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 quantifies 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 at a very early stage of its life-cycle. 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. 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* [2] 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 attack actions by the Attacker. Then, it is a system duty to prevent unauthorized accesses from the Attacker in each state of the graph. 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 succeed. 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

WOA 2022: 23rd Workshop From Objects to Agents, September 1–2, Genova, Italy
[†]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)

Graph under exam) 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* [3]. 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 subset of its edges". We show 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.

2. Sabotage Modal Logic

In this section, we briefly present Sabotage Modal Logic (SML for short). Such logic was proposed in 2005 by Van Benthem [3] as a format for analyzing games that modify graphs they are played on. First, let us fix some notation and terminology that will be used in the following.

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 is inductively defined as follows (clauses for Boolean connectives are immediate and thus omitted):

$M, s \models \bot$ never

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

$M, s \models \Diamond_a \varphi$ 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 $\Diamond_a$ -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.

We conclude the section with the following theorem (the proof can be found in [4]).

Theorem 1 (Model checking problem for SML)**.** *Given a finite rooted Kripke structure M and an SML formula φ the problem of deciding wheter $M \models \varphi$ is PSPACE-complete.*

3. Attack Graphs

The term Attack Graph has been first introduced by Phillips and Swiler [5]. Attack Graphs represent the possible sequence of attacks in a system as a graph. It 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. [6, 7, 8, 9, 10], and [11] 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 [12]. 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 attack action. We define an Attack Graph as follows.

Definition 2. *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 represents the set of target states of the Attacker.

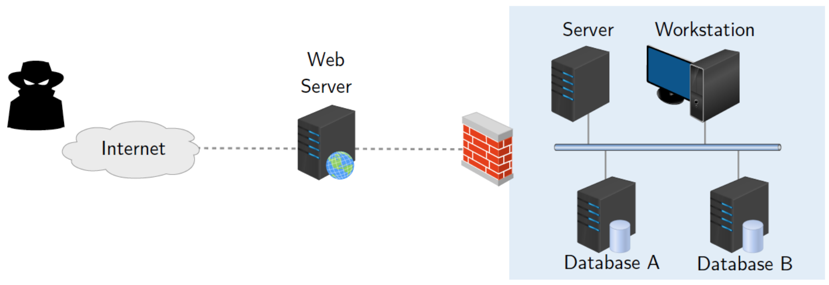


Figure 1: An illustrating LAN architecture example.

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 1. 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	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.

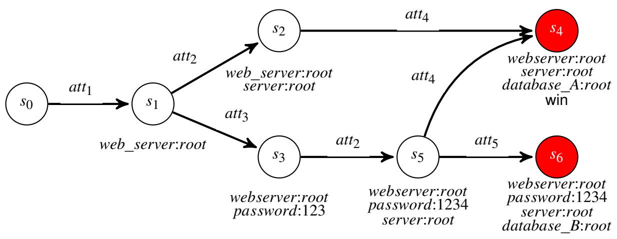


Figure 2: Example of Attack Graph, the atomic proposition satisfied at a given state are listed below the state itself.

Attack Graph built from this set of atomic attacks and collecting possible attack paths as depicted in Figure 2[1]. The Attacker's initial state is a node in the Attack Graph. Let us suppose that the Attacker is in state s_1 and wants to reach state s_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.

4. Games & Subset Sabotage Modal Logic

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 problem. 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

[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.

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, otherwise the Defender wins.

Example 2. *Consider again Figure 2. Suppose that the initial state is s_1 and 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 s_1 to s_3; consequently, such edge is erased from the Attack Graph and the Attacker can only move to s_2. The Defender takes again her 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, he 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. More precisely, the defender can choose to erase all edges that are labeled by the same atomic attack. Given a set of labels Σ, and an attack graph AG whose edges are labeled by members of Σ, a game that is played over AG in which the Defender can erase label that are in $\Delta \subseteq \Sigma$ is called a Δ-game. A winning strategy for a Δ game is called a Δ-winning strategy.

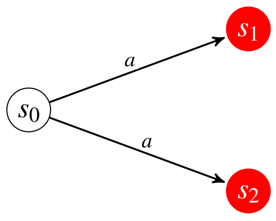
4.1. What is the problem with SML?

The games that we described in the previous section are nothing more than an insidious version of sabotage games. 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 [13] 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 games that we have defined in the previous section, precisely because the Defender can erase *at the same time* all edges that are labeled by the same symbol 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 all the edges 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.

4.2. 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 3. *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 an 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 SSML 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 if $M, s \models \blacksquare_a^{\subseteq} \varphi$ then $M \setminus R_a, s \models \varphi$.*

We now define the model checking problem for SSML. To show that the model checking problem for SSML is decidable, we reduce it 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 $M \models \blacklozenge_a^{\subseteq} \varphi$ iff there is a finite, non-empty subset of edges

$\{e_1, \ldots e_n\}$ of R_a such that $M \setminus \{e_1, \ldots, e_n\} \models \varphi$. If φ is a SML formula then $M' \models \blacklozenge_{n_{e_1}} \cdots \blacklozenge_{n_{e_n}} \varphi$, that is $M' \setminus \{n_{e_1}, \ldots n_{e_n}\} \models \varphi$, 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 4. *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 $f(s, s') = n_e$[2];*
- *the evaluation function $V^{\subset}$ is V.*

If $E = \{e_1, \ldots, e_k\}$ is a subset of edges of M and φ a formula, we write $\blacklozenge_{n_E} \varphi$ as a shortcut for the formula $\blacklozenge_{n_{e1}} \cdots \blacklozenge_{n_{ek}} \varphi$ where each n_{ei} is the label corresponding to e_i in $M^{\subset}$.

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.

$$\begin{array}{ccccccc}
(p)^\star_M & = & p & \quad & (\bot)^\star_M & = & \bot \\
(\neg\varphi)^\star_M & = & \neg(\varphi)^\star_M & & (\varphi \vee \psi)^\star_M & = & (\varphi)^\star_M \vee (\psi)^\star_M \\
(\Diamond_a \varphi)^\star_M & = & \bigvee_{e\in R_a} \Diamond_{n_e} (\varphi)^\star_M & & (\blacklozenge^{\subset}_a \varphi)^\star_M & = & \bigvee_{E\in 2^{R_a}\setminus\emptyset}(\blacklozenge_{n_E} (\varphi)^\star_M)
\end{array}$$

One can prove the following lemma by induction on the structure of φ.

Lemma 1. *For any $M = \langle S, s_o, \Sigma, \{R_a\}_{a\in\Sigma}, V\rangle$, for any SSML formula φ, for any state $s \in S$:*

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

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

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:

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

[2]This implies that each relation on $M^{\subset}$ is either empty or a singleton.

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 alternation 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 5 (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 provide 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. The $\Rightarrow$-direction of the theorem can be proved by induction on the size n of the Attack Graph. The $\Leftarrow$-direction can be proved by contraposition.

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.*

5. 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 prevent him from doing so. To do this, the Defender can eliminate subsets of graph arcs. 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 described 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.

The model checking problem for SSML is decidable. However, we have not investigated the complexity of such 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] R. P. Lippmann, K. W. Ingols, An annotated review of past papers on attack graphs (2005).

[3] 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.

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

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

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

[12] 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).

[13] 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.