Towards Synthesis of Attack Trees for Supporting Computer-Aided Risk Analysis

Sophie Pinchinat, Mathieu Acher, Didier Vojtisek

Towards Synthesis of Attack Trees for Supporting Computer-Aided Risk Analysis

Sophie Pinchinat, Mathieu Acher, and Didier Vojtisek

Inria, IRISA, University of Rennes 1, France

Abstract Attack trees are widely used in the fields of defense for the analysis of risks (or threats) against electronics systems, computer control systems or physical systems. Based on the analysis of attack trees, practitioners can define actions to engage in order to reduce or annihilate risks. A major barrier to support computer-aided risk analysis is that attack trees can become largely complex and thus hard to specify. This paper is a first step towards a methodology, formal foundations as well as automated techniques to synthesize attack trees from a high-level description of a system. Attacks are expressed as a succession of elementary actions and high-level actions can be used to abstract and organize attacks into exploitable attack trees. We describe our tooling support and identify open challenges for supporting the analysis of risks.

1 Introduction

Ensuring the security of an information system means guaranteeing data availability, integrity and confidentiality. In this perspective, a preliminary study of the system and its environment, called *risk analysis*, is necessary [4,13]. The discipline of risk analysis aims to identify and evaluate risks that threaten a given system. Current methods follow mostly the same outline: practitioners decompose the system into subsystems and produce a model, then draw up a list of feared events, and finally determine the potential reasons of the realization of these events. The NATO report [11] showed that the current methods are ill-suited to manage the security of complex systems. Formal methods, well-defined formalisms, and analysis tools have the potential to eliminate current barriers. In particular, *attack trees* are widely used in the fields of defense for the analysis of risks (also called threats) against electronics systems, computer control systems or physical systems [1, 8, 9, 12, 14, 15, 18]. Based on the analysis of attack trees, practitioners can define actions to engage in order to reduce or annihilate risks.

Up to now, the construction of attack trees is made by hand, based on knowledge and experiences of analysts and technicians. There are construction and editing tools of attack trees available (e.g., [1, 14, 15]). This manual effort is time-consuming and error-prone, especially as the size of attack trees can become substantial. Our goal is to create an automated process able to assist practitioners in fulfilling the modeling task. This paper reports on the first steps towards a methodology, the formal foundations as well as automated techniques to *synthesize attack trees* from a high-level description of a system. Though attack

trees have deserved a lot of attention [6–10, 12, 18], we are unaware of existing approaches for (semi-)automatically synthesizing attack trees. Hong *et al.* [3] have addressed synthesis of attack trees, but in a setting lacking the precious hierarchy of actions that enforces the structure of the attack trees. As a result, there is no control over the outcome: deeply flattened and hard to exploit trees can be obtained, precluding a further exploitation by risk analysts and domain experts. For computer system security, Jha *et al.* [5] and Sheyner *et al.* [16, 17] proposed an algorithm to automatically generate an attack graph representing the system under consideration. In our case, attack graphs are generated from a description of a system (e.g., a military building). Our methodology allows the user to specify the intended structure through a hierarchy of actions, with flexibility on the adequate level of abstraction to choose. Moreover we do not synthesize scenarios graphs but attack trees.

Our proposed methodology consists in (1) describing the system to protect as an *attack graph (AG)*; (2) extracting the *attacks* (the solution-paths going from an initial state to a final/goal state); (3) gathering all attacks into an *attack tree*. The exploitation of a high-level specification (e.g., a military building) raises the level of abstraction – practitioners (experts of a domain) can more easily express their knowledge and defense objectives. Moreover automating the approach ensures that the presence of all attacks under study are present in the attack tree. We notice, though, that a fully automated synthesis is likely to produce unexploitable (e.g., deeply flatten) trees. Mauw and Oostdijk [9] showed that numerous structurally different attacks trees can capture the same information, out of which a few are readable and meaningful for an expert. An original and crucial feature of our methodology is the support of *high-level actions* to specify how sequences of actions can be abstracted and structured – a high-level action can be seen as a sub-goal. Likewise experts can control the synthesis process and obtain attack trees close to what they have in mind. We formalize *hierarchies of actions* and use standard pattern-matching techniques to compute a so-called *strategies* of the attacks (corresponding to *attack suites* in [9]), that provide abstractions of the attack descriptions. The methodology comes with an environment and a set of languages for generating attacks, specifying high-level actions, and synthesizing attack trees. We illustrate the synthesis process in the context of analyzing risks of a military building. We also identify some open challenges for fully supporting the approach.

Remainder of the paper. Section 2 introduces a running example in the context of military defense. Section 3 describes background information and notations used throughout the paper. Section 4 defines attack graphs before introducing in Section 5 hierarchy of actions, strategies, and attack trees. Section 6 summarizes the paper and describes future works.

2 Motivation and Running Example

To illustrate the motivation of our work, let us consider the example of Figure 1. Military experts want to protect an armoury, i.e., they do not want that the

stored weapons fall into the wrong hands. For the sake of risk analysis, we put ourselves in the attacker's shoes and look for the ways to intrude the military building. The armoury consists of six rooms, including a hall monitored by a video equipment and the storage room. The building is guarded by defenders: the attacker (ATK) may meet an agent in one of the rooms, a dog in another. The goal of the analysis is then to find all the *relevant* paths to reach the weapons. Relevant means here that we would like to get the successful attacks that are realistic (no invisibility cloak) and without loops – we would like to avoid the cases where the attacker goes from a room (say A) to the next room (say B), goes back to A, then goes to B, and so on. An example of path,

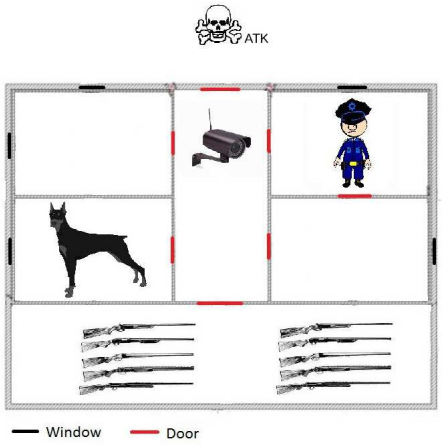


Figure 1. The plan of the armoury to protect

among many others, is that the attacker can cross the hall to go directly into the storage room; to make sure he or she has not been seen, the attacker can cut the video surveillance system. The building of Figure 1 involves 17 elements (seven doors, five windows, three agents, one camera and the arsenal). The number is substantial since numerous paths and attacks can be envisioned. In practice the complexity can be even higher. The goal of the project we are involved in is to assist military experts in synthesizing attack trees. Specifically we want to generate attacks from the description of a system (e.g., military building) and synthesize, with the help of some directives, a readable attack tree that military experts can exploit afterwards.

Running example. In the rest of the paper, we will only consider the end of the attack, that is, the intrusion in the storage room. In our running example (Figure 1), the system consists in a simple room and an attacker who still wants to intrude to steal the weapons located in a locked cabinet of this room. To describe the states of the system, we define the following three two-valued variables: $Pos(ATK) \in \{out., room\}$ ($out.$ stands for $outdoor$) which gives the position of the attacker in the system at each moment, $room \in \{opened, closed\}$ which describes the door status, and $cabinet \in \{opened, closed\}$ which tells whether the weapons are easy to reach or not.

3 Preliminary Notations

For a set X, we let X^* be the set of finite sequences over X, and we denote by ϵ the empty sequence. Given two sequences $s_1, s_2 \in X^*$, we let $s_1.s_2 \in X^*$ denote the concatenation of s_1 and s_2. For $x \in X$, we will simply write x for the set $\{x\} \subseteq X^*$. Given a sequence $s \in X^*$ and $Y \subseteq X^*$ a set of sequences over X^*, we let $s.Y$ denote the sequences s, s' where $s' \in Y$ and we $s^{-1}.Y$ be the set of sequences $s' \in X^*$ such that $s.s' \in Y$. For two subsets Y and Z of X^*, we let $Y.Z = \bigcup_{s \in Y} s.Z$; it is the set of sequences obtained by concatenating a sequence in Y with a sequence in Z. We now recall trees, labeled trees and forests.

A *tree* is a finite set $T \subseteq \mathbb{N}^*$ of *nodes* such that: $t.i \in T$ implies $t \in T$ (prefix-closeness), and $t.i \in T$ implies $t.j \in T$, for all $1 \leq j < i$ (left-closeness).
A node $t \in T$ is a *leaf* if $t.1 \notin T$; we write $leaves(T) \subseteq T$ the set of leaves of T. We let $\deg(t)$ be the greatest n such that $t.n \in T$. We write $children(t)$ for the sequence of ordered children of t in T, i.e., the sequence $t.1 \dots t.\deg(t) \in T^*$. A branch of T is a sequence $i_1 i_2 \dots i_m$ such that $i_1 i_2 \dots i_m \in leaves(T)$.

Let Γ be a set. A *Γ-labeled tree* is a structure $\tau = (T, \ell)$ where T is a tree, and $\ell : T \to \Gamma$ is the *labeling function*; Figures 3(a) and 3(b) depicts two labeled trees τ_1 and τ_2. We write $w(\tau) \in \Gamma^*$ for the sequence of labels of leaf nodes of τ ordered from left to right. Given a Γ-labeled tree $\tau = (T, \ell)$ and a node $t \in T$, we let $\tau_t = (T_t, \ell_t)$ be the sub-tree of τ rooted at node t defined by $T_t = t^{-1}T$ and $\ell_t(t') = \ell(t.t'), for\ all\ t' \in T_t$.
A *Γ-forest* is a finite ordered set $\{\tau_i\}_{i \in I}$ of Γ-labeled trees.

4 Attack Graphs

We use a standard symbolic representation for dynamic systems, where states are characterized by valuations over a finite set of variables (ranging over a finite domain) and transitions between states correspond to actions.

Figure 2 describes the system of our running example: each state, although numbered, is characterized by a valuation of the relevant variables. Initial states are marked by an ongoing arrow and goal states (those the attacker wants to reach) by double row. We can navigate in this graph following arrows that realize actions; label tc/fl means that either action tc or action fl can be chosen.

Definition 1. *An Attack Graph (AG) over a set of actions $\mathbb{A}$ is a structure $\mathcal{G} = (S, f, I, G)$, where S a finite set of* states*, $f : S \times \mathbb{A} \to S$ a partial* transition function*, $I \subseteq S$ is a set of* initial *states, and $G \subseteq S$ is a set of* goal *states.*

The AG $\mathcal{G}_{ex} = (S, f, I, G)$ of Figure 2 is formally defined by:

- $S = \{1, 2, 3, 4, 5, 6, 7, 8\}$: each state of the system is composed of a combination of the possibles of the variables.
- $I = \{1, 5\}$: the initial states are the states where $Pos(ATK) = out.$ and $room = closed.$
- $G = \{7, 8\}$: the final states are the states where attacker is inside and the cabinet opened.

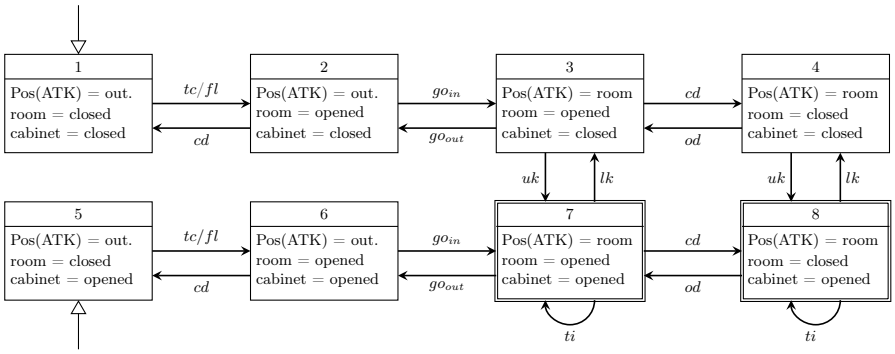


Figure 2. The AG $\mathcal{G}_{ex}$

– $\mathbb{A} = \{tc, fl, od, cd, go_{in}, go_{out}, uk, lk, ti\}$ is the set of primitive actions, which respectively means:

Action	Meaning
tc	type the code (door opener)
fl	force the lock
od	open the door
cd	close the door
go_{in}	go inside
go_{out}	go outside
uk	use a key to open the cabinet
lk	lock the cabinet with a key
ti	take item (weapon)

– $f : S \times \mathbb{A} \to S$ the partial *transition function*

f_{tc}	f_{fl}	f_{od}	f_{cd}	$f_{go_{in}}$	$f_{go_{out}}$	f_{uk}	f_{lk}	f_{ti}
$1 \mapsto 2$	$1 \mapsto 2$	$4 \mapsto 3$	$3 \mapsto 4$	$2 \mapsto 3$	$3 \mapsto 2$	$3 \mapsto 7$	$7 \mapsto 3$	$7 \mapsto 7$
$5 \mapsto 6$	$5 \mapsto 6$	$8 \mapsto 7$	$7 \mapsto 8$	$6 \mapsto 7$	$7 \mapsto 6$	$4 \mapsto 8$	$8 \mapsto 4$	$8 \mapsto 8$
			$2 \mapsto 1$					
			$6 \mapsto 5$					

For example, according to f_{tc}, from the state 1, if we apply the action tc, we get into the state 2. Similarly, from the state 5, we get into the state 6 after the application of the same action.

In the rest of this section let $\mathcal{G} = (S, I, G, f)$ be an AG over $\mathbb{A}$. We define *attacks* as sequences of actions from an initial state to a goal state. We first recall standard notion on labeled graphs.

A *path starting from* $s \in S$ *in* $\mathcal{G}$ is a sequence of states $\pi = s_0 s_1 \dots s_n$ such that $s_0 = s$ and $\exists a \in \mathbb{A}, f(s_i, a) = s_{i+1}$, for all $0 \leq i < n$. A path π *reaches the set* $S' \subseteq S$ if $s_n \in S'$. A path $\pi = s_0 \dots s_n$ is *elementary* if, for all $0 \leq i < j \leq n, s_i \neq s_j$.

Let $a_1 \dots . a_n \in \mathbb{A}^*$ be a sequence of actions and let $s_0 \in S$. A *path induced by* $a_1 \dots . a_n \in \mathbb{A}^*$ is a path $\pi = s_0 s_1 \dots s_n$ such that $f_{a_{i+1}}(s_i) = s_{i+1}$, for all $0 \leq i < n$.

Definition 2. *An attack in* $\mathcal{G}$ *is a sequence of actions* $a_1.a_2 \dots . a_n \in \mathbb{A}^*$ *such that there exists an elementary path from some initial state induced by* $a_1 \dots a_n$ *and which reaches the set* G*. Let* $Attack(\mathcal{G})$ *be the set of attacks in* $\mathcal{G}$*; it is finite.*

In $\mathcal{G}_{ex}$, $tc.go_{in}, fl.go_{in}.cd.uk.ti$ is an attack either along path $5, 6, 7$ or path $1, 2, 3, 4, 8, 8$. Also, $tc.go_{in}.cd.ti \in Attack(\mathcal{G}_{ex})$.

5 High-Level Actions and Attack Trees

We first define *hierarchies of actions* used to describe attacks in a more comprehensible way called *strategy*. Then strategies are gathered into an *attack tree*.

5.1 Hierarchy of actions

The paths extracted from the AG are low-level descriptions of attacks by means of elementary actions. However, sequences of actions may be abstracted as so-called *high-level actions*, explaining some behaviors in a more abstract manner. This abstraction relies on a hierarchy of actions, which may be updated along an analysis process.

Consider our running example. The set $\mathbb{A}$ of elementary actions describe the lowest level actions of level 0, written $\mathcal{H}_0$. "Higher level" actions can be considered for example if one wishes to introduce action *or* for "open room" which may be achieved, or *refined*, by performing either (elementary) actions *tc* ("type the code"), or fl ("force the lock"), or *od* ("open the door"). In the same line, one can define the higher-level action *cr* ("close room") uniquely refined as action *cd* ("close the door"). Since higher-level actions *or* and *cr* can be realized by actions of level 0, they would belong to level 1, that is the set $\mathcal{H}_1$. To $\mathcal{H}_1$, we add action *oc* ("open cabinet") refined by performing *uk*, the action *cc* ("close cabinet") refined by performing *lk* ("lock the cabinet with key"), and *tw* ("take weapons") refined either by *ti*, or *cd.ti*, or *od.ti*, or *ti.tw*.

Based on a (somewhat arbitrary) choice of abstraction given by an expert, we get a hierarchy of actions which consists in a set $\mathcal{H}_0 = \mathbb{A}$ of elementary actions, and a set $\mathcal{H}_1$. The latter is formed of actions whose realizations are sequences of actions of level 0. More generally, we should describe a high-level actions of *level* k by at least a sequence of level $k-1$, i.e., a sequence containing at least an action of level $k-1$ (no action of level greater than $k-1$). We now formalize the notion of hierarchy of high-level actions.

Definition 3. *A hierarchy over a set of actions* $\mathbb{A}$ *is a structure* $\mathbb{H} = (\{\mathcal{H}_k\}_{0 \leq k \leq K}, \mathcal{R})$ *where:*

- *each* $\mathcal{H}_k$ *(*$0 < k \leq K$*) is a finite set of* high-level actions (HLA)*, with* $\mathcal{H}_K \neq \emptyset$ *which forms the* top-level *actions.* $K \in \mathbb{N}$ *is the hierarchy* level *and* $\mathcal{H}_0 = \mathbb{A}$ *is the set of* primitive *actions. We let* $\mathcal{H} = \bigcup_{0 \leq k \leq K} \mathcal{H}_k$*, whose typical elements are* $A, B, A', A_1, A_2, \ldots$.
- $\mathcal{R} \subseteq \mathcal{H} \times \mathcal{H}^*$ *is the set of the* refinement rules *which satisfies* $level(\mathcal{R}(A)) \leq level(A)$*, for all action* $A \in \mathcal{H}$*. In particular,* $\mathcal{R}(A) = A$ *for all* $A \in \mathcal{H}_0$.

We define $level : \mathcal{H} \to \{0, \ldots, K\}$*, the* level function*, by* $level^{-1}(k) = \mathcal{H}_k$[1]*. A hierarchy is* strict *if for all action* $A \in \mathcal{H}$*,* $level(\mathcal{R}(A)) < level(A)$.

[1] We may sometimes display *level* in the structure.

As explained above, we have equipped the AG $\mathcal{G}_{ex}$ with a 2-level hierarchy $\mathbb{H}_{ex}$ over $\mathbb{A}$ defined by $\mathcal{H}_0 = \{tc, fl, od, cd, go_{in}, go_{out}, uk, lk, ti\}$, $\mathcal{H}_1 = \{or, cr, oc, cc, tw\}$ and $\mathcal{H}_2 = \{er, gr, st\}$ and the following refinement rules, where we write $A \rightsquigarrow A_1 \dots A_n$ instead of $(A, A_1 \dots A_n) \in \mathcal{R}$, and even use "|" on the right-hand side of $\rightsquigarrow$ to mimic standard notations in formal grammar rules; for instance, expression $or \rightsquigarrow tc \,|\, fl \,|\, od$ means that $\mathcal{R}(or) = \{tc, fl, od\}$.

$$\begin{cases} er \rightsquigarrow go_{in} \,|\, or.go_{in} \,|\, go_{in}.cr \,|\, or.go_{in}.cr \\ gr \rightsquigarrow go_{out} \,|\, or.go_{out} \,|\, go_{out}.cr \,|\, or.go_{out}.cr \\ st \rightsquigarrow tw \,|\, oc.tw \,|\, tw.lk \,|\, oc.tw.lk \end{cases} \qquad \begin{cases} or \rightsquigarrow tc \,|\, fl \,|\, od \\ cr \rightsquigarrow cd \\ oc \rightsquigarrow uk \\ cc \rightsquigarrow lk \\ tw \rightsquigarrow ti \,|\, cd.ti \,|\, od.ti \,|\, ti.tw \end{cases}$$

with HLAs of level 2 *er* for "enter room", *gr* for "get out of room" and *st* for "steal weapon", and we recall HLAs of level 1: *or* for "open the room", *cr* for "close room", *oc* for "open cabinet", *cc* "close cabinet", and *tw* for "take weapons".

5.2 Strategies

Strategies provide high-level descriptions of attacks in the AG. These objects are (forests of) labelled trees whose sequences of leaves describe attacks, while internal nodes of trees are labelled by HLA.

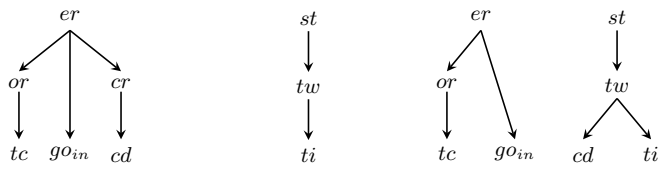
In the rest of this section, we let $\mathcal{G} = (S, I, G, f)$ be an AG over a set of actions $\mathcal{H}_0$, $\mathbb{H} = (\{\mathcal{H}_k\}_{0 \leq k \leq K}, level, \mathcal{R})$ be a hierarchy.

Definition 4. *A* strategy *over $\mathbb{H}$ is an $\mathcal{H}$-forest $\sigma = \{\tau_i\}_{i=1\dots n}$ such that for each $\tau_i = (T_i, \ell_i)$, and for every $t \in T_i$, $\ell_i(t) \rightsquigarrow \ell_i(t_1) \dots \ell_i(t_j)$ is a refinement rule, where $t_1, \dots, t_j$ are the children of t in T_i. Given a strategy σ, we will write $w(\sigma) = w(\tau_1)w(\tau_2)\dots w(\tau_n) \in \mathcal{H}_0^*$ for the* attack *of σ.*
A strategy σ is winning *whenever $w(\sigma) \in Attack(\mathcal{G})$.*

The set $\{\tau_1, \tau_2\}$ (see Figures 3(a) and 3(b)) represents a winning $\mathbb{H}_{ex}$-strategy σ_1 with two trees, whose attack is $tc.go_{in}.cd.ti$. In essence, branching nodes of a strategy have a conjunctive meaning[2].
It should be noted that there may be several strategies associated to an attack. For instance, strategy σ_1 formed by Figures 3(a) and 3(b), and strategy σ_2 of Figure 3(c) have respectively the same attack $tc.go_{in}.cd.ti$. This phenomenon is typical of a hierarchy of actions that is "ambiguous"[3].

For $w \in Attack(\mathcal{G})$, we let $\Sigma(w) := \{\sigma \,|\, w(\sigma) = w\}$ be the set of strategies *associated with* w.
Now that we know how to relate attacks in an AG with strategies that exploit the hierarchy of actions, we can gather strategies into a kind of "and-or" trees called an *attack tree*, in the same line as [6].

[2] And even a "sequential" one, i.e. children of the and-node are considered in order from left to right.
[3] As for context-free grammars.



(a) Tree τ_1 (b) Tree τ_2 (c) Strategy σ_2

Figure 3. Trees τ_1 3(a) and τ_2 3(b) forming strategy $\sigma_1 = \{\tau_1, \tau_2\}$ such that with $w(\sigma_1) = ti.go_{in}.cd.ti \in \mathit{Attack}(\mathcal{G}_{ex})$. 3(c) is another strategy σ_2 for $ti.go_{in}.cd.ti$.

5.3 Attack Trees

Strategies σ_1 and σ_2 of Figure 3, although both abstracting attack $tc.go_{in}.cd.ti$ in $\mathcal{G}_{ex}$, can be distinguished: the former relies on refinement rules $er \rightsquigarrow or.go_{in}.cr$, $cr \rightsquigarrow cd$ and $tw \rightsquigarrow ti$, whereas the latter relies on rules $er \rightsquigarrow or.go_{in}$ and $tw \rightsquigarrow cd.ti$. These alternatives can be expressed using trees including some nodes carrying an "or" semantics.

In the rest of this section, we let $\mathbb{H} = (\{\mathcal{H}_k\}_{0 \leq k \leq K}, \mathit{level}, \mathcal{R})$ be a hierarchy. We consider the signature formed of *and connectors* defined by $\mathcal{C} = \{\textcircled{\wedge}_j | j \in \mathbb{N}\}$, where $\textcircled{\wedge}_j$ has *arity* j (see Definition 5).

Attack tree nodes can be labeled by HLAs or by connectors or by the special symbol *win* which actually characterizes the root; label *win* is somehow a "super" high-level action representing the main goal.

Definition 5. *An Attack Tree (AT) over $\mathbb{H}$ is a $(\mathcal{H} \cup \mathcal{C} \cup \{win\})$-labeled tree $\mathcal{T} = (T, \ell)$ such that for all $t \in T$, if $\ell(t) \in \mathcal{C}_k$, then t has k children, and $t \in \mathit{leaves}(T)$ iff $\ell(t) \in \mathcal{H}_0$.* Virtual *(resp.* true*) nodes of $\mathcal{T}$ are those nodes labeled over $\mathcal{C} \cup \{win\}$ (resp. $\mathcal{H}$).*

Definition 6. *$\mathcal{T} = (T, \ell)$ over $\mathbb{H}$ is* well-formed *if for any branch $n_1 n_2 \dots n_m$ of T, and for any $1 \leq i < j \leq m$, letting $L = \ell(n_1 n_2 \dots n_i)$ and $L' = \ell(n_1 n_2 \dots n_j)$, if $L, L' \in \mathcal{H}$, then $\mathit{level}(L) > \mathit{level}(L')$.*

In *well-formed* ATs, the level of actions along a branch strictly decreases; Figure 4 depicts a well-formed AT over $\mathbb{H}_{ex}$ [4].

From strategies to ATs. We can embed strategies into ATs. This embedding consists in (i) explicitly connecting nodes of the trees to the strategy forest, and in (ii) connecting the roots of all the resulting trees via a "*win*-then-$\textcircled{\wedge}$" mechanism.

We first start explaining (i). Let σ be a strategy over $\mathbb{H}$, and let $\tau = (T, \ell) \in \sigma$. We transform τ into the $(\mathcal{H} \cup \{\textcircled{\wedge}_j | j \in \mathbb{N}\})$-labeled tree $\widehat{\tau} = (\widehat{T}, \widehat{\ell})$ defined by induction over the height h of τ as follows.

[4] In figures, we omit the arity of connectors.

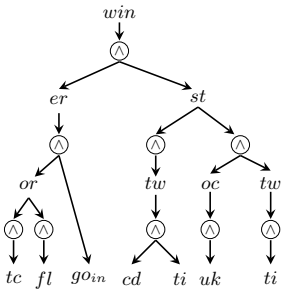
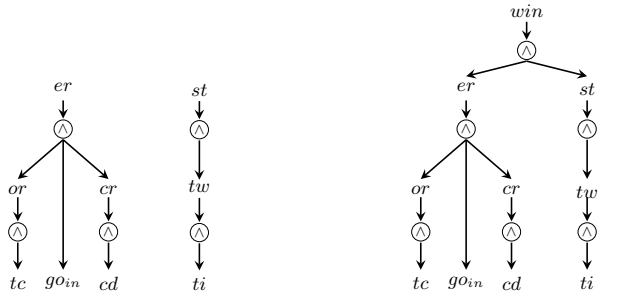


Figure 4. Attack tree $\mathcal{T}_0$

- If $h = 0$, then $\widehat{\tau} = \tau$.
- Otherwise, let $\tau_1 = (T_1, \ell_1), \ldots, \tau_n = (T_n, \ell_n)$ be the ordered sub-trees at τ root (hence with lower height). Then
  - $\widehat{T}$ is the least set containing all the sets $\{1\}$, $1.1.\widehat{T_1}$,...,$1.n.\widehat{T_n}$, and
  - $\widehat{\ell}(\epsilon) = \ell(\epsilon)$, $\widehat{\ell}(1) = \textcircled{\wedge}$ and $\widehat{\ell}(1.i.t) = \widehat{\ell_i}(t)$.

For example, consider the trees τ_1 of Figure 3(a) and τ_2 of Figure 3(b); the trees $\widehat{\tau_1}$ and $\widehat{\tau_2}$ are depicted in Figure 5(a).



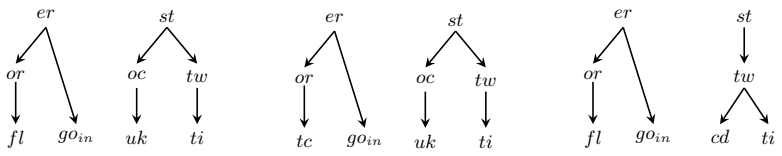
(a) Trees $\widehat{\tau_1}$ and $\widehat{\tau_2}$ (b) The attack tree $\mathcal{T}_{\sigma_1}$

Figure 5. Trees τ_1 of Figure 3(a) and τ_2 of Figure 3(b) transformed before connection and the resulting attack tree $\mathcal{T}_{\sigma_1}$

We can now embed strategies into ATs.

Definition 7. *Let $\sigma = \{(T_i, \ell_i)\}_{i=1,\ldots,n}$ be a strategy over some hierarchy $\mathbb{H}$. The* canonical AT associated to σ *is $\mathcal{T}_\sigma = (T, \ell)$ over $\mathbb{H}$ defined by:*

- *T is the least set containing all the sets $\{1\}, 1.1.\widehat{T_1}$, ..., $1.n.\widehat{T_n}$ (a node of T is then either ϵ or 1 or of the form $1.i.y$ where $y \in \widehat{T_i}$).*
- $\ell(x) = \begin{cases} win & \text{if } x = \epsilon \\ \textcircled{\wedge} & \text{if } x = 1 \\ \widehat{\ell_i}(y) & \text{if } x = 1.i.y \end{cases}$



(a) Strategy σ_3 (b) Strategy σ_4 (c) Strategy σ_5

Figure 6. Strategies included in $\mathcal{T}_0$ of Figure 4

Clearly, $\mathcal{T}_\sigma$ respects Definition 5, hence it is an AT. Note also that if $\mathbb{H}$ is strict, then $\mathcal{T}_\sigma$ is well-formed. Figure 5(b) depicts $\mathcal{T}_{\sigma_1}$ for strategy σ_1 of Figure 3.

From ATs to strategies. A given AT $\mathcal{T}$ naturally denotes a set $\Sigma(\mathcal{T})$ of strategies. Each strategy of $\Sigma(\mathcal{T})$ is obtained by keeping all nodes of $\mathcal{T}$, but by selecting a single child in each $\mathcal{H}$-labeled node of $\mathcal{T}$. Due to lack of space we do not formally define $\Sigma(\mathcal{T})$. Instead, we provide the clarifying following example.

The AT $\mathcal{T}_0$ of Figure 4 is such that $\Sigma(\mathcal{T}_0) = \{\sigma_2, \sigma_3, \sigma_4, \sigma_5\}$ (where the strategies are given in Figure 3(c), Figure 6(a), Figure 6(b) and Figure 6(c) respectively), because in $\mathcal{T}_0$ there are two nodes (with HLAs *or* and *st*), with two possible refinements each, hence the four outcomes for $\Sigma(\mathcal{T}_0)$.

Attack tree synthesis. We equip ATs with a binary operation $\oplus$ of *merging* with the following guaranteed properties.

- $\oplus$ is associative: $(\mathcal{T}_1 \oplus \mathcal{T}_2) \oplus \mathcal{T}_3 = \mathcal{T}_1 \oplus (\mathcal{T}_2 \oplus \mathcal{T}_3)$, for any ATs $\mathcal{T}_1$, $\mathcal{T}_2$ and $\mathcal{T}_3$.
- $\oplus$ is well-formedness preserving.
- For any two ATs $\mathcal{T}_1$ and $\mathcal{T}_2$,

$$\Sigma(\mathcal{T}_1 \oplus \mathcal{T}_2) \supseteq \Sigma(\mathcal{T}_1) \cup \Sigma(\mathcal{T}_2) \tag{1}$$

It should be noted that Equation (1) is not an equality. This is illustrated by the AT $\mathcal{T}_{\sigma_2} \oplus \mathcal{T}_{\sigma_3}$ (for strategies σ_2 and σ_3 of Figures 3(c) and 6(a)), which turns out to correspond to $\mathcal{T}_0$ in Figure 4, and yet $\Sigma(\mathcal{T}_0) = \{\sigma_2, \sigma_3, \sigma_4, \sigma_5\}$. Actually, the fact that Equation (1) is an inclusion instead of an equality is harmless since we claim that the extra strategies are in general also winning – indeed, returning to the matter at hand, HLAs would essentially denote sub-goals. Thus, if the main goal decomposes into HLAs e.g., A then B, and if now A can be refined into either A_1 or A_2 and, similarly, if B can be refined into either B_1 or B_2, the main goal cannot be achieved following any of the four combinations $A_1.A_2$, $A_1.B_2$, $B_1.A_2$ and B_1, B_2.

Operation $\oplus$ is central as it provides the last step for synthesizing an AT: from a finite set Σ of strategies, the synthesis is the result of computing

$$\mathcal{T}(\Sigma) = \oplus_{\sigma \in \Sigma} \mathcal{T}_\sigma$$

How this operation can be implemented in a scalable manner is left as future work (see next section).

6 Towards Attack Tree Synthesis

We proposed a mathematical setting to develop procedures that synthesize attack trees. An original and important aspect of our work is that an expert can participate in the synthesis process through the specification of so-called high-level actions (HLAs). We also sketched an end-to-end tool-supported methodology: starting from a high-level description of a system (e.g., a military building), practitioners can generate attacks through model-checking techniques, and eventually exploit HLAs to synthesize readable and well-structured attack trees. We are implementing ATSyRA[5], a tool built on top of Eclipse and upon the mathematical foundations, to support the synthesis methodology with a set of dedicated languages. Our work opens avenues for further research and engineering effort for supporting the analysis of realistic military risks by practitioners.

Interactive Support. The process is likely to be interactive and incremental: experts obtain an attack tree, add some information, re-synthesize, and so on. At different steps of the methodology, there are opportunities to parametrize the synthesis. For instance, experts can typically fine tune the generation of attacks. Currently, the building specification is transformed into GAL [2] (for Guarded Action Language) a simple yet expressive formalism to model concurrent systems. GAL is supported by efficient decision diagrams for model-checking. We use GAL support to generate attacks. Some predicates can be added to *scope* the space in which model checking mechanisms operate over. We can envision the use of specific languages, independent of GAL and at a higher level of abstraction, for easing the generation of attacks considered as relevant. Besides, experts can guide the synthesis of attack trees through HLAs. Re-structuring the hierarchy or abstracting a set of attacks can arise if the resulting attack tree is not satisfactory (e.g., in case the tree is too flat and hard to understand). Some visualisations and suggestions can help an expert for this task.

Scalability. The number of attacks can be huge, especially if the building specification contains numerous elements, leading to numerous possible paths for an attacker. The *scalability* problem impacts two aspects of the methodology. First, the synthesis of attack trees: HLA can help to reduce the complexity, since a hierarchy is pre-defined, guiding a canonical form they should conform. Yet, implementing the merging operator $\oplus$ of attack trees faces the very complex combinatorics induced by pattern-matching-like issues. Heuristics are likely to be needed both for computing the merge of attack trees and for scaling up. Second, we need to only generate relevant attacks. Again, experts can directly tune GAL to scope the generation: some facilities are needed to ease the task.

Attack Defense. Another step in our work consists in the introduction of the defender. We aim to study his or her actions and reactions. We plan to consider game theory and multi-agents systems, keeping close to the clean foundations

[5] More information about ATSyRA is available online: http://tinyurl.com/ATSyRA

proposed by [7].

Acknowledgements. This work is funded by the Direction Générale de l'Armement (DGA) - Ministère de la Défense, France. We thank Salomé Coavoux and Maël Guilleme for their insightful comments and development around ATSyRA.

References

1. AttackTree+. http://www.isograph.com/software/attacktree/.
2. M. Colange, S. Baarir, F. Kordon, and Y. Thierry-Mieg. Towards distributed software model-checking using decision diagrams. In N. Sharygina and H. Veith, editors, *CAV*, volume 8044 of *Lecture Notes in Computer Science*, pages 830–845. Springer, 2013.
3. J. B. Hong, D. S. Kim, and T. Takaoka. Scalable attack representation model using logic reduction techniques. *12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, pages 404–411, 2013.
4. ISO, Geneva, Switzerland. *Norm ISO/IEC 27002 - Information Technology - Security Techniques - Code of Practice for Information Security Management*, ISO/IEC 27002:2005 edition, 2005. Section 9.
5. S. Jha, O. Sheyner, and J. Wing. Two Formal Analyses of Attack Graphs. In *Proceedings of the 15th Computer Security Foundation Workshop*, pages 49–63, 2002.
6. B. Kordy, S. Mauw, S. Radomirović, and P. Schweitzer. Foundations of attack–defense trees. In *Formal Aspects of Security and Trust*, pages 80–95. Springer, 2011.
7. B. Kordy, S. Mauw, S. Radomirović, and P. Schweitzer. Attack–defense trees. *Journal of Logic and Computation*, 24(1):55–87, 2014.
8. B. Kordy, L. Piètre-Cambacédès, and P. Schweitzer. Dag-based attack and defense modeling: Don't miss the forest for the attack trees. *arXiv preprint arXiv:1303.7397*, 2013.
9. S. Mauw and M. Oostdijk. Foundations of Attack Trees. In *International Conference on Information Security and Cryptology - ICISC 2005. LNCS 3935*, pages 186–198. Springer, 2005.
10. V. Mehta, C. Bartzis, H. Zhu, E. Clarke, and J. Wing. Ranking Attack Graphs. In *Proceedings of Recent Advances in Intrusion Detection*, 2006.
11. N. Research and T. O. (RTO). Improving Common Security Risk Analysis. Technical Report AC/323(ISP-049)TP/193, North Atlantic Treaty Organisation, University of California, Berkeley, 2008.
12. B. Schneier. Attack Trees : Modeling Security Threats. *Dr. Dobb's Journal*, 1999.
13. E. E. Schultz. Risks due to the Convergence of Physical Security and Information Technology Environments. *Inf. Secur. Tech. Rep.*, 12:80–84, 2007.
14. Seamonster. http://sourceforge.net/apps/mediawiki/seamonster/.
15. SecurITree. http://www.amenaza.com/.
16. O. Sheyner, J. Haines, S. Jha, R. Lippman, and J. Wing. Automated Generation and Analysis of Attack Graphs. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, pages 273–. IEEE Computer Society, 2002.
17. O. Sheyner and J. Wing. Tools for Generating and Analyzing Attack Graphs. In *In Proceedings of Formal Methods for Components and Objects, Lecture Notes in Computer Science*, pages 344–371, 2004.
18. O. M. Sheyner. *Scenario Graphs and Attack Graphs*. PhD thesis, 2004.