# Practical security analysis: The distribution of attack trees

Lauri Rätsep

December 3, 2007

## 1 Introduction

Attack tree is a good structure to systematically estimate the practical security of a system by examining possible attacks. Every attack consists of some set of elementary attacks, which represent the leaves of an attack tree. The main target of an attacker is the root node. Every node's children are nodes that correspond to more detailed attacks.

There are two types of nodes in attack trees. The OR-node corresponds to an attack that is successful if it has at least one successful subattack (child). The AND-node corresponds to an attack that is successful only if all its subattacks (child nodes) are successful. The full semantics tree is an attack tree which is in disjunctive normal form. This means it has an OR root node that has AND childnodes, which consist of leafnodes.

Buldas et al. [1] have presented a risk-analysis method for analysing the security of institutions against rational attacks. Their method uses attack trees to estimate the risk by considering the cost, gains, probability, and penalties of an attack. But there exists a problem with this method. Sjouke Mauw and Martijn Oostdijk [3] have presented the reduction rules to convert an attack tree to the full sematics tree. It turns out that if those reductions are applied on trees of [1], then the calculations of a new tree sometimes differ from the original tree.

This paper introduces the calculation rules of the method [1] and explains why those calculations differ from the results of full semantics tree. It also simplifies the parameters of those calculations and gives some ideas to solve this problem.

## 2 Attack trees

### 2.1 Calculation rules

Buldas et al. [1] method uses elementary game theory to estimate the security of a system against gain-oriented attackers. They assume that the attacker is a rationally thinking person who chooses only the most profitable way of attacking. To compare the profits of the attacks, the adversary has to consider the following parameters:

- $\mathsf{Gains}$ - the gains of the attacker, in case the attack succeeds
- $\mathsf{Costs}$ - the cost of the attack
- $p$ - the success probability of the attack
- $\pi$ - the average penalty of an attacker if the attack was successful
- $\pi\_$ - the average penalty of an attacker if the attack was not successful

By using those parameters the attacker calculates the $\mathsf{Outcome}$ of an attack:

$$\mathsf{Outcome} = -\mathsf{Costs} + p \cdot (\mathsf{Gains} - \pi) - \overline{p} \cdot \pi\_ \tag{1}$$

where $\overline{p} = 1 - p$.

The rules for computing the parameters of non-leaf node, which children have the following parameters $(\mathsf{Costs}_i, p_i, \pi_i, \pi_i\_), i = 1, 2$:

- If we have an OR-node, then:

$$(\mathsf{Costs}, p, \pi, \pi\_) = \begin{cases} (\mathsf{Costs}_1, p_1, \pi_1, \pi_1\_), \text{if } \mathsf{Outcome}_1 > \mathsf{Outcome}_2 \\ (\mathsf{Costs}_2, p_2, \pi_2, \pi_2\_), \text{if } \mathsf{Outcome}_1 \leq \mathsf{Outcome}_2 \end{cases}, \tag{2}$$

  where $\mathsf{Outcome}_i = -\mathsf{Costs}_i + p_i \cdot (\mathsf{Gains}_i - \pi_i) - \overline{p}_i \cdot \pi_i\_, i = 1, 2.$

- If we have an AND-node, then:

$$\mathsf{Costs} = \mathsf{Costs}_1 + \mathsf{Costs}_2, \qquad p = p_1 \cdot p_2, \qquad \pi = \pi_1 + \pi_2$$

$$\pi\_ = \frac{p_1\overline{p_2}(\pi_1 + \pi_2\_) + \overline{p_1}p_2(\pi_1\_ + \pi_2\_) + \overline{p_1 p_2}(\pi_1\_ + \pi_2\_)}{1 - p_1 p_2} \tag{3}$$

Let's assume we have a part of an attack tree $T$, which has a root node $x$ that has two child nodes $m$ and $s$. Node $m$ has also two child nodes $t_1$ and $t_2$. Now, if we apply the reduction rules of Sjouke Mauw and Martijn Oostdijk [3] then we get a new tree $T'$ (Figure 1). The branches of AND-nodes are connected with two parallel lines to distinguish AND/OR nodes in this figure.

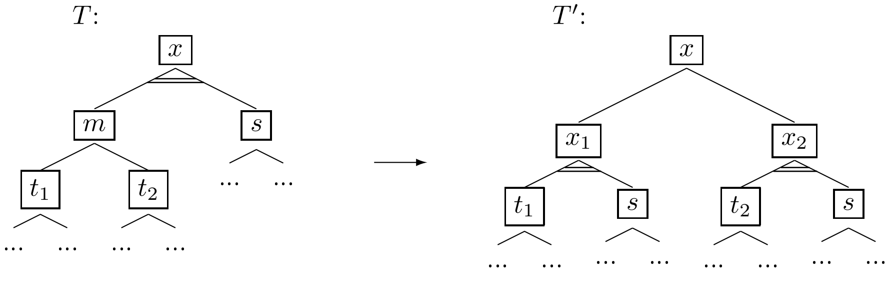

Figure 1: The reduction of an attacktree

## 2.2 The problem

Why those calculations (1,2,3) on a tree $T$ sometimes differ from the tree $T'$ (the example below describes this problematic situation)? Actually we want to know, if we can decide in the OR-node $m$ which child node to choose into the attack suite to get a maximum outcome. It is obvious that $\mathsf{Outcome}(x)$ cannot be larger than $\mathsf{Outcome}(s)$, because every new node in the attack suite causes extra costs and penalties which deduce the outcome of the root node $x$ according to the equation (1).

Let $t_y$ be the child node of $m$ that causes the maximum outcome of an attack suite. So we want to know how much the $\mathsf{Outcome}$ of $s$ will decrease:

$$\Delta\mathsf{Outcome}_{t_y}(x) = \mathsf{Outcome}_x - \mathsf{Outcome}_s =$$

$$= -C_{t_y} - C_s + p_{t_y} \cdot p_s \cdot \mathsf{Gains} - (-C_s + p_s \cdot \mathsf{Gains}) = -C_{t_y} - (1 - p_{t_y}) \cdot p_s \cdot \mathsf{Gains} \quad (4)$$

where $C_z$ is the sum of penalties and costs of a node $z$ ($z = \{t_y, s\}$):

$$C_z = \mathsf{Costs}_z + p_z \cdot \pi_z + (1 - p_z)\pi_{z\_} \quad (5)$$

Now we can see that if we want to decide, which child node to choose, we have to consider the probability $p_s$. So we cannot use the calculation rules of OR-node (2) on a tree $T$ to get correct results.

One possible solution for this problem is to store the list of all parameters of every child node of OR-node and make the choice later when this value $p_s$ is calculated. Since the $\Delta\mathsf{Outcome}$ is a linear function that depends on probability of $s$, we are able to compare those functions by $p_s$ values. When the child node's $\Delta\mathsf{Outcome}$ function is never greater from all of its neighbor nodes' functions on every valuation of $p_s$ then we could leave this node out of the list.

## 2.3 Example

Let $t_1 = (\$0; 0,9; \$0; \$120)$, $t_2 = (\$0; 0,1; \$0; \$1)$, $t_3 = (\$0; 0,5; \$0; \$10)$ and $s = (\$0; 0,5; \$0; \$0)$, $\mathsf{Gains} = \$20$ (Figure 2).

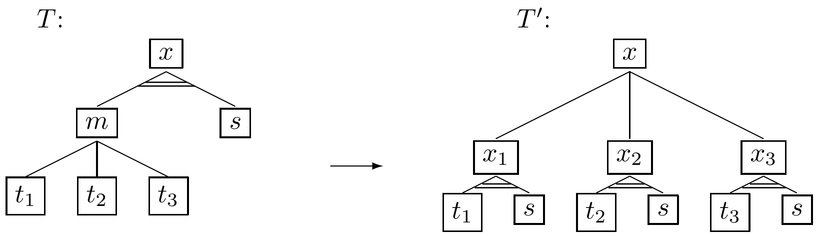

Figure 2: Example

The equation 1 gives $\mathsf{Outcome}_{t_1} = \$6$, $\mathsf{Outcome}_{t_2} = \$1.1$, $\mathsf{Outcome}_{t_3} = \$5$. So the outcome of tree $T$ will be $-\$3$ with attack suite $\{t_1, s\}$. But the outcome of tree $T'$ will be $\$0.1$ with attack suite $\{t_2, s\}$.

Now let's calculate those linear functions of $\Delta\mathsf{Outcome}$:

$$\Delta\mathsf{Out}_{t_1} = -12 - 2p_s$$

$$\Delta\mathsf{Out}_{t_2} = -0.9 - 18p_s$$

$$\Delta\mathsf{Out}_{t_3} = -5 - 10p_s$$

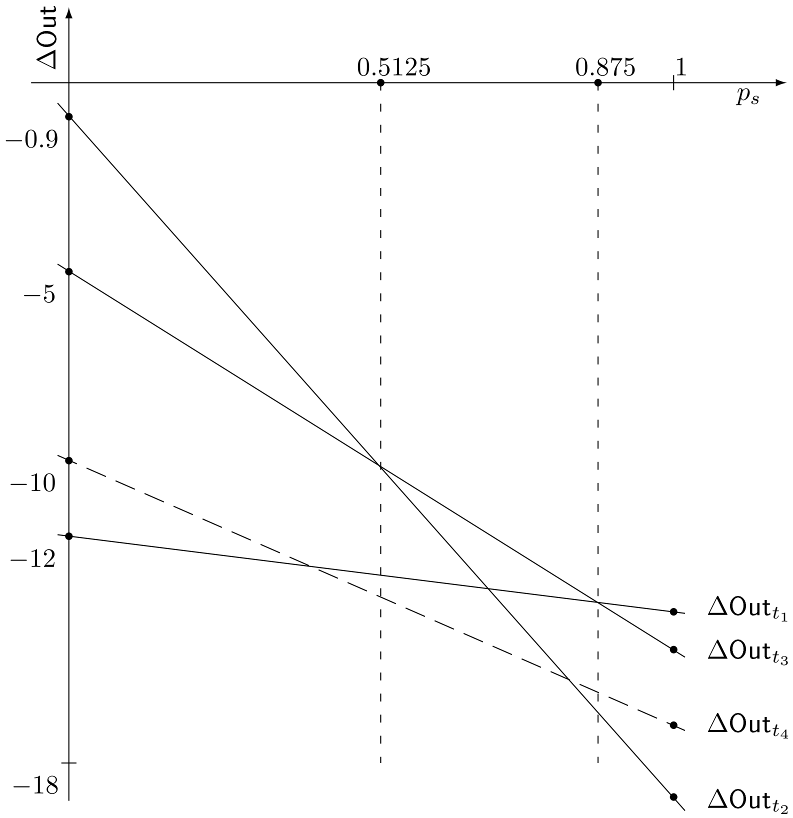

Figure 3: Linear functions of Outcome

According to the figure 3 if the probability of $s$ is less than 0.5125, then the node $t_2$ has to be taken to the attack suite. If $p_s$ is greater than 0.5125, but less than 0.875, then the node $t_3$ gives the best results and finally, if $p_s$ is larger than 0.875, then it is cheaper to use $p_1$. So in this case we have to add every child node of $m$ to the OR-node list, because all of them could belong to the best attack suite.

But if $m$ has a child node $t_4 = (\$10; 0, 7; \$0; \$0)$, then $C_{t_4} = -10$ , and the linear function will be:

$$\Delta\mathsf{Out}_{t_4} = -10 - 7p_s.$$

By using figure 3 it appears that function $\Delta\mathsf{Out}_{t_4}$ is never greater than any other neighbor function (on every $p_s$ valuation), so we can leave the node $t_4$ out from the list.

## 2.4 Improved calculation rules

### 2.4.1 The list method

This list method is quite similar to the idea of Peeter Laud [2], who has given two definitions to make the decision that the outcome function of a node $t_1$ is greater than the outcome function of a node $t_2$:

$(C_{t_1}, p_{t_1})$ *dominates* $(C_{t_2}, p_{t_2})$ if $C_{t_1} \leq C_{t_2}$ and $p_{t_1} \geq p_{t_2}$. It means that the starting point of $t_1$ function is not lower than the starting point of $t_2$ function, because $\Delta\mathsf{Out}_{t_1}(0) = -C_{t_1} \geq -C_{t_2} = \Delta\mathsf{Out}_{t_2}(0)$. Similary, with end points

$$p_{t_1} \geq p_{t_2} \Leftrightarrow -(1-p_{t_1}) \geq -(1-p_{t_2}) \Leftrightarrow$$

$$\Leftrightarrow -(1-p_{t_1}) \cdot \mathsf{Gains} \geq -(1-p_{t_2}) \cdot \mathsf{Gains} \Leftrightarrow$$

$$\Leftrightarrow \Delta\mathsf{Out}_{t_1}(1) = -C_{t_1} - (1-p_{t_1}) \cdot \mathsf{Gains} \geq -C_{t_2} - (1-p_{t_2}) \cdot \mathsf{Gains} = \Delta\mathsf{Out}_{t_2}(1).$$

$(C_{t_1}, p_{t_1})$ and $(C_{t_2}, p_{t_2})$ *multidominates* $(C_{t_3}, p_{t_3})$ if $C_1 < C_3 < C_2$ , $p_1 < p_3 < p_2$ and the crossing-point of functions of $t_1$ and $t_2$ is always higher than the function of $t_3$.

In every node of a tree we store the list of pairs $(C, p)$ which correspond to the subattacks of this node. By this pair we are able to construct the linear function $\Delta\mathsf{Out}$ when we want to compare those subattacks.

- In every leaf we add the pair $(C, p)$ to its list.
- In OR-node we add to its list all the pairs that its children's lists contain and eliminate the pairs which are dominated or multidominated by some other pairs from the list.
- In AND-node we have to find every possibility to form the full attack from its child attacks. In the end the AND-node will get the list of pairs of full attacks, where the parameter $C$ is the sum of all the subattack $C$-s and $p$ is the multiplication of all the subattack $p$-s. (For example, if AND-node's first child has $n$ pairs and the second has $m$ pairs then in the AND-node there has to be considered $n \cdot m$ pairs).

It is clear that this kind of calculations are commutative and associative because addition and multiplication satisfy these conditions. If we leave out the elimination part, then this method finds the full semantics tree and that is why it is distributive. It occurs that the elimination method does not loose this property. The elimination on domination leaves out only the nodes that does not belong to the attack suite, because those pairs cannot generate new pairs (with OR- or AND-calculations) that are not dominated or multidominated. If

$(C_{t_1}, p_{t_1})$ dominates $(C_{t_2}, p_{t_2})$, then in AND-node $C_{t_1} + C_{t_3} \leq C_{t_2} + C_{t_3}$ and $p_{t_1} \cdot p_{t_3} \geq p_{t_2} \cdot p_{t_3}$, because $p_{t_3} \geq 0$. If $(C_{t_1}, p_{t_1})$ and $(C_{t_2}, p_{t_2})$ multidominates $(C_{t_3}, p_{t_3})$, then in AND-node $C_{t_1} + C_{t_4} \leq C_{t_3} + C_{t_4} \leq C_{t_2} + C_{t_4}$ and $p_{t_1} \cdot p_{t_4} \leq p_{t_3} \cdot p_{t_4} \leq p_{t_2} \cdot p_{t_4}$, because $p_{t_4} \geq 0$.

### 2.4.2 Estimation method

The list method is accurate, but actually we can overestimate the outcomes of attacks, because the overestimated attack is always less profitable than the original. In that case it is sufficient to store only one attack in every node, we just change the parameters so that the result will be overestimated. We can store to every node one additional parameter $M$ that shows us how big is the maximum difference between the outcomes of original attack and estimated attack.

Let $[A, B]$ be the range where $A$ corresponds to the least probable attack and $B$ corresponds to the most probable attack, that has to be added to get the full attack. In this method actually $A = 0, B = 1$ but we will give more generalized notation for later work.

In the AND-node the parameter $C$ is the sum of all the children $C$-s and $p$ is the multiplication of all the subattack $p$-s. Unlike the list method, the additional parameter $M$ is also the sum of all the child attack $M$-s. It means that the roundings of all the subattacks are carried over to the AND-node. We can do so, because in every childnode we rounded the outcome at most by $M$. In the AND-node we can do those roundings independently, so we can sum them.

In the OR-node we are only interested in two subattacks. The first subattack $(C_1, p_1, M_1)$ gives us the highest $\Delta\mathsf{Outcome}$ on value $A$ and $(C_2, p_2, M_2)$ gives the highest $\Delta\mathsf{Outcome}$ on value $B$. So these linear functions will look like:

$$\Delta\mathsf{Out}_i(p_s) = -C_i - (1 - p_i) \cdot p_s, i = 1, 2.$$

Now we generate the estimated attack which $\Delta\mathsf{Outcome}$ is never less than the original attack.

$$\Delta\mathsf{Out}_1(B) - \Delta\mathsf{Out}_2(A) = (1 - p) \cdot \mathsf{Gains} \cdot (B - A) \Leftrightarrow$$

$$\Leftrightarrow p = 1 - \frac{\Delta\mathsf{Out}_2(B) - \Delta\mathsf{Out}_2(A)}{(B - A) \cdot \mathsf{Gains}},$$

$$C = \Delta\mathsf{Out}_1(A) - (1 - p) \cdot \mathsf{Gains} \cdot A.$$

Now the new linear function $\Delta\mathsf{Out}_{est} = -C - (1-p) \cdot \mathsf{Gains} \cdot p_s$. Next we have to calculate the error $M$ for $\Delta\mathsf{Out}_{est}$ that shows us how much is this function maximally bigger than the original functions. Because of that, we have to find the crossing point $P$ of $\Delta\mathsf{Out}_1$ and $\Delta\mathsf{Out}_2$:

$$-C_1 - (1 - p_1) \cdot P \cdot \mathsf{Gains} = -C_2 - (1 - p_2) \cdot P \cdot \mathsf{Gains} \Leftrightarrow$$

$$\Leftrightarrow C_2 - C_1 = P \cdot \mathsf{Gains} \cdot (-(1 - p_2) + (1 - p_1)) \Leftrightarrow$$

$$\Leftrightarrow P \cdot \mathsf{Gains} \cdot (p_2 - p_1)) = C_2 - C_1 \Leftrightarrow P = \frac{C_2 - C_1}{\mathsf{Gains} \cdot (p_2 - p_1)}.$$

So the error of this estimated attack is:

$$M = max(M_1, M_2) + \Delta\mathsf{Out}_{est}(P) - \Delta\mathsf{Out}_1(P).$$

Finally, we store the triple $(C, p, M)$ to this OR-node.

The figure 4 illustrates this algorithm, when $A = 0, B = 1$. If we could somehow make $A$ and $B$ closer, then the estimation would be more precise.

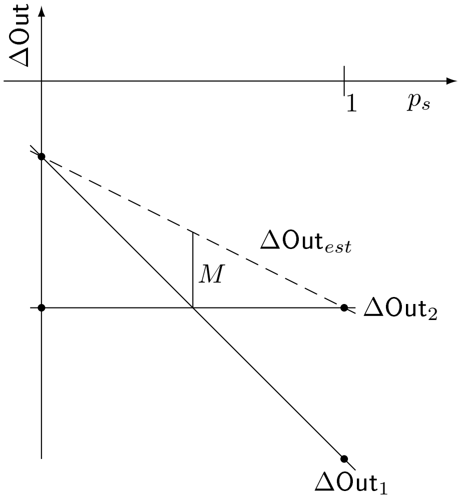

Figure 4: The estimated attack's function

# 3 Conclusion

The risk-analysis method by Buldas et al. [1] could be a fast technique to calculate the risk of different security threats. Unfortunately, there are some bugs that sometimes cause incorrect results.

We examined those trees of [1] and discovered that the mistake is in the way how OR-node's parameters are found. It appears that the decision of which OR-node's child is the best, depends on the probabilities of that node's neighbor nodes' attacks.

One possible, but not fast solution is the list method. Actually it is quite similar to finding the full semantics tree, only this method compares the linear functions of attacks and leaves out the functions that are never greater than the others.

The other solution is to overestimate the outcomes in the range $[A, B]$ where $A = 0, B = 1$. This algorithm is faster, but it is not so accurate than the list method.

Further work will be to minimize the length of $[A, B]$ to get more precise method.

## References

[1] Jaan Priisalu Märt Saarepera Jan Willemson Ahto Buldas, Peeter Laud. Rational choice of security measures via multi-parameter attack trees. *Critical Information Infrastructures Security First International Workshop*, LNCS 4347:235–248, 2006.

[2] Peeter Laud. Unnamed document.

[3] Martijn Oostdijk Sjouke Mauw. Foundations of attack trees. *International Conference on Information Security and Cryptology*, LNCS 3935:186–198, 2005.