Evaluation of Attributes on Attack–Defense Trees

Barbara Kordy, Patrick Schweitzer

Master Project 2010/2011

1 Attack–Defense Trees

Attack–defense trees (ADTrees) have been introduced in [1]. They constitute a natural extension of attack trees that were popularized by Bruce Schneier [3] and formalized by Sjouke Mauw and Martijn Oostdijk in [2].

> *ADTrees are an intuitive, graphical representation of possible actions that an attacker might take in order to attack a system and the countermeasures that a defender can employ to protect the system.*

Consequently, an ADTree has nodes of two opposite types: attack nodes and defense nodes. The root of an ADTree represents the main goal of the attacker. Every node of an ADTree may have one or more children of the same type representing a refinement into sub-goals of the node's goal. The refinement of a node can be either disjunctive or conjunctive. The goal of a disjunctively refined node is achieved when at least one of its refining children goals is achieved. The goal of a conjunctively refined node is achieved when all of its refining children goals are achieved. The nodes without any children of the same type are called non-refined nodes, and they represent basic attack or defense actions. Every node may also have one child of the opposite type, representing a countermeasure. Thus, an attack node may have several children which refine the attack and one child which defends against the attack. A defense node in turn may have several children which refine the corresponding defense and one child being an attack node and countering the defense.

To distinguish between attack nodes and defense nodes, we represent the former by red circles and the latter by green rectangles. Furthermore, every disjunctive node is labeled with the `or` operator, and every conjunctive node is labeled with the `and` operator. Additionally, an arc connects the edges going out from a conjunctive node to its refining children. Finally, the refinements are indicated by solid lines and the countermeasures by dotted lines.

1.1 Example

The ADTree depicted in Figure 1 describes a possible scenario of how to steal a lion. The main goal of an attacker — stealing a lion — is represented by the root 'Lion'. The red circle indicates that stealing a lion is an attack. In this example the 'Lion' node is labeled with the `or` operator. This means that in order to get a

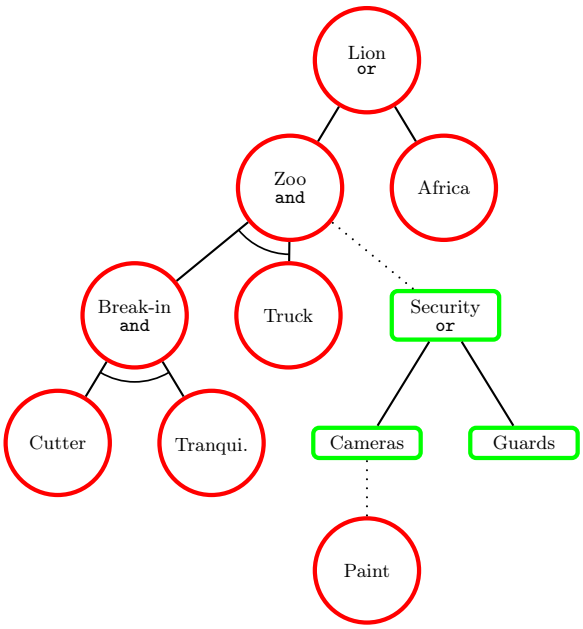

Fig. 1. An ADTree for stealing a lion

lion it is sufficient for the attacker to only fulfill one of the two possible sub-goals: get a lion from a zoo (attack node 'Zoo') or hunt a wild lion in Africa (attack node 'Africa'). In the scenario traveling to Africa is not considered further but instead the emphasis is placed on snatching a lion from a zoo. The zoo heist is refined into two sub-goals — the break-in and the get-away vehicle (the 'Truck' node) — that both have to be fulfilled. Thus the 'Zoo' node is labeled with `and`, and its refining children are connected with an arc. To successfully commit the break-in, a bolt cutter is needed to get into the cage and a tranquilizer gun for obvious reasons.

The scenario considered in the previous paragraph describes only attacks. However, in order to prevent a zoo heist, there are security measures in place. Hence, a defense node 'Security', depicted by a green rectangle, is attached to the 'Zoo' node. This defense node is connected to the 'Zoo' attack node via a dotted line. In our scenario several security measures are possible, so the 'Security' node is refined into: surveillance cameras (defense node 'Cameras') or guards (defense node 'Guards'). The defense node 'Security' is labeled with `or` to indicate that only one of the sub-defenses has to be present so that there is sufficient security to prevent the lion robbery from the zoo. Then, in turn, the attacker can attack those defenses. In order to disable the security measures, the attacker has to spray-paint the lenses of the cameras (attack node 'Paint').

2 Attributes for Attack–Defense Trees

To analyze an attack–defense scenario represented by an ADTree, we use attributes. An attribute is a value assigned to the ADTree, expressing a useful property, such as the minimal cost of an attack, the expected impact, whether special equipment is required, or whether a considered scenario is feasible. In [3], Schneier introduced an intuitive, bottom-up algorithm to calculate the value of an attribute on an attack tree. This idea was extended to ADTrees in [1]. To illustrate the concept of the bottom-up evaluation, let us consider the feasibility of the scenario from Subsection 1.1.

Example 1. In order to express whether a given component is feasible or not, we first assign Boolean values to all non-refined nodes in the tree. Let us assume that, on the one hand, it is feasible for the attacker to have a cutter, a tranquilizer and a truck, and that he can spray-paint the cameras. On the other hand, it is not possible for him to go to Africa. We also assume that both defensive measures — cameras and guards — are in place. Thus, we have

$$\begin{aligned}
&\mathrm{feas}(\mathit{Cutter}) = \top, && \mathrm{feas}(\mathit{Tranquilizer}) = \top, && \mathrm{feas}(\mathit{Truck}) = \top,\\
&\mathrm{feas}(\mathit{Paint}) = \top, && \mathrm{feas}(\mathit{Africa}) = \bot, && \mathrm{feas}(\mathit{Cameras}) = \top,\\
&\mathrm{feas}(\mathit{Guards}) = \top.
\end{aligned}$$

Now, we can compute the feasibility value for the entire scenario, by extending the feasibility function feas to subtrees. Let us denote by t_v the subtree rooted in node v. The feasibility function is then defined recursively, as follows.

- If v is a leaf, then $\mathrm{feas}(t_v) = \mathrm{feas}(v)$.
- If all the children $v_1, \ldots, v_k$ of v have the same type as v, then

$$\mathrm{feas}(t_v) = \begin{cases} \bigwedge_{i=1}^{k} \mathrm{feas}(t_{v_i}), & \text{if the label of } v \text{ is } \texttt{and},\\ \bigvee_{i=1}^{k} \mathrm{feas}(t_{v_i}), & \text{if the label of } v \text{ is } \texttt{or}. \end{cases}$$

- If w is the unique child of v, and the nodes w and v have opposite types, then

$$\mathrm{feas}(t_v) = \mathrm{feas}(v) \wedge \neg\, \mathrm{feas}(t_w).$$

- If v has k children $v_1, \ldots, v_k$ of the same type, and one child w of the opposite type, then

$$\mathrm{feas}(t_v) = \begin{cases} \bigwedge_{i=1}^{k} \mathrm{feas}(t_{v_i}) \wedge \neg\, \mathrm{feas}(t_w), & \text{if the label of } v \text{ is } \texttt{and},\\ \bigvee_{i=1}^{k} \mathrm{feas}(t_{v_i}) \wedge \neg\, \mathrm{feas}(t_w), & \text{if the label of } v \text{ is } \texttt{or}. \end{cases}$$

Thus, we obtain that a break-in sub-attack is feasible, since

$$\begin{aligned}
\mathrm{feas}(t_{\mathit{Break-in}}) &= \mathrm{feas}(t_{\mathit{Cutter}}) \wedge \mathrm{feas}(t_{\mathit{Tranquilizer}})\\
&= \mathrm{feas}(\mathit{Cutter}) \wedge \mathrm{feas}(\mathit{Tranquilizer}) = \top \wedge \top = \top.
\end{aligned}$$

Using a similar reasoning, we can deduce that a zoo sub-attack is not feasible:

$$\begin{aligned}\mathrm{feas}(t_{Zoo}) = &\mathrm{feas}(t_{Break-in}) \wedge \mathrm{feas}(t_{Truck}) \wedge \neg(t_{Security}) = \\ &\mathrm{feas}(Cutter) \wedge \mathrm{feas}(Tranquilizer) \wedge \mathrm{feas}(Truck) \\ &\wedge \neg[(\mathrm{feas}(Cameras) \wedge \neg\,\mathrm{feas}(Paint)) \vee \mathrm{feas}(Guards)] = \bot.\end{aligned}$$

Finally,

$$\mathrm{feas}(t_{Lion}) = \mathrm{feas}(t_{Zoo}) \vee \mathrm{feas}(t_{Africa}) = \bot \vee \bot = \bot,$$

which shows that stealing a lion according to our scenario is not feasible.

Example 1 presents a very simple attribute over a discrete domain. In reality, there are many other possible discrete as well as continuous attribute domains, including the probability of success of a given attack, the likelihood that an attacker will try a given attack, and so on.

3 Project Goals

The main purpose of the project is to make an exhaustive study of interesting attributes for attack–defense trees. The student's task will be to:

- determine interesting attributes and describe the corresponding attribute domains;
- define new semantics for ADTrees compatible with the introduced attribute domains;
- classify which attributes can and which can not be evaluated using the bottom-up approach;
- propose an approach which allows the evaluation of attributes that can not be computed using the bottom-up procedure;
- analyze and illustrate the results on an interesting case study.

References

1. Barbara Kordy, Sjouke Mauw, Saša Radomirović, and Patrick Schweitzer. Foundations of Attack–Defense Trees. In Joshua Guttman Pierpaolo Degano, Sandro Etalle, editor, *FAST*, LNCS. Springer, 2010. To appear.
2. Sjouke Mauw and Martijn Oostdijk. Foundations of Attack Trees. In Dongho Won and Seungjoo Kim, editors, *ICISC*, volume 3935 of *LNCS*, pages 186–198. Springer, 2005.
3. Bruce Schneier. Attack Trees. *Dr. Dobb's Journal of Software Tools*, 24(12):21–29, 1999.