2019 IEEE 32nd Computer Security Foundations Symposium (CSF)

On the Meaning and Purpose of Attack Trees

Heiko Mantel
TU Darmstadt
mantel@cs.tu-darmstadt.de

Christian W. Probst
Unitec Institute of Technology
cprobst@unitec.ac.nz

***Abstract*—Attack trees are a popular notation for describing threats to systems, both in academia and industry. Originally, attack trees lacked a formal semantics, but formal semantics for different variants of attack trees were proposed later. These semantics focus on the attacker's actions defined in the leaves and the logical structure defined by the inner nodes of an attack tree. Surprisingly, they do not clarify the connection to the goal defined at the root node in a satisfactory fashion. In this article, we aim at a better clarification of this connection between the attacks and the attacker goal specified by an attack tree. We argue that there are multiple sensible success criteria for attacks wrt. a given attacker goal and develop a framework for defining such criteria. We exploit our framework to identify similarities and differences between automatic attack-tree generation techniques. Finally, we propose a novel variant of attack trees that allows one to express exploits in an explicit fashion.**

***Index Terms*—Attack trees, threat modeling, security engineering.**

I. Introduction

Attack trees are a pragmatic notation for describing threats to systems and were originally proposed by Schneier [1]. In his definition, the root node of an attack tree specifies the attacker's goal, and each inner node specifies a subgoal of the attacker that contributes to the overall goal. The leaf nodes of an attack tree specify primitive attacker actions. The root and each inner node, in addition, specify whether its subtrees are conjunctively or disjunctively connected. A conjunction expresses that all sub-trees of a node must be processed, while a disjunction expresses that the sub-trees constitute alternatives.

In [1], an attack tree is used to represent one or more attacks, each consisting of one or more attacker actions and each aiming at the attacker's goal specified by the root node. Schneier outlines how the nodes of attack trees can be annotated in order to analyze, for instance, which attacks can be carried out by an attacker with a given skill set, what is the success probability of the specified attacks, what are the costs of carrying out the attacks, and what is the effectiveness of countermeasures.

Albeit we mostly focus on attack trees as a notation for capturing threats in this article, it should be noted that Schneier describes them as part of a methodology for thinking about security. He not only outlines a method for constructing attack trees, but also discusses the process of improving them over time and their use in making security engineering decisions.

Attack trees have gained high popularity and enjoy a widespread use in industrial practice. Their practical impact certainly motivated an intense attention by the research community. Research efforts resulted in novel variants of attack trees that, for instance, feature sequential conjunction as operator for connecting subtrees [2], support the explicit modeling of defenses [3], or compose goals of pre- and post-conditions [4]. There are approaches and tools for generating attack trees automatically (e.g., [5, 6]) and also for supporting security analysts in their manual construction of attack trees [7].

Schneier did not provide a formal semantics for his notion of attack trees in [1], and the resulting ambiguity of their meaning caused criticism by the research community. However, to date, such criticism is unjustified. Multiple research articles were dedicated to clarifying and formalizing the semantics of the original attack trees and of the later proposed variants.

Mauw and Oostdijk define a formal semantics of attack trees [8] using so called attack suites, which are sets of multi-sets of primitive attacker actions (or attack components in their terminology). This formalization nicely reflects the intuition given in [1]. Mauw and Oostdijk used their formal semantics for precisely defining conditions that a transformation on attack trees needs to fulfill in order to be semantics preserving. They also identified conditions guaranteeing an analysis of attribute annotations to deliver results that are robust under semantics-preserving transformations of attack trees. However, their semantics does not establish a relationship to the attacker's goal, and the same observation holds for subsequent action-centered semantics for variants of attack trees, including [2, 9, 10].

In contrast, techniques for the automatic generation of attack trees from system models, such as [6, 11], inherently need a criterion for when an attack against a system is successful. They use the system model and a goal specification to identify successful attacks and then construct an attack tree from these attacks. Unfortunately, automatic attack-tree generation techniques are not yet good at generating declaratively specified subgoals. Usually, there is little conceptual gap between subgoals and actions in an automatically generated attack tree.

In a different direction, variants of attack trees were recently proposed in which all nodes, including leaf nodes, are annotated with goal specifications [4, 7]. This results in elegant frameworks for reasoning declaratively about threats, but again, the overall attacker goal and the annotations of the leaf nodes are formulated at a similar level of abstraction, in this case, because there is no mentioning of attacker actions.

In this article, we study how to relate attacks and the attacker goal specified by an attack tree to each other. Intuitively, whether an attack against a system is successful depends on

DOI 10.1109/CSF.2019.00020

the actions of the attacked system, the actions of the attacker, possibly the actions of other actors, and the interplay between these entities. We identified the following three degrees of freedom in the definition of a success criterion for attacks:

Purity May occurrences of actions of an attack be interleaved with occurrences of other actions and, if yes, which ones?

Persistence Is it sufficient if the attacker's goal is satisfied at some point in time or should it be satisfied persistently?

Causality How much certainty does one desire that the satisfaction of the goal, indeed, results from the attack?

We define a framework for defining criteria for the success of an attack wrt. an attacker goal. Our framework provides multiple options for each of the three degrees of freedom above. Our choice of concrete options aims at illustrating sensible possibilities in the design space, without striving for completeness. Further options could be added later.

We illustrate how our framework for defining success criteria can be used to clarify which success criteria are used, e.g., in automatic attack-tree generation techniques such as [5, 6]. For this purpose, we formulate the success criteria underlying these and other publications in terms of Purity, Persistence, and Causality. The results of this application of our framework are interesting in their own right. We identify similarities and differences in the success criteria generation. In some cases, we argue that the choice of success criteria are at least debatable, for instance, because they are too liberal.

Finally, we propose *exploit tree* as a new term, to enable a clear terminological distinction to attack trees. In our opinion, attack trees should model threats solely from the perspective of an attacker. Similarly, attack-defense trees should focus on two perspectives, the one of the attacker and the one of the defender. Neither attack trees nor attack-defense trees should be cluttered with information about system-internal actions or actions of other actors than the attacker and the defender. Such information could be captured, for instance, by a behavioral model of the system instead. Nevertheless, there apparently is some desire to use trees that carry actions by other actors as annotations. To avoid confusion with attack trees, we suggest the term exploit tree for such trees. From a formal perspective, exploit trees provide no innovation as the definition of their syntax and semantics resembles the one of attack trees. The only difference to attack trees is that the leaves of exploit trees may be annotated with other actions than attacker actions. We argue that the difference is relevant in threat modeling from a methodological perspective. Exploit trees can also be used to clarify better the interplay between attacker actions, actions of the system under attack, and actions of other actors.[1]

As a running example, we use a scenario where an attacker attacks an ATM with the goal to steal money.

[1]Note that this interplay should be relevant for threat modeling: While throwing a snowball on a hilltop should better be avoided, it is not certain to cause disaster. Whether the snowball causes an avalanche depends on many environmental conditions and events that are outside the control of the thrower. Moreover, whether the avalanche results in disaster depends on both the state and the reaction of the system residing at the bottom of the mountain.

Structure of this Article: After introducing basic notions and formal notation in Section II, we define the syntax and semantics of attack trees in Section III. In Section IV, we introduce a logic for describing attacker goals. In Section V, we develop our framework for defining success criteria for attacks. In Section VI, we apply our framework to clarify the success criteria used in prior applications and discuss the lessons learned from this application of our framework. In Section VII, we introduce our novel variant of attack trees, exploit trees, and illustrate it at concrete examples. After discussing related work in Section VIII, we conclude in Section IX.

II. Basic Notions and Notation

We use $\mathbb{B} = \{true, false\}$ to denote the set of booleans and $\mathbb{N}_0 = \{0, 1, 2, \dots\}$ to denote the set of natural numbers. For $i, j \in \mathbb{N}_0$ with $i \leq j$, we write $[i, j]$ for the subset of $\mathbb{N}_0$ containing i, j, and all natural numbers in between, i.e., $[i, j] = \{i, \dots, j\}$. If $i > j$, then $[i, j] = \emptyset$ holds. Each nonempty, finite set $N \subseteq \mathbb{N}_0$ has a unique maximal element denoted by $max(N)$. If N has infinite size, then $max(N) = \infty$.

We use $X \rightharpoonup Y$ and $X \to Y$ to denote the space of partial functions and the space of total functions from a domain X to a co-domain Y, respectively. We use dom and $cdom$ to retrieve the domain and co-domain, respectively, of a function. That is, $dom(f) = dom(g) = X$ and $cdom(f) = cdom(g) = Y$ hold for $f\colon X \rightharpoonup Y$ and $g\colon X \to Y$. Moreover, we use def and img to retrieve the set of elements for which a function is defined and the set of elements that can be reached by a function, respectively. That is, $def(f) = \{\, x \in dom(f) \mid \exists y \in cdom(f)\colon\ f(x) = y \,\}$ and $img(f) = \{\, y \in cdom(f) \mid \exists x \in def(f)\colon\ f(x) = y \,\}$ hold.

A *predicate over a set* X is a function from the space $X \to \mathbb{B}$. We define predicates $even$ and odd over $\mathbb{N}_0$ by $even(n) = true$ if n is even and $even(n) = false$ otherwise and $odd(n) = true$ if n is odd and $odd(n) = false$ otherwise.

A. Finite and Infinite Sequences

Given a set X, we use functions from $\mathbb{N}_0 \to X$ to model infinite sequences over X. We use $SEQ^{\text{inf}}(X)$ to denote the set of all such infinite sequences. Moreover, we use functions $f\colon \mathbb{N}_0 \rightharpoonup X$, with $def(f) = \emptyset$ or $def(f) = [0, i]$, for some $i \in \mathbb{N}_0$, to model finite sequences over X. We define the length of a finite sequence f by $\sharp f = 0$ if $def(f) = \emptyset$, and by $\sharp f = max(def(f)) + 1$ if $def(f) \neq \emptyset$. We use $SEQ^{\text{fin}}(X)$ to denote the set of all such finite sequences. Finally, we use $SEQ(X)$ to denote the set of all infinite and all finite sequences over X (i.e., $SEQ(X) = SEQ^{\text{inf}}(X) \cup SEQ^{\text{fin}}(X)$).

For readability, we use $\langle\rangle$ to denote the empty sequence (i.e., $\langle\rangle\colon \mathbb{N}_0 \rightharpoonup X$ with $def(\langle\rangle) = \emptyset$) and $l = \langle x_0, \dots, x_n \rangle$ to denote a finite sequence with $n + 1$ elements (i.e., $l\colon \mathbb{N}_0 \rightharpoonup X$ with $def(l) = [0, n]$ and $l(i) = x_i$ for each $i \in def(l)$).

For appending to a finite sequence, we use the function

$$\begin{aligned}\circ\colon\ & ((SEQ^{\text{fin}}(X) \times SEQ^{\text{fin}}(X)) \to SEQ^{\text{fin}}(X)) \\ \cup & ((SEQ^{\text{fin}}(X) \times SEQ^{\text{inf}}(X)) \to SEQ^{\text{inf}}(X))\end{aligned}$$

$$(l_1 \circ l_2) : i \mapsto \begin{cases} l_1(i) & \text{, if } i < \sharp l_1 \\ l_2(i - \sharp l_1) & \text{, if } i \geq \sharp l_1 \text{ and } i < \sharp l_1 + \sharp l_2 \\ \text{undefined} & \text{, if } i \geq \sharp l_1 + \sharp l_2 \end{cases}$$

and lift $\circ$ in a pointwise fashion to sets of sequences by

$$(L_1 \circ L_2) = \{l_1 \circ l_2 \mid l_1 \in L_1 \wedge l_2 \in L_2\}\,.$$

A function $emb\colon \mathbb{N}_0 \rightharpoonup \mathbb{N}_0$ is *an embedding* of a sequence $l \in SEQ(X)$ into $l' \in SEQ(X')$ (where $X \subseteq X'$) iff

- $def(emb) = def(l)$,
- $\forall n \in def(emb)\colon\ l(n) = l'(emb(n))$, and
- $\forall n \in (def(emb) \setminus \{0\})\colon\ emb(n) > emb(n-1)$ hold.

We write $emb : l \triangleright l'$ to indicate that emb is an embedding of l into l'. For an embedding $emb\colon \mathbb{N}_0 \rightharpoonup \mathbb{N}_0$ of a nonempty, finite sequence $l \in SEQ^{\text{fin}}(X)$ into a sequence $l' \in SEQ(X')$, we define the *end of* emb by $end(emb) = max(img(emb))$.

A sequence $l \in SEQ(X_1 \cup X_2)$ is an *interleaving* of $l_1 \in SEQ(X_1)$ and $l_2 \in SEQ(X_2)$ (denoted l *interleaves* (l_1, l_2)) iff there exist an embedding emb_1 of l_1 into l and an embedding emb_2 of l_2 into l such that $def(l) = img(emb_1) \cup img(emb_2)$ and $img(emb_1) \cap img(emb_2) = \emptyset$ hold.

For interleaving two sequences, we use the function

$$\sim\colon ((SEQ(X) \times SEQ(X)) \to \mathcal{PS}(SEQ(X)))$$

$$(l_1 \sim l_2) = \{l \in SEQ(X) \mid l \text{ is an interleaving of } l_1 \text{ and } l_2\}$$

and lift $\sim$ in a pointwise fashion to sets of sequences by

$$(L_1 \sim L_2) = \bigcup_{l_1 \in L_1, l_2 \in L_2} l_1 \sim l_2\,.$$

Note that $\circ$ and $\sim$ both are associative operators, i.e.,

$$L_1 \circ (L_2 \circ L_3) = (L_1 \circ L_2) \circ L_3$$
$$L_1 \sim (L_2 \sim L_3) = (L_1 \sim L_2) \sim L_3\,.$$

Hence, the lifting to n-ary operators (for $n \geq 1$) is unambiguous. We use prefix notation for the lifted operators. For instance, the interleaving of three sets L_1, L_2, and L_3 of sequences is denoted by $\sim(L_1, L_2, L_3)$ and equals $L_1 \sim (L_2 \sim L_3)$.

III. Attack Trees

Consider an example scenario where an attacker wants to steal money from an ATM. One possibility might be that she steals the ATM physically. Another possibility might be that she gets physical access to the cash box inside the ATM, for instance, by using explosives or a key. A third possibility might be that she obtains a banking card or a copy of such a card together with the corresponding pin. A fourth possibility might be that she hacks the control software on the ATM to obtain access without proper credentials. This scenario is a subset of a larger set of attacks identified by Fraile *et al.* [12].

An attack tree for this scenario using the usual graphical notation is depicted in Figure 1. Note that the annotation at the root node reflects the attacker's overall goal and that the subtrees of the root reflect the four possibilities sketched above. These subtrees are disjunctively connected, i.e., they describe alternatives for the attacker to achieve her goal. The roots of the subtrees are inner nodes. They are annotated with subgoals that help the attacker in achieving her overall goal.

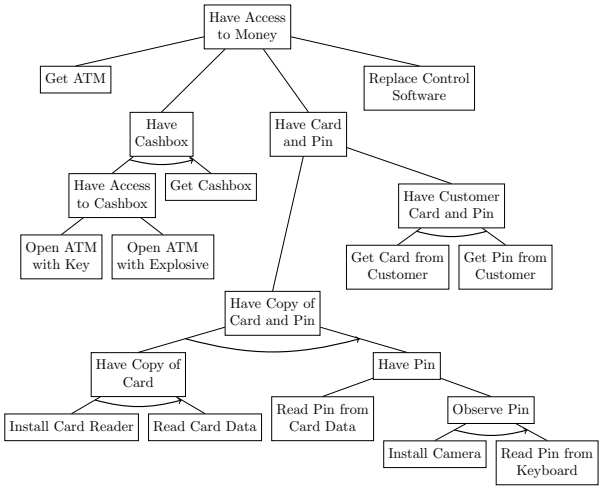

Fig. 1. Example attack tree modeling threats against an ATM (sub-trees for getting the ATM and for replacing the control software are omitted)

The annotations of leaf nodes specify attacker actions. For instance, the two leaves at the bottom left of the tree (labeled "Install Card Reader" and "Read Card Data") specify that the attacker installs a card reader and that she reads the card's data. If executed in this order, these two actions enable the attacker to create a copy of the banking card, which corresponds to the subgoal "Have Copy of Card" of the parent node. That the two leaves are conjunctively connected and need to be processed in a particular order is expressed by the arrow across the lines that connect the parent node to the leaves.

An arc without arrow head across the lines to subtrees indicates a conjunctive connection where it does not matter in which order the subtrees are processed. Consider, for instance, the two leaves on the right-hand side of the tree (labeled "Get Card from Customer" and "Get Pin from Customer"). Here it does not matter whether the attacker obtains the card or the pin first. It only matters that she obtains both.

Overall, the attack tree in Figure 1 models possible ways for an attacker to get access to the money in an ATM.

We now introduce a formal language for describing such attack trees and formally define a semantics for our language.

A. Syntax

Definition 1. *The* language of attack trees *over a set of attacker actions A and a set of attacker goals G is*

$$\mathcal{AT}_{A,G} = \{\mathsf{ROOT}(g\colon at) \mid g \in G \wedge at \in \mathcal{AT}'_{A,G}\}$$

where $\mathcal{AT}'_{A,G}$ is the least set of expressions satisfying:

1) $\mathsf{ACT}(a) \in \mathcal{AT}'_{A,G}$ *for all* $a \in A$ *and*
2) *if* $at_1, \ldots, at_k \in \mathcal{AT}'_{A,G}$ *and* $g \in G$ *then*
   $\mathsf{OR}(at_1, \ldots, at_k)$, $\mathsf{OR}(g\colon at_1, \ldots, at_k) \in \mathcal{AT}'_{A,G}$,
   $\mathsf{AND}(at_1, \ldots, at_k)$, $\mathsf{AND}(g\colon at_1, \ldots, at_k) \in \mathcal{AT}'_{A,G}$, *and*
   $\mathsf{SAND}(at_1, \ldots, at_k)$, $\mathsf{SAND}(g\colon at_1, \ldots, at_k) \in \mathcal{AT}'_{A,G}$.

An expression $\mathsf{ROOT}(g\colon at)$ specifies an attack tree consisting of a root with a single subtree *at* and a goal g. An expression $\mathsf{ACT}(a)$ specifies a subtree consisting of a single node with an action a. The expressions $\mathsf{OR}(at_1,\ldots,at_k) \in \mathcal{AT}_{A,G}$ and $\mathsf{OR}(g\colon at_1,\ldots,at_k) \in \mathcal{AT}_{A,G}$ both specify subtrees consisting of k disjunctively connected subtrees $at_1,\ldots,at_k$. In the second expression, g specifies the subgoal of the tree.

We distinguish the *parallel conjunction* AND (for brevity, *conjunction*) from the *sequential conjunction* SAND. Intuitively, a sequential conjunction (indicated by an arrow in Figure 1) expresses that subtrees shall be traversed in the order in which they occur. In contrast, a parallel conjunction (indicated by an arc) does not prescribe any order for the traversal.

Note that our language mandates the specification of an attacker goal in the root node, but, otherwise, leaves the specification of subgoals in inner nodes of the tree optional.

Example 1. *The subtree with the subgoal "Have Card and Pin" in Figure 1 can be represented in our language by*

OR*(Have Card and Pin:*
  SAND*(Have Copy of Card and Pin:*
    SAND*(Have Copy of Card:*
      ACT*(Install Card Reader),* ACT*(Read Card Data)),*
    OR*(Have Pin:*
      ACT*(Read Pin from Card Data),*
      SAND*(Observe Pin:*
        ACT*(Install Camera),*
        ACT*(Read Pin from Keyboard))))*
  AND*(Have Customer Card and Pin:*
    ACT*(Get Card from Customer),*
    ACT*(Get Pin from Customer)))*.

The overall attack tree can be formalized in our notation, starting from $\mathsf{ROOT}(\textit{Have Access to Money} : \mathsf{OR}(\ldots))$.

B. Semantics

To attack a system, an attacker selects particular actions and a particular order in which she performs these actions.

Definition 2. *An* attack *over a set of attacker actions A is a non-empty sequence of attacker actions $att \in (SEQ^{\text{fin}}(A)\setminus\{\langle\rangle\})$.*

In this article, we focus on attacks that can be performed in finite time. Therefore, we restrict attacks to finite sequences.

We define the semantics of the language $\mathcal{AT}_{A,G}$ in terms of attacks such that the semantics of each expression $at \in \mathcal{AT}_{A,G}$ is a set of attacks $[\![at]\!] \subseteq (SEQ^{\text{fin}}(A)\setminus\{\langle\rangle\})$.

Definition 3. *The function $[\![\cdot]\!]\colon \mathcal{AT}_{A,G} \to \mathcal{PS}(SEQ^{\text{fin}}(A)\setminus\{\langle\rangle\})$ is defined recursively by:*

1) $[\![\mathsf{ACT}(a)]\!] = \{\langle a\rangle\}$,
2) $[\![\mathsf{ROOT}(g\colon at)]\!] = [\![at]\!]$,
3) $[\![\mathsf{OR}(at_1,\ldots,at_k)]\!] = \bigcup_{i\in\{1,\ldots,k\}} [\![at_i]\!]$
   $[\![\mathsf{OR}(g\colon at_1,\ldots,at_k)]\!] = \bigcup_{i\in\{1,\ldots,k\}} [\![at_i]\!]$
   $[\![\mathsf{AND}(at_1,\ldots,at_k)]\!] = \sim([\![at_1]\!],\ldots,[\![at_k]\!])$
   $[\![\mathsf{AND}(g\colon at_1,\ldots,at_k)]\!] = \sim([\![at_1]\!],\ldots,[\![at_k]\!])$
   $[\![\mathsf{SAND}(at_1,\ldots,at_k)]\!] = \circ([\![at_1]\!],\ldots,[\![at_k]\!])$
   $[\![\mathsf{SAND}(g\colon at_1,\ldots,at_k)]\!] = \circ([\![at_1]\!],\ldots,[\![at_k]\!])$

That is, the semantics of a leaf node with an action a is the set containing one attack consisting of one occurrence of a. The semantics of an inner nodes depends on the connective. For OR, AND, and SAND, the operators $\bigcup$, $\sim$, and $\circ$ are used, respectively, to construct a set of attacks from the semantics of an inner node's subtrees. Note that $[\![at]\!]$ is guaranteed to be a non-empty set, and that every element in $[\![at]\!]$ is a non-empty sequence, because each $at \in \mathcal{AT}_{A,G}$ contains at least one subexpression of the form $\mathsf{ACT}(a)$.

Example 2. *For the expression from Example 1, we have*

$$[\![\mathsf{OR}(\textit{Have Card and Pin},\ldots)]\!] = \left\{\begin{array}{l}\{\langle \textit{Install Card Reader}, \textit{Read Card Data},\\ \quad \textit{Read Pin from Card Data}\rangle\\ \langle \textit{Install Card Reader}, \textit{Read Card Data},\\ \quad \textit{Install Camera}, \textit{Read Pin from Keyboard}\rangle\\ \langle \textit{Get Card from Customer}, \textit{Get Pin from Customer}\rangle\\ \langle \textit{Get Pin from Customer}, \textit{Get Card from Customer}\rangle\end{array}\right\}$$

Remark 1. *Our semantics is in the tradition of action-centered semantics for attack trees (e.g., [2, 9, 10]). Among these, the SP semantics [2] is most closely related. The main difference to our semantics is that Jhawar et. al. use a compact graph-based representation where series-parallel graphs represent multiple possible runs, while we prefer a representation where each possible run is explicitly modeled by a separate trace. Since we use traces that linearize occurrences of actions,* $[\![\mathsf{AND}(a,b)]\!] = [\![\mathsf{OR}(\mathsf{SAND}(a,b),\mathsf{SAND}(b,a))]\!]$ *holds according to our semantics (but not in [2]).*

IV. Attackers and their Goals

We consider stateful attackers who are interacting with the systems that they attack via shared-memory and/or message-passing communication. The goal of an attacker might be, e.g., to learn system-internal secrets, to corrupt sensitive data within the system, or to create a situation where the system cannot provide the services anymore that it should provide.

We define a formal language for specifying such attacker goals and use traces to formalize the behavior of systems, of attackers, and of benign parts of a system's environment.

A. Behavioral Model

When an attacker acts then this might affect the attacker's own state (e.g., if she learns a secret), the system's state (e.g., if she provides corrupt data to the system), and the environment's state (e.g., if she broadcasts a secret). Similarly, actions performed by the system or by benign actors in the system's environment might affect the attacker's state, the system's state, and the environment's state, including elements that are shared such as communication lines and shared memory.

The level of detail at which a scenario needs to be modeled depends on many aspects. To ensure the reliability of a security analysis, all aspects of a scenario that are relevant to the attacks

considered must be faithfully captured by a behavioral model. However, cluttering such a model with too many details should be avoided to keep the model construction and the security analysis tractable. For instance, one might decide to not model benign actors in a system's environment at all, if they are irrelevant to the attacks of interest.

For capturing snapshots of a scenario, we employ a formal definition of states, where states are mappings from locations to values. Locations represent containers for storing data such as memory locations, registers, or program variables.

Definition 4. *Let L be a set of locations, and let V be a set of values. A* state over L and V *is a function $s\colon L \to V$.*

For a security analysis of a scenario, one needs to specialize this abstract definition of states by specifying sets L and V concretely. In this process, one might desire to clearly distinguish which locations belong to the system, to the attacker, and to the benign environment. One might also desire to restrict the values that each given location may assume. For the purposes of this article, we use our abstract definition of states and only need to refine it in illustrating examples.

We use traces to capture possible and actual behaviors in a scenario and define traces as sequences of states and actions.

Definition 5. *A* trace *over a set of states S and a set of actions A is an infinite or a finite sequence $tr \in SEQ(S \cup A)$.*

This definition of traces is rather general. In a security analysis, one might decide to use traces of a specific form.

Definition 6. *A trace tr over S and A is a* state trace *iff $img(tr) \subseteq S$ holds, and tr is a* verbose trace *iff*

$$\forall n \in def(tr)\colon (even(n) = true \Rightarrow tr(n) \in S) \\ \wedge (odd(n) = true \Rightarrow tr(n) \in A)\ holds.$$

That is, a trace tr is called a state trace if it does not contain any actions, and tr is called verbose if it starts with a state and if, afterwards, actions and states alternate within tr.

For the reliability of a security analysis, it is essential that a trace faithfully captures the behavior of interest. The order of states and actions in a trace must properly reflect the order of snapshots and events of the given behavior. For the purposes of this article, it is crucial that all snapshots of the behavior and all occurrences of events that are relevant for the attacks of interest are modeled by states and actions in the trace.

As usual, all possible behaviors of a scenario can be modeled by a set of traces. For the reliability of a security analysis, it is essential that each possible behavior of the scenario is faithfully modeled by a trace in this set of traces.

Remark 2. *The causal relationship between actions (of the attacker, of the system, and of the benign environment) and states can be formally captured by defining a transition relation. The set of possible traces can then be inferred from the transition relation as detailed in Appendix A.*

B. Specification Language

We introduce syntax and semantics of a formal language for specifying goals of attackers. The language features predicates, a logical constant to express truth, negation, and conjunction. Locations play the role of term variables in our language, and states play the role of valuations.[2] To keep the exposition simple, we refrain from adding constants and function symbols to the term language and from adding quantifiers and temporal operators to the formula language.

Technically, our definitions follow the usual lines for formally defining syntax and semantics of a logic.

Definition 7. *A* signature *is a pair $(\mathcal{P}, ar)$ consisting of a set $\mathcal{P}$ of* predicates *and a function $ar\colon \mathcal{P} \to \mathbb{N}_0$. For each predicate $P \in \mathcal{P}$, the natural number $ar(P)$ is the* arity *of P.*

Definition 8. *Let $\sigma = (\mathcal{P}, ar)$ be a signature, and let L be a set of locations. The set of formulas over σ and L (denoted by $\mathcal{F}_{\sigma,L}$) is the least set satisfying all of the following conditions:*

1) *$P \in \mathcal{F}_{\sigma,L}$ holds for all $P \in \mathcal{P}$ with $ar(P) = 0$.*
2) *$P(l_1, \ldots, l_n) \in \mathcal{F}_{\sigma,L}$ holds for all $P \in \mathcal{P}$ with $ar(P) = n > 0$ and for all $l_1, \ldots, l_n \in L$.*
3) *$\top \in \mathcal{F}_{\sigma,L}$ holds.*
4) *If $F, F_1, F_2 \in \mathcal{F}_{\sigma,L}$ then*
   a) *$\neg F \in \mathcal{F}_{\sigma,L}$ and*
   b) *$F_1 \wedge F_2 \in \mathcal{F}_{\sigma,L}$ hold.*

Definition 9. *Let $\sigma = (\mathcal{P}, ar)$ be a signature. A* σ-algebra *is a pair $\mathcal{A} = (V, \iota)$ such that V is a set of values and $\iota\colon \mathcal{P} \to \mathcal{PS}(\bigcup_{n\in\mathbb{N}_0} V^n)$ is a function with $\iota(P) \subseteq V^{ar(P)}$ for each $P \in \mathcal{P}$. The set V is the* carrier set *(or brief:* base*) of $\mathcal{A}$, and the function ι is the* interpretation *of $\mathcal{A}$.*

Definition 10. *Let $\sigma = (\mathcal{P}, ar)$ be a signature, and let L be a set of locations. A formula $F \in \mathcal{F}_{\sigma,L}$* is satisfied by a state *$s\colon L \to V$ within a σ-algebra $\mathcal{A} = (V, \iota)$ iff $s \models_{\mathcal{A}} F$, where the* satisfaction relation *$\models$ is defined by*

1) *$s \models_{\mathcal{A}} P$ iff $() \in \iota(P)$ and $ar(P) = 0$;*
2) *$s \models_{\mathcal{A}} P(l_1, \ldots, l_{ar(P)})$ iff $(s(l_1), \ldots, s(l_{ar(P)})) \in \iota(P)$ and $ar(P) \geq 1$;*
3) *$s \models_{\mathcal{A}} \top$ holds;*

4a) *$s \models_{\mathcal{A}} \neg F$ iff $s \models_{\mathcal{A}} F$ does not hold; and*

4b) *$s \models_{\mathcal{A}} F_1 \wedge F_2$ iff $s \models_{\mathcal{A}} F_1$ and $s \models_{\mathcal{A}} F_2$.*

Remark 3. *To keep our definitions simple, we limited the logical connectives to $\neg$ and $\wedge$. As usual, other connective can be added as syntactic abbreviations. For instance, $F_1 \vee F_2$ can be introduced as an abbreviation for $\neg((\neg F_1) \wedge (\neg F_2))$. Moreover, given a set of formulas $\mathcal{F} \subseteq \mathcal{F}_{\sigma,L}$, the expression $\bigvee_{F\in\mathcal{F}} F$ can be introduced as an abbreviation for $(F_1 \vee (F_2 \vee \ldots (F_{n-1} \vee F_n)\ldots))$ where $\langle F_1, F_2, \ldots, F_{n-1}, F_n\rangle$ is a sequence in which each formula of $\mathcal{F}$ occurs once.*

[2]To simplify our technical exposition, we refrain from introducing term variables and valuations as primitive concepts here. Had we introduced term variables and valuations, we would have to equate them explicitly with locations and states, respectively, for the remainder of the article.

C. Specifying Attacker Goals

From an abstract point of view, the attackers that we consider can be viewed as ones who aim at arriving in a *bad state*, where the attacker's goal is made concrete by defining precisely which states are bad. The goals of attackers aiming at breaching confidentiality, integrity, or availability can be described as instances of this abstract notion of attacker goal. If the attacker's goal is to breach confidentiality, then her goal can be characterized by defining the set of bad states to contain all states in which the attacker knows a secret. If the attacker's goal is to breach integrity, then this can be characterized by defining the set of bad states to contain all states in which some location for storing critical data contains a corrupted value. Finally, if the attacker aims at obstructing the system's availability then she has succeeded in this goal, if the system arrives in a state in which the system cannot provide its services.

Example 3. *We consider the subtree with the subgoal "Have Card and Pin" of our running example in Figure 1. We assume that the attacker is interested in the card of a specific person or a copy thereof together with the corresponding pin.*

Assuming two finite sets $Cards$ and $PINs$, whose elements model all customer cards relevant in a scenario and all possible pins, respectively, we introduce four families of predicates

$$(IsPinOfCard_{p,c})_{p\in PINs, c\in Cards},$$
$$(IsCopyOfCard_{c,c'})_{c,c'\in Cards},$$
$$(HasCard_c)_{c\in Cards}, \text{ and } (KnowsPin_p)_{p\in PINs}$$

to model the subgoals in this subtree. We define the arity of predicates $IsPinOfCard_{p,c}$ and $IsCopyOfCard_{c,c'}$ to be zero, and the arity of $HasCard_c$ and $KnowsPin_p$ to be one.

Intuitively, $IsPinOfCard_{p,c}$ captures whether p models the pin of the card modeled by c, and $IsCopyOfCard_{c,c'}$ captures whether c models a copy of the card modeled by c'. Formally, this intuition is captured by defining ι as follows: If p models the pin of the card modeled by c then $\iota(IsPinOfCard_{p,c}) = \{()\}$ and, otherwise, $\iota(IsPinOfCard_{p,c}) = \emptyset$. If c models a copy of the card modeled by c' then $\iota(IsCopyOfCard_{c,c'}) = \{()\}$ and, otherwise, $\iota(IsCopyOfCard_{c,c'}) = \emptyset$.

We introduce the locations Knowledge *and* Storage *to model the knowledge of the attacker and the attacker's storage place for cards, respectively. Moreover, we use subsets of $PINs$ and $Cards$ as values. That is, L and V are chosen such that* $\text{Knowledge}, \text{Storage} \in L$ *and* $\mathcal{PS}(PINs), \mathcal{PS}(Cards) \subseteq V$.

We use states that assign subsets of $PINs$ to Knowledge *and subsets of $Cards$ to* Storage*. If $p \in s(\text{Knowledge})$ then this intuitively means that the pin modeled by p is known to the attacker in the snapshot of the scenario modeled by s. Moreover, if $c \in s(\text{Storage})$ then this means that the attacker has the card modeled by c in the snapshot modeled by s.*

We define the interpretation of the two unary predicates $HasCard_c$ and $KnowsPin_p$ by

$$\iota(HasCard_c) = \{(Cards') \mid Cards' \subseteq Cards \wedge c \in Cards'\}$$
$$\iota(KnowsPin_p) = \{(PINs') \mid PINs' \subseteq PINs \wedge p \in PINs'\}$$

According to Definition 10, $s \models_{(V,\iota)} HasCard_c(l)$ iff $s(l) \subseteq Cards$ and $c \in s(l)$ hold. Moreover, $s \models_{(V,\iota)} KnowsPin_p(l)$ iff $s(l) \subseteq PINs$ and $p \in s(l)$ hold.

Let c^* model the card of the person that the attacker is targeting. We define four auxiliary sets of formulas by

$$\mathcal{F}_1 = \left\{ \begin{array}{l} HasCard_c(\text{Storage}) \\ \wedge IsCopyOfCard_{c,c^*} \end{array} \;\middle|\; c \in Cards \right\}$$

$$\mathcal{F}_2 = \left\{ \begin{array}{l} KnowsPin_p(\text{Knowledge}) \\ \wedge IsPinOfCard_{p,c^*} \end{array} \;\middle|\; p \in PINs \right\}$$

$$\mathcal{F}_3 = \left\{ \begin{array}{l} HasCard_c(\text{Storage}) \\ \wedge IsCopyOfCard_{c,c^*} \\ \wedge KnowsPin_p(\text{Knowledge}) \\ \wedge IsPinOfCard_{p,c^*} \end{array} \;\middle|\; \begin{array}{l} c \in Cards, \\ p \in PINs \end{array} \right\}$$

$$\mathcal{F}_4 = \left\{ \begin{array}{l} HasCard_{c^*}(\text{Storage}) \\ \wedge KnowsPin_p(\text{Knowledge}) \\ \wedge IsPinOfCard_{p,c^*} \end{array} \;\middle|\; p \in PINs \right\}$$

We are now ready to formally model the subgoals in the aforementioned subtree in Figure 1. We use the syntactic abbreviations from Remark 3 in the following.

The subgoal "Have Copy of Card" can be expressed by the formula $F_1 = \bigvee_{F\in\mathcal{F}_1} F$. Both subgoals "Have Pin" and "Observe Pin" can be expressed by $F_2 = \bigvee_{F\in\mathcal{F}_2} F$. The subgoal "Have Copy of Card and Pin" can be expressed by the formula $F_3 = \bigvee_{F\in\mathcal{F}_3} F$. The subgoal "Have Customer Card and Pin" can be expressed by the formula $F_4 = \bigvee_{F\in\mathcal{F}_4} F$. Finally, the goal of the entire subtree, i.e. "Have Card and Pin", can be expressed by $F_3 \vee F_4$.

V. A Framework for Defining Success Criteria

The definition of goals and subgoals is an integral part of Schneier's methodology to model threats against systems [1]: One identifies all possible attacker goals and creates an attack tree for each of them. These trees specify attacker goals but yet lack subtrees that clarify how an attacker can achieve the goals. Afterwards one refines each attack tree in a stepwise fashion. At each step, one chooses a leaf node that specifies a goal or subgoal and then details how it can be achieved by adding subtrees to this leaf node. During the intermediate stages of the construction process, there may be leaf nodes that specify goals rather than attacker actions. However, after the refinement of an attack tree has been completed, all leaf nodes specify attacker actions. Hence, the attack trees resulting from the stepwise refinement process could be specified in $\mathcal{AT}_{A,G}$.

After the refinement process has been completed, the goals and subgoals specified in an attack tree remain valuable. Firstly, the goal specified at the root node clarifies the attacker's intent. This allows one to check whether an attack tree is relevant for an application scenario by checking whether the goal specifies an intent that an attacker might plausibly have in the scenario. Secondly, the goals/subgoals clarify the scope of an attack tree and its subtrees. This provides valuable guidance in reading and understanding attack trees in a modular fashion, hence, reducing conceptual complexity in the usage and maintenance

of attack trees. These two examples illustrate that goals and subgoals in attack trees are of high practical relevance.

A. Treatment of Goals and Subgoals in the Semantics

Given this practical importance of goals, one might wonder why they do not play any role in our formal semantics for attack trees (cf. Definition 3). In fact, adding goals does not add any meaning to an attack tree under our semantics. Rather, our formal semantics of attack trees solely focuses on clarifying which attacks are specified by a given attack tree. That goals are completely transparent from the perspective of our semantics (and, hence, constitute mere decorations) might be surprising and could rightfully raise criticism. Note, however, that our treatment of goals is in-line with other formal semantics of attack trees, including [2, 8, 9, 10], which also do not establish any connection between attacks and the attacker's goal.

If avoiding the transparency of goals in the formal semantics were our only objective, then this could be resolved by modifying Definition 3 such that in addition to attacker actions also goals are collected in the recursive definition. Technically, such an adaptation of our semantics would be straightforward, but it would not provide any clarification of whether the specified attacks, indeed, achieve the specified attacker goal.

Hence, we avoid the cluttering of our formal semantics of attack trees with such a superficial treatment of goals. We aim for connecting the attacks and the attacker goal specified by an attack tree in more meaningful ways in this section.

B. Identification of Key Question

Let us investigate how attacks and attacker goals can be connected on a semantic level in the context of attack trees by focusing primarily on the following question:

Does an attack achieve an attacker goal in a run?

This is, indeed, the key question to be clarified. Once we have a precisely defined criterion for whether a given attack achieves a given goal in a given run, then obtaining criteria for answering questions about entire attack trees like, for instance, "*Does the attack tree specify only attacks that are successful in some possible system run?*" or "*Does the attack tree specify all attacks that are successful in some system run?*" become straightforward based on our semantics of attack trees and formulas (cf. Definitions 3 and 10).

When searching for a suitable criterion for when an attack is successful wrt. an attacker goal in a system run (from now, brief: *success criterion*), we found multiple candidate definitions that each make sense to us. Before clarifying this spectrum of alternative success criteria, we coin and formalize some concepts that we use in the definition of such criteria.

C. Formalization of Relevant Concepts

We define two liftings of the satisfaction of formulas to traces. The first lifting requires the formula to be satisfied at a particular point of a trace, and the second lifting requires the formula to be satisfied from a particular point onwards.

Definition 11. *An* attacker goal $g \in \mathcal{F}_{\sigma,L}$ is satisfied in $tr \in SEQ(S \cup A)$ at $n \in def(tr)$ *iff* $tr(n) \in S$ *and* $tr(n) \models g$ *hold.*

An attacker goal $g \in \mathcal{F}_{\sigma,L}$ is an invariant in $tr \in SEQ(S \cup A)$ from a point $n \in def(tr)$ *iff* g *is satisfied in* tr *at* n *and* $tr(n') \in S$ *implies* $tr(n') \models g$ *for all* $n' \in def(tr)$ *with* $n' \geq n$.

We write $tr@n \models_{\mathcal{A}} g$ to indicate that g is satisfied in tr at n and $tr@n \models^{+}_{\mathcal{A}} g$ if g is an invariant in tr from n.

We define what it means for an attack to occur in a trace.

Definition 12. *Let* $A_{Att} \subseteq A$ *be a set of attacker actions.*

An occurrence of an attack $att \in (SEQ^{\text{fin}}(A_{Att}) \setminus \{\langle\rangle\})$ in a trace $tr \in SEQ(S \cup A)$ *is an embedding* $emb \colon \mathbb{N}_0 \rightharpoonup \mathbb{N}_0$ *of* att *into* tr. *The* start of the occurrence emb *is* $emb(0)$, *and the* end of the occurrence emb *is* $end(emb)$.

There may be zero, one, or multiple occurrences of a given attack in a given trace. We say that *an attack* att *occurs in a trace* tr if there is at least one occurrence of att in tr.

We introduce two restricted forms of occurrences of attacks.

Definition 13. *Let* $A_{Att} \subseteq A$ *be a set of attacker actions.*

An occurrence $emb : \mathbb{N}_0 \rightharpoonup \mathbb{N}_0$ *of an attack* $att \in (SEQ^{\text{fin}}(A_{Att}) \setminus \{\langle\rangle\})$ *in* $tr \in SEQ(S \cup A)$ *is* A_{Att}-pure *iff*

$$\forall n \in [emb(0), end(emb)] \colon (n \in img(emb) \vee tr(n) \notin A_{Att}),$$

and $emb \colon \mathbb{N}_0 \rightharpoonup \mathbb{N}_0$ *is* uninterrupted *iff*

$$\forall n \in [emb(0), end(emb)] \colon (n \in img(emb) \vee tr(n) \notin A).$$

That is, if an occurrence emb of att in tr is A_{Att}*-pure*, then attacker actions may occur in tr at positions between $emb(0)$ and $end(emb)$ (i.e., from where the first action of att occurs in tr to where the last action of att occurs in tr) only if these positions are in $img(emb)$. Intuitively, this means that the actions of the attack occur in the trace without occurrences of other attacker actions in between. Similarly, if an occurrence emb of att in tr is uninterrupted, then the actions of the attack att occur in the trace tr under emb without occurrences of any other actions in between. Note that, if an occurrence emb of att in tr is uninterrupted, then it is also A_{Att}*-pure*.

D. Degrees of Freedom in Defining a Success Criterion

It is fair to assume that any sensible success criterion[3] implies both that the attack must occur in the run and that the attacker goal must be satisfied at some point during the run. At first sight, one might think that the implication in the other direction should then also hold, but this is not such a clear case. In fact, the success criterion for which the implication holds in both directions is the most liberal one in the spectrum of success criteria that we outline in the following.

As stated in the introduction, we identified three degrees of freedom in the definition of a success criterion for attacks:

[3]Recall that we introduced "success criterion" as abbreviation for "criterion for when an attack is successful in a run wrt. a given attacker goal".

Purity May occurrences of actions of an attack be interleaved with occurrences of other actions and, if yes, which ones?

Persistence Is it sufficient if the attacker's goal is satisfied at some point in time or should it be satisfied persistently?

Causality How much certainty does one desire that the satisfaction of the goal is, indeed, a result of the attack?

We present multiple options for resolving each of these three degrees of freedom. For each option that we present, we argue why it makes sense, but we do not strive for covering all sensible options. What we propose here is meant to be an open framework to which further options can be added later.

Let $tr \in SEQ(S \cup A)$ be a trace, $A_{Att} \subseteq A$ be a set of attacker actions, and $att \in (SEQ^{\text{fin}}(A_{Att}) \backslash \{\langle\rangle\})$ be an attack.

Purity: As said before, there must be an occurrence $emb\colon \mathbb{N}_0 \rightharpoonup \mathbb{N}_0$ of att in tr. The options that we present for Purity differ in which actions may occur in the trace tr at positions in $[emb(0), end(emb)]$ that are not in $img(emb)$:

PU1 Arbitrary other actions may occur.

PU2 Other actions may occur except for attacker actions (i.e., emb is an A_{Att}-*pure* occurrence of att in tr).

PU3 No other actions must occur (i.e., emb is an uninterrupted occurrence of att in tr).

Allowing arbitrary other actions to occur in between the actions specified by an attack (i.e., Option PU1) means that the criterion is robust against insertion of occurrences of actions into a trace. This means, if an attacker briefly deviates from the action sequence specified by the attack by trying out other actions, then the attack is still present according to our Option PU1.

In some cases, it is too liberal to allow arbitrary actions in between the actions specified by an attack. For instance, consider an attack in which an attacker gains knowledge about a secret by obtaining a crypto key in an intermediate step. Such an attack has little chance to be successful, if an action occurs in between that makes the attacker forget the key. Here, one might prefer Option PU2, i.e., to not consider the attack to be present when other attacker actions occur in between.

A similar situation occurs, for instance, if a system has actions at its disposal that undo the effects of attacker actions. For instance, if the system can cause a key refresh then this voids the effect of the attacker learnings, because keys she has learned become outdated. This motivates Option PU3.

Be reminded that we are striving here for a declarative criterion that clarifies what it means for an attack to occur in a given trace. We are not concerned with the likelihood of traces to occur. This is why the examples given for Option PU2 and PU3 are relevant, even though an attacker would probably not deliberately forget a key that she knows and needs, and even if an attacker might be able to exploit a key so quickly that the system cannot perform a key refresh quick enough.

Persistence: We consider two options for Persistence:

PE1 The attacker goal must be satisfied in tr at some point $n \in def(tr)$.

PE2 The attacker goal must be an invariant in tr from some point $n \in def(tr)$.

An attacker who corrupts an access control to access a bank safe aims at arriving at a state where she owns a large amount of money. After she successfully stole money from the bank safe, she reaches a state where her goal is satisfied. Later she might perform costly purchases and enjoy a lavish lifestyle. At a certain point in time, her goal to own a large amount of money will not be satisfied anymore. In such a scenario, Option PE1 is fitting, while Option PE2 is too restrictive.

An attacker who aims at learning the master key of a system might aim at keeping the knowledge of the master key even under updates. In this case Option PE2 is more appropriate.

Obviously, there are many other sensible options for Persistence that one could consider like, for instance, that the goal must be satisfied for some duration.

Causality: This might be the most controversial degree of freedom as it gives rise to a great variety of options. It also motivates our proposal of exploit trees, a novel variant of attack trees, in Section VII. In the following, let $emb\colon \mathbb{N}_0 \rightharpoonup \mathbb{N}_0$ be the embedding of att into tr (satisfying the option for Purity) and $n \in def(tr)$ be the point at/from which the attacker goal is satisfied/an invariant (satisfying the option for Persistence).

We consider the following four options to express the causal relationship between emb, the occurrence of the attack, and n, the point where the attacker goal is satisfied. We express all of these options as constraints on n and emb:

C1 n and emb may be arbitrary

C2 n must be greater than the point where the first action of att occurs in tr under emb.

C3 n must be greater than the point where the last action of att occurs in tr under emb.

C4 n must equal the addition of 1 to the point where the last action of att occurs in tr under emb.

Option C1 is the most liberal one. It does not require any causal dependency between the actions in the attack and the point(s) where the attacker's goal is satisfied. One might find it too liberal to say that an attack successfully achieves an attacker goal if the goal was satisfied anyway. This motivates our consideration of the other three, more restrictive options.

Option C2 requires that at least one action from the attack has occurred before the point n. This is a causal dependency between the point(s) where the attacker's goal is satisfied and the occurrence of the attack, but a rather minimal one.

Option C3 requires a stronger causal dependence, namely that the entire attack must have occurred before the point n. Option C4 is a specialization of Option C3 requiring the point at/from which the attacker goal is satisfied/an invariant in tr to be located directly after the last action of the attack.

Many other sensible options for Causality could be defined. For instance, one could consider alternatives that have a less temporal flavor than Options C2, C3, and C4.

E. Representation of Success Criteria and Taxonomy

In Section V-D, we presented multiple options for each degree of freedom in the definition of a criterion for the success

of an attack in a run wrt. a given attacker goal. For each of these options, we provided a motivation and described circumstances under which the option might or might not be suitable.

It remains to give precise characterizations of these options and of the success criteria that result for each combination of options. Since all options are compatible with each other, they can be freely combined in the definition of a success criterion.

The core of our novel framework is the template for the definition of success criteria by 2nd-order formulas in Figure 2. The three grey boxes of increasing darkness in the figure formalize the options for Purity, Persistence, and Causality, respectively, that we informally introduced in Section V-D. If further options shall be added to the framework, their formalizations would need to be added to the template.

$$
\begin{array}{l}
\exists emb : \mathbb{N}_0 \rightharpoonup \mathbb{N}_0 \colon \exists n \in def(tr) \colon \\
emb \colon att \triangleright tr \\
\wedge\, \forall n' \in ([emb(0), end(emb)]) \setminus img(emb) \colon \\
\quad [\ \top \mid tr(n') \notin A_{Att} \mid tr(n') \notin (A_{Att} \cup A_{Sys} \cup A_{Env})\ ] \\
\wedge\ [\ tr@n \models_{\mathcal{A}} g \mid tr@n \models^{+}_{\mathcal{A}} g\ ] \\
\wedge\ [\ \top \mid n > emb(0) \mid n > end(emb) \mid n = 1 + end(emb)\ ]
\end{array}
$$

Fig. 2. A template for defining success criteria (alternative options for Purity, Persistence, and Causality are surrounded by [and], and are separated by |)

We use expressions $\mathsf{T}(i,j,k)$ with $i \in \{1,2,3\}$, $j \in \{1,2\}$, and $k \in \{1,2,3,4\}$ as shorthand for the formulas that result from instantiating our template with the ith option for Purity, the jth option for Persistence, and the kth option for Causality. That is, for instance, $\mathsf{T}(3,1,4)$ corresponds to the formula

$$
\begin{array}{l}
\exists emb : \mathbb{N}_0 \rightharpoonup \mathbb{N}_0 \colon \exists n \in def(tr) \colon \\
emb \colon att \triangleright tr \\
\wedge\, \forall n' \in ([emb(0), end(emb)]) \setminus img(emb) \colon \\
\quad tr(n') \notin (A_{Att} \cup A_{Sys} \cup A_{Env}) \\
\wedge\, tr@n \models_{\mathcal{A}} g \\
\wedge\, n = 1 + end(emb)
\end{array}
$$

Note that, if two instantiations of our template differ in the options chosen, then the resulting formulas are not equivalent.

In some cases, the success criteria resulting from different combinations of options are related by logical implication. This is clarified by the following theorem.

Theorem 1. *$\mathsf{T}(i,j,k)$ implies $\mathsf{T}(i',j',k')$ if $i \geq i'$, $j \geq j'$, and $k \geq k'$ for $i, i' \in \{1,2,3\}$, $j, j' \in \{1,2\}$, and $k, k' \in \{1,2,3,4\}$.*

Proof. The options for Purity become more restrictive in the order in which they appear in the template. That is, $tr(n') \notin (A_{Att} \cup A_{Sys} \cup A_{Env}) \Rightarrow tr(n') \notin A_{Att}$ and $tr(n') \notin A_{Att} \Rightarrow \top$.

Likewise, for Persistence $tr@n \models^{+}_{\mathcal{A}} g \Rightarrow tr@n \models_{\mathcal{A}} g$.

Finally, the options for Causality also become more restrictive in the order in which they appear in the template. That is, the implications $n = 1 + end(emb) \Rightarrow n > end(emb)$, $n > end(emb) \Rightarrow n > emb(0)$, and $n > emb(0) \Rightarrow \top$ hold.

The theorem follows from the fact that the sub-formulas capturing the options for Purity, Persistence, and Causality are conjunctively connected within our template. □

Theorem 1 could be used by a security analyst to migrate to a more suitable success criterion in a stepwise fashion. If the security analyst finds that the currently used success criterion is too liberal, then he could migrate at each step of the search process to an option with a higher number for Purity, Persistence, or Causality until a suitable success criterion is found. Analogously, if the success criterion is too restrictive, he would migrate to options with a smaller number.

Remark 4. *We provided a template for defining criteria for the success of an attack in a trace wrt. a given attacker goal by 2nd-order formulas. For specifying attacker goals, we introduced a propositional logic in Section IV in which propositions may be parameterized by locations. An alternative possibility would have been to employ a temporal logic for specifying attacker goals that is so expressive that both the attacker goal and the desired success criteria can be expressed inside this logic.*

While this might be appealing, in particular if one could benefit from existing model checking techniques and tools, it would not have helped us in achieving the conceptual clarification that we were striving for. In fact, we found it helpful, to have a clear separation between attacker goals and success criteria when deepening our understanding.

Also note that moving to a richer logic for attacker goals does not make the questions addressed in this article void. When using a richer logic, a security analyst still has to face the task to identify the formulas that faithfully capture both the attacker's goal and an appropriate success criterion.

VI. Selection and Application of Success Criteria

We now have a framework for defining success criteria in a fashion that is both precise and uniform.

A precisely defined success criterion can serve as reference point in the evaluation of a given attack tree for a given system model. Based on the success criterion and our formal semantics of attack trees, one can check, e.g., whether the attack tree *at* with goal g is *correct* for a set of possible traces Tr, i.e.,

$$\forall att \in [\![at]\!] \colon \exists tr \in Tr \colon \text{“}att \text{ achieves } g \text{ in } tr\text{”}$$

or *complete* in its specification of successful attacks, i.e.,

$$
\begin{array}{l}
\forall att \in (SEQ^{\text{fin}}(A) \setminus \{\langle\rangle\}) \colon \\
(\exists tr \in Tr \colon \text{“}att \text{ achieves } g \text{ in } tr\text{”}) \Rightarrow att \in [\![at]\!]\,.
\end{array}
$$

In both of these cases, the chosen success criterion provides the precise meaning of “att achieves g in tr”.

A precisely defined success criterion can also serve as a reference point for the evaluation of techniques and tools for the automatic generation of attack trees. Based on the success criterion, one can investigate questions such as “*Is every attack tree generated by the technique/tool correct?*” or “*Is every attack tree generated by the technique/tool complete?*”. Without a precisely defined success criterion, one lacks the reference point needed for verifying attack-tree generation techniques.

Finally, a precisely defined success criterion provides a basis for studying the *effectiveness of defense mechanisms*:

$$\exists att \in (SEQ^{\text{fin}}(A) \setminus \{\langle\rangle\}) : \\ (\exists tr \in Tr : \text{“}att \text{ achieves } g \text{ in } tr\text{”}) \\ \wedge \neg(\exists tr \in Tr' : \text{“}att \text{ achieves } g \text{ in } tr\text{”})$$

where Tr and Tr' specify the set of possible runs without and with the defense being present. Here, a defense mechanism is considered to be effective if at least one attack specified by a given attack tree becomes unsuccessful due to the integration of the defense mechanism into the system.

The uniformity of definitions of success criteria provides a basis for comparisons. One can investigate questions such as "*In which respect do the success criteria underlying two correct attack-tree generation techniques differ?*" or "*Does one attack-tree generation technique assume a more restrictive success criterion than another attack-tree generation technique?*".

In the following, we illustrate how the success criteria underlying prior publications can be captured in our framework. We consider techniques for the automatic generation of attack trees [5, 6, 13, 14] and a technique for guiding security analysts in the stepwise refinement of attack trees [7].

A. Identification of Underlying Success Criteria

We specify success criteria of prior publications by identifying the relevant options for Purity, Persistence, and Causality. The formal characterization of the success criteria then follows from the template provided by our framework.

Our overall findings are summarized in Figure 3.

In the figure, we name the relevant option for each degree of freedom in the 2nd, 3rd, and 4th column. In case, a degree of freedom could not be resolved from the publication, we write "-". In case, a degree of freedom could only be resolved for illustrating examples, we indicate this by "- (NO)", where NO names the option underlying the examples. The 5th column specifies the alphabet of actions that may be used in an attack tree. We write A_{Att} for the set of attacker actions, A_{Sys} for the set of system actions, A_{Env} for the set of environment actions, and A for $A_{Att} \cup A_{Sys} \cup A_{Env}$. Some approaches consider variants of attack trees in which actions must not be used as annotations. We indicate this by $\emptyset$. Moreover, we use '? (A_{Att})" to indicate that it did not become sufficiently clear from the respective publication which actions are permitted in an attack tree, but our impression from text and illustrating examples is that likely, only actions from A_{Att} should occur.

Vigo et al. [5] propose an approach to generate attack trees using static analysis based on a process calculus. They model the behavior of systems by process terms in the Quality Calculus, a variant of the π-calculus that features flexible binders. The attacker is modeled as a process running concurrently with the system. The attacker and the system communicate via channels, where the attacker can use a channel only if she knows the channel's name. The goal of the attacker is to communicate in a way such that the system reaches a particular system state (i.e., Option PE1 for Persistence).

	Purity	Persistence	Causality	actions in attack tree
Vigo et al. [5]	PU1	PE1	-	A_{Att}
Ivanova et al. [6]	PU3	PE1	C4	? (A_{Att})
Gadyatskaya et al. [13]	PU3	PE1	C4	A
Kammmüller [14]	PU3	- (PE1)	- (C4)	? (A_{Att})
Audinot et al. [7]	PU1	PE1	C4	$\emptyset$

Fig. 3. Identification of the success criteria that underly prior work

Vigo et al. [5] generate attack trees for a given system, modelled by a process term, using propositional formulas as an intermediate representation. In the first step, the process term is translated in a syntax-directed fashion to a set of propositional formulas of the forms $\Phi \Rightarrow \overline{l}$, $\Phi \Rightarrow \overline{c}$, and $\Phi \Rightarrow \overline{x}$, where l is a location in the process term, c is a channel name and x is a variable name. Intuitively, a formula $\overline{l}$ means that the attacker can make the system reach location l, $\overline{c}$ means that the attacker can learn channel name c, and $\overline{x}$ means that the attacker can learn the variable x. A formula $\Phi \Rightarrow \overline{l}$ means that the attacker can reach the location l if she can fulfill Φ.

Attack trees are generated from the set of generated implications for a given attacker goal (i.e., a location l that shall be reached) by backwards chaining. In this process, the knowledge that an attacker needs to have in order to reach l is recorded. The resulting attack tree is an AND/OR-tree where leaves are annotated by channel names. That is, channel names correspond to attacker actions in our setting. The semantics of a generated attack tree is the set of all combinations of channel names that allow an attacker to successfully reach l. The underlying semantics of attack trees is closer in spirit to [8] than to the semantics given in Section III as it remains under-specified in which order channel names need to be used.

A consequence of the semantics not being specific in this respect is that our definition of occurrences of attacks cannot be used. Consequently, it remains unclear which option to choose for Causality. An attack tree may only contain attacker actions (i.e., channel names), but the attacker, the system and other actors may perform steps in between the use of channel names specified by an attack (i.e., Option PU1 for Purity).

Ivanova et al. [6] propose an approach to generating attack trees by policy invalidation. In their terminology, a "global policy" specifies either that some action must not be enabled for a certain actor or that some asset must not reach a certain location. The attacker's goal is to invalidate such a global policy. They use "local policies" to specify for each action, which credentials are needed for performing this action. For connecting subtrees, they support SAND and OR.

They use a fairly sophisticated model for socio-technical systems that supports the modeling of geographic locations using two levels, "locations" and "domains", where "locations" may belong to multiple "domains" (to model, e.g, that a room

is part of some building and also of some city). They provide concepts for modeling actors and passive assets (e.g., data), where actors are associated with a behavior, assets carry a value, and both have a location. Assets may be connected to actors such that they move around with that actor. In addition, assets may be associated with a "metric" to capture, e.g., price, time, impact, or a probability. They use credentials to specify the enabledness of actions and introduce four special actions on credentials, geographic locations, and "metrics".

Ivanova et al. perform a forward reachability analysis to identify whether some state is reachable in which a global policy is violated (i.e., Option PE1 for Persistence and Option C4 for Causality). In their reachability analysis, they consider all actions that can be performed by actors whom they consider as attackers. The occurrence of other actions in between attacker actions is not permitted (i.e., Option PU3 for Purity). From their description it does not become clear whether one is allowed to consider also the system as an attacking actor. This ambiguity is relevant for understanding which actions may occur in an attack tree. In their illustrating example, only attacker actions occur in the attack tree (indicated by "? (A_{Att})" in Figure 3).

They claim that an attack tree generated by their approach is correct, in the sense that every attack specified by the tree is a successful attack, and also complete, in the sense that every successful attack is specified by the tree. These claims are plausible, but made informally and without proof.

Gadyatskaya et al. [13] observe that automatically generated attack trees lack informative subgoals and a refinement structure that manually created attack trees usually feature and that is helpful for security analysts to obtain an overview of potential vulnerabilities. They aim at bridging this gap and propose the use of refinement specifications for constraining the structure of generated attack trees. They consider SAND and OR for connecting subtrees and build on the SP semantics from [2], which is similar to our semantics in Section III.

Sets of predicates are used both to specify attacker goals and to characterize system states. That is, an attacker goal is satisfied if a system state is reached that features the desired predicates (i.e., Option PE1 for Persistence and Option C4 for Causality). They define abstraction-based refinement specifications as a constraint system for soundly propagating subgoals of subtrees to parent nodes. This propagation does not depend on the behavioral specification of the system, and it is rather conservative because, for an SAND-connection of subtrees, only the rightmost subtree may contribute to the goal defined in the parent. The propagation assumes that no other actions occur in between the actions specified by an attack (i.e., Option PU3 for Purity). Interestingly, they do not limit the actions that may occur in an attack tree to attacker actions, but also permit the use of actions of benign actors to occur. This understanding is confirmed by their illustrating example.

Gadyatskaya et al. propose a technique for the generation of "correct" attack trees. They call an attack tree "correct" if it represents a given set of attacks and satisfies given refinement specifications. This notion of correctness does not require any relation between the specified attacks and the attacker's goal.

Kammmüller [14] presents an Isabelle/HOL formalization of attack trees building on the Isabelle Insider framework. His formalization covers three designated base attacks: Goto "location", Perform "action", and Credential "location" that model an actors move to a location, the performance of an action by an actor, and the stealing of credentials at a location. He considers AND and OR for connecting subtrees.

Kammmüller uses labeled transition relations to capture the semantics of actions, by sets of triples of the form (s, a, s') where a is the action, s is state before a occurs, and s' is the state after a occurred. This gives rise to a Kripke structure over which CTL formulas can be interpreted. Surprisingly, the transition relation leaves the actor implicit, which makes it hard to distinguish between attacker actions and other actions, unless one implicitly assumes that all actions are performed by the attacker, which is the case in the illustrating example (hence, the "? (A_{Att})" in the last column of Figure 3).

For generating attack trees, a reachability analysis is performed similar in spirit to [6], where the attacker's goal is to invalidate a policy that is specified by a CTL formula. While, in principle, one could use Isabelle/HOL to deal with arbitrary CTL formulas (including ones that characterize success criteria beyond the ones representable by our schema from Section V), Kammmüller provides no guidance for how to construct such CTL formulas (cf. Remark 4). Moreover, the expressiveness of CTL is not needed for expressing the attacker goal used in the illustrating example from the health-care domain. Here, the attacker strives to reach a state in which an action is enabled that should not be executed. This attacker goal could also be formulated in our goal language from Section IV. We indicate that although, in general, one may use success criteria that are not definable in our framework, the success criteria in the illustrating example are definable by the entries "- (PE1)" for Persistence and "- (C4)" for Causality in Figure 3. No other actions may occur in between the actions specified by the attack tree (i.e., Option PU3 for Purity).

Audinot et al. [7] propose an approach to guide security analysts in the stepwise refinement of attack trees. They provide a trace-based semantics that is similar in spirit to our semantics of attack trees in Section III. A technical difference is that they support a form of true concurrency in their construction. They use labeled transition systems to specify the behavior of a system and model runs by sequences of states, which they call "paths". Their terminology differs and relates to ours as follows: their "traces" correspond to our "attacks", their "paths" correspond to our "traces", and them calling a "path π a τ-attack on a system $\mathcal{S}$" corresponds in our terminology to some attack specified by the attack tree τ being successful.

In their setting, goals may only appear in leaf nodes. When a tree consists only of a single node, then the root is also a leaf and specifies the attacker's goal. Whenever a node is refined using AND, SAND, OR, this node loses its goal annotation,

but each leaf that is added specifies a goal. At all stages of the refinement process, each goal specifies the reachability of a state with certain properties (i.e., Option PE1 for Persistence).

The fourth column in our figure is not applicable (indicated by "$\emptyset$") as leaf nodes do not specify actions. Their traces (in our terminology: attacks) are words over a set of propositions. A state matches a leaf node if the goal of the leaf node is entailed by the propositions satisfied by the state. They define a notion of embedding that requires that the states in a path entail the goals in a trace in a point-wise fashion. Their embeddings nevertheless corresponds to Option PU1 for Purity (and not to Option PU3). This is due to a technical trick in their construction of the semantics of attack trees, in which they permit the satisfaction of an arbitrary number of dummy goals $\top^*$ before the satisfaction of any goal specified by a leaf node. After the goal specified by a leaf node, they do not insert any dummy goals (i.e., Option C4 for Causality).

B. Lessons Learned from the Application of Our Framework

We showed how to specify with our framework the success criteria that underly selected prior publications. There are multiple lessons learned from this application of our framework:

Firstly, and for us very surprisingly, it turned out to be much more difficult than expected to develop from the publications a sufficiently detailed understanding of the respectively underlying success criteria such that we could formalize these criteria in a faithful manner. We had expected our distinction of Purity, Persistence, and Causality to reduce the conceptual complexity of the task to a level that would have allowed us to proceed much more quickly. Nevertheless, using Purity, Persistence, and Causality was helpful for deepening our understanding in a stepwise fashion and for checking the (non-)faithfulness of candidate formalizations of the success criteria modularly.

Secondly, we encountered publications on attack trees that we could not cover in this section, because they did not describe any success criterion (and for which the success criterion also did not become clear from the context). For instance, a recent publication on the correctness of attack trees [4] proposes criteria for the admissibility and the correctness of attack trees, but does so without introducing a notion of "attack" (and, hence, naturally also without a success criterion). While the concept of "attacks" was simply not needed to derive the theoretical results in this publication (i.e., from a formal methods perspective), the omission of such a key concept of threat modeling might make it hard for security analysts to benefit from the insights of the publication (i.e., from a security perspective).[4]

Thirdly, in the representation of success criteria, only Option C4 was used for Causality, meaning that the attacker's goal must be satisfied directly after the last action of an attack occurred. Our study of prior work might even suggest that there is a general acceptance for using Option C4 for Causality. We will get back to this point in Section VII.

It is also interesting to observe that Causality is a degree of freedom that could not be resolved for the oldest publication we covered, i.e., [5], due to the use of a semantics of attack trees that does not define any order for occurrences of attacker actions. However, the ordering of attacker actions might be vital for the success of an attack. All of the newer publications use semantics where the ordering matters.

Fourthly, there is heterogeneity within the column for Purity in Figure 3, but only Options PU1 and PU3 are used.

As we mentioned in Section V, the use of Option PU1 for Purity can be too liberal, but let us elaborate this further now where we have encountered this option in prior work. Consider, for instance, the following classical bank robbery attack:

enter(Eva,bank), draw(Eva,pistol),
take(Eva,1M$), leave(Eva,bank) ,

where Eva's goal is to obtain 1M$ from the bank. Recall that Option PU1 for Purity considers this attack to be present in a trace even if other actions (including attacker actions) occur in between the actions of the attack. This means that the above bank robbery attack is considered present in the following trace

s_0, enter(Eva,bank), leave(Eva,bank), draw(Eva,pistol),
store(Eva,pistol), enter(Eva,bank), apply(Eva,loan(1M)),
take(Eva,1M), leave(Eva,bank), s_f

After the last action of the attack occurred, Eva has 1M$, i.e., the attacker goal is satisfied in state s_f. The success criterion $\mathsf{T}(1,1,4)$ classifies this occurrence of the bank robbery attack in the trace as successful wrt. Eva's goal. Intuitively, this makes little sense, because Eva owes 1M$ of debts to the bank in s_f. This illustrates the mismatches to one's intuition that can occur if Option PU1 is chosen for Purity (as in [7] and [5]).[5]

Fifthly, there is also heterogeneity in the last column of Figure 3, where the entries are A_{Att}, $?(A_{Att})$, and A. Regarding the actions that may occur in an attack tree, [13] takes a rather non-conventional viewpoint by permitting arbitrary actions to occur in an attack tree. This raises the question whether the attack trees generated by this approach are as suitable for understanding the threats to a system from an attacker's perspective as traditional attack trees are. We will get back to this point in Section VII.

Noteworthy, but less interesting than the above observations is that, for Persistence, we only encountered Option PE1. None of the publications required the attacker goal to be an invariant from some point in a run.

Remark 5. *As explained, Audinot et. al. [7] use a technical trick in their construction of the semantics of attack trees. They permit the satisfaction of an arbitrary number of dummy goals $\top^*$ before the satisfaction of any goal specified by a leaf node. This corresponds to choosing Option PU1 for Purity.*

[4]In [4], sequences of states (which are referred to as *paths*) are used to give attack trees a meaning. Formally, paths differ from our attacks, which are sequences of actions (cf. Definition 2), and this difference has consequences. Given a path, a security analyst cannot easily infer which attacker actions contribute to the attacker's goal. In contrast, given an attack, a security analyst can directly see which attacker actions should be prevented. The two concepts complement each other. While attacks capture which actions an attacker needs to perform, paths capture the effects of the attacker's actions.

[5]See Remark 5 for how the success criterion of [7] could be modified.

Our bank-robbery example indicated that Option PU1 might be too liberal. Hence, one might want to move to Option PU2. This could be done by inserting a formula that is only fulfilled by system and environment actions instead of $\top$ in the definition of the semantics of attack trees.

Remark 5 is not meant as the development of fully-fledged variant of [7], but to illustrate a possibility. The implementation of this idea will likely require the addition of some technical machinery. Our main point here was, to sketch how the concepts introduced in this article can guide one in the migration from one success criterion to a more suitable one.

VII. Exploit Trees

When developing an attack tree, one should focus on the viewpoint of one attacker and on a single attacker goal. Both contributes to simplifying the threat modelling task. Such a simplification is desirable because an increase of conceptual complexity not only increases the effort but also the likelihood of omissions and other mistakes during threat modeling.

To keep the focus on the attacker's viewpoint, only attacker actions should be used as annotations of leaves in an attack tree. Hence, we criticized the use of arbitrary actions by Gadyatskaya et al. [13] from the perspective of threat modeling in Section VI. From the perspective of a reachability analysis, however, their viewpoint makes perfect sense. They search for sequences of actions that lead to a state that satisfies an attacker goal, and this might very well be combinations of actions by the attacker, by the system, and by other, benign actors in the system's environment. Using trees as a graphical notation to present such sequences can be helpful. We only prefer to give such trees a different name than attack trees, namely *exploit trees*. Moreover, we use the term *exploit* for sequences of actions by the attacker, by the system, and by benign actors in system's environment. We capture the semantics of exploit trees in terms of such exploits.

The formal definitions of exploits and exploit trees are analogous to the ones for attacks and attack trees, the only difference being the use of a larger set of actions. Consequently, our observations about criteria for the success of an attack in a run wrt. an attacker goal are directly applicable to exploits. Note also, that the use of Option C4 for Causality in combination with exploits offers one the possibility to precisely specify when the attacker's goal shall be satisfied, which need not be after an attacker action. This opens up interesting possibilities.

A. Syntax and Semantics

Definition 14. *The* language of exploit trees *over a set of attacker actions A_{Att}, a set of system actions A_{Sys}, a set of environment actions A_{Env}, and a set of attacker goals G is*

$$\mathcal{ET}_{A,G} = \{\mathsf{ROOT}(g\colon et) \mid g \in G \wedge et \in \mathcal{ET}'_{A,G}\}$$

where $A = A_{Att} \cup A_{Sys} \cup A_{Env}$ is the set of all actions, and $\mathcal{ET}'_{A,G}$ is the least set of expressions satisfying:

1) $\mathsf{ACT}(a) \in \mathcal{ET}'_{A,G}$ *for all* $a \in A$ *and*

2) if $et_1, \ldots, et_k \in \mathcal{ET}'_{A,G}$ *and* $g \in G$ *then*
$\mathsf{OR}(et_1, \ldots, et_k),\ \mathsf{OR}(g\colon et_1, \ldots, et_k) \in \mathcal{ET}'_{A,G}$,
$\mathsf{AND}(et_1, \ldots, et_k),\ \mathsf{AND}(g\colon et_1, \ldots, et_k) \in \mathcal{ET}'_{A,G}$, *and*
$\mathsf{SAND}(et_1, \ldots, et_k),\ \mathsf{SAND}(g\colon et_1, \ldots, et_k) \in \mathcal{ET}'_{A,G}$.

Definition 15. *An* exploit *over a set of attacker actions A_{Att}, a set of system actions A_{Sys}, and a set of environment actions A_{Env} is a non-empty sequence $att \in (SEQ^{\text{fin}}(A)\backslash\{\langle\rangle\})$, where $A = A_{Att} \cup A_{Sys} \cup A_{Env}$ is the set of all actions.*

The formal semantics $[\![et]\!]$ of an exploit tree *et* is a set of exploits (instead of a set of attacks). Otherwise, the definition of the formal semantics of exploit trees follows the same lines as the formal semantics of attack trees (cf. Definition 3). We, hence, refrain from presenting it here in detail.

B. Illustrating Example

The attack tree in Figure 1 specifies only actions of the attacker. However, for an attack to be successful other actions have to occur. For instance, the camera can only be used to observe the pin when the customer uses the ATM where the camera is installed. Similarly, copying the banking card is only possible if the customer puts the card into the reader that the attacker installed. Adding these actions of the customer to the attack tree leads to the exploit tree visualized in Figure 4.

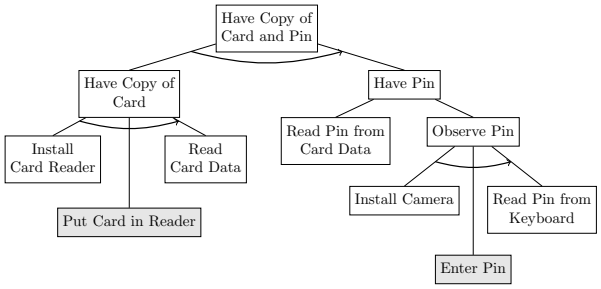

Fig. 4. Exploit tree showing actions by the card owner (light grey boxes) and the attacker (white boxes) for a sub-tree of the running example

VIII. Related Work

In Section V, we proposed a framework for defining criteria for when an attack is successful wrt. an attacker goal in a system run. To our knowledge, the fact that there are multiple sensible definitions of such success criteria has not been clarified before.

The first formal semantics for attack trees was proposed by Mauw and Oostdijk [8]. Their semantics gives a meaning to attack trees in terms of sets of multisets of attacker actions (cf. Section I). More recent semantics in this tradition include [2, 9, 10]. All of these semantics focus on the attacks that are represented by an attack tree or an attack-defense tree without establishing a connection to the attacker's goal. As explained in Remark 1, our semantics of attack trees defined in Section III is similar in spirit to the SP-semantics in [2].

Recently, Audinot, Pinchinat, and Kordy developed criteria for the admissibility and correctness of attack trees [4] using a

declarative approach. They use a syntax of attack trees, where the annotations of nodes are specifications consisting of a pre- and a postcondition. This uniformity allows them to formulate their definitions and to derive their results in a rather elegant fashion. They capture the effects of attacks using sequences of states. Attacker actions and attacks in the sense of our article are treated implicitly except for in concrete examples. As explained in Section VI-B, we therefore did not cover this approach in the application of our framework in Section VI-A.

In a later article [7], the same authors developed a technique for guiding security analysts in the stepwise refinement of attack trees. The syntax of attack trees in this article differs from the one in their earlier article. Again, they do not permit actions as annotations of nodes. Instead of using pre-/post-condition pairs as annotations, they use reachability goals in the new article. We clarified in Section VI how the success criterion from this article can be captured in our framework.

For automatically generating attacks, one can employ a reachability analysis to find out how a state can be reached that satisfies a given attacker goal. Such approaches to generating attacks inherently need a success criterion to determine which attacks achieve the attacker goal (as outlined in Section I).

For instance, Dimkov et al. [15] propose an approach to generate so called attack scenarios in which an attacker combines digital, physical, and social means to achieve her goals. Attack scenarios are expressions that specify a sequential composition of actions. In their approach, attack scenarios are generated in three phases. In the first phase, pre- and post-conditions for each relevant action are determined, and the earliest iteration is inferred when the execution of the action is possible. This results in a set of so called action templates. In the second phase, a backwards search from the attacker's goal is performed to determine action templates that contribute to reaching this goal. In the third and final phase, the set of action templates is completed to reach the attacker's goal.

Since [15] does not discuss how attack trees should be computed from attack scenarios, we did not cover it in Section VI. Nevertheless, the assumptions underlying this approach can be categorized using the success criteria from our framework. Dimkov et al. assume that the actions specified by an attack scenario occur subsequently without any occurrences of other actions in between (which corresponds to Option PU3 for Purity). They require the attacker goal to be satisfied by the state after the last action of the attack scenario occurred (which corresponds to Option C4 for Causality) and they do not consider what happens after the attacker goal has been achieved (which corresponds to Option PE1 for Persistence). The computed attack scenarios may contain attacker actions as well as other actions (which means that these attack scenarios correspond to exploits rather than to attacks in our terminology).

Reachability analyses have also been used to automatically generate attack trees. However, publications proposing such attack-tree generation techniques tend to leave the underlying success criteria implicit, or to describe them only informally or in semi-formal terms. This leaves room for interpretation and can become the source of mistakes. We clarified for four such attack-tree generation techniques (i.e., [5, 6, 13, 14]) in Section VI how the respectively underlying success criteria can be expressed in our framework.

Kumar and Stoelinga [16] and recently André *et al.* [17] introduced attack-fault trees to assess safety and security aspects of cyber-physical systems. The focus of this work is different from our approach in that they target quantitative analyses of the resulting trees to identify values of properties of interest or the effect of countermeasures, or to compare effects of design alternatives on a system's safety and security. The actions in attack-fault trees can be attacker actions or failures of basic components, which occur in the system being modelled. In this sense, attack-fault trees are similar to exploit trees.

Many other researchers work on quantitative analyses for attack trees. Aslanyan et al. [18] proposed an extension of attack-defense trees in which temporal dependencies among subgoals can be expressed, and encode these as stochastic two-player games to verify formally specified security properties and to synthesize strategies for attackers or defenders that guarantee or optimize some quantitative property. In earlier work, Aslanyan and Nielson [19] also evaluate the Pareto efficiency for trees with multiple conflicting parameters. Hermanns et al. [20] explore how stochastic timed automata can be used to study attack-defense scenarios where timing plays a central role, similar to David et al. [21]. Buldas and Lenin [22] explore the games as modelling the utility of attackers in the presence of defenses. Kordy et al. [23] combine trees with Bayesian networks to identify probabilistic measures of attack-defense trees with dependent actions.

IX. Conclusion

Attack trees have been introduced by Schneier [1] as a pragmatic notation for describing threats to systems. Due to their universal applicability, numerous refinements and enhancements of attack trees have been suggested, some with and some without semantics. Other approaches consider the automatic generation of attack trees from system models.

In this article, we aimed at better clarifying the relationship between the attacks and the attacker goal specified by an attack tree. We identified "*Does an attack achieve an attacker goal in a run?*" as the key question in this respect. We argued that there is spectrum of sensible answers rather than one answer that suits all purposes. This observation has been confirmed by our study in Section VI. Our clarification of success criteria used in prior publications revealed differences between the criteria that have been used so far.

The framework for defining success criteria that we proposed in Section V can express also success criteria that, to our knowledge, have not been used in publications so far. We have sketched at one example, how our framework could inspire modifications in the work of others (cf. Remark 5). We have also outlined possible uses for precisely defined success criteria (cf. the beginning of Section VI). They can serve as reference points for the evaluation of attack trees, as a reference point

of a correctness proof for attack-tree generation and attack-tree transformation techniques, and also serve as a basis for comparisons of such techniques.

Why the connection between the attacks and the attacker goal specified by an attack tree has not been studied before? This still remains a mystery to us, after having written this article. We hope that our clarification of the connection between the meaning (i.e., the attacks) and the purpose (i.e., the attacker goals) of attack trees is useful for others. We needed the clarification ourselves in the context of other research questions involving attack trees that we are pursuing.

Acknowledgments

We thank the anonymous reviewers for their helpful detailed comments and Barbara Sprick for feedback and discussions.

References

[1] B. Schneier, "Attack trees: Modeling security threats," *Dr. Dobbs Journal*, December 1999.

[2] R. Jhawar, B. Kordy, S. Mauw, S. Radomirović, and R. Trujillo-Rasua, "Attack trees with sequential conjunction," in *ICT Systems Security and Privacy Protection*. Springer, 2015, pp. 339–353.

[3] B. Kordy, S. Mauw, S. Radomirovic, and P. Schweitzer, "Attack-defense trees," *Journal of Logic and Computation*, vol. 24, no. 1, pp. 55–87, 2014.

[4] M. Audinot, S. Pinchinat, and B. Kordy, "Is my attack tree correct?" in *Computer Security - ESORICS 2017 - 22nd European Symposium on Research in Computer Security, Part I*, 2017, pp. 83–102.

[5] R. Vigo, F. Nielson, and H. R. Nielson, "Automated generation of attack trees," in *IEEE 27th Computer Security Foundations Symposium, CSF*, 2014, pp. 337–350.

[6] M. Ivanova, C. Probst, R. Hansen, and F. Kammüller, "Attack tree generation by policy invalidation," in *Information Security Theory and Practice*. Springer, 2015, pp. 249–259.

[7] M. Audinot, S. Pinchinat, and B. Kordy, "Guided design of attack trees: A system-based approach," in *31st IEEE Computer Security Foundations Symposium, CSF*, 2018, pp. 61–75.

[8] S. Mauw and M. Oostdijk, "Foundations of attack trees," in *Information Security and Cryptology - ICISC 2005, 8th International Conference, Revised Selected Papers*, 2005, pp. 186–198.

[9] B. Kordy, S. Mauw, S. Radomirovic, and P. Schweitzer, "Foundations of attack-defense trees," in *Formal Aspects of Security and Trust - 7th International Workshop, FAST 2010, Pisa, Italy, September 16-17, 2010. Revised Selected Papers*, 2010, pp. 80–95.

[10] R. Horne, S. Mauw, and A. Tiu, "Semantics for specialising attack trees based on linear logic," *Fundamenta Informaticae*, vol. 153, no. 1–2, pp. 57–86, 2017.

[11] O. Sheyner, J. W. Haines, S. Jha, R. Lippmann, and J. M. Wing, "Automated generation and analysis of attack graphs," in *IEEE Symposium on Security and Privacy*, 2002, pp. 273–284.

[12] M. Fraile, M. Ford, O. Gadyatskaya, R. Kumar, M. Stoelinga, and R. Trujillo-Rasua, "Using attack-defense trees to analyze threats and countermeasures in an atm: A case study," in *9th IFIP WG 8.1 Working Conference on The Practice of Enterprise Modeling, PoEM*, vol. 267, 2016, pp. 326–334.

[13] O. Gadyatskaya, R. Jhawar, S. Mauw, R. Trujillo-Rasua, and T. A. C. Willemse, "Refinement-aware generation of attack trees," in *Security and Trust Management - 13th International Workshop, STM*, 2017, pp. 164–179.

[14] F. Kammüller, "A proof calculus for attack trees in isabelle," in *Data Privacy Management, Cryptocurrencies and Blockchain Technology - ESORICS International Workshops, DPM and CBT*, 2017, pp. 3–18.

[15] T. Dimkov, W. Pieters, and P. H. Hartel, "Portunes: Representing attack scenarios spanning through the physical, digital and social domain," in *Automated Reasoning for Security Protocol Analysis and Issues in the Theory of Security - Joint Workshop, ARSPA-WITS, Revised Selected Papers*. Springer, 2010, pp. 112–129.

[16] R. Kumar and M. Stoelinga, "Quantitative security and safety analysis with attack-fault trees," in *18th IEEE International Symposium on High Assurance Systems Engineering, HASE*, 2017, pp. 25–32.

[17] É. André, D. Lime, M. Ramparison, and M. Stoelinga, "Parametric analyses of attack-fault trees," in *19th International Conference on Application of Concurrency to System Design, ACSD*, 2019, to appear.

[18] Z. Aslanyan, F. Nielson, and D. Parker, "Quantitative verification and synthesis of attack-defence scenarios," in *29th IEEE Computer Security Foundations Symposium, CSF*. IEEE Computer Society, 2016, pp. 105–119.

[19] Z. Aslanyan and F. Nielson, "Pareto efficient solutions of attack-defence trees," in *Principles of Security and Trust - 4th International Conference, POST*. Springer, 2015, pp. 95–114.

[20] H. Hermanns, J. Krämer, J. Krčál, and M. Stoelinga, "The value of attack-defence diagrams," in *Proceedings of the 5th International Conference on Principles of Security and Trust*. Springer, 2016, pp. 163–185.

[21] N. David, A. David, R. R. Hansen, K. G. Larsen, A. Legay, M. C. Olesen, and C. W. Probst, "Modelling social-technical attacks with timed automata," in *7th ACM CCS International Workshop on Managing Insider Security Threats, MIST*, 2015, pp. 21–28.

[22] A. Buldas and A. Lenin, "New efficient utility upper bounds for the fully adaptive model of attack trees," in *Decision and Game Theory for Security - 4th International Conference, GameSec*, 2013, pp. 192–205.

[23] B. Kordy, M. Pouly, and P. Schweitzer, "A probabilistic framework for security scenarios with dependent actions," in *Integrated Formal Methods - 11th International Conference, IFM*. Springer, 2014, pp. 256–271.

APPENDIX A
LABELED TRANSITION SYSTEMS AND TRACES

Labeled transition systems can be used to model the possible initial configurations of a scenario and the causal relationship between events and snapshots in a scenario.

Definition 16. *A* labelled transition system *(or short* LTS*) is a tuple* $Lts = (S, S^0, A, \rightarrow)$ *consisting of a non-empty set of states* S*, a non-empty set of initial states* $S^0 \subseteq S$*, a set of actions* A*, and a step relation* $\rightarrow \subseteq S \times A \times S$*.*

Intuitively $(s, a, s') \in \rightarrow$ means that the event modeled by action a is enabled in the snapshot modeled by state s, and that s' models a snapshot that might result after the event occurred.

A labeled transition system *faithfully captures* a given scenario if each initial configuration possible in the scenario is faithfully modeled by a state $s \in S^0$, and if each possible effect of an event in the scenario is modeled by a transition $(s, a, s') \in \rightarrow$. A labeled transition system *precisely captures* a given scenario if each state $s \in S^0$ faithfully models a possible initial configuration, and if each transition $(s, a, s') \in \rightarrow$ faithfully models possible effect of an event in the scenario.

The notions of traces from Definition 6 can be related to labelled transition systems as follows:

Definition 17. *Let* $Lts = (S, S^0, A, \rightarrow)$ *be an LTS.*

- *A* state trace $tr : \mathbb{N}_0 \rightharpoonup S$ is compatible with Lts *iff* $tr(0) \in S^0$ *and for all* $n \in def(tr)$ *holds*
  $(n+1) \in def(tr) \Rightarrow \exists a \in A\colon (tr(n), a, tr(n+1)) \in \rightarrow$.
- *A* verbose trace $tr\colon \mathbb{N}_0 \rightharpoonup (S \cup A)$ is compatible with Lts *iff for all* $n \in def(tr)$ *holds*
  $$(even(n) \wedge (n+1, n+2) \in def(tr)) \Rightarrow (tr(n), tr(n+1), tr(n+2)) \in \rightarrow .$$

Labeled transition systems are often used to capture the possible behaviors of a scenario. In this article, we used sets of traces for this purpose instead. The following definition bridges the gap between these concepts and illustrates how labeled transition systems can be translated into sets of traces.

Definition 18. *Let* $Lts = (S, S^0, A, \rightarrow)$.

- *The* set of state traces induced by Lts *is* $\{tr : \mathbb{N}_0 \rightarrow (S \cup A) \mid tr$ *is a state trace compatible with* $Lts\}$.
- *The* set of verbose traces induced by Lts *is* $\{tr : \mathbb{N}_0 \rightarrow (S \cup A) \mid tr$ *is a verbose trace compatible with* $Lts\}$.