Computing Exact Outcomes of Multi-parameter Attack Trees

Aivo Jürgenson[1,2] and Jan Willemson[3]

[1] Tallinn University of Technology, Raja 15, 12618 Tallinn, Estonia
aivo.jurgenson@eesti.ee
[2] Elion Enterprises Ltd, Endla 16, 15033 Tallinn, Estonia
[3] Cybernetica, Aleksandri 8a, Tartu, Estonia
jan.willemson@gmail.com

Abstract. In this paper we introduce a set of computation rules to determine the attacker's exact expected outcome based on a multi-parameter attack tree. We compare these rules to a previously proposed computational semantics by Buldas *et al.* and prove that our new semantics always provides at least the same outcome. A serious drawback of our proposed computations is the exponential complexity. Hence, implementation becomes an important issue. We propose several possible optimisations and evaluate the result experimentally. Finally, we also prove the consistency of our computations in the framework of Mauw and Oostdijk and discuss the need to extend the framework.

1 Introduction

Attack tree (also called threat tree) approach to security evaluation is several decades old. It has been used for tasks like fault assessment of critical systems [1] or software vulnerability analysis [2,3]. The approach was first applied in the context of information systems (so-called *threat logic trees*) by Weiss [4] and later more widely adapted to information security by Bruce Schneier [5]. We refer to [6,7] for good overviews on the development and applications of the methodology.

Even though already Weiss [4] realised that nodes of attack trees have many parameters in practise, several subsequent works in this field considered attack trees using only one estimated parameter like the cost or feasibility of the attack, skill level required, etc. [3,5,8]. Opel [9] considered also multi-parameter attack trees, but the actual tree computations in his model still used only one parameter at a time. Even though single-parameter attack trees can capture some aspects of threats reasonably well, they still lack the ability to describe the full complexity of the attacker's decision-making process.

A substantial step towards better understanding the motivation of the attacker was made in 2006 by Buldas *et al.* [10]. Besides considering just the cost of the attack, they also used success probability together with probabilities and amount of penalties in the case of success or failure of the attack in their analysis. As a result, a more accurate model of the attack game was obtained and it was later used to analyse the security of several e-voting schemes by Buldas and

Mägi [11]. The model was developed further by Jürgenson and Willemson [12] extending the parameter domain from point values to interval estimations.

However, it is known that the computational semantics given in [10] is both imprecise and inconsistent with the general framework introduced by Mauw and Oostdijk [8] (see Section 2). The motivation of the current paper is to develop a better semantics in terms of precision and consistency. For that we will first review the tree computations of [10] in Section 2 and then propose an improved semantics in Section 3. However, it turns out that the corresponding computational routines are inherently exponential, so optimisation issues of the implementation become important; these are discussed in Section 4. In Section 5 we prove that the new semantics always provides at least the same expected outcome for an attacker as the tree computations of [10]. We also argue that the new semantics is consistent with the framework of Mauw and Oostdijk. Finally, in Section 6 we draw some conclusions and set directions for further work.

2 Background

In order to better assess the security level of a complex and heterogeneous system, a gradual refinement method called *threat tree* or *attack tree method* can be used. The basic idea of the approach is simple — the analysis begins by identifying one or more *primary threats* and continues by splitting the threat into subattacks, either all or some of them being necessary to materialise the primary threat. The subattacks can be divided further etc., until we reach the state where it does not make sense to split the resulting attacks any more; these kinds of non-splittable attacks are called *elementary* or *atomic attacks* and the security analyst will have to evaluate them somehow. During the splitting process, a tree is formed having the primary threat in its root and elementary attacks in its leaves. Using the structure of the tree and the estimations of the leaves, it is then (hopefully) possible to give some estimations of the root node as well. In practise, it mostly turns out to be sufficient to consider only two kinds of splits in the internal nodes of the tree, giving rise to AND- and OR-nodes. As a result, an AND-OR-tree is obtained, forming the basis of the subsequent analysis. An example attack tree originally given by Weiss [4] and adopted from [6] is presented in Figure 1.

We will use the basic multi-parameter attack tree model introduced in [10]. Let us have the AND-OR-tree describing the attacks and assume all the elementary attacks being pairwise independent. Let each leaf X_i have the following parameters:

– Cost_i – the cost of the elementary attack
– p_i – success probability of the attack
– π_i^- – the expected penalty in case the attack was unsuccessful
– π_i^+ – the expected penalty in case the attack was successful.

Besides these parameters, the tree has a global parameter Gains showing the benefit of the attacker in the case he is able to mount the root attack. For practical

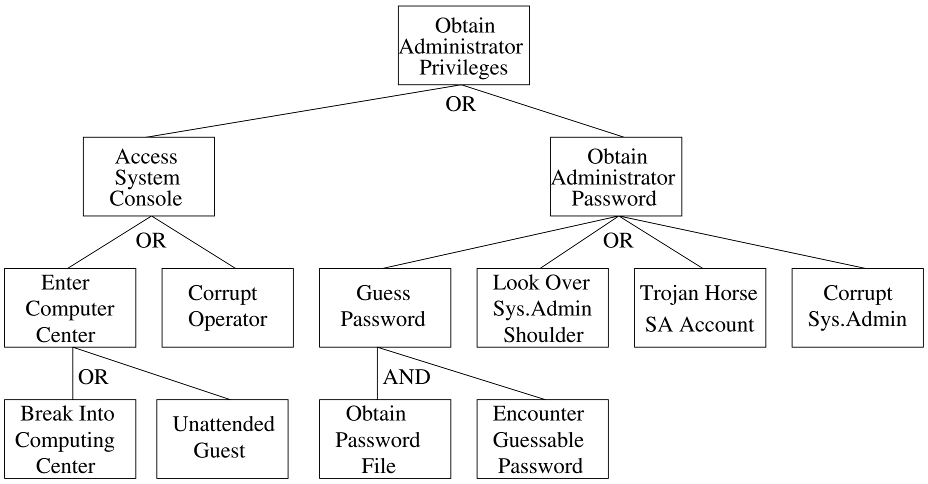

Fig. 1. Example of an attack tree

examples on how to evaluate those parameters for real-life attacks, please refer to [11] and [13].

The paper [10] gives a simple computational semantics to the attack trees, which has further been extended to interval estimates in [12]. After the above-mentioned parameters have been estimated for the leaf nodes, a step-by-step propagation algorithm begins computing the same parameters for all the internal nodes as well, until the root node has been reached. The computational routines defined in [10] are the following:

– For an OR-node with child nodes with parameters $(\mathsf{Cost}_i, p_i, \pi_i^+, \pi_i^-)$ $(i = 1, 2)$ the parameters $(\mathsf{Cost}, p, \pi^+, \pi^-)$ are computed as:

$$(\mathsf{Cost}, p, \pi^+, \pi^-) = \begin{cases} (\mathsf{Cost}_1, p_1, \pi_1^+, \pi_1^-), \text{ if } \mathsf{Outcome}_1 > \mathsf{Outcome}_2 \\ (\mathsf{Cost}_2, p_2, \pi_2^+, \pi_2^-), \text{ if } \mathsf{Outcome}_1 \leq \mathsf{Outcome}_2 \end{cases},$$

$$\mathsf{Outcome}_i = p_i \cdot \mathsf{Gains} - \mathsf{Cost}_i - p_i \cdot \pi_i^+ - (1 - p_i) \cdot \pi_i^- .$$

– For an AND-node with child nodes with parameters $(\mathsf{Cost}_i, p_i, \pi_i^+, \pi_i^-)$ $(i = 1, 2)$ the parameters $(\mathsf{Cost}, p, \pi^+, \pi^-)$ are computed as follows:

$$\mathsf{Costs} = \mathsf{Costs}_1 + \mathsf{Costs}_2, \quad p = p_1 \cdot p_2, \quad \pi^+ = \pi_1^+ + \pi_2^+,$$
$$\pi^- = \frac{p_1(1-p_2)(\pi_1^+ + \pi_2^-) + (1-p_1)p_2(\pi_1^- + \pi_2^+)}{1 - p_1 p_2} + \\ + \frac{(1-p_1)(1-p_2)(\pi_1^- + \pi_2^-)}{1 - p_1 p_2} .$$

The formula for π^- represents the average penalty of an attacker, assuming that at least one of the two child-attacks was not successful. For later computations, it will be convenient to denote expected expenses associated with the node i as $\mathsf{Expenses}_i = \mathsf{Cost}_i + p_i \cdot \pi_i^+ + (1-p_i) \cdot \pi_i^-$. Then it is easy to see that in an AND-node the equality $\mathsf{Expenses} = \mathsf{Expenses}_1 + \mathsf{Expenses}_2$ holds. Note that the formulae above have obvious generalisations for non-binary trees.

At the root node, its Outcome is taken to be the final outcome of the attack and the whole tree is considered to be beneficial for a rational attacker if $\mathsf{Outcome} > 0$. Following the computation process it is possible to collect the corresponding set of leaves which, when carried out, allow the attacker to mount the root attack and get the predicted outcome. Such leaf sets will subsequently be called *attack suites.*[1]

However, while being very fast to compute, this semantics has several drawbacks:

1. In order to take a decision in an OR-node, the computational model of [10] needs to compare outcomes of the child nodes and for that some local estimate of the obtained benefit is required. Since it is very difficult to break the total root gain into smaller benefits, the model of [10] gives the total amount of Gains to the attacker for each subattack. This is clearly an overestimation of the attacker's outcome.
2. In an OR-node, the model of [10] assumes that the attacker picks exactly one descendant. However, it is clear that in practise, it may make sense for an attacker to actually carry out several alternatives if the associated risks and penalties are low and the success probability is high.
3. There is a general result by Mauw and Oostdijk [8] stating which attack tree computation semantics are inherently consistent. More precisely, they require that the semantics of the tree should remain unchanged when the underlying Boolean formula is transformed to an equivalent one (e.g. to a disjunctive normal form). Semantics given in the [10] are not consistent in this sense. For example, lets take two attack trees, $T_1 = A \vee (B\&C)$ and $T_2 = (A \vee B)\&(A \vee C)$, both having same parameters $\mathsf{Gains} = 10000$, $p_A = 0.1$, $p_B = 0.5$, $p_C = 0.4$, $\mathsf{Expenses}_A = 1000$, $\mathsf{Expenses}_B = 1500$, $\mathsf{Expenses}_C = 1000$. Following the computation rules of [10], we get $\mathsf{Outcome}_{\mathsf{T}_1} = 8000$ and $\mathsf{Outcome}_{\mathsf{T}_2} = 6100$, even though the underlying Boolean formulae are equivalent.

The aim of this paper is to present an exact and consistent semantics for attack trees. The improved semantics fixes all the three abovementioned shortcomings. However, a major drawback of the new approach is the increase of the computational complexity from linear to exponential (depending on the number of elementary attacks). Thus finding efficient and good approximations becomes a vital task. In this paper, we will evaluate suitability of the model of [10] as an approximation; the question of better efficient approximations remains an open problem for future research.

[1] Note that our terminology differs here from the one used by Mauw and Oostdijk [8]. Our attack suite would be just attack in their terms and their attack suite would be the set of all possible attack suites for us.

3 Exact Semantics for the Attack Trees

3.1 The Model

In our model, the attacker behaves as follows.

– First, the attacker constructs an attack tree and evaluates the parameters of its leaves.
– Second, he considers all the potential attack suites, i.e. subsets $\sigma \subseteq \mathcal{X} = \{X_i : i = 1, \dots, n\}$. Some of these materialise the root attack, some of them do not. For the suites that do materialise the root attack, the attacker evaluates their outcome for him.
– Last, the attacker decides to mount the attack suite with the highest outcome (or he may decide not to attack at all if all the outcomes are negative).

Note that in this model the attacker tries all the elementary attacks independently. In practise, this is not always true. For example, if the attacker has already failed some critical subset of the suite, it may make more sense for him not to try the rest of the suite. However, the current model is much more realistic compared to the one described in [10], since now we allow the attacker to plan its actions with redundancy, i.e. try alternative approaches to achieve some (sub)goal.

3.2 Formalisation

The attack tree can be viewed as a Boolean formula $\mathcal{F}$ composed of the set of variables $\mathcal{X} = \{X_i : i = 1, \dots, n\}$ (corresponding to the elementary attacks) and conjunctives $\vee$ and &. Satisfying assignments $\sigma \subseteq \mathcal{X}$ of this formula correspond to the attack suites sufficient for materialising the root attack.

The exact outcome of the attacker can be computed as

$$\mathsf{Outcome} = \max\{\mathsf{Outcome}_\sigma : \sigma \subseteq \mathcal{X},\, \mathcal{F}(\sigma := \mathsf{true}) = \mathsf{true}\}\,. \tag{1}$$

Here $\mathsf{Outcome}_\sigma$ denotes the expected outcome of the attacker if he decides to try the attack suite σ and $\mathcal{F}(\sigma := \mathsf{true})$ denotes evaluation of the formula $\mathcal{F}$, when all of the variables of σ are assigned the value true and all others the value false. The expected outcome $\mathsf{Outcome}_\sigma$ of the suite σ is computed as follows:

$$\mathsf{Outcome}_\sigma = p_\sigma \cdot \mathsf{Gains} - \sum_{X_i \in \sigma} \mathsf{Expenses}_i\,, \tag{2}$$

where p_σ is the success probability of the attack suite σ.

When computing the success probability p_σ of the attack suite σ we must take into account that the suite may contain redundancy and there may be (proper) subsets $\rho \subseteq \sigma$ sufficient for materialising the root attack. Because we are using the full suite of σ to mount an attack, those elementary attacks in the $\sigma \setminus \rho$ will contribute to the success probability of p_ρ with $(1-p_j)$. Thus, the total success probability can be computed as

$$p_\sigma = \sum_{\substack{\rho\subseteq\sigma \\ \mathcal{F}(\rho:=\mathsf{true})=\mathsf{true}}} \prod_{X_i\in\rho} p_i \prod_{X_j\in\sigma\setminus\rho} (1-p_j)\,. \tag{3}$$

Note that the formulae (1), (2) and (3) do not really depend on the actual form of the underlying formula $\mathcal{F}$, but use it only as a Boolean function. As a consequence, our framework is not limited to just AND-OR trees, but can in principle accommodate other connectives as well. Independence of the concrete form will also be the key observation when proving the consistency of our computation routines in the framework of Mauw and Oostdijk (see Proposition 1 in Section 5).

3.3 Example

To explain the exact semantics model of the attack trees, we give the following simple example. Lets consider the attacktree with the Boolean formula $T = (A \vee B)\&C$ with all elementary attacks (A, B, C) having equal parameters $p = 0.8$, $\mathsf{Cost} = 100$, $\pi^+ = 1000$, $\pi^- = 1000$ and $\mathsf{Gain} = 10000$. That makes $\mathsf{Expenses} = 1100$ for all elementary attacks. When we follow the approximate computation rules in the [10], we get the $\mathsf{Outcome}_T = 4200$.

By following the computation rules in this article, we have the attack suites $\sigma_1 = \{A, C\}, \sigma_2 = \{B, C\}, \sigma_3 = \{A, B, C\}$, which satisfy the original attack tree T. The outcome computation for attack suites σ_1 and σ_2 is straightforward and $\mathsf{Outcome}_{\sigma_1} = \mathsf{Outcome}_{\sigma_2} = 4200$. The $\mathsf{Outcome}_{\sigma_3}$ is a bit more complicated as there are three subsets $\rho_1 = \{A, C\}$, $\rho_2 = \{B, C\}$, $\rho_3 = \{A, B, C\}$ for the suite σ_3, which also satisfy the attack tree T. Therefore we get the $p_{\sigma_3} = p_A p_B p_C + p_A p_C(1 - p_B) + p_B p_C(1 - p_A) = 0.768$ and $\mathsf{Outcome}_{\sigma_3} = 4380$. By taking the maximum of the three outcomes, we get $\mathsf{Outcome}_T = 4380$.

As the Cost parameters in this example for elementary attacks A and B were chosen quite low and the success probability p_A and p_B of these attacks were quite high, it made sense for an attacker to mount both of these subattacks and get bigger expected outcome, even though the attack tree would have been satisfied as well by only one of them.

4 Implementation

The most time-consuming computational routine among the computations given in Section 3.2 is the generation of all the satisfiable assignments of a Boolean formula $\mathcal{F}$ in order to find the maximal outcome by (1). Even though the computation routine (3) for finding p_σ formally also goes through (potentially all) subsets of σ, it can be evaluated in linear time in the number of variables n. To do so we can set $p_i = 0$ for all $X_i \notin \sigma$ and leave all the p_i for $X_i \in \sigma$ untouched.

Then for each internal node of the tree with probabilities of the child nodes being $p_{i_1}, p_{i_2}, \ldots, p_{i_k}$ we can compute the probability of the parent node to be

$$\prod_{j=1}^{k} p_{i_j} \quad \text{or} \quad 1 - \prod_{j=1}^{k} (1 - p_{i_j})$$

depending on whether it is an AND or an OR node. Propagating throughout the tree, this computation gives exactly the success probability p_σ of the suite σ at the root node.

The routine (1) can be optimised as well by cutting off hopeless cases (see Theorem 1), but it still remains worst-case exponential-time. Thus for performance reasons it is crucial to have an efficient implementation of this routine. We are using a modified version of DPLL algorithm [14] to achieve this goal. The original form of the DPLL algorithm is only concerned about satisfiability, but it can easily be upgraded to produce all the satisfying assignments as well. Note that all the assignments are not needed at the same time to compute (1), but rather one at a time. Hence we can prevent the exponential memory consumption by building a serialised version, obtaining Algorithm 1.

Algorithm 1 works recursively and besides the current Boolean formula $\mathcal{F}$ it has two additional parameters. The set S contains the variables of which the satisfying assignments should be composed from. The set A on the other hand contains the variables already chosen to the assignments on previous rounds of recursion. As a technical detail note that the satisfying assignments are identified by the subset of variables they set to true.

The computation starts by calling process_satisfying_assignments($\mathcal{F}$, $\mathcal{X}$, $\emptyset$). Note that Algorithm 1 does not really produce any output, a processing subroutine is called on step 1 instead. This subroutine computes $\mathsf{Outcome}_\sigma$ for the given assignment σ and compares it with the previous maximal outcome.

4.1 Optimisations

Even with the help of a DPLL-based algorithm, the computations of (1) remain worst-case exponential time. In order to cut off hopeless branches, we can make some useful observations.

When we consider a potential attack suite σ and want to know, whether it is sufficient to materialise the root attack, we will set all the elements of σ to true, all the others to false and evaluate the formula $\mathcal{F}$ corresponding to the attack tree. In the process, all the internal nodes of the tree get evaluated as well (including the root node, showing whether the suite is sufficient). In Section 3, we allowed the suites σ to have more elements than absolutely necessary for materialising the root node, because in OR-nodes it often makes a lot of sense to try different alternatives. In AND nodes, at the same time, no choice is actually needed and achieving some children of an AND node without achieving some others is just a waste of resources.

Thus, intuitively we can say that it makes no sense to have AND-nodes with some children evaluating to true and some children to false. Formally, we can state and prove the following theorem.

Algorithm 1. Processing all the satisfying assignments of a formula

```
Procedure process_satisfying_assignments($\mathcal{F}, S, A$)
Input: Boolean CNF-formula $\mathcal{F}$, a subsets $S$ of its variables and a subset $A \subseteq \mathcal{X} \setminus S$

1. If $\mathcal{F}$ contains true in every clause then
     – Process the assignment $A \cup T$ for every $T \subseteq S$; return
2. If $\mathcal{F}$ contains an empty clause or $S = \emptyset$ then return #no output in this branch
3. If $\mathcal{F}$ contains a unit clause $\{X\}$, where $X \in S$ then
     – Let $\mathcal{F}'$ be the formula obtained by setting $X =$ true in $\mathcal{F}$
     – process_satisfying_assignments($\mathcal{F}', S \setminus \{X\}, A \cup \{X\}$)
     – Return
4. Select a variable $X \in S$
5. Let $\mathcal{F}'$ be the formula obtained by setting $X =$ true in $\mathcal{F}$
6. process_satisfying_assignments($\mathcal{F}', S \setminus \{X\}, A \cup \{X\}$)
7. Let $\mathcal{F}''$ be the formula obtained by deleting $X$ from $\mathcal{F}$
8. process_satisfying_assignments($\mathcal{F}'', S \setminus \{X\}, A$)
9. Return
```

Theorem 1. *Let $\mathcal{F}$ be a Boolean formula corresponding to the attack tree T (i.e. AND-OR-tree, where all variables occur only once) and let σ be its satisfying assignment (i.e. an attack suite). Set all the variables of σ to* true *and all others to* false *and evaluate all the internal nodes of T. If some AND-node has children evaluating to* true *as well as children evaluating to* false*, then there exists a satisfying assignment $\sigma' \subset \sigma$ ($\sigma' \neq \sigma$) such that* $\mathsf{Outcome}_{\sigma'} \geq \mathsf{Outcome}_{\sigma}$.

Proof. Consider an AND-node Y having some children evaluating to true and some evaluating to false. Then the node Y itself also evaluates to false, but the set of variables of the subformula corresponding to Y has a non-empty intersection with σ; let this intersection be τ. We claim that we can take $\sigma' = \sigma \setminus \tau$. First it is clear that $\sigma' \subset \sigma$ and $\sigma' \neq \sigma$. Note also that σ' is a satisfying assignment and hence $\sigma' \neq \emptyset$. Now consider the corresponding outcomes:

$$\mathsf{Outcome}_{\sigma} = p_{\sigma} \cdot \mathsf{Gains} - \sum_{X_i \in \sigma} \mathsf{Expenses}_i \,,$$

$$\mathsf{Outcome}_{\sigma'} = p_{\sigma'} \cdot \mathsf{Gains} - \sum_{X_i \in \sigma'} \mathsf{Expenses}_i \,.$$

Since $\sigma' \subset \sigma$, we have

$$\sum_{X_i \in \sigma} \mathsf{Expenses}_i \geq \sum_{X_i \in \sigma'} \mathsf{Expenses}_i \,,$$

as all the added terms are non-negative.

Now we claim that the equality $p_{\sigma} = p_{\sigma'}$ holds, which implies the claim of the theorem. Let

$$R_{\sigma} = \{\rho \subseteq \sigma \;:\; \mathcal{F}(\rho := \mathsf{true}) = \mathsf{true}\}$$

and define $R_{\sigma'}$ in a similar way. Then by (3) we have

$$p_\sigma = \sum_{\rho \in R_\sigma} \prod_{X_i \in \rho} p_i \prod_{X_j \in \sigma \setminus \rho} (1 - p_j)\,,$$
$$p_{\sigma'} = \sum_{\rho' \in R_{\sigma'}} \prod_{X_i \in \rho'} p_i \prod_{X_j \in \sigma' \setminus \rho'} (1 - p_j)\,.$$

We claim that $R_\sigma = \{\rho' \cup \tau' : \rho' \in R_{\sigma'}, \tau' \subseteq \tau\}$, i.e. that all the satisfying subassignments of σ can be found by adding all the subsets of τ to all the satisfying subassignments of σ'. Indeed, the node Y evaluates to false even if all the variables of τ are true, hence the same holds for every subset of τ due to monotonicity of AND and OR. Thus, if a subassignment of σ satisfies the formula $\mathcal{F}$, the variables of τ are of no help and can have arbitrary values. The evaluation true for the root node can only come from the variables of σ', proving the claim.

Now we can compute:

$$\begin{aligned}
p_\sigma &= \sum_{\rho \in R_\sigma} \prod_{X_i \in \rho} p_i \prod_{X_j \in \sigma \setminus \rho} (1 - p_j) = \sum_{\substack{\rho = \rho' \cup \tau' \\ \rho' \in R_{\sigma'}, \tau' \subseteq \tau}} \prod_{X_i \in \rho} p_i \prod_{X_j \in \sigma \setminus \rho} (1 - p_j) = \\
&= \sum_{\rho' \in R_{\sigma'}} \sum_{\tau' \subseteq \tau} \prod_{X_i \in \rho' \cup \tau'} p_i \prod_{X_j \in \sigma \setminus (\rho' \cup \tau')} (1 - p_j) = \\
&= \sum_{\rho' \in R_{\sigma'}} \sum_{\tau' \subseteq \tau} \prod_{X_i \in \rho'} p_i \prod_{X_i \in \tau'} p_i \prod_{X_j \in \sigma' \setminus \rho'} (1 - p_j) \prod_{X_j \in \tau \setminus \tau'} (1 - p_j) = \\
&= \sum_{\rho' \in R_{\sigma'}} \prod_{X_i \in \rho'} p_i \prod_{X_j \in \sigma' \setminus \rho'} (1 - p_j) \sum_{\tau' \subseteq \tau} \prod_{X_i \in \tau'} p_i \prod_{X_j \in \tau \setminus \tau'} (1 - p_j) = \\
&= \sum_{\rho' \in R_{\sigma'}} \prod_{X_i \in \rho'} p_i \prod_{X_j \in \sigma' \setminus \rho'} (1 - p_j) \prod_{X_i \in \tau} [p_i + (1 - p_i)] = \\
&= \sum_{\rho' \in R_{\sigma'}} \prod_{X_i \in \rho'} p_i \prod_{X_j \in \sigma' \setminus \rho'} (1 - p_j) = p_{\sigma'}\,,
\end{aligned}$$

since $\sigma \setminus (\rho' \cup \tau') = (\sigma' \setminus \rho') \dot{\cup} (\tau \setminus \tau')$. The claim of the theorem now follows easily. □

Note that Theorem 1 really depends on the assumption that $\mathcal{F}$ is an AND-OR-tree and that all variables occur only once. Formulae (1) and (3) together with Algorithm 1 can still be applied if the structure of the formula $\mathcal{F}$ is more complicated (say, a general DAG with other connectives in internal nodes), but the optimisation of Theorem 1 does not necessarily work.

This theorem allows us to leave many potential attack suites out of consideration by simply verifying if they evaluate children of some AND-node in a different way.

4.2 Performance

We implemented Algorithm 1 in Perl programming language and ran it on 500 randomly generated trees. The tests were ran on a computer having 3GHz dual-core Intel processor, 1GB of RAM and Arch Linux operating system.

The tree generation procedure was the following:

1. Generate the root node.
2. With probability 50% let this node have 2 children and with probability 50% let it have 3 children.
3. For every child, let it be an AND-node, an OR-node or a leaf with probability 40%, 40% and 20%, respectively.
4. Repeat the steps number 2 and 3 for every non-leaf node until the tree of depth up to 3 has been generated and let all the nodes on the third level be leaves.
5. To all the leaf nodes, generate the values of Cost, π^+ and π^- as integers chosen uniformly from from the interval $[0, 1000)$, and the value of p chosen uniformly from the interval $[0, 1)$.
6. Generate the value of Gains as an integer chosen uniformly from the interval $[0, 1000000)$.

Thus, the generated trees may in theory have up to 27 leaves. That particular size limit for the trees was chosen because the running time for larger trees was already too long for significant amount of tests.

Performance test results showing the average running times and the standard deviation of the running times of the algorithm depending on the number of leaves are displayed in Figure 2. Note that the time scale is logarithmic. The times are measured together with the conversion of the attack tree formula to the conjunctive normal form. In Figure 2 we have included the trees with only up to 19 leaves, since the number of larger trees generated was not sufficient to produce statistically meaningful results. The number of the generated trees by the number of leaves is given later in Figure 3.

5 Analysis

In this Section we provide some evaluation of our tree computations compared to the ones given by Buldas *et al.* [10] and within the framework of Mauw and Oostdijk [8].

5.1 Comparison with the Semantics of Buldas *et al.*

Our main result can be shortly formulated as the following theorem.

Theorem 2. *Let us have an attack tree T. Let the best attack suites found by the routines of the current paper and the paper [10] be σ and σ' respectively. Let the corresponding outcomes (computed using the respective routines) be $\mathsf{Outcome}_\sigma$ and $\mathsf{Outcome}_{\sigma'}$. The following claims hold:*

1. *If $\sigma = \sigma'$ then $\mathsf{Outcome}_\sigma = \mathsf{Outcome}_{\sigma'}$.*
2. *$\mathsf{Outcome}_\sigma \geq \mathsf{Outcome}_{\sigma'}$.*

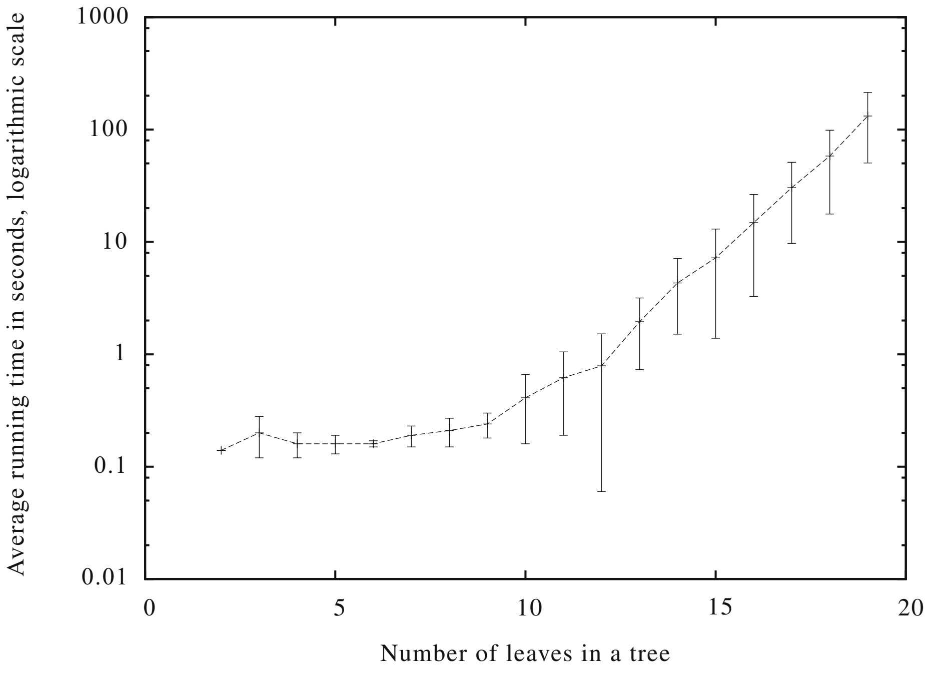

Fig. 2. Performance test results

Proof.

1. We need to prove that if $\sigma = \sigma'$ then

$$\mathsf{Outcome}_{\sigma'} = p_\sigma \cdot \mathsf{Gains} - \sum_{X_i \in \sigma} \mathsf{Expenses}_i \,.$$

First note that the attack suite output by the routine of [10] is minimal in the sense that none of its proper subsets materialises the root node, because only one child is chosen in every OR-node. Hence, $p_\sigma = \prod_{X_i \in \sigma} p_i$. Now consider how $\mathsf{Outcome}_{\sigma'}$ of the root node is computed in [10]. Let the required parameters of the root node be p', Gains' and $\mathsf{Expenses}'$. Obviously, $\mathsf{Gains}' = \mathsf{Gains}$. By looking at how the values of the attack success probability and the expected expenses are propagated throughout the tree, we can also conclude that

$$p' = \prod_{X_i \in \sigma} p_i = p_\sigma \quad \text{and} \quad \mathsf{Expenses}' = \sum_{X_i \in \sigma} \mathsf{Expenses}_i \,,$$

finishing the first part of the proof.

2. Since σ' is a satisfying assignment of the Boolean formula underlying the tree T, we can conclude that σ' is considered as one of the attack suite candidates in (1). The conclusion now follows directly from the first part of the proof. □

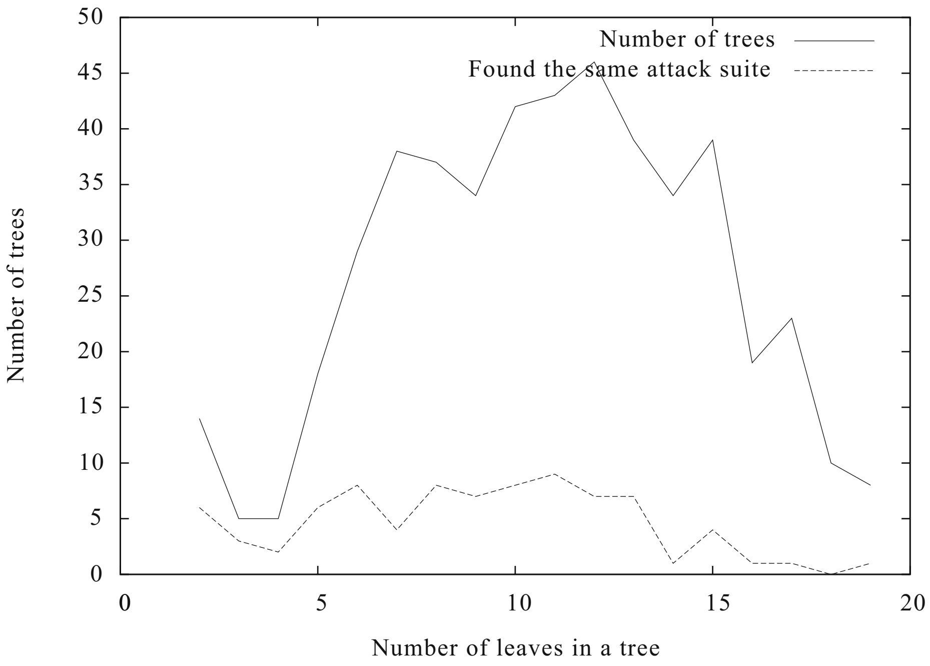

Fig. 3. Precision of the computational routine of Buldas *et al.* [10]

Theorem 2 implies that the exact attack tree computations introduced in the current paper always yield at least the same outcome compared to [10]. Thus, the potential use of the routine of [10] is rather limited, because it only allows us to get a lower estimate of the attacker's expected outcome, whereas the upper limit would be of much higher interest. We can still say that if the tree computations of [10] show that the system is insufficiently protected (i.e. $\mathsf{Outcome}_{\sigma'} > 0$) then the exact computations would yield a similar result ($\mathsf{Outcome}_{\sigma} > 0$).

Following the proof of Theorem 2, we can also see that the semantics of [10] is actually not too special. Any routine that selects just one child of every OR-node when analysing the tree would essentially give a similar under-estimation of the attacker's expected outcome.

Together with the performance experiments described in Section 4.2 we also compared the outcome attack suites produced by the routines of the current paper and [10] (the implementation of the computations of [10] was kindly provided by Alexander Andrusenko [15]). The results are depicted in Figure 3.

The graphs in Figure 3 show the number of the generated trees by the number of leaves and the number of such trees among them, for which the routine of [10] was able to find the same attack suite that the exact computations introduced in the current paper. Over all the tests we can say that this was the case with 17.4% of the trees.

5.2 Consistency with the Framework of Mauw and Oostdijk

Working in a single parameter model, Mauw and Oostdijk [8] first define a set V of attribute values and then consider an attribute function $\alpha : \mathbb{C} \to V$, where $\mathbb{C}$ is the set of elementary attacks (called *attack components* in [8]). In order to extend this attribution to the whole tree, they essentially consider the tree corresponding to the disjunctive normal form of the underlying Boolean formula. To obtain the attribute values of the conjunctive clauses (corresponding to our attack suites), they require a conjunctive combinator $\triangle : V \times V \to V$, and in order to get the value for the whole DNF-tree based on clause values they require a disjunctive combinator $\bigtriangledown : V \times V \to V$. Mauw and Oostdijk prove that if these combinators are commutative, associative and distributive, all the nodes of the tree in the original form can also be given attribute values and that the value of the (root node of the) tree does not change if the tree is transformed into an equivalent form. This equivalence is denoted as $\equiv$ and it is defined by the set of legal transformations retaining logical equivalence of the underlying Boolean formulae (see [8]). The structure $(\alpha, \bigtriangledown, \triangle)$ satisfying all the given conditions is called *distributive attribute domain*.

Even though the semantics used in [10,12] formally require four different parameters, they still fit into a single parameter ideology, since based on the quadruples of the child nodes, similar quadruples are computed for parents when processing the trees. However, it is easy to construct simple counterexamples showing that the computation rules of [10,12] are not distributive, one is given in the Section 2.

The computation rules presented in the current paper follow the framework of Mauw and Oostdijk quite well at the first sight. Formula (1) essentially goes through all the clauses in the complete disjunctive normal form of the underlying formula $\mathcal{F}$ and finds the one with the maximal outcome. So we can take $V = \mathbb{R}$ and $\bigtriangledown = \max$ in the Mauw and Oostdijk framework. However, there is no reasonable way to define a conjunctive combinator $\triangle : V \times V \to V$, since the outcome of an attack suite can not be computed from the outcomes of the elementary attacks; the phrase "outcome of an elementary attack" does not even have a meaning.

Another possible approach is to take $V = [0,1] \times \mathbb{R}^+$ and to interpret the first element of $\alpha(X)$ as the success probability p and the second element as $\mathsf{Expenses}$ for an attack X. Then the disjunctive combinator can be defined as outputting the pair which maximises the expression $p \cdot \mathsf{Gains} - \mathsf{Expenses}$. This combinator has a meaning in the binary case and as such, it is both associative and commutative, giving rise to an obvious n-ary generalisation. For the conjunctive combinator to work as expected in the n-ary case, we would need to achieve

$$\triangle_{X_i \in \sigma} \alpha(X_i) = (p_\sigma, \Sigma_{X_i \in \sigma} \mathsf{Expenses}_i) .$$

However, it is easy to construct a formula and a satisfying assignment σ such that constructing p_σ from success probabilities of the descendant instances using a conjunctive combinator is not possible. For example, we can take the formula

$\mathcal{F} = X_1 \vee X_2 \& X_3$, where X_1, X_2, X_3 are elementary attacks with success probabilities p_1, p_2, p_3, respectively. Let α_1 denote the first element of the output of α and let $\triangle_1$ denote the combinator $\triangle$ restricted to the first element of the pair (so $\triangle_1(X_i) = p_i$, $i = 1, 2, 3$). Then for $\sigma = \{X_1, X_2, X_3\}$ we would need to obtain

$$(p_1 \triangle_1 p_2) \triangle_1 p_3 = (\alpha_1(X_1) \triangle_1 \alpha_1(X_2)) \triangle_1 \alpha_1(X_3) = p_\sigma = p_1 + p_2 p_3 - p_1 p_2 p_3$$

for any $p_1, p_2, p_3 \in [0, 1]$, which is not possible. Indeed, taking $p_3 = 0$ we have $(p_1 \triangle_1 p_2) \triangle_1 0 = p_1$. In the same way we can show that $(p_2 \triangle_1 p_1) \triangle_1 0 = p_2$, which is impossible due to commutativity of $\triangle_1$ when $p_1 \neq p_2$.

All of the above is not a formal proof that our computations do not form a distributive attribute domain, but we can argue that there is no obvious way to interpret them as such. Additionally, if we had a distributive attribute domain then Theorem 3 with Corollary 2 of [8] would allow us to build a linear-time value-propagating tree computation algorithm, but this is rather unlikely.

However, we can still state and prove the following proposition.

Proposition 1. *Let T_1 and T_2 be two attack trees. If $T_1 \equiv T_2$, we have* $\mathsf{Outcome}(T_1) = \mathsf{Outcome}(T_2)$.

Proof. It is easy to see that the formulae (1), (2) and (3) do not depend on the particular form of the formula, but use it only as a Boolean function. Since the tree transformations defined in [8] keep the underlying Boolean formula logically equivalent, the result follows directly. □

In the context of [8], this is a somewhat surprising result. Even though the attribute domain defined in the current paper is not distributive (and it can not be easily turned into such), the main goal of Mauw and Oostdijk is still achieved. This means that the requirement for the attribute domain to be distributive in the sense of Mauw and Oostdijk is sufficient to have semantically consistent tree computations, but it is not really necessary. It would be interesting to study, whether the framework of Mauw and Oostdijk can be generalised to cover non-propagating tree computations (like the one presented in the current paper) as well.

6 Conclusions and Further Work

In this paper we introduced a computational routine capable of finding the maximal possible expected outcome of an attacker based on a given attack tree. We showed that when compared to rough computations given in [10], the new routine always gives at least the same outcome and mostly it is also strictly larger. This means that the tree computations of [10] are not very useful in practise, since they strongly tend to under-estimate attacker's capabilities. We also proved that unlike [10], our new semantics of the attack tree is consistent with the general ideology of the framework of Mauw and Oostdijk, even though our attribute domain is not distributive. This is a good motivation to start looking for further generalisations of the framework.

On the other hand, the routines of the current paper are computationally very expensive and do not allow practical analysis of trees with the number of leaves substantially larger than 20. Thus, future research needs to address at least two issues. First, there are some optimisations possible in the implementation (e.g. precomputation of frequently needed values), they need to be programmed and compared to the existing implementations. Still, any optimisation will very probably not decrease the time complexity of the algorithm to a subexponential class. Thus the second direction of further research is finding computationally cheap approximations, which would over-estimate the attacker's exact outcome.

As a further development of the attack tree approach, more general and realistic models can be introduced. For example, the model presented in the current paper does not take into account the possibility that the attacker may drop attempting an attack suite after a critical subset of it has already failed. Studying such models will remain the subject for future research as well.

Acknowledgments

This research has been supported by the Estonian Science Foundation grant no. 7081. Also, we would like to thank Ahto Buldas, Peeter Laud and Sven Laur for helpful discussions.

References

1. Vesely, W.E., Goldberg, F.F., Roberts, N.H., Haasl, D.F.: Fault Tree Handbook. US Government Printing Office, Systems and Reliability Research, Office of Nuclear Regulatory Research, U.S. Nuclear Regulatory Commission (January 1981)
2. Viega, J., McGraw, G.: Building Secure Software: How to Avoid Security Problems the Right Way. Addison Wesley Professional, Reading (2001)
3. Moore, A.P., Ellison, R.J., Linger, R.C.: Attack modeling for information security and survivability. Technical Report CMU/SEI-2001-TN-001, Software Engineering Institute (2001)
4. Weiss, J.D.: A system security engineering process. In: Proceedings of the 14th National Computer Security Conference, pp. 572–581 (1991)
5. Schneier, B.: Attack trees: Modeling security threats. Dr. Dobb's Journal 24(12), 21–29 (1999)
6. Edge, K.S.: A Framework for Analyzing and Mitigating the Vulnerabilities of Complex Systems via Attack and Protection Trees. Ph.D thesis, Air Force Institute of Technology, Ohio (2007)
7. Espedahlen, J.H.: Attack trees describing security in distributed internet-enabled metrology. Master's thesis, Department of Computer Science and Media Technology, Gjøvik University College (2007)
8. Mauw, S., Oostdijk, M.: Foundations of attack trees. In: Won, D., Kim, S. (eds.) ICISC 2005. LNCS, vol. 3935, pp. 186–198. Springer, Heidelberg (2006)
9. Opel, A.: Design and implementation of a support tool for attack trees. Technical report, Otto-von-Guericke University, Internship Thesis (March 2005)
10. Buldas, A., Laud, P., Priisalu, J., Saarepera, M., Willemson, J.: Rational Choice of Security Measures via Multi-Parameter Attack Trees. In: López, J. (ed.) CRITIS 2006. LNCS, vol. 4347, pp. 235–248. Springer, Heidelberg (2006)

11. Buldas, A., Mägi, T.: Practical security analysis of e-voting systems. In: Miyaji, A., Kikuchi, H., Rannenberg, K. (eds.) Advances in Information and Computer Security, Second International Workshop on Security, IWSEC. LNCS, vol. 4752, pp. 320–335. Springer, Heidelberg (2007)
12. Jürgenson, A., Willemson, J.: Processing multi-parameter attacktrees with estimated parameter values. In: Miyaji, A., Kikuchi, H., Rannenberg, K. (eds.) IWSEC 2007. LNCS, vol. 4752, pp. 308–319. Springer, Heidelberg (2007)
13. Rätsep, L.: The influence and measurability of the parameters of the security analysis of the Estonian e-voting system. M.Sc thesis, Tartu University (2008) (in Estonian)
14. Davis, M., Logemann, G., Loveland, D.: A machine program for theorem proving. Communications of the ACM 5(7), 394–397 (1962)
15. Andrusenko, A.: Multiparameter attack tree analysis software. B.Sc thesis, Tartu University (2008) (in Estonian)