A Game Theoretic Approach to Attack Graphs

Davide Catta, Antonio Di Stasio, Jean Leneutre, Vadim Malvone, Aniello Murano

A Game Theoretic Approach to Attack Graphs

Davide Catta[1], Antonio Di Stasio[2], Jean Leneutre[1], Vadim Malvone[1] and Aniello Murano[3]
[1]*Télécom-Paris, Paris, France*
[2]*Sapienza University of Rome, Rome, Italy*
[3]*University of Naples Federico II, Naples, Italy*

Keywords: Attack Graphs, Game Theory, Automata Theoretic Approach.

Abstract: An attack graph is a succinct representation of all the paths in an open system that allow an attacker to enter a forbidden state (e.g., a resource), besides any attempt of the system to prevent it. Checking system vulnerability amounts to verifying whether such paths exist. In this paper we reason about attack graphs by means of a game-theoretic approach. Precisely, we introduce a suitable game model to represent the interaction between the system and the attacker and an automata-based solution to show the absence of vulnerability.

1 INTRODUCTION

The inherent complexity of modern systems came with a cost: as they became more complex, it also becomes harder and harder to assure their security. When dealing with security, one should be true to the motto "Better safe than sorry". This is because the cost of repairing a system flaw during maintenance is at least two order of magnitude higher, compared to a fixing at an early design. As a consequence, in order to develop a secure system, one should come up with tools able to detect vulnerability and unexpected behaviors at a very early stage of its lifecycles (Clarke et al., 1999). To check systems reliability a story of success is the use of *formal methods* techniques (Clarke et al., 1999). They allow checking whether a system is correct by formally checking whether a mathematical model of it meets a formal representation of its desired behaviour.

Recently, classic approaches such as *model checking* and automata-theoretic techniques, originally developed for monolithic systems (Clarke and Emerson, 1981; Kupferman et al., 2000), have been meaningfully extended to handle *open* and *multi-agent systems* (Kupferman et al., 2001; Alur et al., 2002; Lomuscio et al., 2009; Mogavero et al., 2014; Jamroga and Murano, 2015). These are systems that encapsulate the behaviour of two or more rational agents interacting among them in a cooperative or adversarial way, aiming at a designed goal (Jennings and Wooldridge, 1998).

In system security checking, a malicious attack can be seen as an attempt of an attacker to gain an unauthorized resource access or compromise the system integrity. In this setting, *attack graph* (Lippmann and Ingols, 2005) is one of the most prominent attack model developed and receiving much attention in recent years. This encompasses a graph where each state represents an attacker at a specified network location and edges represent state transitions, i.e., attack actions by the attacker. Then, it is a system duty to prevent unauthorized accesses from the attacker in each state of the graph.

In this paper, we reason about attack graphs by introducing game models and an automata-based solution to evaluate system reliability. We first set a two-player turn-based reachability game between the (system) defender and the (external and potential) attacker, where in turn the latter moves along adjacent states (w.r.t. the attack graph under exam) and the former inhibits some attacks by taking countermeasures. We show how simple attack graphs can be reduced to such game-model. We then build a finite tree automaton that accepts all the walking trees that allow the attacker to reach the designated states, no matter how the defender behaves. By checking the emptiness of the automaton, we show the robustness of the system (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.

Outline. In section 2, we present related works. In section 3 we formally introduce attack graphs and two-player turn-based games, show a reduction from

Catta, D., Di Stasio, A., Leneutre, J., Malvone, V. and Murano, A.
A Game Theoretic Approach to Attack Graphs.
DOI: 10.5220/0011776900003393
In *Proceedings of the 15th International Conference on Agents and Artificial Intelligence (ICAART 2023) - Volume 1*, pages 347-354
ISBN: 978-989-758-623-1; ISSN: 2184-433X

the former to the latter. Furthermore, we show how to represent attacker's strategies via trees and a tree automaton accepting all such trees, where the latter is used to prove whether an attacker has a winning strategy. Finally, section 4 concludes the paper and presents some future directions.

2 RELATED WORK

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 works in (Durkota et al., 2015a; Durkota et al., 2015b) 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 (Nguyen et al., 2017) 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 (Zhang and Malacaria, 2021) 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.

The work in (Bursztein and Goubault-Larrecq, 2007) 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.

Besides the problem of attack graph generation, a large body of works on attack graphs proposes methods to analyze them, as surveyed in (Zeng et al., 2019). These methods can be roughly divided into two groups: the risk assessment methods aiming at predicting the attacker's behavior and the risk treatment methods aiming at deploying new security countermeasures. 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. However, most of these works consider a static view corresponding to a prevention approach. A more dynamic view or reaction approach is needed when facing attack: given an action of the attacker, which countermeasure the defender must deploy in priority to minimize the risk on the system. Resource consumption in games is a well established area of research.

The practical study on attack graphs mainly refers to "non model-based" approaches, with few exceptions. (Al Ghazo et al., 2020) introduces A2g2v, a model checker that generates attack graphs and detects an attack sequence by means of a counterexample. (Ritchey and Ammann, 2000) introduces a model checker for vulnerability analysis via attack graphs; it uses the verification tool SMV (McMillan, 1993), so it can only show one attack (counterexample) at the time. Differently, (Jha et al., 2002) uses a modified version of the tool NuSMV (Cimatti et al., 1999) to represents all possible attacks. (Ou et al., 2006) introduces 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 (Vardi, 2011), useful to specify service guarantees against real malicious activity.

3 ATTACK GRAPHS

The term attack graph has been first introduced by Phillips and Swiler (Phillips and Swiler, 1998). 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., (Sheyner et al., 2002; Ammann et al., 2002; Noel et al., 2003; Ou et al., 2006; Ingols et al., 2006), and (Kaynar, 2016) 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

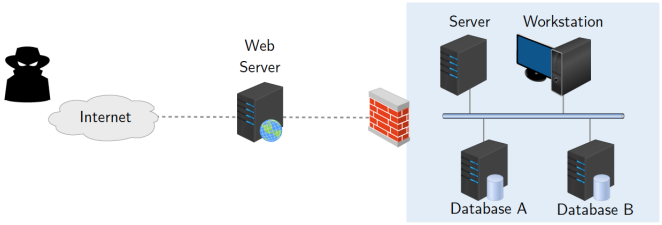

Figure 1: An illustrating LAN architecture example.

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

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

(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 (Heberlein et al., 2012). 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).

We will consider in the rest of the paper *monotonic* attack graphs, i.e., acyclic graphs. 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:

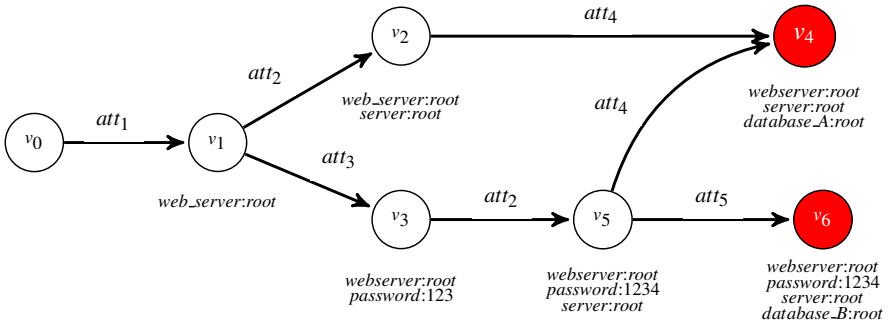

Figure 2: Example of attack graph.

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*).
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).

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.

3.1 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:*

- V *is a set of states;*
- v_0 *is an element of* V *(the initial state);*
- $\mathsf{E} \subseteq \mathsf{V} \times \mathsf{V}$ *is a set of edges;*
- $\mathsf{L} : \mathsf{E} \to \mathbb{N}$ *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* $\mathcal{G} = \langle \mathsf{S}, \mathsf{s}_0, \mathcal{R}, \mathsf{W} \rangle$ *where:*

- $\mathsf{S} = \mathsf{S}_1 \cup \mathsf{S}_2$ *is the set of states.* S_1 *and* S_2 *are two disjoint sets of states corresponding to Player* 1 *and Player 2 states;*
- s_0 *is a member of* S_1 *(the initial state);*
- $\mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2$ *is the transition relation, where* $\mathcal{R}_1 \subseteq \mathsf{S}_1 \times \mathsf{S}_2$ *and* $\mathcal{R}_2 \subseteq \mathsf{S}_2 \times \mathsf{S}_1$*,*
- $\mathsf{W} \subseteq \mathsf{S}_2$ *is the set of states that are winning for Player* 1*.*

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 $\mathcal{R}$, starting from the initial state, with Player 1 moving first. If the token is in a Player 1's (resp., Player 2's) state, then he can move in a subset of states that belongs to Player 2 (resp., Player 1). A *play* $\rho = \rho_0, \ldots, \rho_n$ over $\mathcal{G}$ is a finite, non-empty, sequence of states in S such that $\rho_0 = \mathsf{s}_0$ and $(\rho_i, \rho_{i+1}) \in \mathcal{R}$, for $i \in \{0, \ldots, n-1\}$. We use ρ and π to denote plays. A play $\rho = \rho_0, \ldots, \rho_n$ is *won by Player 1* iff $\rho_n \in \mathsf{W}$.

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 Player 2's (the opponent) states s, has

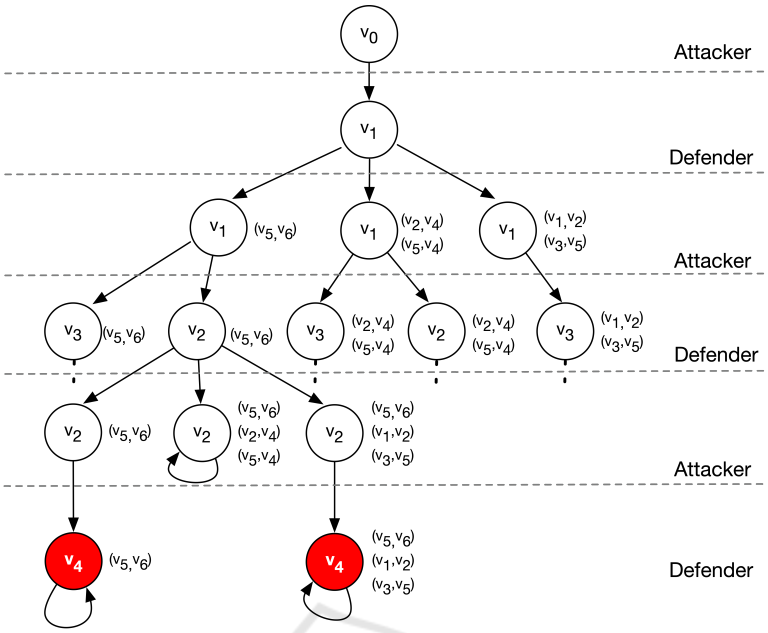

Figure 3: Part of the 2TG generated from the AT in Figure 2.

Algorithm 1: From Attack Graph to Two-Player Game.

procedure REDUCETOGAME($\mathcal{M}$, Act_d)
  $token = 1$ ▷ Set the turn
  $rm_{v_0} = \text{nil}$
  $S_1 = \{(v_0, rm_{v_0})\}$
  $queue = [(v_0, rm_{v_0})]$
  while $queue \neq \emptyset$ **do**
    for $i = 1$ **to** $size(queue)$ **do**
      $(v, rm_v) = dequeue(queue)$
      if $token = 1$ **then**
        for $v' \in \Pi_2(E(v) \setminus rm_v)$ **do** ▷ chose a reachable state
          $enqueue(queue, (v', rm_v))$
          $S_2 = S_2 \cup \{(v', rm_v)\}$ ▷ update Player 2 states
          $\mathcal{R}_1 = \mathcal{R}_1 \cup \{((v, rm_v), (v', rm_v))\}$
        if $\Pi_2(E(v) \setminus rm_v) = \emptyset$ **then**
          $\mathcal{R}_1 = \mathcal{R}_1 \cup \{((v, rm_v), (v, rm_v))\}$ ▷ Update Player 1 transitions
        $token = 2$
      else
        for $a \in Act_d$ **do**
          $rm'_v = \text{UPDATE}(a, \mathcal{M})$ ▷ Update the list of removed arcs
          $enqueue(queue, (v, (rm_v \cup rm'_v)))$
          $S_1 = S_1 \cup \{(v, (rm_v \cup rm'_v))\}$ ▷ Update Player 1 states
          $\mathcal{R}_2 = \mathcal{R}_2 \cup ((v, rm_v), (v, (rm_v \cup rm'_v))$ ▷ Update Player 2 transitions
          if $v \in \text{Tr}$ **then**
            $W = W \cup \{(v, (rm_v \cup rm'_v))\}$ ▷ Update the list of winning states
        $token = 1$
procedure UPDATE($a, \mathcal{M}$)
  $temp = \emptyset$
  for $e \in E$ **do**
    if $L(e) = a$ **then**
      $temp = temp \cup \{e\}$
  return $(temp)$

as many children as there are available $\mathcal{R}_2$-reachable state from s and each play ending in one of Player 1's (the proponent) state has at most one child. Recall that a *tree* is a (finite or infinite) connected directed graph, with one node designated as the root, in which every non-root node as a unique parent, and the root has no parent (s is the parent of t, and t is the child of s if there is an edge from s to t). A path $\mathcal{P} = x_0, x_1, \ldots$ is a (finite or infinite) sequence of nodes such that x_i is the parent of x_{i+1} for all $i \geq 0$. A branch is a path that is maximal and whose first node is the root of the tree.

Definition 3. *An attacker strategy σ for a game $\mathcal{G} = \langle \mathsf{S}, \mathsf{s}_0, \mathcal{R}, \mathsf{W} \rangle$ 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 $s \in \mathsf{S}_1$ then s has at most one child;*
2. *For each node s of σ: if $s \in \mathsf{S}_2$ and $s \notin W$ then s has as many child as there are nodes s' such that $(\mathsf{s}, \mathsf{s}') \in \mathcal{R}_2$.*

An attacker strategy σ is winning whenever each leaf of the strategy belongs to W.

Let $\mathcal{M} = \langle \mathsf{V}, \mathsf{v}_0, \mathsf{E}, \mathsf{L}, \mathsf{Tr} \rangle$ be an attack graph, we denote by Act_d the set of actions of the defender in $\mathcal{M}$. If $\mathsf{v} \in \mathsf{V}$ we define $\mathsf{E}(\mathsf{v}) = \{(\mathsf{v}, \mathsf{v}') \in \mathsf{E}\}$. If $e = (\mathsf{v}, \mathsf{v}') \in \mathsf{E}$, $\pi_i(e)$ for $i \in \{1,2\}$ denote the i-projection of e. If $\mathsf{E}' \subseteq \mathsf{E}$ is a set of edges $\Pi_i(\mathsf{E}') = \{\mathsf{v} \in \mathsf{V} \mid \mathsf{v} = \pi_i(e)$ for $e \in \mathsf{E}'\}$. 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 $\mathcal{G}$ in which the attacker is represented by Player 1 and the defender by Player 2. 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 6-25) that is divided in two different part w.r.t. the token value. If token = 1, i.e., it is the turn of the attacker, then given the state $(\mathsf{v}, rm_{\mathsf{v}})$ 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 new state in S_2, a new transition, and add it in the queue (lines 9-15). 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_1, the correspondent transition, add it in the queue, and check whether it is a final state in the attack graph (lines 18-24). 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 28-30).

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.

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.

3.2 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 attacker is a tree that takes for each node corresponding to a state s in the game, one successor if s belongs to the attacker, or all successors, otherwise. If a strategy is winning, all the leaves of this tree are target states of $\mathcal{G}$, then surely the attacker has a winning strategy over the game.

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

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 1. *Given a 2TG $\mathcal{G}$ it is possible to decide in linear time w.r.t the size of $\mathcal{G}$ whether the attacker has a winning strategy over $\mathcal{G}$.*

Proof. We build a *NTA* $\mathcal{A}$ that accepts all the winning strategies for the attacker 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:

$$\delta(q,a) = \begin{cases} (\mathsf{s}'_1) \vee \cdots \vee (\mathsf{s}'_n) & \text{if } \mathsf{s} \in \mathsf{S}_1 \text{ and } q = a \\ (\mathsf{s}'_1, \ldots, \mathsf{s}'_n) & \text{if } \mathsf{s} \in \mathsf{S}_2 \text{ and } q = a \\ \emptyset & \text{otherwise} \end{cases}$$

where $s'_i \in \mathsf{S}$ and $(s, s'_i) \in \mathcal{R}$, for all $1 \leq i \leq n$. Note that, $n = |\{s' \in \mathsf{S} \mid (s, s') \in \mathcal{R}\}|$. Finally, the set of accepting states is equal to W. The size of the automaton is linear in the size of the game, and from (Thomas, 1990) we know that checking the emptiness of a *NTA* can be done in linear time. So, the desired complexity result follows. □

4 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 provided an automata solution to the game, 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 (Löding and Rohde, 2003; Kupferman et al., 2001; Alur et al., 2002).

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.

Finally, 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 (van Benthem, 2005) is proposed in (Catta et al., 2022). Following this line, we can study logics for the strategic reasoning such as ATL (Alur et al., 2002) and Strategy Logic (Mogavero et al., 2014) 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 (Faella et al., 2010; Aminof et al., 2018). However, the more realistic setting for games is with imperfect information, but unfortunately, the model checking problem with imperfect information for strategic logics is undecidable in general (Dima and Tiplea, 2011). Given the relevance of this setting, even partial solutions to the problem can be useful, such as abstractions either on the information (Belardinelli et al., 2019; Belardinelli and Malvone, 2019) or on the strategies (Belardinelli et al., 2022) or on the formulas (Ferrano and Malvone, 2022). In conclusion, we can embed the mentioned techniques to provide a more powerful framework.

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.

REFERENCES

Al Ghazo, A. T., Ibrahim, M., Ren, H., and Kumar, R. (2020). A2g2v: Automatic attack graph generation and visualization and its applications to computer and scada networks. *IEEE TSMCS*, 50(10):3488–3498.

Alur, R., Henzinger, T., and Kupferman, O. (2002). Alternating-Time Temporal Logic. *JACM*, 49(5):672–713.

B. Aminof, V. Malvone, A. Murano, and S. Rubin (2018). Graded modalities in strategy logic. *Inf. Comput.*, 261:634–649.

Ammann, P., Wijesekera, D., and Kaushik, S. (2002). Scalable, graph-based network vulnerability analysis. In CCS 2002, page 217–224.

F. Belardinelli, A. Lomuscio, and V. Malvone (2019). An abstraction-based method for verifying strategic properties in multi-agent systems with imperfect information. In *AAAI 2019*, pages 6030–6037.

F. Belardinelli, A. Lomuscio, V. Malvone, and E. Yu (2022). Approximating perfect recall when model checking strategic abilities: Theory and applications. *J. Artif. Intell. Res.*, 73:897–932.

F. Belardinelli and V. Malvone (2020). A three-valued approach to strategic abilities under imperfect information. In KR 2020, pages 89–98.

Bursztein, E. and Goubault-Larrecq, J. (2007). A logical framework for evaluating network resilience against faults and attacks. In ASIAN 2007, pages 212–227.

D. Catta, J. Leneutre, and V. Malvone (2022). Subset sabotage games & attack graphs. In WOA, pages 209–218.

Cimatti, A., Clarke, E., Giunchiglia, F., and Roveri, M. (1999). Nusmv: A new symbolic model verifier. In *CAV* 2009, pages 495–499.

Clarke, E. and Emerson, E. (1981). Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic. In *LP 1981*, pages 52–71.

Clarke, E. M., Grumberg, O., and Peled, D. A. (1999). *Model Checking*. The MIT Press, Massachusetts.

C. Dima and F. Tiplea (2011). Model-checking ATL under imperfect information and perfect recall semantics is undecidable. *CoRR*, abs/1102.4225, 2011.

Durkota, K., Lisý, V., Bosanský, B., and Kiekintveld, C. (2015a). Approximate solutions for attack graph games with imperfect information. In GameSec 2015, pages 228–249.

Durkota, K., Lisy, V., Bošansky, B., and Kiekintveld, C. (2015b). Optimal network security hardening using attack graph games. In IJCAI 2015, pages 526–532.

M. Faella, M. Napoli, and M. Parente (2010). Graded alternating-time temporal logic. *Fundam. Informaticae*, 105(1-2):189–210, 2010.

A. Ferrando and V. Malvone (2022). Towards the combination of model checking and runtime verification on multi-agent systems. In PAAMS 2022, pages 140–152.

Heberlein, T., Bishop, M., Ceesay, E., Danforth, M., Senthilkumar, C., and Stallard, T. (2012). A taxonomy for comparing attack-graph approaches. *[Online] http://netsq. com/Documents/AttackGraphPaper. pdf*.

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

Ingols, K., Lippmann, R., and Piwowarski, K. (2006). Practical attack graph generation for network defense. In ACSAC 2006, pages 121–130.

Jamroga, W. and Murano, A. (2015). Module checking of strategic ability. In *AAMAS 2015*, pages 227–235.

Jennings, N. R. and Wooldridge, M. (1998). Application of intelligent agents. In *Agent Technology: Foundations, Applications, and Markets*. Springer-Verlag.

Jha, S., Sheyner, O., and Wing, J. (2002). Two formal analyses of attack graphs. In CSFW-15, pages 49–63.

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

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

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

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

Löding, C. and Rohde, P. (2003). Solving the sabotage game is pspace-hard. In MFCS 2003, pages 531–540.

Lomuscio, A., Qu, H., and Raimondi, F. (2009). MCMAS: A model checker for the verification of multi-agent systems. In CAV 2009, pages 682–688.

McMillan, K. L. (1993). Symbolic model checking. In *Symbolic Model Checking*, pages 25–60.

Mogavero, F., Murano, A., Perelli, G., and Vardi, M. Y. (2014). Reasoning about strategies: On the model-checking problem. *ACM Transactions in Computational Logic*, 15(4):34:1–34:47.

Nguyen, T. H., Wright, M., Wellman, M. P., and Baveja, S. (2017). Multi-stage attack graph security games: Heuristic strategies, with empirical game-theoretic analysis. MTD 2017, pages 87–97.

Noel, S., Jajodia, S., O'Berry, B., and Jacobs, M. (2003). Efficient minimum-cost network hardening via exploit dependency graphs. In ACSAC 2003, page 86.

Ou, X., Boyer, W. F., and McQueen, M. A. (2006). A scalable approach to attack graph generation. In CCS 2006, pages 336–345.

Phillips, C. and Swiler, L. P. (1998). A graph-based system for network-vulnerability analysis. In NSPW 1998, pages 71–79.

Ritchey, R. W. and Ammann, P. (2000). Using model checking to analyze network vulnerabilities. In S&P 2000, pages 156–165.

Sheyner, O., Haines, J., Jha, S., Lippmann, R., and Wing, J. (2002). Automated generation and analysis of attack graphs. pages 273– 284.

Thomas, W. (1990). Automata on Infinite Objects. In *Handbook of Theoretical Computer Science (vol. B)*, pages 133–191. MIT Press.

J. van Benthem (2005). *An Essay on Sabotage and Obstruction*. In MMR, pages 268–276.

Vardi, M. Y. (2011). The rise and fall of LTL. In *Gandalf*, 54.

Zeng, J., Wu, S., Chen, Y., Zeng, R., Wu, C., and Caballero-Gil, P. (2019). Survey of attack graph analysis methods from the perspective of data and knowledge processing. *SCN*.

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