Foundations of Attack Trees

Sjouke Mauw[1] and Martijn Oostdijk[2]

[1] Eindhoven University of Technology
sjouke@win.tue.nl
[2] Radboud University Nijmegen
martijno@cs.ru.nl

Abstract. Attack trees have found their way to practice because they have proved to be an intuitive aid in threat analysis. Despite, or perhaps thanks to, their apparent simplicity, they have not yet been provided with an unambiguous semantics. We argue that such a formal interpretation is indispensable to precisely understand how attack trees can be manipulated during construction and analysis. We provide a denotational semantics, based on a mapping to attack suites, which abstracts from the internal structure of an attack tree, we study transformations between attack trees, and we study the attribution and projection of an attack tree.

Keywords: attack trees, semantics, threat analysis.

1 Introduction

Attack trees (the term is introduced by Schneier in [10, 11]) form a convenient way to systematically categorize the different ways in which a system can be attacked. The graphical, structured tree notation is appealing to practitioners, yet also seems promising for tool builders attempting to partially automate the threat analysis process. As such, attack trees may turn out to be of interest to the security community at large as a standard notation for threat analysis documents.

An attack tree is a tree in which the nodes represent attacks. The root node of the tree is the global goal of an attacker. Children of a node are refinements of this goal, and leafs therefore represent attacks that can no longer be refined. A refinement can be conjunctive (aggregation) or disjunctive (choice). Figure 1 shows an example of an attack tree. In this tree, the goal of the attacker is to obtain a free lunch. The tree lists three possible ways to reach this goal. Lower levels in the tree explain how these sub-goals are refined as well. For instance, the "Eat-n-run" attack requires the attacker to order a meal and to leave the restaurant without paying. The arc connecting these two components expresses that this is a conjunctive refinement, which means that all sub-goals have to be fulfilled. Refinements without such a connecting arc are disjunctive, expressing that satisfying one sub-goal suffices.

Once the possible attacks on a system have been modeled in an attack tree, the tree can be used to analyze attributes of the security of the system. Schneier

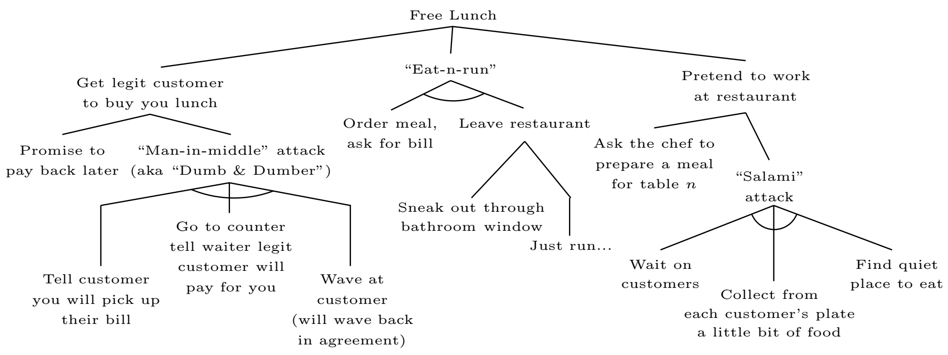
Free Lunch
Get legit customer to buy you lunch
"Eat-n-run"
Pretend to work at restaurant
Promise to pay back later
"Man-in-middle" attack (aka "Dumb & Dumber")
Order meal, ask for bill
Leave restaurant
Ask the chef to prepare a meal for table n
"Salami" attack
Tell customer you will pick up their bill
Go to counter tell waiter legit customer will pay for you
Wave at customer (will wave back in agreement)
Sneak out through bathroom window
Just run...
Wait on customers
Collect from each customer's plate a little bit of food
Find quiet place to eat

Fig. 1. Example attack tree

suggests several such attributes, for example the (im)possibility, the cost, and whether special tools are needed. The analysis proceeds in two steps: First, the value for each leaf node is determined. Second, the value in non-leaf nodes is synthesized from the value of its children. Thus, creativity on the part of the analyst is only needed in figuring out good values for the leaf nodes, as the rules for synthesis in both the conjunctive and the disjunctive case is usually determined by the nature of the attribute.

The result of an analysis can be the value of an attribute in the root node (for example the cost of the cheapest attack), but it could also be a sub-tree consisting of nodes adhering to some predicate (for example those attacks costing less than 100K Euro or those attacks that do not require use of special equipment). Also, values of different attributes can be defined (for example to determine the cheapest attack not using special equipment).

At a conceptual level attack trees are well understood and the above description is enough to work with them and even develop tool support. However, there are some questions that require a more fundamental answer: What is an attack? Is it just a collection of steps that should be performed or does it have some internal structure? Which conditions should an attribute satisfy before it allows to be synthesized bottom-up? Under what conditions may a projection of a predicate be executed bottom-up? When do two attack trees represent the same set of attacks? How should combined attributes be treated? And which extensions of the formalism (forests, directed acyclic graphs, attack graphs) are possible?

In order to be able to answer these questions, and in order to determine what computer aided threat analysis tools could look like, it is necessary to provide attack trees with foundations. Specifically, this paper provides attack trees with a semantics in terms of attack suites and defines valuations and projections in a formal way. Furthermore we show which algebraic conditions on attributes are sufficient to allow correct inference of values.

Related work. While the foundations of other sub-disciplines of computer security have had plenty of attention from formal methods researchers (e.g. access control [4], cryptographic protocols [7]), the sub-discipline of threat analysis has had little attention thus far. Attack trees are usually attributed to Schneier. They seem, however, to have a much longer tradition. Witnessed, for instance, by work on fault trees [16, 2]. Other research considers attack graphs [9, 13], in which event sequences are the central topic of research, rather than event abstractions. This seems to be an entirely different field with no cross-references to the kind of structures we study in the current paper. This field has its own set of tools (reachability analysis, etc.) In fact, the risk analysis community seems to have brought forth many such event based formalisms, for example cause-effect diagrams. Although mostly used to describe system-internal events, these can be applied to active external attacker scenarios as well. See for instance [3]. In [12] attack trees are compared to attack nets (a threat analysis formalism based on petri nets, described in [6]) with regard to sharing of security knowledge between collaborators. As far as collaborative attack modeling is concerned, the authors prefer attack nets over attack trees. Much of their criticism on attack trees seems to be based on a lack of semantics of the formalism. Tidwell et al. extend the attack tree formalism with parameters in [15], and successfully apply these trees to model multi-stage Internet attacks. The trees are used inside intrusion detection systems. A commercial tool [5] and some rudimentary tools [1, 8] are already available for Schneier's attack trees.

Outline. This paper is structured as follows. In Section 2 we introduce the notions of attack suites and attack trees and define a mapping from attack trees to attack suites, expressing the semantics of an attack tree. Section 3 provides an alternative characterization of the semantics through rewriting. This makes it possible to transform equivalent attack trees into each other. In Section 4 we define attributes on attack trees and discuss under which conditions they can be synthesized bottom-up. Finally, we consider how attacks can be singled out that satisfy some given property. Such projections are discussed in Section 5.

2 Attack Suites and Attack Trees

The purpose of an attack tree is to define and analyze possible attacks on a system in a structured way. This structure is expressed in the node hierarchy, allowing one to decompose an abstract attack or attack goal into a number of more concrete attacks or sub-goals. Although this structure carries information on the interpretation and grouping of attacks, we will discard it when determining the meaning of an attack tree. An attack tree simply defines a collection of possible attacks which we call an *attack suite*. Each attack consists of the components required to perform this attack. A component may occur more than once in an attack, so an attack is a multi-set of attack components. These attack components are at the lowest level of abstraction that we consider and thus have no internal structure. By describing an attack as a set of attack components we will also abstract from any causal relations between the components, such as being ordered in time.

First we introduce some common notation. We use $\mathcal{P}(V)$ to denote the power set of a set V and $\mathcal{P}^+(V)$ to denote the set of all non-empty subsets of V.

Likewise, we use $\mathcal{M}(V)$ and $\mathcal{M}^+(V)$ for multi-sets. The multi-set consisting of elements a, a and b is denoted by $\{\!|a, a, b|\!\}$. The difference of (multi-)sets V and W is denoted by $V \setminus W$. The substitution of element x for y in (multi-)set V is denoted by $V[x/y]$. The *distributed product* of two sets of multi-sets V and W is the set defined by $V \otimes W = \{v \uplus w \mid v \in V, w \in W\}$, where $\uplus$ denotes multi-set union. The operator $\bigotimes_{i\in I}$ is the generalization of $\otimes$ (with unit element $\{\emptyset\}$). The set of end nodes of a tree T is denoted by $E(T)$.

Definition 1. *Let $\mathbb{C}$ denote a set of attack components. An* attack *is a finite non-empty multi-set of $\mathbb{C}$ and an* attack suite *is a finite set of attacks. The universe of attacks is denoted by $\mathbb{A} = \mathcal{M}^+(\mathbb{C})$ and the universe of attack suites is denoted by $\mathbb{S} = \mathcal{P}(\mathbb{A})$.*

Example 1. If we have $\mathbb{C} = \{$*open door*, *steal key*, *force lock*, *pick lock*$\}$ then the following attack suite defines three ways to illegally enter a building $\{\{\!|$*steal key*, *open door*$|\!\}, \{\!|$*force lock*, *open door*$|\!\}, \{\!|$*pick lock*, *open door*$|\!\}\}$.

Attack trees as defined by Schneier have two types of nodes: *and-nodes* and *or-nodes*. The children of an and-node should all be executed to reach the goal represented by the and-node, while execution of any child of an or-node suffices to reach the goal of the or-node. By considering only one type of nodes, we will follow a slightly different, but equivalent, approach. Rather than considering edges from a node to its children, we consider connections from a node to a multi-set of nodes. Such a connection is called a *bundle*. A node may contain several such bundles. The nodes in a bundle must all be executed to form an attack. Execution of any bundle of a node will suffice to reach the goal of that node. Our approach differs in one more aspect from Schneier's attack trees. We allow sharing of nodes as a means to express that a sub-attack occurs more than once. Although a node may be contained in several bundles, we will not allow the construction of cycles. So, formally speaking, we study *rooted directed acyclic bundle graphs*, but we will still call them *attack trees*.

Definition 2. *An* attack tree *is a 3-tuple $(N, \rightarrow, n_0)$, where N is a finite set of nodes, $\rightarrow$ is a finite acyclic relation of type $\rightarrow \subseteq N \times \mathcal{M}^+(N)$ and $n_0 \in N$ is the root node, such that every node in N is reachable from n_0. The universe of attack trees is denoted by $\mathbb{T}$.*

We will use the informal terminology "A is a bundle of m" to express that $m \rightarrow A$. Whenever we speak of attack suites in the context of some attack tree T, we will identify the universe of attack components $\mathbb{C}$ with T's end nodes, $E(T)$.

Next, we define the semantics of attack trees by interpreting them in the domain of attack suites. As stated before, the internal branching structure of an attack tree will not be expressed in the attack suite. The semantics only expresses which combinations of attack components (i.e. end nodes of the tree) form an attack. The attack suite defined by a node in the tree can be determined recursively from its bundles. Since the bundles define alternative attacks, the attack suite defined by a node consists of the union of the attack suites defined by its bundles. The attack suite defined by a single bundle is determined by the attack suites of the nodes contained in the bundle. In order to construct an attack in a bundle, we must take an attack from each of the nodes in the bundle and join these together. The definition below formalizes this recursive construction. We use the distributed product, as defined above, to join the attacks within a bundle and we use the normal set union to join the bundles. The base case of the recursive definition considers the end nodes of the tree. They define an attack suite consisting of one attack which contains a single attack component.

Definition 3. *Let T be an attack tree $(N, \rightarrow, n_0)$, then the semantics of a node $[\![\_]\!] : N \rightarrow \mathbb{S}$ is defined recursively by*

$$[\![n]\!] = \begin{cases} \{\{\!|n|\!\}\} & \text{if } n \in E(T) \\ \bigcup_{n \rightarrow A} \bigotimes_{m \in A} [\![m]\!] & \text{if } n \notin E(T) \end{cases}$$

The semantics of an attack tree is defined as the semantics of its root node, $[\![T]\!] = [\![n_0]\!]$.

3 Transformations

Figure 2 illustrates that two structurally different attack trees may intuitively capture the same information. The difference in structuring can arise from a different approach towards partitioning the attacks. One may also want to simplify or rebalance an attack tree in order to change the view on the described attack suite, without changing its meaning. These two observations lead to the definition of semantics-preserving transformations of attack trees. The class of allowed transformations can be characterized by just two reduction rules. These rules are illustrated in Figure 3 and formalized in Definition 4. In the figure we consider a node which has a bundle A and possibly some other bundles (illustrated by W). Bundle A contains node m, which has a bundle B. We make a distinction between the two cases that B is the only bundle of m and that m has more bundles.

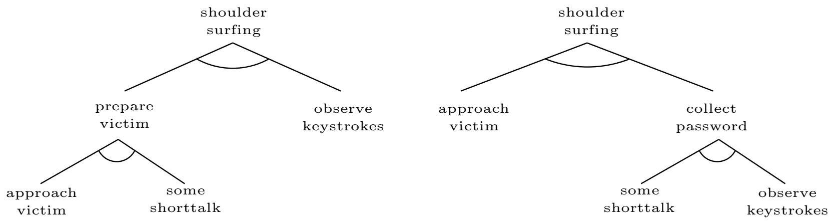

Fig. 2. Two equivalent trees

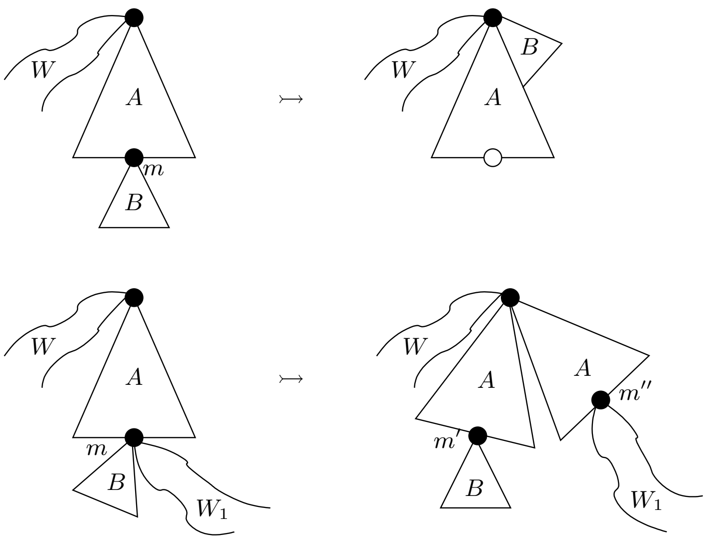

Fig. 3. Two reduction rules for attack trees (cf. Definition 4)

The first rule is based on the associativity of conjunction. If a bundle contains a node with only one sub-bundle, then this node can be deleted and its sub-bundle can be lifted one level, so as to become part of the bundle. Intuitively, this captures the fact that if we want to perform an attack that contains one sub-attack, we can simply take the components of the sub-attack and add them to the attack.

The second rule is based on the distributivity of conjunction over disjunction. If a bundle contains a node with two (or more) sub-bundles, then we can replace the bundle by two copies with the difference that the first copy only contains the first sub-bundle and the second copy only contains the second sub-bundle. Intuitively, this captures the fact that if we want to perform an attack that contains a sub-attack which can be performed in two ways, we have actually described two attacks. Please notice that the duplication of attack A in the figure is displayed in a somewhat misleading way. The picture suggests that the nodes in bundle A are also copied, but it is our intention that common nodes are shared, without duplicating nodes.

The next definition of an Abstract Reduction System on attack trees formally captures this intuition. We use the function reachable to remove all nodes and bundles that through rewriting become unreachable from the root node. The definition of this function is straightforward and will be omitted.

Definition 4. *The reduction relation* $\rightarrowtail : \mathbb{T} \rightarrow \mathbb{T}$ *is defined by the following two reduction rules. Let* T *be an attack tree* $(N, \rightarrow, n_0)$, $n, m \in N$ *and* $A, B \in \mathcal{M}^+(N)$ *such that* $n \rightarrow A$, $m \in A$, $m \rightarrow B$.

1. *if* B *is the only bundle in* m *(i.e.* $\forall_{C:m\rightarrow C} \cdot C = B$*), then*
   $(N, \rightarrow, n) \rightarrowtail \text{reachable}(N, \rightarrow', n)$*, where*

$$\to' = (\to \setminus \{(n, A)\}) \cup \{(n, (A \setminus \{m\}) \uplus B)\}$$

2. *if B is not the only bundle in m (i.e. $\exists_{C:m\to C} \cdot B \neq C$), then* $(N, \to, n) \rightarrowtail \text{reachable}(N', \to', n)$*, where*

$$\begin{aligned} \to' &= \big(\to \setminus \{(n, A)\}\big) \\ &\quad \cup \big\{(n, A[m'/m]), (n, A[m''/m]), (m', B)\big\} \\ &\quad \cup \{(m'', B') \mid m \to B', B \neq B'\}, \\ N' &= N \cup \{m', m''\} \quad \textit{where } m', m'' \notin N \end{aligned}$$

Because we are interested in transformations (without a preferred direction), we introduce the reflexive, transitive, symmetric closure of $\rightarrowtail$, denoted by $\equiv$.

Later we will prove that these reduction rules are *sound* with respect to the semantics. Interestingly enough, the rules turn out to be complete as well. Therefore, the reduction rules defined above are not only useful for manipulating attack trees; they also provide an alternative characterization of the semantics of an attack tree. It can be easily seen that the normal forms are the “one-level attack trees” which are in one-to-one correspondence to the universe of attack suites. In order to obtain these results, we first we have to show that the reduction rules are well-behaved, i.e. that there are no infinite reduction sequences and, roughly speaking, that any two diverging reductions can be reduced to a common attack tree.

Lemma 1. *The reduction relation on attack trees is strongly terminating.*

Proof. The proof is not given here because of space restrictions.

Lemma 2. *The reduction relation on attack trees is weakly confluent.*

Proof. This follows by observing that all critical pairs are convergent. (The rules are not overlapping anyway.)

As a standard corollary of the above lemmas we have the unique normal forms property (see e.g. [14]). The set of normal forms can be determined easily.

Lemma 3. *If we denote the set of attack trees with depth $\leq d$ by $\mathbb{T}_{\leq d}$, then the set of normal forms of $\rightarrowtail$ is $\mathbb{T}_{\leq 1}$.*

Proof. This follows easily by inspection of the left-hand sides of the reduction rules. An attack tree with depth ≥ 2 still has a redex and, conversely, to have a redex an attack tree must have depth ≥ 2.

Example 2. The tree presented in Figure 1 has a normal form consisting of six bundles, each of depth one, corresponding to the following attack suite. (The lengthy attack component descriptions of Figure 1 have been replaced by mnemonic names.)

$$\begin{aligned} \{\{&\textit{promise pay later}\}, \{\textit{tell victim}, \textit{go counter}, \textit{wave}\}, \{\textit{order}, \textit{bathroom}\}, \\ &\{\textit{order}, \textit{just run}\}, \{\textit{ask chef}\}, \{\textit{wait}, \textit{collect little}, \textit{quiet place}\}\} \end{aligned}$$

By verifying that the reduction rules preserve the semantics of an attack tree, we obtain their soundness. However, we will postpone the soundness proof until Section 4, where we will prove a more general soundness property of which this is an instantiation.

Theorem 1 (Soundness). *If $T_1 \equiv T_2$, then $[\![T_1]\!] = [\![T_2]\!]$.*

Proof. Postponed until Corollary 1.

The following lemma helps in proving completeness. It establishes the one-to-one correspondence between normal forms and attack suites.

Lemma 4. *Let T_1, $T_2 \in \mathbb{T}_{\leq 1}$ such that $[\![T_1]\!] = [\![T_2]\!]$ then $T_1 = T_2$.*

Proof. It follows easily from the definitions that an attack suite has a unique representation in $\mathbb{T}_{\leq 1}$. Every attack in the attack suite gives rise to a bundle from the root node.

Theorem 2 (Completeness). *If $[\![T_1]\!] = [\![T_2]\!]$, then $T_1 \equiv T_2$.*

Proof. If we have $[\![T_1]\!] = [\![T_2]\!]$, then we can reduce T_1 and T_2 to normal form, denoted by $\mathrm{nf}(T_1)$ and $\mathrm{nf}(T_2)$, and obtain

$$[\![\mathrm{nf}(T_1)]\!] = [\![T_1]\!] = [\![T_2]\!] = [\![\mathrm{nf}(T_2)]\!]$$

Now, Lemma 4 yields equality of the normal forms: $\mathrm{nf}(T_1) = \mathrm{nf}(T_2)$. So, $T_1 \equiv T_2$.

4 Attributes

In order to calculate e.g. the cost or impact of an attack, an attack tree can be decorated with attributes. The attribute value of an attack tree can be calculated by first determining the semantics of the tree followed by calculating the attribute value of the defined attack suite. However, under some conditions it is possible to synthesize the attribute value of the attack tree without first having to reduce the tree to normal form. This can be done by calculating the attribute values of the nodes in a bottom-up way.

Before defining attributes in an attack tree, we will first define the attribution of attack suites.

Given a set V of attribute values, an attribute α is a function that assigns a value to every attack component, $\alpha\colon \mathbb{C} \rightarrow V$. An attack consists of a number of attack components that must all be executed, so we assume that the attribution of an attack can be calculated from the attributions of its attack components. Therefore, we extend α to attacks, $\alpha\colon \mathcal{M}^+(\mathbb{C}) \rightarrow V$. In the same way, because an attack suite consists of a number of attacks, we extend attributions to attack suites, $\alpha : \mathcal{P}(\mathcal{M}^+(\mathbb{C})) \rightarrow V$. In order to calculate the attribution of an attack from the attributions of its attack components we use a *conjunctive combinator* $\triangle$, while for determining the value of an attack suite from its attacks we use a *disjunctive combinator* $\triangledown$. We require that the combinators are associative and commutative and that the conjunctive combinator distributes over the disjunctive combinator. These properties follow from the structure of attack trees.

Definition 5. *Let $\mathbb{C}$ be a set of attack components. An attribute domain is a structure $(V, \triangledown, \vartriangle)$ where V is the set of attribute values, $\triangledown : V \times V \to V$ is the disjunctive combinator for attribute values and $\vartriangle : V \times V \to V$ is the conjunctive combinator for attribute values. We require that the combinators are associative and commutative:*

$$\begin{gathered}(x \triangledown y) \triangledown z = x \triangledown (y \triangledown z)\\ x \triangledown y = y \triangledown x\\ (x \vartriangle y) \vartriangle z = x \vartriangle (y \vartriangle z)\\ x \vartriangle y = y \vartriangle x\end{gathered}$$

Given an attribute domain $(V, \triangledown, \vartriangle)$, an attribute α is a function from $\mathbb{C}$ to V. An attribute domain is distributive if the following property holds:

$$x \vartriangle (y \triangledown z) = (x \vartriangle y) \triangledown (x \vartriangle z)$$

Generalization of the combinators to $\mathcal{M}^+(V) \to V$ is defined in the usual way. An attribute domain is often called a semi-ring. However in published literature, the notion of a semi-ring often occurs with the additional requirement of a unit element for one or both operators. In the setting of attack trees this is not required, since we assumed that bundles are not empty.

Definition 6. *Let S be an attack suite and α an attribute with attribute domain $(V, \triangledown, \vartriangle)$ then the value attributed by α to S is*

$$\alpha(S) = \bigtriangledown_{A \in S} \bigtriangleup_{c \in A} \alpha(c)$$

Example 3. The structure $(\mathbb{N}, \min, +)$ is an example of a distributive attribute domain. An attribute with this attribute domain could be interpreted as "cost of the cheapest attack". Other examples: $(\mathbb{N}, \max, +)$ "maximal damage", $(\mathbb{N}, \min, \max)$ "minimum skill level required to perform attack", $(\mathbb{B}, \wedge, \vee)$ "is the attack possible", $(\mathbb{B}, \vee, \wedge)$ "special equipment needed".

It is interesting to see that there are also examples of attributes that do not satisfy the requirements. Consider, for instance, the structure $(\mathbb{N}, +, \min)$. This could express the costs to defend against all attacks from an attack suite: to defend against one attack one only has to find the cheapest defense against any of its attack components, and to defend against an attack suite, one has to add the defense costs to all its attacks. However, this structure is not a distributive attribute domain because it does not satisfy the required distribution property. At the end of this section we will come back to this counter example.

Now that we have defined the calculation of an attribute for attack suites, we will define the attribution of an attack tree.

Definition 7. *Let T be an attack tree $(N, \to, n_0)$ and let $\alpha : E(T) \to V$ be an attribute with a distributive attribute domain. Then we define the extension of α to the nodes of the attack tree, $\alpha : N \to V$, inductively by*

$$\alpha(n) = \bigtriangledown_{n \to A} \bigtriangleup_{m \in A} \alpha(m) \quad \textit{for } n \notin E(T).$$

The value of an attack tree is determined by the value of its root node: $\alpha(T) = \alpha(n_0)$.

This clearly defines a unique attribution of the attack tree. Moreover, the value attributed to an attack tree is respected by the rewrite rules.

Theorem 3. *Let* T_1 *and* T_2 *be attack trees and let* $\alpha\colon E(T) \to V$ *be an attribute with distributive attribute domain, then* $T_1 \equiv T_2$ *implies that* $\alpha(T_1) = \alpha(T_2)$.

Proof. The proof is not given here because of space restrictions.

It is notable that Definition 7 looks similar to Definition 3. In fact semantics can be seen as an attribute, since $(\mathcal{P}(\mathcal{M}^+(\mathbb{C})), \cup, \otimes)$ is a distributive attribute domain. Associativity and commutativity are standard, while $x \otimes (y \cup z) = (x \otimes y) \cup (x \otimes z)$ can be verified easily. Thus, the postponed soundness proof of Theorem 1 is a corollary of Theorem 3.

Corollary 1. *The reduction rules are sound with respect to the semantics, i.e. if* $T_1 \equiv T_2$*, then* $[\![T_1]\!] = [\![T_2]\!]$.

Finally, we observe that the value of a tree is equal to the value of the attack suite that is formed by taking the semantics of the tree.

Corollary 2. *Given attribute* α *and attack tree* T*, we have* $\alpha(T) = \alpha([\![T]\!])$.

Proof. From Theorem 3 it follows that $\alpha(T) = \alpha(\mathrm{nf}(T))$. The required equality now follows from the observed correspondence between normal forms and attack suites (Lemma 4), and by comparing Definitions 6 and 7.

Clearly, all reasonable attributes encountered in practice satisfy the requirements. However, as shown in Example 3, there are also attributes that make sense at first sight, but which are not consistent with the semantics. The inconsistency shows when comparing the value of the attribute calculated on the original tree to the value of the attribute calculated on the normal form of this tree. These values can differ if e.g. the law of distributivity does not hold. Thus we can conclude that there are attributes which cannot be synthesized bottom up. The counter example shows the usefulness of our formalization. We are now able to make the distinction between *sound* attributes and attributes that are inconsistent with the algorithms intuitively sketched by Schneier (which form the basis of our semantics).

5 Projections

By manipulating attack trees one can get answers to questions like "Show all attacks that do not require special equipment", or "Which attacks incur a damage over 1000 dollars"? The last question e.g. requires an attribute *incurred damage* and a predicate on its domain, $P(n) \equiv n \geq 1000$. Taking the projection of an attack suite boils down to selecting the attacks that satisfy the predicate.

Definition 8. *Let α be an attribute with distributive attribute domain $(V, \triangledown, \triangle)$ and let $P \subseteq V$ be a predicate. Then the projection of attack suite $S \in \mathbb{S}$ onto P is defined by $\Pi^{\alpha}_{P}(S) = \{A \in S \mid P(\alpha(A))\}$.*

The definition of projections on attack trees follows from the algorithm sketched by Schneier: remove all attacks that do not satisfy the predicate.

Definition 9. *Let α be an attribute with distributive attribute domain $(V, \triangledown, \triangle)$ and let $P \subseteq V$ be a predicate. Then the projection of attack tree $T = (N, \rightarrow, n_0)$ onto P is defined by $\Pi^{\alpha}_{P}(T) = \text{reachable}((N', \rightarrow', n_0))$, where $N' = \{n \in N \mid P(\alpha(n))\} \cup \{n_0\}$ and $\rightarrow' = \{(n, A) \in \rightarrow \mid P(\alpha(A))\}$.*

Again we want to know under which conditions these two definitions are consistent with each other. More precisely, we want to know if the projection of the semantics of an attack tree is equal to the semantics of the projection of that tree. Phrased in terms of the rewriting rules, we want to know if rewriting and projection commute.

A simple example shows that this does not hold in general. Consider the attribute domain $(\mathbb{N}, \min, +)$ to calculate the cost of an attack and look at the first tree of Figure 2. Suppose that all leafs have cost 5, which implies cost 10 for the intermediate node and cost 15 for the root. Now, if we take predicate $P(n) \equiv n \neq 10$, then for the projection of this tree we have to remove the intermediate node (and its dangling leaf nodes) which gives a tree with a single attack component. However, if we first reduce the tree to normal form, the projection would not affect the tree. Clearly, this cannot be a reduct of the first projected tree.

Monotonicity of predicate P suffices to prevent such problems. We say that predicate $P \subseteq V$ is monotonic with respect to attribute domain $(V, \triangledown, \triangle)$ iff

$$P(x \triangle y) \Rightarrow P(x) \wedge P(y)$$
$$P(x \triangledown y) \Rightarrow P(x) \vee P(y)$$

We first prove an auxiliary lemma, where we denote a sequence of zero or more reductions by $\rightarrowtail^*$. After that, we state the main result for projections.

Lemma 5. *Let α be an attribute with distributive attribute domain $(V, \triangledown, \triangle)$ and let $P \subseteq V$ be a monotonic predicate. If T and T' are attack trees such that $T \rightarrowtail T'$, then $\Pi^{\alpha}_{P}(T) \rightarrowtail^* \Pi^{\alpha}_{P}(T')$.*

Proof. The proof proceeds by inspecting the two rewrite rules, while looking at the possible values of P for the nodes of interest. If P is false in the top node, then the sub-trees are all removed and the lemma trivially holds. If P is true in the top node of the first redex, then monotonicity with respect to conjunction implies that the predicate is also true in all sub-nodes, making the same reduction step possible on the projected trees. If P is true in the top node of the second redex, then monotonicity with respect to conjunction and disjunction yields the same reasoning as for the first redex.

Theorem 4. *Let α be an attribute with distributive attribute domain $(V, \triangledown, \vartriangle)$ and let $P \subseteq V$ be a monotonic predicate. Then we have for $T \in \mathbb{T}$ that $[\![\Pi_P^\alpha(T)]\!] = \Pi_P^\alpha([\![T]\!])$.*

Proof. Due to the correspondence between attack suites and normal forms, it suffices to prove that $\mathrm{nf}(\Pi_P^\alpha(T)) = \Pi_P^\alpha(\mathrm{nf}(T))$. Now, suppose that T reduces to $\mathrm{nf}(T)$ via the sequence $T \rightarrowtail T' \rightarrowtail \ldots \rightarrowtail \mathrm{nf}(T)$. Then by applying Lemma 5 we obtain the reduction sequence $\Pi_P^\alpha(T) \rightarrowtail^* \Pi_P^\alpha(T') \rightarrowtail^* \ldots \rightarrowtail^* \Pi_P^\alpha(\mathrm{nf}(T))$. From the fact that $\mathrm{nf}(T)$ is a normal form, we can derive that $\Pi_P^\alpha(\mathrm{nf}(T))$ is a normal form as well. Because normal forms are unique, we have the desired property $\mathrm{nf}(\Pi_P^\alpha(T)) = \Pi_P^\alpha(\mathrm{nf}(T))$.

Summarizing, we have found that the projection algorithm informally presented by Schneier can only be applied to a restricted class of predicates. Monotonicity of predicate P suffices, but it is easy to see that monotonicity of $\neg P$ is also sufficient.

6 Conclusions

The main result of our work is a formalization of the concepts informally introduced by Schneier. This formalization clarifies which manipulations of attack trees are allowed under which conditions. Such an understanding is a prerequisite for building adequate tool support. A simple experiment with building a prototype tool confirmed the feasibility of our approach.

Central to our work is the observation that an attack tree describes an attack suite. We argue that the structural information that we lose in this way is a residual of the modeling strategy, rather than an intrinsic property of the described set of attacks. Therefore, attack suites form the appropriate level of abstraction. This semantics can be characterized in two ways: by traversing the tree from the leaves to the root and by rewriting the tree to normal form. Both strategies can be easily implemented, but in practice it is more interesting to build and manipulate attack trees than to calculate their semantics. Rewriting is more suited for this purpose. The rewrite rules can be used e.g. to add structure to an unstructured attack suite or to rebalance an attack tree.

It turns out that in order to recursively calculate attributes and projections, as introduced by Schneier, certain conditions have to be met. The condition for attributes (the combination operators form a semi-ring) is rather natural and no serious restriction. The condition for sound projections is somewhat stronger, but still satisfied by Schneier's examples. As mentioned before, our formalization motivates why certain attributes and predicates are not compatible with the informal algorithms presented by Schneier.

Having formalized the basic concepts of attack trees, it is of interest to study the extension with e.g. cycles, ordered attacks and pre- and post-conditions. Furthermore, our experience with using attack trees in practice indicated the need for *defense trees* and *attack forests* (i.e. attack libraries). Finally, we mention that although we have represented an attack as a multi-set, we could have used normal sets as well. The main difference would be the following extra requirement for attributes: $x \triangle x = x$.

Acknowledgments. We thank Alexander Opel, Leon Schrijvers and Martijn Wolfs for performing some initial studies with respect to implementation, use and analysis of attack trees. Hans Zantema is acknowledged for suggesting the norm used in the termination proof.

References

1. TANAT – Threat ANd Attack Tree Modeling plus Simulation, 2004. `http://www13.informatik.tu-muenchen.de:8080/tanat/`.
2. ARES corporation. Risk techniques, fault trees. Technical report. Available from `http://www.arescorporation.com/`.
3. Stefán Einarsson and Marvin Rausand. An approach to vulnerability analysis of complex industrial systems. *Risk Analysis*, 18(5):535 – 545, 1998.
4. Carl E. Landwehr. Formal models for computer security. *Computing Surveys*, 13(3):247 – 277, September 1981.
5. Amaneza Technologies Limited. A quick tour of attack tree based risk analysis using SecurITree. Technical report, 2002.
6. J.P. McDermott. Attack net penetration testing. In *Proc. 2000 workshop on New Security Paradigm*, pages 15–20. ACM, 2001.
7. Catherine Meadows. Open issues in formal methods for cryptographic protocol analysis. In *DARPA Information Survivability Conference & Exposition*, volume 1, pages 237 – 250, 2000.
8. Alexander Opel. Design and implementation of a support tool for attack trees, 2005.
9. Cynthia Phillips and Laura Painton Swiler. A graph-based system for network-vulnerability analysis. In *Proc. New Security Paradigms Workshop*, pages 71–79, 1998.
10. Bruce Schneier. Attack trees: Modeling security threats. *Dr. Dobb's journal*, December 1999.
11. Bruce Schneier. *Secrets & Lies: Digital Security in a Networked World.* Wiley, 2000.
12. Jan Stefan and Markus Schumacher. Collaborative attack modeling. In *Proc. SAC 2002*, pages 253–259. ACM, 2002.
13. L.P. Swiler, C. Philips, D. Ellis, and S. Chakarian. Computer-attack graph generation tool. In *Proc. DARPA Information Survivability Conference and Exposition*, volume 2, pages 307–321, June 2001.
14. Terese. *Term Rewriting Systems*, volume 55 of *Cambridge Tracts in Theoretical Computer Science.* Cambridge University Press, 2003.
15. T. Tidwell, R. Larson, K. Fitch, and J. Hale. Modeling internet attacks. In *Proc. of the 2001 IEEE Workshop on Information Assurance and Security*, 2001.
16. W.E. Vesely et al. Fault tree handbook. Technical Report NUREG-0492, U.S. Nuclear Regulatory Commission, January 1981.