Subset Sabotage Games & Attack Graphs

Davide Catta[1,†], Jean Leneutre[1] and Vadim Malvone[1]

[1]LTCI, Télécom Paris, Institut Polytechnique de Paris, Paris, France

Abstract

We consider an extended version of sabotage games played over Attack Graphs. Such games are two-player zero-sum reachability games between an Attacker and a Defender. This latter player can erase particular subsets of edges of the Attack Graph. To reason about such games we introduce a variant of Sabotage Modal Logic (that we call Subset Sabotage Modal Logic) in which one modality quantify over non-empty subset of edges. We show that we can characterize the existence of winning Attacker strategies by formulas of Subset Sabotage Modal Logic.

Keywords

Attack Graphs, Sabotage Games, Logics in Games

1. Introduction

Modern systems are inherently complex and security plays a crucial role. The main challenge in developing a secure system is to come up with tools able to detect vulnerability and unexpected behaviors at a very early stage of its life-cycles. These methodologies should be able to measure the grade of resilience to external attacks. Crucially, 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 stage [1].

In the past fifty years several solutions have been proposed and a story of success is the use of *formal methods* techniques [1]. They allow checking whether a system is correct by formally checking whether 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 meaningfully extended to handle *open* and *multi-agent systems* [4, 5, 6, 7, 8]. These are systems that encapsulate the behavior of two or more rational agents interacting among them in a cooperative or adversarial way, aiming at a designed goal [9].

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* [10] 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.

WOA: Workshop From Objects to Agents 2022

[†]These authors contributed equally.

davide.catta@telecom-paris.fr (D. Catta); jean.leneutre@telecom-paris.fr (J. Leneutre); vadim.malvone@telecom-paris.fr (V. Malvone)

Said more precisely, the Attacker goal is to reach a certain state of the Attack Graph by traveling thought its edges, while the Defender goal is to prevent him from doing so. To do this, the Defender can dynamically deploy countermeasures preventing the attack to succeeds. The attacks are represented by the edges of the Attack Graph. If the Defender deploys a countermeasure and such countermeasure is successful, the Attacker will no longer be able to use an attack. We can formalize such scenario as a two-player turn based game between the Defender and the Attacker, where in turn the latter moves along adjacent states (w.r.t. the Attack Graph under examen) and the former inhibits some attacks by erasing some subset of edges of the Attack Graph itself. The goal of the Defender is to block the Attacker to reach some designated states, while not blocking the entire system functionality. The kind of scenario that we have just sketched is an example of **extended** *sabotage game* [11]. Sabotage games were introduced by van Benthem in 2005 with the aim of studying the computational complexity of a special class of graph-reachability problems. Namely, graph reachability problems in which the set of edges of the graph became thinner and thinner as long as a path of the graph is constructed. To reason about sabotage games, van Benthem introduced Sabotage Modal Logic. Such Logic is obtained by adding to the $\Diamond$-modality of classical modal logic another modality $\blacklozenge$. Let G be a directed graph and s one of its vertex (or states); the intended meaning of a formula $\blacklozenge\varphi$ is " $\blacklozenge\varphi$ is true at a state s of G iff φ is true at s in the graph obtained by G by erasing an edge e".

Our sabotage games differs from the one introduced by van Benthem because one of the player can erase *an entire subset of edges of a given graph*. To reason about such games, we introduce a variant of Sabotage Modal Logic that we call, for lack of wit, Subset Sabotage Modal Logic (SSML for short). The logic SSML is obtained by adding a modality $\blacklozenge^{\subset}$ to the language of classical modal logic. The intended meaning of a formula $\blacklozenge^{\subset}\varphi$ is " $\blacklozenge^{\subset}\varphi$ is true at a state s of a directed graph G iff φ is true at s in the graph G' that is obtained from G by erasing a non-empty set of its edges". We prove that the model checking problem for SSML is decidable and that the existence of an Attacker winning strategy over an Attack Graph can be expressed by using an SSML formula.

Structure of the work. The rest of the paper is structured as follows. In Sec 2 we discuss the related works. In Sec 3 we briefly present Sabotage Modal Logic. In Sec. 4 we present Attack Graphs, propose a formal definition of such objects in terms of Kripke structures and introduce, informally, subset sabotage games on Attack Graphs. In the following section (Sec 5), we formally define the class of subset sabotage games on Attack Graphs that we are interested on. We define plays on such games and Attacker winning strategies. In the penultimate section (Sec 6), we introduce Subset Sabotage Modal Logic and show that the model checking problem for such logic is decidable. To prove such result, we reduce the model checking problem of SSML to the one of SML by giving a translation of SSML-formulas into SML formulas. We then show that the existence of an Attacker winning strategies for a subset sabotage game over an Attack Graph, is equivalent to the satisfability of a certain SSML formula at the root of the considered Attack Graph. The last section concludes by discussing possible future works.

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 work in [12, 13] studies 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 [14] 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 [15] proposes an approach to select 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.

Since the games we introduced are a variant of sabotage games, our work is indebted to those dedicated to this type of games and to Sabotage Modal Logic [11, 16, 17]. In particular, the authors of [16] shows the decidability of the model-checking problem for Sabotage Modal Logic. Such result is important to our work because we use it to show the decidability of the model-checking problem for Subset Sabotage Modal Logic. In [17] the authors develops a complete proof system for Sabotage Modal Logic, they define and study the notion of bisimulation, and they with an extensive discussion of sabotage games and their characterisation via the logic.

3. Sabotage Modal Logic

In this section, we briefly present Sabotage Modal Logic (SML for short). Such logic was proposed in 2005 by Van Benthem [11] as a format for analyzing games that modify graphs they are played on. SML was later investigated in a series of papers . Before entering into the formal matter of SML, let us fix some notation and terminology that will be used in the following.

Notation & Terminology. *Given a sequence ρ, we denote is length as $|\rho|$, and its $j+1$-th element as ρ_j. For $j \leq |\rho|$, let $\rho_{\geq j}$ be the suffix of ρ starting at ρ_j and $\rho_{\leq j}$ the prefix $\rho_0 \cdots \rho_j$ of ρ. The empty sequence will be denoted by ϵ. Given a set of sequences X, we say that X is prefix-closed whenever given $\rho \in X$ then $\rho' \in X$ for any prefix ρ' of ρ. If X is a prefix-closed set of sequences, then $\langle X, \sqsubseteq \rangle$ is a tree where $\sqsubseteq$ denotes the prefix order. Accordingly, if X is finite, we will call* leaves *the $\sqsubseteq$-maximal elements of X. If $G = \langle V, E \rangle$ is a directed graph, a path is a sequence of vertices $v_0 \cdots v_n$ such that $(v_i, v_{i+1}) \in E$ for all $0 \leq i \leq n-1$.*

Given a non-empty set $\mathcal{P}$ of atomic formulas, and a finite non-empty set Σ of labels, we define SML formulas by the following grammar:

$$\varphi ::= p \mid \bot \mid \neg\varphi \mid \varphi \vee \varphi \mid \Diamond_a \varphi \mid \blacklozenge_a \varphi$$

where $p \in \mathcal{P}$ and $a \in \Sigma$. We define $\top \doteq \neg\bot$. If φ and ψ are formulas, we define $\varphi \to \psi \doteq \neg\varphi \vee \psi$, $\phi \wedge \psi \doteq \neg(\neg\varphi \vee \neg\psi)$, $\Box_a \psi \doteq \neg \Diamond_a \neg\varphi$ and $\blacksquare_a \doteq \neg \blacklozenge_a \neg\varphi$.

We now define the structures that will serve as interpretation of SML-formulas

Definition 1. *A rooted Kripke structure is a tuple $M = \langle S, s_0, \Sigma, \{R_a\}_{a\in\Sigma}, V\rangle$ where S is a non-empty set of states, $s_0 \in S$ is the initial state, Σ is a finite set of labels, $\{R_a\}_{a\in\Sigma}$ is a family of binary relations over the set of states such that there is one relation for each label in Σ; we will sometimes call elements of a binary relation* edges. *Finally, $V : M \to 2^{\mathcal{P}}$ is the evaluation function, assigning a set of atomic formulas to any state $s \in S$.*

If $M = \langle S, s_0, \Sigma, \{R_a\}_{a\in\Sigma}, V\rangle$ is a Kripke structure and $(s_1, s_2) \in R_a$ for some $a \in \Sigma$, we write $M \setminus (s_1, s_2)$ to denote the Kripke structure obtained by erasing the pair (s_1, s_2) from the binary relation R_a.

The notion of satisfaction of a formula φ at a given state s of a Kripke structure M (written $M, s \models \varphi$) is inductively defined as follows:

$M, s \models \bot$ never

$M, s \models p$ iff $p \in V(s)$

$M, s \models \neg\varphi$ iff not $M, s \models \varphi$

$M, s \models \varphi \vee \psi$ iff $M, s \models \varphi$ or $M, s \models \psi$

$M, s \models \Diamond_a \varphi$ iff there is a $s' \in S$ such that $(s, s') \in R_a$ and $M, s' \models \varphi$.

$M, s \models \blacklozenge_a \varphi$ iff there is $(s_1, s_2) \in R_a$ such that $M \setminus (s_1, s_2), s \models \varphi$

We say that a formula φ is true in a rooted Kripke structure M (written $M \models \varphi$) iff φ is true at the initial state of M. As we can see from the above definition, the meaning of the boolean connectives of SML logic is the standard one. Equally, the meaning of the $\Diamond_a$ -connective is the standard meaning in modal logic: a formula $\Diamond_a \varphi$ is true at a given state s of a Kripke structure whenever s is adjacent (with respect to an edge labeled by a) to a vertex s' for which the property expressed by φ is true. The meaning of the $\blacklozenge_a$ -connective can be spelled out as follows: a formula $\blacklozenge_a \varphi$ is true at a given state s of a Kripke structure M whenever φ is true at s in the Kripke structure M' that is obtained by erasing a pair (s', s'') from the relation R_a of M.

Definition 2. *The model checking problem for Sabotage Modal Logic consists of the following data and question:*

Data 1*: an SML formula φ,*

Data 2*: a rooted finite Kripke structure M*

Question*: Is it the case that $M \models \varphi$?*

The proof of the followin theorem can be found in [16].

Theorem 1. *The model checking problem for Sabotage Modal Logic is PSPACE-complete*

4. Attack Graphs

The term Attack Graph has been first introduced by Phillips and Swiler [18]. Attack Graphs represent the possible sequence of attacks in a system as a graph. An Attack Graph may be generated by using the following informations:

1. a description of the system architecture (topology, configurations of components, etc.);
2. the list of the known vulnerabilities of the system;
3. the Attacker's profile (his capabilities, password 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. [19, 20, 21, 22, 23], and [24] for a survey. Each of the previously cited works introduced its own Attack Graph model with its specificity, and thus there is no standard definition of an Attack Graph. However, all introduced models can be mapped into a *canonical Attack Graph* as introduced in [25]. It is a labelled oriented graph, where:

1. 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 on the network topology, privilege level, obtained credentials, etc.);
2. 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).

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.

By abstracting all the data of the above discussion, one can see an Attack Graph as a directed graph togheter with a labeling of its vertices and edges. The labeling of vertices is used to specify which properties (the kind of properties mentionned in 1) are true at a certain vertex, while the edge labeling specifies the name of the action of the Attacker. As we have seen in the example above, it is also useful to specify the set of Attacker's target states. We thus define an Attack Graph as follows:

Definition 3. *Suppose that the set $\mathcal{P}$ contains an atomic proposition* win. *An* ***Attack Graph*** *is a tuple $AG = \langle S, s_0, \Sigma, \{R_a\}_{a\in\Sigma}, V, T\rangle$ where:*

– *$\langle S, s_0, \Sigma, \{R_a\}_{a\in\Sigma}, V\rangle$ is a rooted Kripke structure where the set S of states is finite and $V(s)$ is non empty for any $s \in S$;*

– *T is a non-empty subset of S such that* win $\in V(s)$ *for all $s \in T$.*

The set T represent the set of target states of the Attacker.

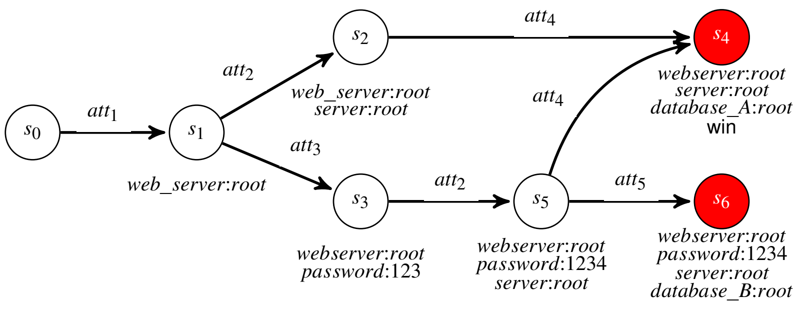


Figure 1: Example of Attack Graph, the atomic proposition satisfied at a given state are listed below the state itself.

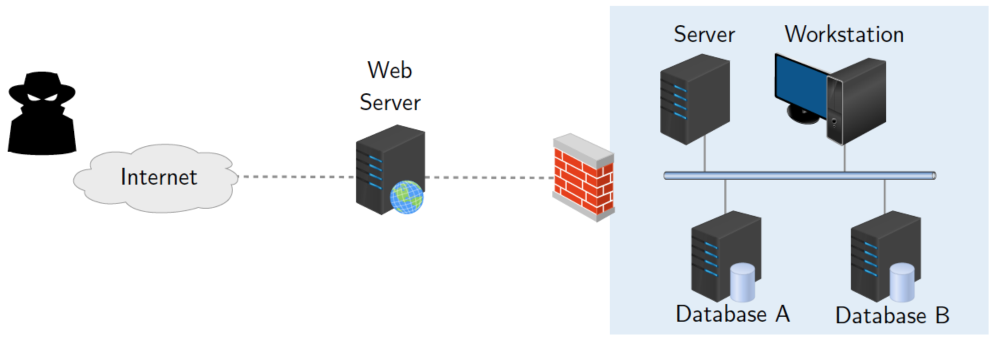


Figure 2: An illustrating LAN architecture example.

Example 1. *Consider the following scenario: an enterprise has a local area network (LAN) that features a* Server*, a* Workstation *and two databes A and B. The LAN also provided a* Web Server*. Internet's access to the LAN are controlled by a firewall. Such scenario is decipted in Figure 2. Suppose that we know some vulnerabilities and that we have established that a malevolent user can make the attack listed in Table 1, e.g., by making att_2 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.

Then we can construct an Attack Graph built from this set of atomic attacks and collecting possible attack paths as depicted in Figure 1[1]. 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, she can perform the sequences of atomic attacks att_2, att_4 or att_3, att_2, att_4.

[1]This Attack Graph is not complete w.r.t. our previous description, since some possible sequences of atomic attacks are not listed: for instance $att_1, att_2, att_3, att_5$ are not taken into account.

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 2.

5. Attack Graphs & Games

In the previous section, we saw that given a specific description of a system together with its vulnerabilities, we can generate a graph representing the dynamics of attacks that are possible over such system. Given a set of target states over such a graph, one can ask whether there is a path from an initial state to one of these target states, i.e., by reasoning over Attack Graphs, we can encode a security problem as a graph-reachability problem. Let us make one step more by adding a dynamic to such reachability problems. An Attack Graph represents a sequence of possible actions made by an Attacker to reach a specific goal. Let us add another character to this story, the Defender, whose objective is to counter the attack. Suppose that she has the power 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. Now that we have specified what is the Defender's power, we can consider a turn based game between an Attacker and a Defender. Both players play on an Attack Graph $\langle S, s_0, \Sigma, \{R_a\}_{a\in\Sigma}, V, T\rangle$. The Attacker's goal is to reach one of the states in T, while the Defender's goal is to prevent him from doing so. The Defender starts the game by selecting a certain countermeasure. By choosing such a countermeasure, she deletes a subset of edges of the Attack Graph. The Attacker take his turn and move from s_0 to one of its successor along on the edges that have not being erased (if any). The game evolve following this pattern. The Attacker won if he can reach one of the states in T in a finite number of moves, the Defender wins otherwise.

Example 2. *Consider again the Attack Graph of Figure 1. Suppose that the initial state is v_1. 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 as reported in the last column of Table 1. The Defender starts the game by deploying countermeasure c_3. The only edge of the Attack Graph labeled by att_3 is the one going from v_1 to v_3; consequently, such edge is erased from the Attack Graph and the Attacker can only move to v_2. The Defender take again his turn and now deploys countermeasure c_2. There are two edges that are labeled by att_2 and both are* *erased from the Attack Graph. The Attacker moves from v_2 to v_4 and, since v_4 is a target state, she wins the game.*

There is a natural way to look at the games described above: at each round the Attacker can follow an edge from his current position in the given graph, but the Defender can choose a new graph (which is a subgraph of the current graph) missing a subset of edges. To precisely define such plays, we first need of an auxiliary notion. We know that it is reasonable to assume that the Defender is unable to counter each of the Attacker's possible attacks. At each step of the game, she can only deactivate a relation contained in a certain subset of the Attack Graph relation set. As mentioned above, we can consider that the Defender selects a sub-graph of the current Attack Graph at each instant of the game. According to the limitation set out above, this sub-graph must be obtained by deleting some specific relations, i.e., it must be a sub-graph that is reachable modulo a subset of relations of the original graph. Since we can identify a given set of relations by considering their labels, we define this notion of reachability as follows.

Definition 4. *Let $AG = \langle S, s_0, \Sigma, \{R_a\}_{a\in\Sigma}, V, T\rangle$ be an Attack-graph and $\Delta \subseteq \Sigma$. The Attack Graph AG' is Δ-reachable from AG iff $AG' = \langle S, s_0, \Sigma, \{R_a\}_{a\in\Sigma\setminus\{b\}}, V, T\rangle$ for some $b \in \Delta$ or $AG' = AG$.*

We have now all the ingredients to define a play in our game.

Definition 5. *Let $AG = \langle S, s_0, \Sigma, \{R_a\}_{a\in\Sigma}, V, T\rangle$ be an Attack-graph and Δ a non-empty subset of Σ. A Δ-play ρ is a non-empty sequence ρ of lenght at least 2 where $\rho_0 = AG$, $\rho_1 = s_0$ and for all $2 < j \leq |\rho|$:*

1. *ρ_j is an Attack Graph if j is even.*
2. *ρ_j is a state of S if j is odd.*
3. *$(\rho_{j-2}, \rho_j) \in R_a$ is an edge in the Attack Graph $AG' = \rho_{j-1}$ for some $a \in \Sigma$, if j is odd.*
4. *ρ_j is Δ-reachable from ρ_{j-2}, if j is even.*

Let ρ be a finite Δ-play whose last element is stase s. We say that the Attack Graph AG' is legal for ρ iff $\mathsf{win} \notin V(s)$ and the sequence obtained by concatenating ρ to AG' is a Δ-play

Remark that given any Δ-play ρ and any even natural number $j < |\rho|$, the sequence $\rho_{\geq j}$ is a Δ-play over the Attack Graph obtained by setting the state ρ_{j+1} as initial state of the Attack Graph ρ_j.

Definition 6. *Let ρ be a Δ-play over an Attack Graph AG. We say that ρ is won by the Attacker iff $|\rho| = 2n$ for some $n \in \mathbb{N}$ and there is no Attack Graph AG' legal for ρ.*

Said plainly: the Attacker wins the game whenever she reaches one of her target states.

According to the above definition of play, the same state can appear many times in the same play. For instance, consider the Attack Graph AG shown in Figure 3. Let AG' be the Attack Graph obtained by deleting from AG the edge labeled by c. Consider the following $\{c\}$-play ρ over AG:

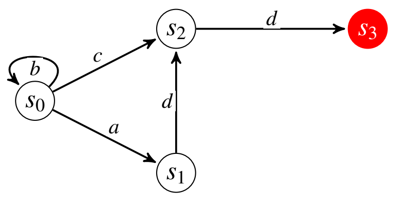


Figure 3: Another Example of Attack Graph. The red state is the only state in T

$$\begin{array}{cccccccccccc} AG & s_0 & AG' & s_0 & AG' & s_0 & AG' & s_1 & AG' & s_2 & AG' & s_3 \\ \rho_0 & \rho_1 & \rho_2 & \rho_3 & \rho_4 & \rho_5 & \rho_6 & \rho_7 & \rho_8 & \rho_9 & \rho_{10} & \rho_{11} \end{array}$$

We can see that the vertex v_0 appears three times in such a play. We define another class of plays in which this phenomenon cannot happen.

Definition 7. *Let ρ be a Δ-play over an Attack Graph AG. We say that ρ is* stubborn *whenever for any $0 \leq j \leq |\rho|$ if j is odd then there is no $i < j$ such that $\rho_i = \rho_j$.*

The definition above precludes the Attacker from choosing the same vertex of an Attack Graph in two differents turns of the play. As an example, the $\{a\}$-play presented above is not a $\{a\}$ stubborn play over AG because $\rho_1 = \rho_3 = \rho_5$.
Every stubborn Δ-play is a Δ-play. If ρ is a stubborn Δ-play over an Attack Graph AG, then $|\rho| \leq 2(|S| - 1)$, where S is the set of states of AG. This is because if s and s' are two distinct states of AG and there is a path $p_0 \cdots p_k$ where all the states p_i are distinct, then $k \leq |S| - 1$. Also, we remark that there is a path ρ from s to s' if and only such two verices are connected by a path ρ' in which no state appears twice. We thus obtain the following proposition for free:

Proposition 1. *For any Attack Graph $AG = \langle S, s_0, \Sigma, \{R_a\}_{a\in\Sigma}, V, T\rangle$ for any non-empty subset Δ of Σ, for any ρ : if ρ is Δ-play over AG, then there is a stubborn Δ-play ρ' over AG such that ρ' is won by the Attacker iff ρ is won by the Attacker.*

A strategy 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 history of the game is. We choose another equivalent definition of strategy. We informally describe how a strategy should operate and then formalize this notion. Imagine being engaged in a game $\mathcal{G}$, that the last move of $\mathcal{G}$ was played according to the strategy, and that it is now your Opponent's turn to play. Your Opponent could extend the game in different ways: for example if you are playing chess, you are white, and you just made your first move by moving a pawn to a certain position of the chessboard, black can in turn move a pawn or move a horse. If you are playing according to the strategy, the strategy should tell you how to react against either type of move. If black moves a pawn to $C6$ and you just moved your pawn to $C3$, then you move the horse to $H3$. If black moves a horse to $H6$ and you just moved your pawn to $C3$, then you move your pawn in $B4$. Therefore, a strategy can be viewed as a tree in which each node is a play in the game, the moves of my Opponent have at most one daughter, and my moves have as many daughters as there are available moves (with respect to the considered play) for my Opponent. A tree can be seen as a prefix-closed set of sequences; we can thus define strategies as follows.

Definition 8. *An Attacker Δ-strategy $\mathcal{S}$ for an Attack Graph AG is a non empty-prefix closed set of Δ-plays over AG, such that for any play Δ-play ρ and σ, for any move m, n, and n'*

- *if $\rho\, m \in \mathcal{S}$ and $\rho\, n \in \mathcal{S}$ and ρ has even length then $m = n$.*
- *if $\rho \in \mathcal{S}$ and ρ has odd length k, then $\rho\, m \in \mathcal{S}$ for any m that is Δ-reachable from ρ_{k-1}.*

A strategy is winning iff every of its elements has finite length, and every maximal (with respect to the prefix order) element of the strategy is won by the Attacker. Finally, we say that a strategy $\mathcal{S}$ is stubborn, *if ρ is a stubborn Δ-play, for any $\rho \in \mathcal{S}$.*

By proposition 1 above, we immediately obtain the following.

Proposition 2. *For any Attack Graph AG, for any non-empty subset Δ of the set Σ of labels of AG, there is a winning Δ-strategy over AG if and only if there is a winning Δ stubborn strategy over AG.*

6. Games & Subset Sabotage Modal Logic

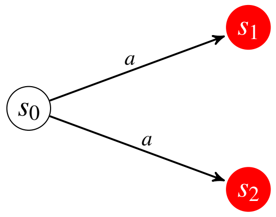
6.1. What is the problem with SML?

Plays, that we defined in the previous sections, are nothing more than runs in a insidious sabotage game. In a sabotage game, one of the two players can delete an edge of the graph when it is her turn to move. In our games, one of the two players can delete a *subset* of edges of the graph when it is her turn to move. In [17] the authors claims that, by using our terminology, the existence of an Attacker winning strategy for a Sabotage Game over a finite rooted Kripke structure $M = \langle S, s_0, \Sigma, \{R_a\}_{a\in\Sigma}, V\rangle$ can be expressed by using a particular SML-formula. Such formula is defined by induction on n,

$$\lambda_n = \begin{cases} \mathsf{win} & \text{if } n = 0 \\ \Diamond\top \wedge \blacksquare\Diamond(\lambda_{n-1} \vee \mathsf{win}) & \text{otherwise} \end{cases} \tag{1}$$

where $\blacksquare\varphi \doteq \bigwedge_{a\in\Sigma} \blacksquare_a \varphi$ and $\Diamond\varphi \doteq \bigvee_{a\in\Sigma} \Diamond_a \varphi$ for a given SML formula φ, and $\Diamond\top$ is used to check that the $\blacksquare$ is not satisfied because some relation is empty.

More precisely, if $M = \langle S, s_o, \Sigma, \{R_a\}, V\rangle$ is a Kripke structure such that $\mathsf{win} \in V(s)$ for some $s \in S$ and $|S| = n \geq 1$ then $M, s_o \models \mathsf{win} \vee \lambda_{n-1}$ if and only if there is an Attacker winning strategy for the sabotage game played over M. Such result is false in the case of the particular class of sabotage game that we have defined in the previous section, precisely because the Defender erases a relation R_a at each step of the game. For instance, consider the following Attack Graph:



We can see that there is no $\{a\}$-winning strategy over M. If the Defender erases any edge labeled with a, the Attacker would not be able to move from s_o to one of the winning states s_1 or s_2. On the contrary, the formula $\mathsf{win} \vee \lambda_2 = \mathsf{win} \vee (\Diamond\top \wedge \blacksquare\Diamond(\lambda_1 \vee \mathsf{win}))$ is true at s_o in M. This is because if the top-most a-edge is removed, the Attacker can pass from the lowermost edge to reach a winning state, and he can pass from the top-most one if the lower one is removed.

6.2. Subset Sabotage Modal Logic

To speak about our particular games we introduce a variant of Sabotage Modal Logic. We call such variant of Sabotage Modal Logic, Subset Sabotage Modal Logic.

Definition 9. *Given a non-empty set $\mathcal{P}$ of atomic propositions and a finite non-empty set Σ of labels, formulas of Subset Sabotage Modal Logic (SSML for short) are defined by the following grammar:*

$$\varphi ::= p \mid \bot \mid \neg\varphi \mid \varphi \vee \varphi \mid \Diamond_a \varphi \mid \blacklozenge_a^{\subseteq} \varphi$$

where $p \in \mathcal{P}$ and $a \in \Sigma$. Given a rooted Kripke structure $M = \langle S, s_0, \Sigma, \{R_a\}_{a\in\Sigma}, V\rangle$, the definition of satisfaction of a SSML formula at a state s of a Kripke is defined inductively. Such definition is the same as SML for atomic proposition, negated formulas, disjunction and the diamond modality. It is defined as follows for the $\blacklozenge_a^{\subseteq}$-modality:

$M, s \models \blacklozenge_a^{\subseteq} \varphi$ iff there is a non-empty subset E of R_a such that $M \setminus E, s \models \varphi$

where $M \setminus E$ denotes the structure M from which we have erased the subset E of edges. If φ is an SSLML formula we define $\blacksquare_a^{\subseteq} \varphi \doteq \neg \blacklozenge_a^{\subseteq} \neg\varphi$.

Remark 1. *$M, s \models \blacksquare_a^{\subseteq} \varphi$ if and only if for any non-empty subset E of R_a we have that $M\setminus E, s \models \varphi$, R_a included. This implies that $M, s \models \blacksquare_a^{\subseteq} \varphi$ iff $M \setminus R_a, s \models \varphi$.*

We now define the model checking problem for SSML.

Definition 10. *The model checking problem for SSML consists of the following data and question:*

***Data 1**: a finite rooted Kripke structure M;*

***Data 2**: a SSML formula φ,*

***Question**: is it the case that $M \models \varphi$?*

In what follows, we show that the model checking problem for SSML is decidable. To do so, we reduce the model checking problem for SSML to the model checking problem of SML. The idea of the proof is the following: we are considering a finite Kripke structure. As a consequence, any of its non-empty subset of edges is finite. We give a name n_e to each edge e of M obtaining a Kripke structure M'. We remark that $M \models \blacklozenge_a^{\subset} \varphi$ iff there is a finite, non-empty subset of edges $\{e_1, \dots e_n\}$ of R_a such that $M \setminus \{e_1, \dots, e_n\} \models \varphi$. If φ is a SML formula then $M' \models \blacklozenge_{n_{e_1}} \varphi \wedge \cdots \wedge \blacklozenge_{n_{e_n}} \varphi$, that is $M' \setminus \{n_{e_1}, \dots n_{e_n}\} \models \varphi$, where each n_{e_i} is the name of edge e_i. Thus, we need to give a translation from SSML-formulas to SML formulas. Such translation will be parametrized by Kripke structures, because we need to consider their subsets of edges.

Definition 11. *Let $M = \langle S, s_o, \Sigma, \{R_a\}_{a \in \Sigma}, V \rangle$ be a finite rooted Kripke structure. For all $a \in \Sigma$, let $f : R_a \to \mathbb{N}$ be an injective function associating to each member e of R_a a natural number n_e. We define the structure $M^{\subset} = \langle S^{\subset}, s_0^{\subset}, \Sigma^{\subset}, \{R_a\}_{a \in \Sigma^{\subset}}, V^{\subset} \rangle$ as follows:*

- *the set of states of $M^{\subset}$ and its initial state are the same of M*
- *the set of labels $\Sigma^{\subset}$ is $\bigcup_{a \in \Sigma} \{n_e \mid e \in R_a\}$*
- *a pair of states $(s, s') \in R_{n_e}$ iff $f(s, s') = n_e$. Remark that this implies that each relation on $M^{\subset}$ is either empty or a singleton.*
- *the evaluation function $V^{\subset}$ is V.*

Lemma 1. *Let M be a finite rooted Kripke Structure, R_a a relation on M, E any non-empty subset of R_a and $M^{\subset}$ the corresponding Kripke structure introduced above. If $E = \{e_1, \dots, e_n\}$ then:*

$$(M \setminus E)^{\subset} = M^{\subset} \setminus \{R_{n_{e_1}}, \dots, R_{n_{e_n}}\}$$

Proof. It follows immediately from the definition of $M^{\subset}$. □

We define a function that maps an SSML formula φ to an SML formula $(\varphi)^{\star}_M$. Such function takes as argument a Kripke Structure M and an SSML formula φ, and gives as result an SML formula $(\varphi)^{\star}_M$. The function is defined on the structure of φ.

$$\begin{aligned}
(p)^{\star}_M &= p \\
(\bot)^{\star}_M &= \bot \\
(\neg\varphi)^{\star}_M &= \neg(\varphi)^{\star}_M \\
(\varphi \vee \psi)^{\star}_M &= (\varphi)^{\star}_M \vee (\psi)^{\star}_M \\
(\Diamond_a \varphi)^{\star}_M &= \bigvee_{e \in R_a} \Diamond_{n_e} (\varphi)^{\star}_M \\
(\blacklozenge_a^{\subset} \varphi)^{\star}_M &= \bigvee_{E \in 2^{R_a} \setminus \emptyset} (\bigwedge_{e \in E} \blacklozenge_{n_e} (\varphi)^{\star}_M)
\end{aligned}$$

Lemma 2. *For any $M = \langle S, s_o, \Sigma, \{R_a\}_{a\in\Sigma}, V\rangle$, for any SSML formula φ, for any state $s \in S$:*

$$M, s \models \varphi \iff M^{\subset}, s \models (\varphi)^{\star}_{M}$$

Proof. The proof is by induction on the structure of φ. We omit the subscript M since it is clear from the context. If $\varphi = p$ or $\varphi = \bot$ the result is immediate since $V(s) = V^{\subset}(s)$ for any state s. Suppose that the statement of the lemma holds for any formula of size $k \leq n$, and let φ be a formula of size $n + 1$. If the main connective of φ is a boolean connective, the result follows by the induction hypothesis. If $\varphi = \Diamond_a \psi$:

($\Rightarrow$) $M, s \models \Diamond_a \psi$ iff there is s' such that $(s, s') \in R_a$ and $M, s' \models \psi$. By induction hypothesis $M^{\subset}, s' \models (\psi)^{\star}$. By the definition of $M^{\subset}$ we have that $e = (s, s') \in R_{n_e}$ thus s and s' are adjacent, with respect to R_{n_e}, in $M^{\subset}$. It follows that $M^{\subset}, s \models \Diamond_{n_e} (\psi)^{\star}$. But this means that $M^{\subset}, s \models \bigvee_{e\in R_a} \Diamond_{n_e} (\psi)^{\star}$ as we wanted.

($\Leftarrow$) $M^{\subset}, s \models (\Diamond_a \psi)^{\star}$ iff $M^{\subset}, s \models \bigvee_{e\in R_a} \Diamond_{n_e} (\psi)^{\star}$. If t his is the case, then there is at least an edge $e = (s, s') \in R_a$ such that $M^{\subset}, s \models \Diamond_{n_e} (\psi)^{\star}$ which implies that $M^{\subset} s' \models (\psi)^{\star}$. By induction hypothesis $M, s' \models \psi$. Since, by definition of $M^{\subset}$, $(s, s') \in R_a$ in M, we get the result.

If $\varphi = \blacklozenge^{\subset}_{a} \psi$:

($\Rightarrow$) $M, s \models \blacklozenge^{\subset}_{a} \psi$ iff there is a non-empty subset $E = \{e_1, \dots e_n\}$ of R_a such that $M \setminus E, s \models \psi$. By induction hypothesis $(M \setminus E)^{\subset}, s \models (\psi)^{\star}$. By definition, $(M \setminus E)^{\subset} = M^{\subset} \setminus \{R_{n_{e_1}}, \dots R_{n_{e_n}}\}$. Since $M^{\subset} \setminus \{R_{n_{e_1}}, \dots R_{n_{e_n}}\}, s \models (\psi)^{\star}$ we deduce that $M^{\subset}, s \models \bigwedge_{e\in E} \blacklozenge_{n_e}$ and we can conclude.

($\Leftarrow$) $M^{\subset}, s \models (\blacklozenge^{\subset}_{a} \psi)^{\star}$ iff $M^{\subset}, s \models \bigvee_{E\in 2^{R_a}\setminus\emptyset}(\bigwedge_{e\in E} \blacklozenge_{n_e} (\psi)^{\star})$. This means that there is a E subset of R_a such that $E = \{e_1, \dots e_n\}$ and $M^{\subset}, s \models \blacklozenge_{n_{e_1}} (\psi)^{\star} \wedge \cdots \wedge \blacklozenge_{n_{e_n}} (\psi)^{\star}$. By definition of satisfaction, this means that $M^{\subset} \setminus R_{n_{e_i}}, s \models (\psi)^{\star}$ for any $i \leq n$, thus that $M^{\subset} \setminus \{R_{n_{e_1}}, \dots, R_{n_{e_n}}\}, s \models (\varphi)^{\star}$. By lemma 1, this implies that $(M \setminus E)^{\subset}, s \models (\psi)^{\star}$. By induction hypothesis, $M \setminus E, s \models \psi$ and thus $M, s \models \blacklozenge^{\subset}_{a} \psi$ as we wanted.

□

From the above lemma and the fact the model checking problem is decidable for SML, we immediately deduce the following theorem:

Theorem 2. *The model checking problem for SSML is decidable: if $M = \langle S, s_0, \sigma, \{R_a\}_{a\in\sigma}, V\rangle$ is a finite rooted Kripke Structure and φ an SSML formula, we can decide whether $M \models \varphi$ or not.*

It should be no surprise that we can express the existence of Attacker winning strategies for our games by using SSML: we have designed such logic with precisely this goal in mind.

Let $AG = \langle S, s_0, \{R_a\}_{a\in\Sigma}, V, T\rangle$ be an Attack Graph and Δ a non-empty subset of Σ. If φ is an SSML formula, we define the two SSML-formulas:

$$\blacksquare^{\subset}_{\Delta} \varphi \doteq \bigwedge_{a\in\Delta} \blacksquare^{\subset}_{a} \varphi \qquad \Diamond \varphi \doteq \bigvee_{a\in\Sigma} \Diamond_a \varphi$$

A strategy is a plan of action. As it is logical, the plan is winning when it leads me to victory, whatever my opponent's plan of action. Thus, a winning strategy can be expressed as an alternance of universally quantified sentences and existentially quantified sentences "for all actions of my Opponent, there is an action that I can make that leads me to victory". Let us put ourselves in the villain's shoes: suppose that we are the Attacker, and that, by playing, we have reached a certain state s of an Attack Graph AG. It is now Defender's turn. If a have a winning strategy, I must be able to reach a successor state s' of s in whatever subgraph AG' of AG that is Δ-reachable from AG. Said differently, we must have that $AG, s \models \blacksquare^{\subset}_{\Delta} \diamond \varphi = (\blacksquare^{\subset}_{a} \diamond \varphi) \wedge \cdots \wedge (\blacksquare^{\subset}_{b} \diamond \varphi)$ for some formula φ that expresses the winning condition. Such formula is nothing but the SSML version of the one we have defined in 1.

Definition 12 (Winning Formulas)**.** *The family $\{\psi^n_{\Delta}\}_{n\in\mathbb{N}}$ of Winning formulas is defined by induction on n as follows:*

$$\psi^n_{\Delta} = \begin{cases} \mathsf{win} & \text{if } n = 0 \\ \diamond\top \wedge \blacksquare^{\subset}_{\Delta} \diamond(\psi^{n-1}_{\Delta} \vee \mathsf{win}) & \text{otherwise} \end{cases} \tag{2}$$

We are now ready to prove the main result of our paper. Namely, that the existence of a winning Attacker Δ-strategy over an Attack Graph AG is equivalent to the truth of a winning formula over AG.

Theorem 3. *For any Attack Graph $AG = \langle S, s_0, \Sigma, \{R_a\}_{a\in\Sigma}, V, T\rangle$ for any non-empty subset Δ of Σ, if $|S| = n$, then $AG, s_0 \models \mathsf{win} \vee \psi^{n-1}_{\Delta}$ iff there is a winning Δ-strategy over AG.*

Proof. We prove the $\Rightarrow$-direction of the theorem by induction on n. An Attack Graph has a non-empty set of states by definition. Thus, the base case is $n = 1$. $AG, s_0 \models \mathsf{win} \vee \mathsf{win} \iff T = \{s_0\}$. The strategy we are looking for is AG, s_o.

Suppose that the statement of the theorem holds for any Attack Graph AG' of size $k \leq n$, and let AG be an Attack Graph of size $n+1$. Suppose that $AG, s_0 \models \mathsf{win} \vee \psi^n_{\Delta}$. Without loss of generality, we can suppose that $AG, s_0 \not\models \mathsf{win}$ because otherwise the result is trivial. Thus $AG, s_0 \models \psi^n_{\Delta}$. This means that $AG, s_0 \models \diamond\top \wedge (\blacksquare^{\subset}_{\Delta} \diamond (\psi^{n-1}_{\Delta} \vee \mathsf{win}))$. By the definition of satisfaction, $AG, s_0 \models (\diamond\top \wedge \blacksquare^{\subset}_{a} \diamond (\psi^{n-1}_{\Delta} \vee \mathsf{win}))$ for any $a \in \Delta$. Let a one of the element of Δ. We have that, $AG, s_0 \models \diamond\top \wedge \blacksquare^{\subset}_{a} \diamond (\psi^{n-1}_{\Delta} \vee \mathsf{win})) \iff AG, s_0 \models \diamond\top$ and $AG, s_o \models \blacksquare^{\subset}_{a} \diamond (\psi^{n-1}_{\Delta} \vee \mathsf{win}))$. We can thus conclude that there is a $s' \neq s$ such that $(s, s') \in R_c$ for some $c \neq a$ and $s' \models \psi^{n-1}_{\Delta} \vee \mathsf{win}$. Consider the subgraph AG_a of AG obtained by erasing the vertex s_o, any edge that has s_o as source, any edge that is labeled by the letter a and in which the initial state is s'. The size of such graph is n and $AG_a, s' \models \psi^{n-1}_{\Delta} \vee \mathsf{win}$. By induction hypothesis, there is a winning Δ-strategy $\mathcal{S}_a$ over AG_a. We write $AG, s_0 \star \mathcal{S}_a$ for the set $\{\sigma \mid \sigma = AG, s_0, \rho_a$ and $\rho_a \in \mathcal{S}_a$ and $\rho_a \neq \epsilon\}$. It is clear that

$$\mathcal{S} = \{\rho \mid \rho \sqsubseteq \rho' \text{ and } \rho' \in \bigcup_{a\in\Delta} (AG, s_0 \star \mathcal{S}_a)\}$$

is a winning Δ-strategy over AG.

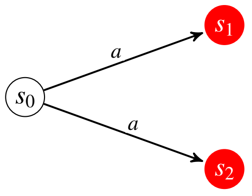
For the other direction, we prove that if $AG, s_0 \not\models \mathsf{win} \vee \psi^{n-1}_{\Delta}$ then there is no winning strategy $\mathcal{S}$ over AG. We have that $AG, s_0 \not\models \mathsf{win}$ thus $s_0 \notin T$. From $AG, s_0 \not\models \psi^{n-1}_{\Delta}$ we deduce that there is a $b \in \Delta$ such that $AG, s_0 \not\models \diamond\top \wedge \blacksquare^{\subset}_{b} \diamond \psi^{n-2}_{\Delta}$. There are two possibilities:

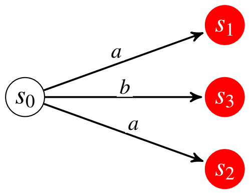
1. If $AG, s_0 \not\models \Diamond\top$, then there is no edge having s_0 has source in the graph AG' obtained from AG by deleting each edge in R_b. Thus, we cannot construct a play ρ won by the Attacker such that $AG, s_o, AG' \sqsubseteq \rho$.

2. if $AG, s_0 \models \Diamond\top$ then $AG, s_0 \not\models \blacksquare^{\subseteq}_b \Diamond \psi^{n-2}_{\Delta}$ for any $s' \in S$ such that $(s_0, s') \in R_c$ for $c \neq b$ we have that $AG, s' \models \mathsf{win} \vee \blacksquare^{\subseteq}_c \Diamond \psi^{n-3}_{\Delta}$, for some $c \in \Delta$ (not necessarily different from b). There are again two possibilities, and we can use the same line of reasoning used here or in (1). We can thus conclude.

□

Example 3. *Conside the two following Attack Graphs:*





There is no $\{a\}$-winning strategy over the Attack Graph on the left. In fact, it does not satisfy the formula $\mathsf{win} \vee (\Diamond\top \wedge \blacksquare^{\subseteq}_a \Diamond ((\Diamond\top \blacksquare^{\subseteq}_a \Diamond \mathsf{win}) \vee \mathsf{win}))$ *In fact* $s_0 \not\models \mathsf{win}$ *and* $s_0 \not\models \blacksquare^{\subseteq}_a \Diamond ((\Diamond\top \blacksquare^{\subseteq}_a \Diamond \mathsf{win}) \vee \mathsf{win})$. *If we erase any edge labeled by the letter a, no edges are left. Thus* $s_0 \not\models \Diamond\varphi$ *for any* φ.

Consider the Attack Graph on the right, call it AG_r and let AG^a_r be AG_r without edges labeled by a and AG^b_r be the AG_r without the edge labeled by b. The formula $\mathsf{win} \vee \psi^{\{a,b\}}_3$ *is satisfied by AG_r, and there is a winning strategy $\mathcal{S}$ (we display only maximal plays).*

$$\mathcal{S} = \{AG_r\, s_o\, AG^a_r\, s_3,\ AG_r\, s_0\, AG^b_r\, s_1\}$$

7. Conclusion and Future Work

We have presented a natural class of two-player games over Attack Graphs. Such games are played by an Attacker and a Defender. The Attacker tries to reach some vertex of the Attack Graph while the Defender tries to tries to prevent him from doing so. To do this, the Defender can eliminate subsets of graph arcs. We have seen how these games can be viewed as a generalized version of sabotage games, we have formally defined plays of such games and winning strategies. Finally, we have introduced a variant of Sabotage Modal Logic, showed the the model checking problem for such logic is decidable and that we can express the existence of a winning strategies for our subset sabotage games by formulas of the logic.

The games we have defined are perfect information games; both players know, at every point in the game, the location of the other player. This assumption is unrealistic: during a cyber attack, a possible Defender may not know what state an Attacker is in, and conversely, an Attacker may not be aware of changes made by the Defender to counter his attack. We would therefore like to extend our play model in order to include this type of imperfect information. From the Defender's point of view this could be implemented as an equivalence class between Attack Graph states. From the Attacker's point of view, on the other hand, we could think of a notion of weak bisimulation between Attack Graphs: the Attacker considers as equal two models that are bisimilar up to identification of some subset of arcs.

We have shown that the model checking problem for SSML logic is decidable. However, we have not investigated the complexity of that problem (which must, however, be at least P-Space). We leave this investigation for future work. We suspect that a bisimulation notion for SSML logic can be obtained by slightly modifying the one for SML and that, at the same, a complete proof system for SSML can be obtained in terms of tableaux. In conclusion, we suspect that the satisfiability problem for SSML logic is undecidable. Indeed, the same problem is undecidable for SML and since our logic is SML in which we quantify over subset of arcs of a graph, our intuition tells us that the satisfiability problem for SSML can only be more difficult than the one of SML.

References

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

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

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

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

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

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

[7] F. Mogavero, A. Murano, G. Perelli, M. Y. Vardi, Reasoning about strategies: On the model-checking problem, ACM Transactions in Computational Logic 15 (2014) 34:1–34:47. URL: http://doi.acm.org/10.1145/2631917. doi:10.1145/2631917.

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

[9] N. R. Jennings, M. Wooldridge, Application of intelligent agents, in: Agent Technology: Foundations, Applications, and Markets, Springer-Verlag, 1998.

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

[11] J. van Benthem, An Essay on Sabotage and Obstruction, Springer Berlin Heidelberg, Berlin, Heidelberg, 2005, pp. 268–276. URL: https://doi.org/10.1007/978-3-540-32254-2_16. doi:10.1007/978-3-540-32254-2_16.

[12] K. Durkota, V. Lisý, B. Bosanský, C. Kiekintveld, Approximate solutions for attack

graph games with imperfect information, in: M. H. R. Khouzani, E. A. Panaousis, G. Theodorakopoulos (Eds.), Decision and Game Theory for Security - 6th International Conference, GameSec 2015, London, UK, November 4-5, 2015, Proceedings, volume 9406 of *Lecture Notes in Computer Science*, Springer, 2015, pp. 228–249. URL: https://doi.org/10.1007/978-3-319-25594-1_13. doi:10.1007/978-3-319-25594-1_13.

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

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

[15] Y. Zhang, P. Malacaria, Bayesian stackelberg games for cyber-security decision support, Decis. Support Syst. 148 (2021) 113599. URL: https://doi.org/10.1016/j.dss.2021.113599. doi:10.1016/j.dss.2021.113599.

[16] C. Löding, P. D. Rohde, Model checking and satisfiability for sabotage modal logic, in: FSTTCS, 2003.

[17] G. Aucher, J. V. Benthem, D. Grossi, Modal logics of sabotage revisited, Journal of Logic and Computation 28 (2018) 269 – 303. URL: https://hal.inria.fr/hal-01827076. doi:10.1093/logcom/exx034.

[18] C. Phillips, L. P. Swiler, A graph-based system for network-vulnerability analysis, in: Proceedings of the 1998 workshop on New security paradigms, 1998, pp. 71–79.

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

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

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

[22] X. Ou, W. F. Boyer, M. A. McQueen, A scalable approach to attack graph generation, in: Proceedings of the 13th ACM conference on Computer and communications security, 2006, pp. 336–345.

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

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

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