, M)
33:     temp = ∅
34:     for e ∈ E do
35:         if L(e) = a then
36:             temp = temp ∪ {e}
37:         end if
38:     end for
39:     return (temp)
40: end procedure
```

new state in S_D, a new transition, and add it in the queue (lines 9-13). Then (lines 14-15) we check whether the edge belongs to the set of target states of the attack graph. If so, we add it to the set of winning states of the game. Otherwise, if it is the defender's turn, we analyze each possible action for the defender (defined with the set Act_d) and create a new state in S_A, the correspondent transition, add it in the queue. In this second case, we use an auxiliary procedure called Update, to update the list $rm_{\mathsf{v}'}$ (line 19) by adding edges in accordance with the action a of the defender (lines 27-28).

Note that, for every state v, we associate a list of removed edges rm_{v} to memorize the actions selected by the defender along the current computation from the initial state. To conclude, since the attack graph is monotonic, i.e. it is acyclic, it is easy to see that the algorithm terminates. Now, we have all the ingredients to provide the following result.

Theorem 1. *Algorithm 1 is sound and complete.*

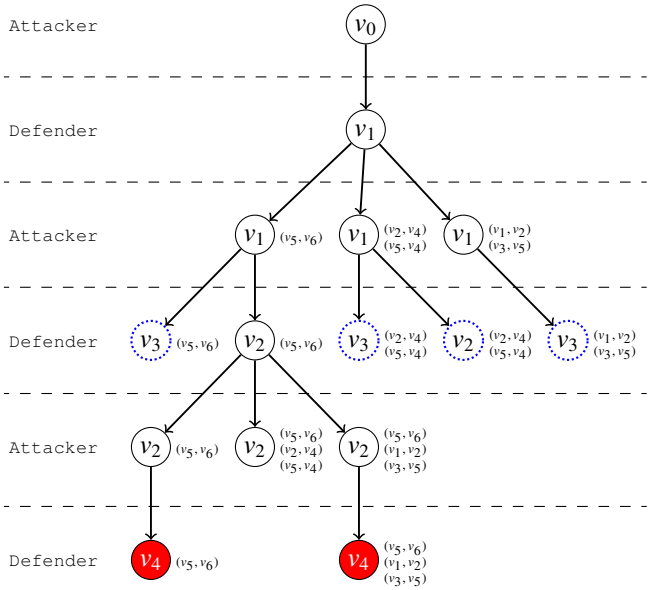

Fig. 3 Part of the 2TG generated from the AT in Figure 2. Blue dotted vertex are those that have a successor that is not showed. Red vertex are winning

Proof. For completeness, note that the algorithm generates a two-player turn-based game for any given input attack graph. Correctness follows trivially by construction. In fact, we ensure that the attacker can navigate the model according to the defender's choices equivalently to what is allowed in an attack graph. To guarantee the correctness of the game execution, the defender's choices (i.e. the eliminated edges) are saved within the states and the winning states are labeled following those of the attack graph. □

Figure 3 shows an application of Algorithm 1 for the attack graph depicted in Figure 2 by considering $Act_d = \{c_2, c_4, c_5\}$ and initial state for the attacker v_0.

3.3 Automata-Based Approach for Solving 2TG

We now present a top-down automata-theoretic approach to solve our game. According to definition 3, a strategy for the defender is a tree that takes for each node corresponding to a state s in the game, one successor if s belongs to the defender, or all successors, otherwise. If a strategy is winning, then no leaf of this tree is a winning state for the attacker in $\mathcal{G}$.

Now, we define the automaton that accepts all the trees that are winning strategies for the defender.

Definition 4. *A nondeterministic tree automaton (NTA, for short) is a tuple* $\mathcal{A} = \langle Q, \Sigma, q_0, \delta, F \rangle$*, where:* Q *is a set of states,* $q_0 \in Q$ *is an initial state,* $\delta : Q \times \Sigma \to 2^Q$ *is a transition function mapping pairs of states and symbols to a set of tuples of states, and* $F \subseteq Q$ *is a set of the accepting states.*

A *NTA* $\mathcal{A}$ recognizes trees and works as follows. For a node tree labelled by a and $\mathcal{A}$ being in a state q, it sends different copies of itself to successors in accordance with δ. By

$L(\mathcal{A})$ we denote the set of trees accepted by $\mathcal{A}$. The automaton is not empty if $L(\mathcal{A}) \neq \emptyset$. We now give the main result of this section.

Theorem 2. *Given an attack graph $\mathcal{M}$ and its* 2TG $\mathcal{G}$*, it is possible to decide in linear time w.r.t the size of $\mathcal{G}$ whether the Defender has a winning strategy over $\mathcal{G}$.*

Proof. We build a *NTA* $\mathcal{A}$ that accepts all the winning strategies for the Defender over $\mathcal{G}$. We briefly describe the automaton. The set of states Q is the set of states S of the game. We use the alphabet $\Sigma = \mathsf{S}$. For the initial state, we set $q_0 = \mathsf{s}_0$. For the transitions, starting from a state $q = \mathsf{s}$, we have the following cases:

$$\delta(q,a) = \begin{cases} (\mathsf{s}'_1) \vee \cdots \vee (\mathsf{s}'_n) & \text{if } \mathsf{s} \in \mathsf{S}_D \text{ and } q = a \\ (\mathsf{s}'_1, \ldots, \mathsf{s}'_n) & \text{if } \mathsf{s} \in \mathsf{S}_A \text{ and } q = a \\ \emptyset & \text{otherwise} \end{cases}$$

where $\mathsf{s}'_i \in \mathsf{S}$ and $(\mathsf{s}, \mathsf{s}'_i) \in \mathsf{R}$, for all $1 \leq i \leq n$. Note that, $n = |\{\mathsf{s}' \in \mathsf{S} \mid (\mathsf{s}, \mathsf{s}') \in \mathsf{R}\}|$. Finally, the set of accepting states is equal to the set of leafs of the tree $\langle \mathsf{S}, \mathsf{s}_0, \mathsf{R} \rangle$ that are not in W. The size of the automaton is linear in the size of the game, and from [59] we know that checking the emptiness of a *NTA* can be done in linear time. So, the desired complexity result follows. Note that, completeness comes from the fact that for each attack graph we generate an *NTA*, while correctness directly derives from the construction of the automaton. In fact, the accepting trees are those where for a defender's node it is sufficient to consider only one successor of the automaton (disjunction of the transition function), while for the attacker's nodes, all possible successors must be considered. These types of trees are by definition a strategy for the defender. The strategy becomes winning if every path of the tree does not visit winning states for the attacker. □

4 Quantitative attack graphs

As initially suggested by Bruce Schneier in the case of attack trees[1], attack graphs have also been extended to incorporate some attributes such as costs of atomic attacks in order to deduce the cheapest attack scenarios, but also other types of attributes such as probability of attack success, severity, impact (see for instance [60]). Usually the values of these attributes are inferred from vulnerability metrics and databases such as the NIST *Common Vulnerability Scoring System / Network Vulnerability Database (CVSS/ CNVD)*[2]. In this section, we introduce a quantitative extension of attack graphs that we call energy attack graph (or EAG for short). We use such attack graphs to model situations in which an attacker spends a finite (possibly null) amount of resources to move from one state to another in the graph. These resources can be heterogeneous in nature. For example: a monetary cost, units of time, units of computation, etc. We thus associate to each edge of the attack graph a finite vector of natural numbers. Each vector's component represents the amount of resources of a certain type needed to go from the vertex-source of the edge to its vertex-target. The attacker has an initial amount of resources. His goal is to find a path in the attack graph reaching one of the target states that does not have a cost that exceeds his initial amount of resources. Consequently, the defender's goal is to prevent the attacker from doing so. The cost of a path $\mathsf{v}_0, \ldots, \mathsf{v}_n$ being the sum of the resources associated to the edges going from v_i to v_{i+1} for $0 \leq i \leq n-1$.

[1] See https://www.schneier.com/academic/archives/1999/12/attack_trees.html
[2] See https://nvd.nist.gov/

Algorithm 2 From EAG to Two-Player Game

procedure REDUCETOGAME($\mathcal{M}, Act_d$)
  $token = 1$
  $rm_{\mathsf{v}_0} = \mathsf{nil}$
  $c_{\mathsf{v}_0} = [0_1, \ldots, 0_r]$
  $\mathsf{S}_A = \{(\mathsf{v}_0, rm_{\mathsf{v}_0})\}$
  $queue = [(\mathsf{v}_0, rm_{\mathsf{v}_0})]$ ▷ Initialize the counter
  while $queue \neq \emptyset$ **do**
    for $i = 1$ **to** $size(queue)$ **do**
      $(\mathsf{v}, rm_{\mathsf{v}}) = dequeue(queue)$
      if $token = 1$ **then**
        for $\mathsf{v}' \in \Pi_2(E(\mathsf{v}) \setminus rm_{\mathsf{v}})$ **do**
          $enqueue(queue, (\mathsf{v}', rm_{\mathsf{v}}))$
          $c_{\mathsf{v}'} = c_{\mathsf{v}} + \$(\mathsf{v}, \mathsf{v}')$ ▷ Update the counter for the node v'
          $\mathsf{S}_D = \mathsf{S}_D \cup \{(v', rm_v)\}$
          $\mathsf{R}_A = \mathsf{R}_A \cup \{((v, rm_v,), (v', rm_v))\}$
          if $\mathsf{v}' \in \mathsf{Tr}$ **then**
            if $c'_{\mathsf{v}} \leq \vec{m}$ **then** ▷ Check the value of the counter w.r.t. the initial amount
              $\mathsf{W} = \mathsf{W} \cup \{(\mathsf{v}', rm_v)\}$
            end if
          end if
        end for
        $token = 2$
      else
        for $a \in Act_d$ **do**
          $rm'_{\mathsf{v}} = \text{UPDATE}(a, \mathcal{M})$
          $enqueue(queue, (\mathsf{v}, (rm_{\mathsf{v}} \cup rm'_{\mathsf{v}})))$
          $\mathsf{S}_A = \mathsf{S}_A \cup \{(\mathsf{v}, (rm_{\mathsf{v}} \cup rm'_{\mathsf{v}}))\}$
          $\mathsf{R}_D = \mathsf{R}_D \cup ((\mathsf{v}, rm_{\mathsf{v}}), (\mathsf{v}, (rm_{\mathsf{v}} \cup rm'_{\mathsf{v}}))$
        end for
        $token = 1$
      end if
    end for
  end while
end procedure
return $\langle \mathsf{S}_A \cup \mathsf{S}_D, v_0, \mathsf{R}_A \cup \mathsf{R}_D, \mathsf{W} \rangle$

4.1 From Quantitative Attack Graphs to Two-Player Games

We now formally define EAG and then reduce them to our simple two-player game model.

Definition 5. *An energy attack graph is a tuple* $\mathcal{M} = \langle \mathsf{V}, \mathsf{v}_0, \mathsf{E}, \mathsf{L}, \mathsf{Tr}, r, \vec{m}, \$ \rangle$*, such that:*

- $\langle \mathsf{V}, \mathsf{v}_0, \mathsf{E}, \mathsf{L}, \mathsf{Tr} \rangle$ *is an attack graph;*
- $\vec{m}$ *is a vector of natural numbers of length* r*;*
- r *is a positive natural number;*
- $\$: \mathsf{E} \to \mathbb{N}^r$ *is a function that assigns to each edge a vector* $\vec{n}$ *of natural numbers such that* $|\vec{n}| = r$*.*

The natural number r represents the number of different types of resources that are needed to make an attack, e.g., money, time, energy, etc. The vector $\vec{m}$ represents the initial amount of resources that are given to an attacker: each component of the vector represents the amount of a different type of resource. The function $\$$ assigns to each transition $e \in \mathsf{E}$ the cost of the transition. Such cost is represented as a vector of natural numbers of length r.

To reduce an EAG to a 2TG one natural option would be to extend our game model. In particular, we could opt for energy games. In such games, each transition from one state to

Attack	Cost
att_1	$\langle 0, 0\rangle$
att_2	$\langle 2, 1\rangle$
att_3	$\langle 4, 1\rangle$
att_4	$\langle 3, 2\rangle$
att_5	$\langle 2, 5\rangle$

Table 2 Cost of each attack for the attack graph of Figure 2

another consumes a finite amount of resources. Although this extension seems natural, we opt for another solution that allows us not to modify our game model in any way. In fact, given an EAG $\mathcal{M}$, we can transform it in a 2TG $\mathcal{G}$ of definition 2, by a slight modification of Algorithm 1.

To obtain such solution (displayed in Algorithm 2), we add a counter c_v (a vector of length r) for each node v (the counter for the initial node v_0 has value 0 in each of its coordinates). Such counter keeps track of the resources used to reach the node v along the path of the attack graph starting at the initial node. The counter is updated using the function \$ of the EAG. To check whether a node $v \in S_D$ belongs to the set of winning states W, we check if it belongs to the set of target states of the EAG and if the counter is not bigger than the initial amount of resources $\vec{m}$ given to the attacker in the EAG.

Now, we can provide the following result.

Theorem 3. *Algorithm 2 is sound and complete.*

Proof. For completeness, note that the algorithm generates a two-player turn-based game for any given input attack graph. Correctness follows trivially by construction. In fact, by Algorithm 1, given an attack graph we can generate a turn-based game. For the quantitative part, we have added a counter for each path of the turn-based model that guarantees us a condition on the energy threshold provided to the attacker. In this way, if the attacker's energy is sufficient for an attack, the attack is successful, whereas if the energy is not sufficient, the attack fails. We model this last part by considering that if an attacker has exhausted their resources, they will never be able to reach a winning state. □

For instance, consider again the attack graph of Figure 2. Suppose that, $\vec{m} = \langle 5, 2\rangle$ and that the cost of each attack is specified as in Table 2. Part of the 2TGobtained from this EAG is displayed in Figure 4

Remark that given an EAG, the corresponding game $\mathcal{G}$ produced by algorithm 2 is a 2TG. Thus, it is possible to use theorem 2 and check, in linear time, if a winning strategy for $\mathcal{G}$ exists.

Corollary 1. *Given an energy attack graph $\mathcal{M}$ and its 2TG $\mathcal{G}$, it is possible to decide in linear time w.r.t the size of $\mathcal{G}$ whether the Defender has a winning strategy over $\mathcal{G}$.*

5 Imperfect Information

In this section, we introduce a notion of imperfect information for the defender. We consider that in some situations, the defender cannot tell which attack has been made by the attacker.

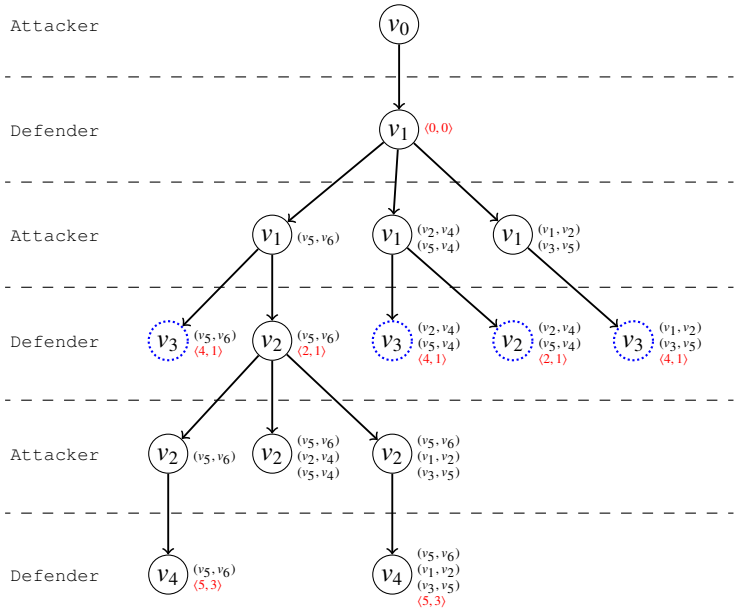

Fig. 4 Part of the 2TG generated from the AT in Figure 2 provided with the costs of Table 2. Blue dotted vertex are those that have a successor that is not showed. The value of the counter for Defender's node is the red array.

Following this intuition, we consider that some edges of the attack graph are in an equivalence relation. Intuitively, this equivalence relation represents the fact that a defender cannot distinguish two attacks that belong to this class. Since, in the game model, edges of the attack graph corresponds to attacker's actions, the equivalence relation on attack's graph edge will induce an equivalence relation on attacker's edge in the game. Using these latter relation, we can define plays over the game that are indistinguishable from the defender's point of view.

5.1 From Quantitative Attack Graphs with Imperfect information to Two-Player Games with Imperfect information

We now formally define attack graphs with imperfect information and show a reduction to turn-based games with imperfect information.

Definition 6. *An energy attack graph with imperfect information (*EAGI *for short) is a tuple* $\mathcal{M} = \langle \mathsf{V}, \mathsf{v}_0, \mathsf{E}, \mathsf{L}, \mathsf{Tr}, \$, r, \vec{m}, \$, \sim_d \rangle$ *such that* $\langle \mathsf{V}, \mathsf{v}_0, \vec{m}, \mathsf{E}, \mathsf{L}, \mathsf{Tr}, \$, r, \vec{m}, \$ \rangle$ *is an* EAG *and* $\sim_d \subseteq \mathsf{E} \times \mathsf{E}$ *is an equivalence relations over the set of edges.*

We reduce an EAGI to 2TG with imperfect information. We define such games as follows.

Definition 7. *Given* L_A *and* L_D *two sets of any symbols, a* 2TG *with imperfect information is a tuple* $\mathcal{G} = \langle \mathsf{S}, \mathsf{s}_0, \mathsf{R}, \mathsf{W}, f_A, f_D, \sim^d_{R_A} \rangle$ *where:* $\langle \mathsf{S}, \mathsf{s}_0, \mathsf{R}, \mathsf{W} \rangle$ *is a* 2TG, $f_A : \mathsf{R}_A \to L_A$ *and* $f_D : \mathsf{R}_D \to L_D$ *are two labelling functions such that* $L_A \cap L_D = \emptyset$ *and* $\sim^d_{R_A} \subseteq \mathsf{R}_A \times \mathsf{R}_A$ *is an equivalence relation on edges.*

If ρ and τ are two plays over a 2TG with imperfect information, we say that ρ and τ are equivalent for the defender (denoted $\rho \sim \tau$) whenever $|\rho| = |\tau|$ and for all $i < |\rho|$ either $f_D((\rho_i, \rho_{i+1})) = f_D((\tau_i, \tau_{i+1}))$ or $((\rho_i, \rho_{i+1}), (\tau_i, \tau_{i+1})) \in \sim^d_{R_A}$.

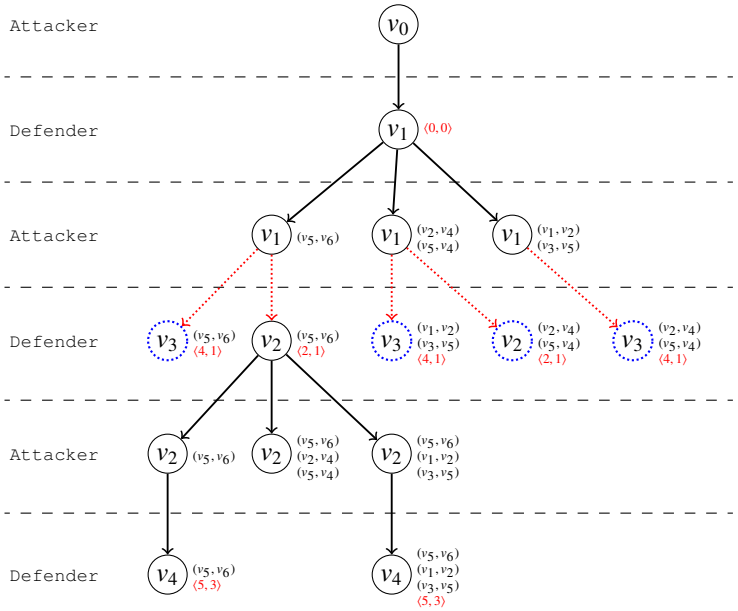

Fig. 5 Part of the 2TG generated from the AT in Figure 2 provided with the costs of Table 2 and with the equivalence relation $\{(v_1,v_2),(v_1,v_3)\}$. Blue dotted vertex are those that have a successor that is not showed. The value of the counter for Defender's node is the red array. Equivalent edges are the red dotted ones. The labeling of edges is not showed

Given an EAGI $\mathcal{M}$, it is easy to obtain a 2TG with imperfect information through some modifications of the algorithm described in subsection 4.1. Such a procedure is given in Algorithm 3. In the algorithm L is the labelling function given in the definition of the attack graph. Remark that we label edges of the 2TG that belongs to the given EAGI with the same label that they have in the EAGI (line 18) while we label an edge $((\mathsf{v}, rm_{\mathsf{v}}),(\mathsf{v},(rm_{\mathsf{v}} \cup rm'_{\mathsf{v}}))) \in \mathsf{R}_D$ with $rm_{\mathsf{v}} \cup rm'_{\mathsf{v}}$ (line 39).

Theorem 4. *Algorithm 3 is sound and complete.*

Proof. The key point of this transformation is the equivalence relation. Here, we transparently propagate the equivalence relation on the edges of the attack graph to the equivalence relation of the defender, thanks to line 22 of Algorithm 3. Then, the result follows by Algorithm 1 and Algorithm 2. □

As an example, consider again the attack graph of Figure 2. Suppose that the cost of the edges of the attack graph is specified according to Table 2, that the initial amount of resources given to the Attacker is, as before, $\langle 5,2\rangle$. Moreover, suppose that for the Defender, the two edges (v_1,v_2) and (v_1,v_3) are indistinguishable, that is $((v_1,v_2),(v_1,v_3)) \in \sim^d_{R_A}$. Figure 5 shows a partial output of Algorithm 3 given this latter attack graph.

We now define the defender's strategies taking into account the imperfect information.

Definition 8. *A defender* uniform *strategy for a* 2TG *with imperfect information* $\mathcal{G} = \langle \mathsf{S}, \mathsf{s}_0, \mathsf{R}, \mathsf{W}, f_A, f_D, \mathsf{R}^{\sim_d}\rangle$ *is a defender strategy* σ *such that for any two paths* ρ, s *and* τ, s' *of*

Algorithm 3 From EAGI to Two-Player Game

procedure REDUCETOGAME($\mathcal{M}, Act_d^{\mathcal{M}}$)
  $token = 1$
  $rm_{\mathsf{v}_0} = \mathsf{nil}$
  $f_A = \emptyset$
  $\sim_{R_A}^{d} = \emptyset$
  $c_{\mathsf{v}_0} = [0_1, \ldots, 0_r]$
  $\mathsf{S}_A = \{(\mathsf{v}_0, rm_{\mathsf{v}_0})\}$
  $queue = [(\mathsf{v}_0, rm_{\mathsf{v}_0})]$
  while $queue \neq \emptyset$ **do**
    for $i = 1$ **to** $size(queue)$ **do**
      $(\mathsf{v}, rm_{\mathsf{v}}) = dequeue(queue)$
      if $token = 1$ **then**
        for $\mathsf{v}' \in \Pi_2(E(\mathsf{v}) \setminus rm_{\mathsf{v}})$ **do**
          $enqueue(queue, (\mathsf{v}', rm_{\mathsf{v}}))$
          $c_{\mathsf{v}'} = c_{\mathsf{v}} + \$(\mathsf{v}, \mathsf{v}')$
          $\mathsf{S}_D = \mathsf{S}_D \cup \{(v', rm_v)\}$
          $\mathsf{R}_A = \mathsf{R}_A \cup \{((v, rm_v,), (v', rm_v))\}$
          $f_A = f_A \cup \{(((v, rm_v,), (v', rm_v)), L((v, v')))\}$
          for $((\mathsf{v}, rm_{\mathsf{v}}), (\mathsf{v}', rm_{\mathsf{v}'})) \in \mathsf{R}_A$ **do**
            for $((\mathsf{v}_1, rm_{\mathsf{v}_1}), (\mathsf{v}_2, rm_{\mathsf{v}_2})) \in \mathsf{R}_A$ **do**
              if $((\mathsf{v}, \mathsf{v}'), (\mathsf{v}_1, \mathsf{v}_2)) \in \sim_d$ **then**
                $\sim_{R_A}^{d} = \sim_{R_A}^{d} \cup (((\mathsf{v}, rm_{\mathsf{v}}), (\mathsf{v}', rm_{\mathsf{v}'})), ((\mathsf{v}_1, rm_{\mathsf{v}_1}), (\mathsf{v}_2, rm_{\mathsf{v}_2})))$ ▷ Update the eq.relation
              end if
            end for
          end for
          if $\mathsf{v}' \in \mathsf{Tr}$ **then**
            if $c_{\mathsf{v}}^{'} \leq \vec{m}$ **then**
              $\mathsf{W} \cup \{(\mathsf{v}', rm_{\mathsf{v}})\}$
            end if
          end if
        end for
      end if
      $token = 2$
      for $a \in Act_d^{\mathcal{M}}$ **do**
        $rm'_{\mathsf{v}} = \text{UPDATE}(a, \mathcal{M})$
        $enqueue(queue, (\mathsf{v}, (rm_{\mathsf{v}} \cup rm'_{\mathsf{v}})))$
        $\mathsf{S}_A = \mathsf{S}_A \cup \{(\mathsf{v}, (rm_{\mathsf{v}} \cup rm'_{\mathsf{v}}))\}$
        $\mathsf{R}_D = \mathsf{R}_D \cup ((\mathsf{v}, rm_{\mathsf{v}}), (\mathsf{v}, (rm_{\mathsf{v}} \cup rm'_{\mathsf{v}})))$
        $f_D = f_D \cup \{((\mathsf{v}, rm_{\mathsf{v}}), (\mathsf{v}, (rm_{\mathsf{v}} \cup rm'_{\mathsf{v}})), rm_{\mathsf{v}} \cup rm'_{\mathsf{v}})\}$
      end for
      $token = 1$
    end for
  end while
end procedure
return $\langle \mathsf{S}_A \cup \mathsf{S}_D, v_0, \mathsf{R}_A \cup \mathsf{R}_D, \mathsf{W}, f_A, f_D, \sim_{R_A}^{d} \rangle$

σ if s *and* s' *are in* S_A *and* $\rho \sim \tau$*, then* $f_D((x, s)) = f_D((x', s'))$ *where* x *is the parent of* s *and* x' *is the parent of* s'.

To conclude this section, we need to provide a solution for 2TG with imperfect information, i.e. the structures that encapsulate the behavior of EAGI. Given the nature of the 2TG with imperfect information, we can use again a tree automata to check whether the attacker has a winning strategy. However, games with imperfect information over the actions of the agents have already been solved in [11]. In particular, the authors have proposed a solution

on Concurrent Games Structures with imperfect information about the actions (ICGS). So, what we need to do is to provide a reduction from 2TGwith imperfect information to ICGS and then to use the automata theoretic approach used in [11]. In what follows, we recall the definition of ICGS with imperfect information about actions.

Definition 9. *A concurrent (two player) game structure with imperfect information about actions (ICGS for short) is a tuple* $\mathfrak{M} = \langle \mathsf{Ag}, \mathsf{S}', \mathsf{s}'_0, \{\mathsf{Act}_i\}_{i\in\mathsf{Ag}}, d, \delta, \mathsf{W}', \{\sim\}\rangle$ *where:*

- $\mathsf{Ag} = \{1, 2\}$ *is the set of agents;*
- S' *is a finite, non-empty set of states, with initial state* $\mathsf{s}'_0 \in \mathsf{S}$*;*
- $\mathsf{Act}_i = \mathsf{Act}_1 \cup \mathsf{Act}_2$ *is a finite non-empty set of actions;*
- d *is the protocol function mapping an agent* i *and a state to a non-empty subset of* Act_i*;*
- $\delta : \mathsf{S}' \times \mathsf{Act}_1 \times \mathsf{Act}_2 \to \mathsf{S}'$ *is the transition function;*
- $\mathsf{W}' \subseteq \mathsf{S}'$ *is a set of target states.*
- $\sim \subseteq \mathsf{Act}_1 \times \mathsf{Act}_1$ *is an equivalence relation;*

Without loss of generality, we can assume that for each pair of states s and s' there is at most one pair of actions that lets to transit from s to s'. The following definitions are taken from [11]. A *track* τ is a finite[3] sequence of states $\tau_0, \ldots, \tau_n$ such that for all i, $0 \leq i \leq n-1$ there exists two action $a_1 \in \mathsf{Act}_1$, $a_2 \in \mathsf{Act}_2$ such that $\tau_{i+1} = \delta(\tau_i, a_1, a_2)$. Let tr be a partial function that given two states s and s' returns the pair of actions a_1, a_2 such that $\mathsf{s}' = \delta(\mathsf{s}, a_1, a_2)$ if such pair exists.

Definition 10. *If* $\mathfrak{M} = \langle \mathsf{Ag}, \mathsf{S}', \mathsf{s}'_0, \{\mathsf{Act}_i\}_{i\in\mathsf{Ag}}, d, \delta, \mathsf{W}', \{\sim\}\rangle$ *is an ICGS with imperfect information about actions, then we say that two tracks* τ *and* τ' *are indistinguishable for Player 2 whenever they have the same length* n *and, for each* $i \leq n-1$, $\mathsf{tr}(\tau_i, \tau_{i+1}) \sim \mathsf{tr}(\tau'_i, \tau'_{i+1})$.

We denote the indistinguishability relation between tracks by $\tau \sim \tau'$. A strategy σ_i for Player i is a function mapping each track to an action of Player i. A strategy is *uniform* iff for any two tracks τ, τ' such that $\tau \sim \tau'$, we have that $\sigma_i(\tau) = \sigma_i(\tau')$. Assume that Player 1 and Player 2 choose two strategies σ_1 and σ_2. Their composition induces a play, i.e., a finite sequence of states $\rho_0, \ldots \rho_n$ such that $\rho_0 = \mathsf{s}'_0$ and for all $1 \leq i \leq n-1$, $\rho_{i+1} = \sigma(\rho_i, \sigma_1(\rho_{\leq i}), \sigma_2(\rho_{\leq_i}))$ where $\rho_{\leq_i}$ is the prefix of ρ ending at ρ_i. Finally, we say that a strategy ρ is winning for Player 2, iff for any Player 1 strategy the resulting induced play ends in a state that is not in W.

Theorem 5. *Given a* 2TG *with imperfect information* $\mathcal{G} = \langle \mathsf{S}, \mathsf{s}_0, \mathsf{R}, \mathsf{W}, f_A, f_D, \sim^d_{R_A}\rangle$*, there is an Exp-time algorithm deciding whether there is a uniform, winning strategy in* $\mathcal{G}$*.*

Proof. In [11] the authors show that the problem of determining whether there exists a winning strategy in an ICGS with imperfect information about actions is Exp-time complete. Thus, it suffices to reduce our game model to an ICGS. Given a 2TG with imperfect information, $\langle \mathsf{S}, \mathsf{s}_0, \mathsf{R}, \mathsf{W}, f_A, f_D, \sim^d_{R_A}\rangle$, ,we define an ICGS $\mathfrak{M} = \langle \mathsf{Ag}, \mathsf{S}', \mathsf{s}'_0, \{\mathsf{Act}_i\}_{i\in\mathsf{Ag}}, d, \delta, \mathsf{W}', \{\sim\}\rangle$ as follows:

- $\mathsf{Ag} = \{A, D\}$;
- $\mathsf{S}' = \mathsf{S}$;
- $\mathsf{s}'_0 = \mathsf{s}_0$;
- $\mathsf{Act}_A = \mathsf{R}_A \cup \{\star\}$ and $\mathsf{Act}_D = \{f_D(x) \mid x \in \mathsf{R}_D\} \cup \{\star\}$ where $\star$ is the idle action;

[3] in [11] the authors considers infinite tracks. However, in the case of reachability games we can restrict ourselves to finite ones.

– for the attacker, we set $d(A, s) = \{e \in \mathsf{R}_A \mid e = (\mathsf{s}, \mathsf{s}') \text{ for some } \mathsf{s}' \in \mathsf{S}_D\}$ if $s \in \mathsf{S}_A$, $d(A, s) = \{\star\}$ otherwise. For the defender we set $d(D, \mathsf{s}) = \{f_D((\mathsf{s}, \mathsf{s}')) \mid (\mathsf{s}, \mathsf{s}') \in \mathsf{R}_D\}$ if $\mathsf{s} \in S_D$ and $d(D, \mathsf{s}) = \{\star\}$ otherwise;
– remark that by the previous point given $i, j \in \{A, D\}$ such that $i \neq j$, if $\mathsf{s} \in \mathsf{S}_i$ then $d(j, \mathsf{s}) = \{\star\}$. Thus, we can define the function δ as follows: $\{(\mathsf{s}, (\mathsf{s}, \mathsf{s}'), \star, \mathsf{s}') \mid \mathsf{s} \in \mathsf{S}_A,\ \mathsf{s}' \in \mathsf{S}_D,\ \} \cup \{(\mathsf{s}, \star, f_D((\mathsf{s}, \mathsf{s}')), \mathsf{s}') \mid \mathsf{s} \in \mathsf{S}_D,\ \mathsf{s}' \in \mathsf{S}_A\}$;
• the set of winning states of the ICGS is W.
– $\sim = \{(e, e') \mid (e, e') \in \sim^{d}_{R_A}\}$;

Remark that the size of the so obtained IGCS is linear in the size of the 2TG.
For the completeness of this approach, it is sufficient to note that for every two-player game with imperfect information, we generate an ICGS. As for correctness, to transition from a turn-based game to a concurrent game, it is enough to explicitly associate a distinct action with each transition of a turn-based game to the ICGS and make only one player 'active' in each state. To accomplish this latter operation, we have inserted the idle action in each state, which makes a player 'inactive' and leaves the decision-making power to determine a successor to the other player. Given these assumptions, the correctness follows. □

6 Conclusion

In this paper, we restated the attack graph framework by means of a two-player turn-based game, defender vs attacker: the defender deactivates resource accesses while the attacker chooses adjacent states along which to move. We have first considered games with perfect information, we have then included quantitative constraint for the attacker, and we have finally extended our framework to take into account imperfect information for the defender.

We provided an automata solution to these games, which amounts to show that the defender can always prevent the attacker to enter forbidden states. Since the automata solution requires linear-time, we justify the introduction of an ad-hoc game model instead of using more expensive existing frameworks [4, 5, 61].

We plan to continue the work in a number of directions. First, we want to extend our procedure to attack graphs with cycles. Second, we aim to add weights that represent the resources available for the attacker and the defender. Furthermore, we want to investigate more complex situations, involving multiple attackers. In this setting, we also plan to exploit resilient solutions with the aim of reducing a damage when an attack cannot be stopped. In addition, we can consider to study formal logics to gain expressive power to define the attackers' objectives and check more intricate solution concepts. On this respect, an approach to Sabotage Logic [62] is proposed in [63]. Following this line, we can study logics for the strategic reasoning such as ATL [5] and Strategy Logic [7] to capture the features on attackers vs. defenders games. Furthermore, in this context, we can also study if an attacker has some backup strategies to achieve his objectives by following the line on graded modalities as done in [64, 65]. Finally, beyond theoretical aspects to improve the expressiveness of our framework, we would also like to analyze purely practical aspects and implement our foundational results in verification tools such as VITAMIN [66].

To the best of our knowledge, this is the first work providing a game-theoretic approach with an automata solution to attack graphs. We hope that this will serve as a fertilization for new solutions to challenging question in attack graphs.

Acknowledgments

This research has been supported by the EU Horizon 2020 Marie Sklodowska-Curie project with grant agreement NO 101105549, ANR project AGAPE ANR-18-CE23-0013, the PNRR MUR projects PE0000013 and the ECS0000037-MUSA-INFANT, and the PRIN 2020 project-RIPER

References

[1] Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. The MIT Press, Cambridge, Massachusetts (1999)

[2] Clarke, E.M., Emerson, E.A.: Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic. In: LP'81. LNCS 131, pp. 52–71. Springer, (1981)

[3] Kupferman, O., Vardi, M.Y., Wolper, P.: An Automata Theoretic Approach to Branching-Time ModelChecking. Journal of the ACM **47**(2), 312–360 (2000)

[4] Kupferman, O., Vardi, M.Y., Wolper, P.: Module Checking. Information and Computation **164**(2), 322–344 (2001)

[5] Alur, R., Henzinger, T.A., Kupferman, O.: Alternating-Time Temporal Logic. Journal of the ACM **49**(5), 672–713 (2002)

[6] Lomuscio, A., Qu, H., Raimondi, F.: MCMAS: A model checker for the verification of multi-agent systems. In: Proceedings of the 21th International Conference on Computer Aided Verification (CAV09). Lecture Notes in Computer Science, vol. 5643, pp. 682–688. Springer, (2009)

[7] Mogavero, F., Murano, A., Perelli, G., Vardi, M.Y.: Reasoning about strategies: On the model-checking problem. ACM Transactions in Computational Logic **15**(4), 34–13447 (2014) https://doi.org/10.1145/2631917

[8] Jamroga, W., Murano, A.: Module checking of strategic ability. In: AAMAS 2015, pp. 227–235 (2015)

[9] Jennings, N.R., Wooldridge, M.: Application of intelligent agents. In: Agent Technology: Foundations, Applications, and Markets. Springer, (1998)

[10] Lippmann, R.P., Ingols, K.W.: An annotated review of past papers on attack graphs (2005)

[11] Malvone, V., Murano, A., Sorrentino, L.: Hiding actions in multi-player games. In: Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2017, São Paulo, Brazil, May 8-12, 2017, pp. 1205–1213 (2017)

[12] Malvone, V.: Strategic reasoning in game theory. PhD thesis, University of Naples Federico II, Italy (2018)

[13] Ma, J., Zhang, D., Xu, G., Yang, Y.: Model checking based security policy verification and validation. In: 2010 2nd International Workshop on Intelligent Systems and Applications, pp. 1–4 (2010). https://doi.org/10.1109/IWISA.2010.5473291

[14] Baliosian, J., Serrat, J.: Finite state transducers for policy evaluation and conflict resolution. In: Proceedings. Fifth IEEE International Workshop on Policies for Distributed Systems and Networks, 2004. POLICY 2004., pp. 250–259 (2004). IEEE

[15] Catta, D., Stasio, A.D., Leneutre, J., Malvone, V., Murano, A.: A Game Theoretic Approach to Attack Graphs. In: Rocha, A.P., Steels, L., Herik, H.J. (eds.) Proceedings of the 15th International Conference on Agents and Artificial Intelligence, ICAART 2023, Volume 1, Lisbon, Portugal, February 22-24, 2023, pp. 347–354. SCITEPRESS, (2023). https://doi.org/10.5220/0011776900003393 . https://doi.org/10.5220/0011776900003393

[16] Al Ghazo, A.T., Ibrahim, M., Ren, H., Kumar, R.: A2g2v: Automatic attack graph generation and visualization and its applications to computer and scada networks. IEEE Transactions on Systems, Man, and Cybernetics: Systems **50**(10), 3488–3498 (2020) https://doi.org/10.1109/TSMC.2019.2915940

[17] Ritchey, R.W., Ammann, P.: Using model checking to analyze network vulnerabilities. In: Proceeding 2000 IEEE Symposium on Security and Privacy. S&P 2000, pp. 156–165 (2000). IEEE

[18] McMillan, K.L.: Symbolic model checking. In: Symbolic Model Checking, pp. 25–60. Springer, (1993)

[19] Jha, S., Sheyner, O., Wing, J.: Two formal analyses of attack graphs. In: Proceedings 15th IEEE Computer Security Foundations Workshop. CSFW-15, pp. 49–63 (2002). IEEE

[20] Cimatti, A., Clarke, E., Giunchiglia, F., Roveri, M.: Nusmv: A new symbolic model verifier. In: International Conference on Computer Aided Verification, pp. 495–499 (1999). Springer

[21] Ou, X., Boyer, W.F., McQueen, M.A.: A scalable approach to attack graph generation. In: Proceedings of the 13th ACM Conference on Computer and Communications Security, pp. 336–345 (2006)

[22] Vardi, M.Y.: The rise and fall of ltl. GandALF **54** (2011)

[23] Zeng, J., Wu, S., Chen, Y., Zeng, R., Wu, C., Caballero-Gil, P.: Survey of attack graph analysis methods from the perspective of data and knowledge processing. Sec. and Commun. Netw. **2019** (2019) https://doi.org/10.1155/2019/2031063

[24] Zenitani, K.: Attack graph analysis: An explanatory guide. Computers Security **126**, 103081 (2023)

[25] Ge, Y., Shen, X., Xu, B., He, G.: A hybrid attack graph analysis method based on model checking. In: 2022 Tenth International Conference on Advanced Cloud and Big Data (CBD), pp. 258–263 (2022)

[26] Noel, S., Jajodia, S., O'Berry, B., Jacobs, M.: Efficient minimum-cost network hardening via exploit dependency graphs. In: 19th Annual Computer Security Applications Conference, 2003. Proceedings., pp. 86–95 (2003)

[27] Wang, L., Noel, S., Jajodia, S.: Minimum-cost network hardening using attack graphs. Comput. Commun. **29**(18), 3812–3824 (2006)

[28] Albanese, M., Jajodia, S., Noel, S.: Time-efficient and cost-effective network hardening using attack graphs. In: IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2012), pp. 1–12 (2012)

[29] Durkota, K., Lisý, V., Bosanský, B., Kiekintveld, C.: Approximate solutions for attack graph games with imperfect information. In: Decision and Game Theory for Security - 6th International Conference, GameSec 2015, London, UK, November 4-5, 2015, Proceedings. Lecture Notes in Computer Science, vol. 9406, pp. 228–249. Springer, (2015)

[30] Durkota, K., Lisy, V., Bošansky, B., Kiekintveld, C.: Optimal network security hardening using attack graph games. IJCAI'15, pp. 526–532. AAAI Press, (2015)

[31] Nguyen, T.H., Wright, M., Wellman, M.P., Baveja, S.: Multi-stage attack graph security games: Heuristic strategies, with empirical game-theoretic analysis. MTD '17, pp. 87–97. Association for Computing Machinery, New York, NY, USA (2017)

[32] Zhang, Y., Malacaria, P.: Bayesian stackelberg games for cyber-security decision support. Decis. Support Syst. **148**, 113599 (2021)

[33] Bursztein, E., Goubault-Larrecq, J.: A logical framework for evaluating network resilience against faults and attacks. In: Advances in Computer Science - ASIAN 2007. Computer and Network Security, 12th Asian Computing Science Conference, Doha, Qatar, December 9-11, 2007, Proceedings. Lecture Notes in Computer Science, vol. 4846, pp. 212–227. Springer, Berlin, Heidelberg (2007)

[34] Khakpour, N., Parker, D.: Partially-Observable Security Games for Automating Attack-Defense Analysis (2022)

[35] Chen, T., Forejt, V., Kwiatkowska, M., Parker, D., Simaitis, A.: Automatic verification of competitive stochastic systems. In: Flanagan, C., König, B. (eds.) Tools and Algorithms for the Construction and Analysis of Systems, pp. 315–330 (2012)

[36] Chatterjee, K., Doyen, L., Henzinger, T.A., Raskin, J.-F.: Generalized Mean-payoff and Energy Games. In: IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010). Leibniz International Proceedings

in Informatics (LIPIcs), vol. 8, pp. 505–516. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany (2010). https://doi.org/10.4230/LIPIcs.FSTTCS.2010.505 . http://drops.dagstuhl.de/opus/volltexte/2010/2848

[37] Jurdziński, M., Lazić, R., Schmitz, S.: Fixed-dimensional energy games are in pseudo-polynomial time. In: Halldórsson, M.M., Iwama, K., Kobayashi, N., Speckmann, B. (eds.) Automata, Languages, and Programming, pp. 260–272. Springer, Berlin, Heidelberg (2015)

[38] Alechina, N., Logan, B., Nga, N.H., Rakib, A.: Resource-bounded alternating-time temporal logic. In: Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: Volume 1 - Volume 1. AAMAS '10, pp. 481–488. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC (2010)

[39] Alechina, N., Logan, B.S., Nga, N.H., Rakib, A.: A logic for coalitions with bounded resources. In: IJCAI (2009)

[40] Alechina, N., Logan, B.S., Nga, N.H., Raimondi, F.: Decidable model-checking for a resource logic with production of resources. In: ECAI (2014)

[41] Kupferman, O., Vardi, M.Y.: Module checking revisited. In: CAV'97, pp. 36–47 (1997). Springer

[42] Reif, J.H.: The complexity of two-player games of incomplete information. J. Comput. Syst. Sci. **29**(2), 274–301 (1984)

[43] Dima, C., Tiplea, F.L.: Model-checking ATL under Imperfect Information and PerfectRecall Semantics is Undecidable. Technical report, arXiv (2011)

[44] Belardinelli, F., Lomuscio, A., Malvone, V.: An abstraction-based method for verifying strategic properties in multi-agent systems with imperfect information. In: The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019, pp. 6030–6037 (2019). https://doi.org/10.1609/aaai.v33i01.33016030 . https://doi.org/10.1609/aaai.v33i01.33016030

[45] Belardinelli, F., Malvone, V.: A three-valued approach to strategic abilities under imperfect information. In: Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning, KR 2020, Rhodes, Greece, September 12-18, 2020, pp. 89–98 (2020). https://doi.org/10.24963/kr.2020/10 . https://doi.org/10.24963/kr.2020/10

[46] Belardinelli, F., Lomuscio, A., Malvone, V.: Approximating perfect recall when model checking strategic abilities. In: Thielscher, M., Toni, F., Wolter, F. (eds.) Principles of Knowledge Representation and Reasoning: Proceedings of the Sixteenth International Conference, KR 2018, Tempe, Arizona, 30 October - 2 November 2018, pp. 435–444

(2018). https://aaai.org/ocs/index.php/KR/KR18/paper/view/18010

[47] Belardinelli, F., Lomuscio, A., Malvone, V., Yu, E.: Approximating perfect recall when model checking strategic abilities: Theory and applications. J. Artif. Intell. Res. **73**, 897–932 (2022) https://doi.org/10.1613/jair.1.12539

[48] Jamroga, W., Malvone, V., Murano, A.: Natural strategic ability. Artif. Intell. **277** (2019) https://doi.org/10.1016/j.artint.2019.103170

[49] Belardinelli, F., Lomuscio, A., Murano, A., Rubin, S.: Verification of multi-agent systems with public actions against strategy logic. Artif. Intell. **285**, 103302 (2020) https://doi.org/10.1016/j.artint.2020.103302

[50] Berthon, R., Maubert, B., Murano, A., Rubin, S., Vardi, M.Y.: Strategy logic with imperfect information. ACM Trans. Comput. Log. **22**(1), 5–1551 (2021) https://doi.org/10.1145/3427955

[51] Ferrando, A., Malvone, V.: Towards the combination of model checking and runtime verification on multi-agent systems. In: Proceedings of the 20th International Conference on Advances in Practical Applications of Agents, Multi-Agent Systems, and Complex Systems Simulation PAAMS 2022. Lecture Notes in Computer Science, vol. 13616, pp. 140–152. Springer, (2022)

[52] Phillips, C., Swiler, L.P.: A graph-based system for network-vulnerability analysis. In: Proceedings of the 1998 Workshop on New Security Paradigms, pp. 71–79 (1998)

[53] Sheyner, O., Haines, J., Jha, S., Lippmann, R., Wing, J.: Automated generation and analysis of attack graphs, pp. 273–284 (2002). https://doi.org/10.1109/SECPRI.2002.1004377

[54] Ammann, P., Wijesekera, D., Kaushik, S.: Scalable, graph-based network vulnerability analysis. CCS '02, pp. 217–224. Association for Computing Machinery, New York, NY, USA (2002)

[55] Noel, S., Jajodia, S., O'Berry, B., Jacobs, M.: Efficient minimum-cost network hardening via exploit dependency graphs. In: Proceedings of the 19th Annual Computer Security Applications Conference. ACSAC '03, p. 86. IEEE Computer Society, USA (2003)

[56] Ingols, K., Lippmann, R., Piwowarski, K.: Practical attack graph generation for network defense. In: 2006 22nd Annual Computer Security Applications Conference (ACSAC'06), pp. 121–130 (2006)

[57] Kaynar, K.: A taxonomy for attack graph generation and usage in network security. J. Inf. Secur. Appl. **29**(C), 27–56 (2016)

[58] Heberlein, T., Bishop, M., Ceesay, E., Danforth, M., Senthilkumar, C., Stallard, T.:

A taxonomy for comparing attack-graph approaches. [Online] http://netsq. com/Documents/AttackGraphPaper. pdf (2012)

[59] Thomas, W.: Automata on Infinite Objects. In: Handbook of Theoretical Computer Science (vol. B), pp. 133–191 (1990)

[60] Homer, J., Zhang, S., Ou, X., Schmidt, D., Du, Y., Rajagopalan, S.R., Singhal, A.: Aggregating vulnerability metrics in enterprise networks using attack graphs. J. Comput. Secur. **21**(4), 561–597 (2013)

[61] Löding, C., Rohde, P.: Solving the sabotage game is pspace-hard. In: International Symposium on Mathematical Foundations of Computer Science, pp. 531–540 (2003). Springer

[62] Benthem, J.: An essay on sabotage and obstruction. In: Mechanizing Mathematical Reasoning, Essays in Honor of Jörg H. Siekmann on the Occasion of His 60th Birthday. Lecture Notes in Computer Science, vol. 2605, pp. 268–276 (2005). https://doi.org/10.1007/978-3-540-32254-2_16 . https://doi.org/10.1007/978-3-540-32254-2_16

[63] Catta, D., Leneutre, J., Malvone, V.: Subset sabotage games & attack graphs. In: Ferrando, A., Mascardi, V. (eds.) Proceedings of the 23rd Workshop "From Objects to Agents", Genova, Italy, September 1-3, 2022. CEUR Workshop Proceedings, vol. 3261, pp. 209–218 (2022). https://ceur-ws.org/Vol-3261/paper16.pdf

[64] Faella, M., Napoli, M., Parente, M.: Graded alternating-time temporal logic. Fundam. Informaticae **105**(1-2), 189–210 (2010) https://doi.org/10.3233/FI-2010-363

[65] Aminof, B., Malvone, V., Murano, A., Rubin, S.: Graded modalities in strategy logic. Inf. Comput. **261**, 634–649 (2018) https://doi.org/10.1016/j.ic.2018.02.022

[66] Ferrando, A., Malvone, V.: VITAMIN: A compositional framework for model checking of multi-agent systems. CoRR **abs/2403.02170** (2024) https://doi.org/10.48550/ARXIV.2403.02170 2403.02170