A Formal Verification Approach to Handle Attack Graphs

Davide Catta[1], Jean Leneutre[2], Antonina Mijatovic[2], Johanna Ulin[2] and Vadim Malvone[2]
[1] *Università degli Studi di Napoli Federico Secondo, Naples, Italy*
[2] *Télécom Paris, Palaiseau, France*

Keywords: Attack Graphs, Formal Verification, Multi-Agent Systems

Abstract: We propose a formalization of attack graphs through a multi-agent approach. Specifically, we focus on dynamic scenarios that capture the interaction between an attacker and defenders during a cyberattack. We introduce a formal definition of an attack graph using interpreted systems, demonstrating how this formalization enables us to express interesting security properties. Finally, we present a tool AG2IS, which we have developed as an implementation of our formal definitions, to perform the formal verification of attack graphs.

1 Introduction

With the increasing use of computer networks and the internet, cybersecurity has become a major concern. Malevolent users constantly attempt to exploit vulnerabilities in computer systems to gain unauthorized access to sensitive information, disrupt critical infrastructure, or commit other malicious activities. One of the ways to analyze the security of computer networks is by using Attack Graphs.

An Attack Graph is a graphical representation of all possible ways a malevolent user can penetrate a computer network by exploiting its vulnerabilities. Attack Graphs are useful for identifying the weakest links in a network and devising strategies to improve its security. However, the size and complexity of Attack Graphs can make them challenging to analyze and verify manually.

In recent years, researchers have been exploring the use of formal methods to analyze Attack Graphs automatically (Jha et al., 2002; Catta et al., 2023a; Catta et al., 2023c). Formal methods are mathematical techniques used to verify the correctness of computer systems. They can be used to model, specify, and verify the behavior of a system. More in detail, they can help detect and fix errors early in the design process, before the system is implemented, thus reducing the risk of costly errors and security breaches. Most approaches to Attack Graph's formal verification are static and focus on verifying the existence of critical paths in the graph. Few works address a dynamic scenario, that captures the interactions between a defender and an attacker during an ongoing attack.

In this paper, we demonstrate that multi-agent systems can offer a more dynamic approach to address the formal verification of Attack Graphs. We begin by showing how Attack Graphs can be modeled by means of interpreted systems. In particular, we will consider interpreted systems with two kinds of agents: *attacker agents* and *defender agents*. An attacker agent represents a malevolent user who is trying to penetrate the network by exploiting its vulnerabilities, while a defender agent represents the security measures and policies that are in place to protect the network. The goal of the defender agent is to prevent the attacker agent from achieving its objectives, while the attacker agent tries to bypass the security measures and successfully carry out its attack. We then present an implementation of our theoretical definitions. More precisely, we build a converter called AG2IS between the Attack Graph generated by the Multihost, Multistage Vulnerability Analysis Language (MulVAL) (Ou et al., 2005) tool to an interpreted system following the Interpreted Systems Programming Language (ISPL) format (Lomuscio and Raimondi, 2006). In addition to specify the attacker's goal, the tool AG2IS provides a formula for verifying the possibility of a successful attack by the attacker in the interpreted system. The formal verification part is done via the Model Checking of Multi-Agent Systems (MCMAS) tool (Lomuscio and Raimondi, 2006).

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 goal is to find the Nash equilibrium in order to characterize the most promising attacker/defender's actions. 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. A few works adopt a formal approach to analyze games defined on Attack Graphs. 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. Finally, the works in (Catta et al., 2023a; Catta et al., 2023c; Catta et al., 2023b) share some ideas with ours on the cybersecurity side. However, the authors consider turn-based models and do not consider coalitions of defenders as we do. Finally, on the implementation side, the PRISM model-checker https://www.prismmodelchecker.org/ have been used several times to model attack- defence scenarios. See, for example: https://www.prismmodelchecker.org/casestudies/index.php#security and https://www.prismmodelchecker.org/games/casestudies.php.

2 Preliminaries

If X is a set, then $\overline{X}$ denotes its complement. We denote by X^+ the set of finite sequences of elements of X and by X^ω the set of infinite (countable) sequences of elements of X. If $\rho \in X^+ \cup X^\omega$ then $|\rho|$ denote its length, which is ω if ρ is infinite. If $i \leq |\rho|$ then ρ_i denotes the i-th element of ρ, $\rho_{\leq i}$ denotes the finite prefix $\rho_1, \ldots, \rho_i$ of ρ and $\rho_{\geq i}$ denotes the suffix of ρ starting at ρ_i. Finally, if ρ is finite $last(\rho)$ denotes its last element $\rho_{|\rho|}$. If $l = \langle x_1, \ldots, x_n \rangle$ is a tuple, we denote by $l[i]$ the i-th component x_i of the tuple.

Definition 1. *Given a non-empty set* Ap *of atomic propositions and a non-empty set* Σ *of labels, a Kripke Structure based on* Ap *and* Σ *is a tuple* $\mathfrak{M} = \langle W, R, \mathcal{L} \rangle$ *where* W *is a non-empty set of worlds (or states),* $R \subseteq W \times \Sigma \times W$ *is a ternary relation (the transition relation), and* $\mathcal{L}$ *is the labeling function sending any world to (an eventually empty) subset of atomic propositions.*

Given a subset Δ of Σ, we write R_Δ for the restriction of R to elements of Δ, that is $R_\Delta = \{\langle w, a, w' \rangle \in R \mid a \in \Delta\}$. In the rest of the paper, we will suppose that given a Kripke structure $\mathfrak{M}$, it holds that $\mathsf{Ap} = \bigcup_{w \in W} \mathcal{L}(w)$. A **rooted** Kripke structure is a pair formed by a Kripke structure $\mathfrak{M}$ and a subset W_I of the set of worlds of $\mathfrak{M}$. Elements of W_I are called initial worlds. We now recall the definition of interpreted systems (Fagin et al., 1995). First, we define agents.

Definition 2 (Agent)**.** *Given a finite set of agent index* $\mathcal{A} = \{1, \ldots, k\}$*, an* agent *is a tuple* $i = \langle L_i, act_i, P_i, t_i \rangle$*, where:* L_i *is the finite non-empty set of local states;* act_i *is the finite non-empty set of individual actions. We denote by* ACT *the product set* $\Pi_{i \in \mathcal{A}} act_i$ *and we call its element* joint actions. $P_i : L_i \to (2^{act_i} \setminus \emptyset)$ *is the local protocol function.* $t_i : L_i \times ACT \to L_i$ *is the local transition function. Such function takes an agent's state* l *and a joint action* $\mathbf{a} = \langle a_1, \ldots a_k \rangle$ *and outputs an agent's state. The function* $t_i(l_i, \mathbf{a})$ *is defined if and only if we have that* $\mathbf{a}[i] \in P_i(l_i)$*.*

By the above definition, an agent i is situated in a local state $l \in L_i$ representing the information it has about the system. At any state, the agent can perform the actions in act_i according to the protocol function P_i. A joint action determines a change in the state of the agent according to the transition function t_i.

If $\mathcal{A}$ is a set of agents of length k, a **global state** $s \in G$ is a tuple $s = \langle l_1, \ldots, l_k \rangle$ where each l_i is an i agent's state for $i \leq k$. A **history** is a finite sequence $h = s_1, \ldots, s_n$ of global states. We denote by H the set of histories. Two global states s and s', are **equivalent** for the agent i whenever $s[i] = s'[i]$. We denote such notion by $s \sim_i s'$. Two histories h and h' are equivalent for the agent i whenever they have the same length m and $h_j \sim_i h'_j$ for any $j \leq m$.

Definition 3. *An interpreted system is a tuple* $I = \langle \mathcal{A}, G_I, T, \Pi \rangle$ *where* $\mathcal{A}$ *is a set of agents,* $G_I \subseteq G$ *is the set of global initial states.* $T : G \times ACT \to G$ *is the global transition function such that* $T(s, \mathbf{a}) = s'$ *iff for every* $i \in \mathcal{A}$*,* $t_i(s[i], \mathbf{a}[i]) = s'[i]$. $\Pi : G \to 2^{\mathsf{Ap}}$ *is the labelling function.*

A path ρ is an infinite sequence of global states, $\rho = s_1, s_2, s_3, \ldots$ respecting the following constraint: for every $i \geq 1$, there is a joint action $\mathbf{a}$ such that $T(s_i, \mathbf{a}) = s_{i+1}$. We denote paths by ρ, τ, and π.

Definition 4. *A uniform strategy (strategy for short) for an agent* i *is a function* $\sigma_i : H \to act_i$ *such that:*

1. *for every* $h \in H$ *we have that* $\sigma_i(h) \in P_i(last(h))$*.*
2. *For every pair of histories* h *and* h'*,* $h \sim_i h'$ *implies* $\sigma_i(h) = \sigma_i(h')$*.*

A strategy is memoryless when for every pair of histories h *and* h' *we have that* $last(h) = last(h')$ *implies* $\sigma(h) = \sigma(h')$*.*

Given a coalition Γ, a uniform strategy for Γ, or simply Γ-strategy, is a collection σ_Γ of uniform strategies comprising one strategy σ_i for each $i \in \Gamma$. Given a path ρ, we say that ρ is σ_Γ**-compatible** iff for every $j \geq 1$, $\rho_{j+1} = T(\rho_j, \mathbf{a})$ for some joint action $\mathbf{a}$ such that for every $i \in \Gamma$, $\mathbf{a}[i] = \sigma_i(\rho_{\leq j})$, and for every $k \in \overline{\Gamma}$, $\mathbf{a}[k] \in P_k(\rho_j)$. We denote with $Out(s, \sigma_\Gamma)$ the set of all σ_Γ-compatible paths whose first state is s.

We now introduce Alternating-time Temporal Logic (Alur et al., 1997) (ATL, for short) to reason about strategic abilities of the agents. Formulas of ATL are defined by the grammar:

$$\varphi := \top \mid p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \langle\Gamma\rangle \mathsf{X} \varphi \mid \langle\Gamma\rangle(\varphi \,\mathsf{U}\, \varphi) \mid \langle\Gamma\rangle(\varphi \,\mathsf{R}\, \varphi)$$

where $p \in \mathsf{Ap}$ and $\Gamma \subseteq \mathcal{A}$. The boolen connective can be defined as usual as well as the temporal connectives G (globally), F (eventually) and (weak-until) W.

Definition 5. *Given an interpreted system I, a global state s of I, and an ATL formula φ, the satisfaction relation $I, s \models \varphi$ is defined by induction on the structure of φ as follows:*

- *$I, s \models \langle\Gamma\rangle \mathsf{X} \psi$ iff there is a Γ-strategy σ_Γ such that for every $\pi \in Out(s, \sigma_\Gamma)$ we have that $I, \pi_2 \models \psi$;*
- *$I, s \models \langle\Gamma\rangle \psi \,\mathsf{U}\, \theta$ iff there is a Γ-strategy σ_Γ such that for every $\pi \in Out(s, \sigma_\Gamma)$ there is a $j \geq 1$ such that $I, \tau_j \models \theta$ and $I, \tau_i \models \psi$ for each $1 \leq i < j$;*
- *$I, s \models \langle\Gamma\rangle \psi \,\mathsf{R}\, \theta$ iff there is a Γ-strategy σ_Γ such that for every $\pi \in Out(s, \sigma_\Gamma)$ either $I, \tau_i \models \theta$ for all $i \geq 1$ or there is a $j \geq 1$ such that $I, \tau_j \models \psi$ and $I, \tau_i \models \theta$ for each $1 \leq i \leq j$.*

where the satisfaction clauses for atoms and boolean connective are standard and thus omitted.

3 Formal Models

Now that we have laid out the preliminaries for our work, we can gather together the elements exposed so far and present our formalization of Attack Graphs.

3.1 Attack Graphs as Kripke Structures

The term attack graph was first introduced by Phillips and Swiler (Phillips and Swiler, 1998). The general idea is to represent the possible attack paths in a system by means of a directed labeled graph. This graph is generated based on a description of the system's architecture (including topology and component configurations), along with a list of existing vulnerabilities, the attacker's profile (including capabilities, knowledge of passwords, and privileges), and attack templates (which encompass the attacker's atomic actions, including preconditions and postconditions). Each attack path within the graph represents a sequence of atomic attacks. Numerous studies have adopted this approach, 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. Despite the existence of various, and quite different, formalizations of the Attack Graph notion, most authors agree that a formalization of this concept should adhere to the following two conditions: an Attack Graph is said to be **complete** whenever 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 labeled with *att*. An attack graph is said to be **monotone** whenever for every state q and q', if there is an atomic attack that leads from q to q' then the conditions true at q are also true at q'.

As we have said, Attack Graphs are generated from a series of heterogeneous pieces of information. The primary one among them is provided by the conditions that must be satisfied in a given situation for an attacker to launch an attack and the effects that such an attack generates. Such type of information can be expressed through atomic propositions that serve as labels for the nodes of the graph. Consequently, the required information about the preconditions and postconditions of an attack can be expressed with the aid of a ternary relation. In each tuple $\langle X, a, Y\rangle$ of this relation, X is a set of atomic propositions representing the necessary conditions for launching the attack, a is the action undertaken by the attacker (the attack itself), and Y is the set of postconditions generated by the attack a starting from X. We call this type of relation an "attack relation". It will serve as the foundation for our definition of an Attack Graph.

Definition 6 (Attack Graph)**.** *Suppose that* $\mathsf{win} \in \mathsf{Ap}$*, let Σ be a finite alphabet and* $\mathsf{Att} \subseteq 2^{\mathsf{Ap}} \times \Sigma \times (2^{\mathsf{Ap}} \setminus \emptyset)$ *be a ternary relation dubbed Attack Relation. An Attack Graph is a tuple $AG = \langle \mathfrak{M}, W_I, \mathsf{Att}\rangle$ where $\langle \mathfrak{M}, W_I\rangle$ is a rooted Kripke structure based on* Ap *and Σ such that:*

1. *There is a non-empty subset* T *of W such that* $\mathsf{win} \in \mathcal{L}(w)$ *for all* $w \in \mathsf{T}$.
2. *$\langle w, a, w'\rangle \in R$ if and only if $w \neq w'$,* $w \notin \mathsf{T}$ *and there is a tuple* $\langle X, a, Y\rangle \in \mathsf{Att}$ *such that $X \subseteq \mathcal{L}(w)$ and $\mathcal{L}(w') = \mathcal{L}(w) \cup Y$.*
3. *The relation R is functional (or deterministic): for every $a \in \Sigma$ and $w \in W$ there is at most one $w' \in W$ such that $\langle w, a, w'\rangle \in R$.*

As the reader can see, our definition implies that the structure of an Attack Graph is determined by the given attack relation. For instance, suppose that the relation Att is specified accordingly to Table 1. An Attack Graph based on this attack relation is showed in Figure 1.

As an immediate consequence of condition 2 in the definition of Attack Graph, we get the following, in which we express that our definition of an At-

Precondition	Label	Postcondition
webserver:root	att_1	server:root
webserver:root	att_2	password:1234
server:root	att_3	databaseA:root, win
server:root, password:1234	att_4	databaseB:root, win

Table 1: Table specifying the relation Att.

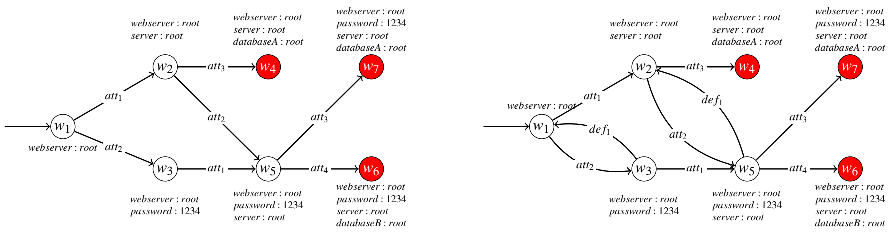


Figure 1: Attack Graph (left) and of Attack-Defense Graph (right). Red nodes are those in which win is true.

tack Graph implies the properties of monotonicity and completeness.

Proposition 1. *If $AG = \langle \mathfrak{M}, W_I, \mathsf{Att}\rangle$ is an Attack Graph, then the following holds; AG is **monotonous**: if $\langle w, a, w'\rangle \in R$ then $\mathcal{L}(w) \subset \mathcal{L}(w')$. AG is **complete**: if $\langle X, a, Y\rangle \in \mathsf{Att}$ and there is a world w of AG such that $X \subseteq \mathcal{L}(w)$, then there is a world w' of AG such that $\langle w, a, w'\rangle \in R$ and $Y \subseteq \mathcal{L}(w')$.*

3.2 The Countermeasure Relation

In the last subsection, we defined an Attack Graph as a special kind of Kripke structure in which the transition relation is determined by an Attack Relation and in which some other conditions are satisfied. The Attack Relation represents the possible attacks over the system, where an attack is identified with a set of preconditions and postconditions. Our goal is to model dynamic interaction scenarios between an attacker and defenders using an Attack Graph to represent the attack scenario. To achieve this, we will subsequently define interpreted system agents (attacker and defenders) and use ATL to specify their objectives. As an intermediate step towards defining these agents, we introduce the notion of an Attack-Defense graph. This graph has two types of edges: edges representing the attack relation and edges representing the defense relation. In this context, a defense is understood as follows: a defense consists of nullifying the effects of an attack, thereby restoring the system state and privileges obtained by the attacker to a state prior to the attack itself.

Let Ap be a set of atomic propositions, Σ a set of labels, and $\mathfrak{M}$ a Kripke structure based on Ap and Σ, if $\Delta \subseteq \Sigma$, the Δ-restriction $\mathfrak{M}_\Delta$ of $\mathfrak{M}$ is the tuple $\langle W_\Delta, R_\Delta, \Sigma_\Delta\rangle$ where $W_\Delta = \{w \in W \mid \exists w' \in W \wedge \exists a \in \Delta \wedge (\langle w, a, w'\rangle \in R_\Delta \vee \langle w', a, w\rangle \in R_\Delta)\}$ and $\mathcal{L}_\Delta(w) = \mathcal{L}(w)$ for every $w \in W_\Delta$. We remark that $\mathfrak{M}_\Delta$ is a Kripke structure whenever R_Δ is not-empty.

Definition 7. *Given Ap, Σ, an Attack Relation Att based on a subset Δ of Σ, and an indexed family $\{X_b\}_{b\in\overline{\Delta}}$ of pairwise disjoints subsets of 2^{Ap}, an Attack-Defense Graph is a tuple $ADG = \langle \mathfrak{M}, W_I, \mathsf{Att}, \{\mathsf{C}_b\}_{b\in\overline{\Delta}}\rangle$ where:*

1. *$\langle \mathfrak{M}_\Delta, W_I, \mathsf{Att}\rangle$ is an Attack Graph;*
2. *for every $b \in \overline{\Delta}$, $\mathsf{C}_b \subseteq W \times \{b\} \times W$ is a ternary relation such that $\langle w, b, w'\rangle \in \mathsf{C}_b$ if and only $X_b \subseteq \mathcal{L}(w)$, $\mathcal{L}(w') = \mathcal{L}(w) \setminus X_b$ and $w, w' \notin \mathsf{T}$;*
3. *the relation $\mathsf{C} = \bigcup_{b\in\overline{\Delta}} \mathsf{C}_b$ is functional: if $\langle w, b, w'\rangle \in \mathsf{C}$ and $\langle w, b', w''\rangle \in \mathsf{C}$ then $b = b'$ and $w' = w''$.*

In the above definition, the set of labels Σ is partitioned into two disjoint subsets, Δ and its complement; Δ represents the set of actions (attacks) of the attacker, while each element of b of $\overline{\Delta}$ represents a countermeasure for a defender. To each countermeasure b is associated a subset of atomic propositions X_b. This subset represents the effect of an attack that a defender can nullify by using the given countermeasure b. To make things more concrete, let us give an example. Suppose that $\Sigma = \{att_1, att_2, att_3, att_4, def_1\}$ that $\Delta = \Sigma \setminus \{def_1\}$ and that $X_{def_1} = \{password : 1234\}$. Suppose that Att is specified as in Table 1. An Attack-Defense graph is shown in Figure 1 (right).

3.3 Attack Graphs as Interpreted Systems

We can now finally define agents corresponding to the attacker and the various defenders of a system. In the following, we consider an Attack-Defense graph $ADG = \langle \mathfrak{M}, W_I, \mathsf{Att}, \{\mathsf{C}_b\}_{b\in\overline{\Delta}}\rangle$ based over an indexed family $\{X_b\}_{b\in\overline{\Delta}}$ of pairwise disjoints subsets of 2^{Ap}, and whose set of agents is $\mathsf{Ag} = \{att, d_{b_1}, \dots, d_{b_n}\}$ where the b_is are all only the elements $\overline{\Delta}$. We denote by ACT the set of tuples of length $n+1$ where, in each tuple $\mathbf{a}$, the first member $\mathbf{a}[1]$ is either $\star$ (the idle action) or an element of Δ and for each $1 < i \leq |\mathbf{a}|$, the i-th component $\mathbf{a}[i]$ is either $\star$ or b_i.

Definition 8 (Attacker agent)**.** *Given* Ag *as above, the attacker att is the agent* $\langle L_{att}, act_{att}, P_{att}, t_{att} \rangle$ *such that:*

- *the set of local state of the agent is equal to set of worlds W of the given ADG;*
- *the set of actions* act_{att} *of the agent is equal to* $\Delta \cup \{\star\}$*;*
- *the protocol function* P_{att} *assigns to each local state l the idle action* $\star$ *and each member a of* Δ *such that* $\langle l, a, l' \rangle \in R$*, where R is the transition relation of the Attack Graph;*
- *given a local state l an element* $\mathbf{a}$ *of ACT such that* $\mathbf{a} \in P_{att}(l)$ *the transition function is defined as follows:*

$$t_{att}(l, \mathbf{a}) = \begin{cases} l' & \text{if } \mathbf{a}[1] = a, \mathbf{a}[i] = \star \text{ for all } i \geq 2 \text{ and } \langle l, a, l' \rangle \in R \\ l'' & \text{if there is } i \geq 2 \text{ such that } \mathbf{a}[j] = b \text{ and } \langle l, b, l'' \rangle \in C_b \\ l & \text{otherwise} \end{cases}$$

Definition 9 (Defender agent)**.** *For* $i \geq 2$*, the i-Defender agent* d_{b_i} *is the agent* $\langle L_{d_{b_i}}, act_{d_{b_i}}, P_{d_{b_i}}, t_{d_{b_i}} \rangle$ *such that:*

- *the set of local states of the i-Defender is the set of equivalences classes of worlds of the AGD generated by the following relation:* $w \sim_{d_i} w'$ *if and only if* $\mathcal{L}(w) \cap X_{b_i} = \mathcal{L}(w') \cap X_{b_i}$*. Remark that this implies that each defender has exactly two local states: a state l such that* $\mathcal{L}(w) \cap X_{b_i} = \emptyset$ *for all* $w \in L$ *and a state* l' *such that* $\mathcal{L}(w') \cap X_{b_i} = X_{b_i}$ *for all* $w' \in l'$*. We refer to the former as the* ***empty state*** *and to the latter as the* ***full state****;*
- *the set of actions of the i-Defender is equal to* $\{\star, b_i\}$*;*
- *the protocol function assigns* $\star$ *to the empty state and* $\{\star, b_i\}$ *to the full state;*
- *Given a state l and a tuple of actions* $\mathbf{a} \in ACT$ *such that* $\mathbf{a}[i] \in P_{d_i}(l)$*, the transition function is defined as follows:*

$$t_{d_i}(l, \mathbf{a}) = \begin{cases} l' & \text{if } \mathbf{a}[i] = b_i \text{ and} \\ & \langle w, b_i, w' \rangle \in C_{b_i} \text{ for some } w \in l, w' \in l' \\ l' & \text{if } \mathbf{a}[1] = a \in \Delta,\ \mathbf{a}[i] = \star \text{ for all } i \geq 2 \text{ and} \\ & \langle w, a, w' \rangle \in R \text{ for some } w \in l \text{ and } w' \in l' \\ l & \text{otherwise} \end{cases} \tag{1}$$

Finally, an Attack-Defense Interpreted System *ADI* is an interpreted system in which the set of players is composed of an attacker and $n \geq 1$ defenders generated from the same Attack-Defense graph *ADG*, and the labeling function is equal to $\Pi(s) = \mathcal{L}(s[1])$[1] for every global state s. We will also say that the Interpreted System is **based** on the *ADG*.

Why Interpreted System Matters. As we have argued in the introduction, the risk analysis through Attack Graph is usually based on detecting critical paths within the system. In other words, paths that lead from one of the initial states of the graph to a target state. This type of analysis is easily simulated in our framework. Such a reachability goal can be expressed by the ATL formula $\langle att \rangle \mathsf{F}\,\mathsf{win}$ in an Attack-Defense interpreted system, generated from an attack-defense graph in which the countermeasure relation is the empty relation $\emptyset$. In our framework, however, we can specify much more complex objectives. For example, suppose Γ is a coalition formed by a group of defenders; we might be interested in knowing if the following specifications are satisfied in a dynamic attack-defense context:

O_1 *the attacker is never able to obtain root privilege on server s unless the defenders are able to obtain information on its identity*;

O_2 *while the defenders have not obtained information about the attacker identity, the attacker does not have root privilege on server s.*

Let a be an atomic proposition that expresses the fact that the identity of the attacker is known. Let r_s be an atomic proposition expressing the fact that the attacker has root privilege on server s. The two security objectives O_1 and O_2 presented above can be expressed by ATL formulae. Objective O_1 says that either we want the attacker to never reach a state satisfying r_s *or* if the attacker reaches such a state, then the defender wants to be able to identify it (a). We can express O_1 as $\varphi_1 := \langle \Gamma \rangle \mathsf{G}(\neg r_s \vee (r_s \rightarrow \langle \Gamma \rangle (\mathsf{F}\,a)))$. Objective O_2 says that we want r_s to be false *until* we have identified the attacker (a) if such identification ever occurs. Thus, we can write O_2 using the weak-until connective as: $\varphi_2 := \langle \Gamma \rangle (\neg r_s \,\mathsf{W}\, a)$.

4 Implementation

In this section, we present a tool called `AG2IS`[2] that represents the practical development of the previous definitions and concepts. We first provide a high-level description of the two tools `AG2IS` uses.

MulVal. `MulVal` is a tool with two objectives: to generate an Attack Graph that allows the automated inte-

[1] We remark that every first component of a global state s is a world of the ADG. Thus, $\mathcal{L}(s[1])$ is well defined.

[2] You can find the sources at the following link (Important: pay attention to the underscore in “view_only”, sometimes copy and paste does not include it): https://osf.io/5x6j2/?view_only=127a477518fc48e2bd5af9b89f937b37.

gration of formal vulnerability specifications, and to generate an Attack Graph that has the ability to model very large networks. To obtain these objectives, the creators of `MulVal` have chosen to use logic programming to model the elements of the analysis and Attack Graph generation. In this work, we do not discuss the theory behind the Attack Graph generation, as we use `MulVal` as a black-box tool. However, we will discuss the format of the network specification which `MulVal` employs to generate Attack Graphs and the format of the generated Attack Graph.[3]

MCMAS. `MCMAS` is a Model Checker for Multi-Agent Systems. The tool takes an interpreted system and verification formulae as input and employs Ordered Binary Decision Diagrams (Bryant, 1986) to return the truth value of the formulae. `MCMAS` allows definition of formulae in Alternating-time Temporal Logic (ATL). Therefore, `MCMAS` is able to handle both temporal and strategic properties. Here, we will employ ATL to verify the existence of attack paths.

4.1 Our tool AG2IS

The goal of `AG2IS` is to construct an interpreted system in the Interpreted Systems Programming Language (ISPL) format from a generated Attack Graph by `MulVAL`. The multi-agent systems that can be handled by `MCMAS` are described as interpreted systems in the ISPL. This language allows the definition of agents in terms of variables and evolution by using Boolean expressions. In particular, in ISPL we consider that there will be three different types of agents: the attacker, the defender[4], and the *environment*. This latter represents the interpreted system itself.

`AG2IS` takes three files as input: one file containing the Attack Graph generated by `MulVal` on the basis of a given attack relation. A second file that specifies what are the initial nodes and target nodes of the aforementioned Attack Graph; initial nodes will often be referred to as initial assumptions in the following. A third file containing the countermeasures for the defender which corresponds to the countermeasure relation introduced in the previous section.

Our tool then outputs one ISPL file containing the full interpreted system as well as the formula which needs to be verified by the model checker `MCMAS`.

The underlying algorithm of `AG2IS` works in three stages; first it constructs the necessary data structures from the input files, then it writes the interpreted system, and it finishes with constructing the evaluation and formula for verification. In what follows, we give a high-level exposition of each of these three stages. In the first phase, we construct the data structure we use to define the attacker's local model and the defender's local model. The attacker's local model is based upon the information contained in the `MulVal`-generated Attack Graph and the corresponding input of `MulVal`. The defender's local model is based on the information in the file containing the countermeasures for the defender, and relate them to nodes in the `MulVal` generated Attack Graph.

In the second phase, we specify the interpreted system based on the data structures obtained in the preceding phase. First, we construct the *Environment Agent*. Then we build the local model for the attacker using the data structures defined for the attacker. Then we construct the local model for the defender using the defender's data structure.

Finally, we construct the *Evaluation* of the interpreted system, which depends on the attacker's target, which is defined in the initial assumptions and its corresponding nodes in the `MulVal` generated Attack Graph. Lastly, we construct the formula that needs to be verified over the interpreted system, which is fully independent of the input files and therefore constant for any Attack Graph.

Example 1. *We give an example of input for `AG2IS` consisting of: (1) an attack (hyper)graph generated by `MulVal`; (2) a simplified[5] example of input file that specifies the initial node of the Attack Graph and the attacker's target state; (3) an example of a file corresponding to the specification of the defender's countermeasures.*

***MulVal Hypergraph**. In Figure 2 (left) we display a hypergraph corresponding to a `MulVal`-generated Attack Graph. On the right part , we display an Attack Graph generated from the one above that respects our definition of Attack Graph In the hypergraph, NODE 1 and NODE 3 correspond to the initial assumptions of the system, NODE 2 is a rule that the attacker can activate and NODE 4 is the postcondition of activating NODE 2. In the corresponding graph NODE A corresponds to the state containing both preconditions NODE 1 and NODE 2. NODE B contains both the postcondition NODE 4 and the preconditions of the transition. EDGE 1 describes the transition from NODE A to the node containing the postcondition NODE B.*

***Initial nodes and target node**. The initial nodes (or initial assumption) of the Attack Graph described in the preceding paragraph are NODE 1 and NODE 3 while the attacker's target state is NODE 4. This kind*

[3]Note that `MulVal` generates hypergraphs. However, this is resolved because it is manageable to generate a graph from a hypergraph (Bartolomeo et al., 2022).

[4]Note that, in the current implementation we only consider one defender.

[5]This file is not in the datalog format of a `MulVal` input file.

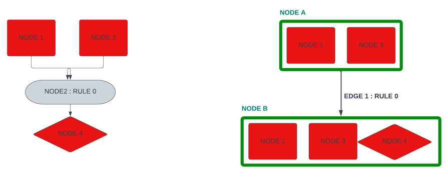
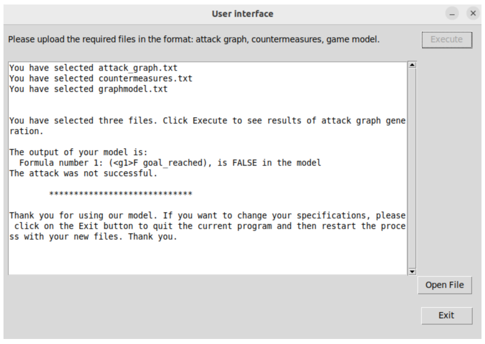


Figure 2: `MulVal`-generated Attack Graph (left) and corresponding Attack Graph respecting our definition (right).

of information can be easily specified in `MulVal` *as follows: attackGoal(node4), node1, node3.*

Countermeasures*. In our implementation, we have made the following choice: the defender can apply countermeasures by considering the actual state of the Attack Graph. Specifying the availability of a countermeasure for the defender is quite straightforward. Indeed, the file that specifies the countermeasure is composed of datalog commands with the following syntax:* $Countermeasure = Bool : preCondition(\ldots)$*. The interpretation of this command is straightforward. If the Boolean variable Bool is* True*, then the defender has a countermeasure against the vulnerability described with the predicate preCondition(...). If the Boolean variable Bool is* False*, then the defender does not have a countermeasure against the related condition. The format allows the user to not input conditions for any precondition of the transitions of the Attack Graph corresponding to the system. In fact, if an assumption is not contained in the countermeasure input, it is considered to be assigned with the Boolean variable Bool as* False.

ISPL Generation. After having constructed the necessary data structures, the algorithm begins constructing the interpreted system, in an ISPL format. It constructs three agents, an attacker, a defender, and an environment agent as well as the initial states for the system, as presented by the specification of `MCMAS` in the dedicated paragraph. The implementation has the following structure: the environment agent contains all the variables, i.e. all preconditions and postconditions, of the Attack Graph representing the modeled system, as well as the the evolution of these variables. The attacker and defender agents contains solely their own actions and the protocol of these actions, i.e. when these actions can be activated. Once we have obtained an interpreted system, we need to determine the evaluation and formula to be verified for our model. The evaluation corresponds with the atomic propositions of the system, which we define with win. Simply, this propositions is verified if the attacker can reach any of its objectives. Then, we define the formula to be verified by `MCMAS`. The formula is constructed in ATL as follows: $\langle g_1 \rangle \mathsf{F}\,\mathsf{win}$, where g_1 corresponds to the group which only contains the attacker. The formula reads as *"the attacker has a strategy to reach in the future one of his objectives"*.



Please upload the required files in the format: attack graph, countermeasures, game model.

```
You have selected attack_graph.txt
You have selected countermeasures.txt
You have selected graphmodel.txt


You have selected three files. Click Execute to see results of attack graph gene
ration.

The output of your model is:
  Formula number 1: (<g1>F goal_reached), is FALSE in the model
The attack was not successful.

        *****************************

Thank you for using our model. If you want to change your specifications, please
 click on the Exit button to quit the current program and then restart the proce
ss with your new files. Thank you.
```

Figure 3: The user interface.

User Interface (UI). Our tool `AG2IS` is provided with a user interface. In Figure 3, we can observe a sample view of the interface, which has been designed for a user-friendly interaction. Upon launching the interface, users are greeted with a message at the top, instructing them to upload the required files in a specific format: Attack Graph, countermeasures, and game model (containing the specification of the initial nodes and target states) using an "Open File" button. As each file is uploaded, the user receives confirmation through messages displayed on the screen. Once all three files are uploaded, the user is informed that they can proceed by clicking the "Execute" button in the upper right corner, which will generate the truth value for the attacker's objective written in ATL.

Behind the scenes, the UI utilizes the `MCMAS` tool with the provided command, displaying only the relevant output to the user. Additionally, users are informed that if they wish to modify the specifications, they can click the "Exit" button to restart the process.

5 Conclusion and Future Works

In this paper, we present a formal technique for transforming an Attack Graph, along with countermeasures, into a game between an attacker and a coalition of defenders using interpreted systems. We have proven the correctness of this transformation and subsequently developed a tool called `AG2IS` that establishes a connection between the output of an Attack Graph generator (namely, `MulVal`) and the input of a model checker for multi-agent systems (namely, `MCMAS`). The ultimate goal of this entire framework is to determine whether an attacker possesses a strategy to gain access to certain private resources, regardless of the countermeasures employed by the defenders.

As future work, we aim to explore various directions from both theoretical and practical perspectives. On the theoretical side, we intend to investigate scenarios where the game structure involves multiple attackers. This presents a challenging objective due to our current approach of transforming Attack Graphs into interpreted systems. To address multiple attackers, we must consider transforming multiple Attack Graphs into interpreted systems. Currently, merging several Attack Graphs has not yet been studied. Initially, we will delve into the amalgamation of multiple Attack Graphs and subsequently devise a formal framework for generating an interpreted system. On the practical side, there is much to accomplish. Primarily, we need to enhance our tool to handle multiple defenders. Additionally, we must expand the capabilities of `AG2IS` on the specification side. Our current implementation only defines a reachability objective for the attacker. Nonetheless, as demonstrated in our work, incorporating more intricate temporal objectives can enhance the verification process. Moreover, our approach in this study involved utilizing two existing tools for Attack Graph generation and verification. However, since `MulVal` is an outdated tool without active maintenance, we are considering developing a new Attack Graph generator. Furthermore, `MCMAS` lacks certain extensions in terms of logics for strategic reasoning. Thus, we have plans to extend it with more robust logics that are particularly relevant within the realm of cybersecurity.

Acknowledgement. The first author is supported by the PRIN project RIPER (No. 20203FFYLK).

REFERENCES

Alur, R., Henzinger, T. A., and Kupferman, O. (1997). Alternating-time temporal logic. In *FOCS97*, pages 100–109.

Ammann, P., Wijesekera, D., and Kaushik, S. (2002). Scalable, graph-based network vulnerability analysis. CCS '02, page 217–224. Association for Computing Machinery.

Bartolomeo, S. D., Pister, A., Buono, P., Dunne, C., and Fekete, J.-D. (2022). Six methods for transforming layered hypergraphs to apply layered graph layout algorithms. *EuroVis 2022*, 41(3).

Bryant, R. E. (1986). Graph-based algorithms for boolean function manipulation. *IEEE Trans. Computers*, 35(8):677–691.

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

Catta, D., Leneutre, J., and Malvone, V. (2023a). Attack graphs & subset sabotage games. *Intelligenza Artificiale*, 17(1):77–88.

Catta, D., Leneutre, J., and Malvone, V. (2023b). Obstruction logic: A strategic temporal logic to reason about dynamic game models. In *ECAI 2023*, pages 365–372.

Catta, D., Stasio, A. D., Leneutre, J., Malvone, V., and Murano, A. (2023c). A game theoretic approach to attack graphs. In *ICAART 2023*, pages 347–354.

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. Springer.

Durkota, K., Lisy, V., Bošansky, B., and Kiekintveld, C. (2015b). Optimal network security hardening using attack graph games. IJCAI'15, page 526–532. AAAI Press.

Fagin, R., Halpern, J., Moses, Y., and Vardi, M. (1995). *Reasoning about Knowledge.* MIT.

Ingols, K., Lippmann, R., and Piwowarski, K. (2006). Practical attack graph generation for network defense. In *ACSAC'06*, pages 121–130.

Jha, S., Sheyner, O., and Wing, J. M. (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.

Lomuscio, A. and Raimondi, F. (2006). MCMAS: A model checker for multi-agent systems. In *TACAS 2006*, pages 450–454.

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 '17, page 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.

Ou, X., Govindavajhala, S., and Appel, A. W. (2005). Mulval: A logic-based network security analyzer. In *USENIX Security '05'*, page 8.

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

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