Serial Model for Attack Tree Computations

Aivo Jürgenson[1,2], 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 51004, Estonia
jan.willemson@gmail.com

Abstract. In this paper we extend the standard attack tree model by introducing temporal order to the attacker's decision making process. This will allow us to model the attacker's behaviour more accurately, since this way it is possible to study his actions related to dropping some of the elementary attacks due to them becoming obsolete based on the previous success/failure results. We propose an efficient algorithm for computing the attacker's expected outcome based on the given order of the elementary attacks and discuss the pros and cons of considering general rooted directed acyclic graphs instead of plain trees as the foundations for attack modelling.

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.

Since their first introduction, attack trees have been used to describe attacks against various real-world applications like Border Gateway Protocol [8], SCADA protocols [9] and e-voting infrastructures [10]. Attack trees have found their place in computer science education [11] and several support tools like AttackTree+[4] and SecurITree[5] have been developed.

Early approaches to attack tree modelling were mostly concerned with just categorising the attacks [8] or modelling the attacker's behaviour by one specific parameter of the attacks like the cost, difficulty or severity [5,

[4] http://www.isograph-software.com/atpover.htm
[5] http://www.amenaza.com/

9, 12]. A substantial step forward was taken by Buldas *et al.* [13] who introduced the idea of game-theoretic modelling of the attacker's decision making process based on several interconnected parameters like the cost, risks and penalties associated with different elementary attacks. This approach was later refined by Jürgenson and Willemson [14, 15] and applied to the analysis of the security of several e-voting solutions by Buldas and Mägi [10].

So far, practically all the research in the field of attack trees has concentrated on what one could call a parallel model [4, 5, 3, 8, 9, 16, 12–14, 10, 15]. Essentially, the model assumes that all the elementary attacks take place simultaneously and hence the attacker's possible decisions based on success or failure of some of the elementary attacks are ignored. However, as noted already in [15], this model is unrealistic. In practice, the attacker is able to order his actions and try different alternative scenarios if some others fail or to stop trying altogether if some critical subset of elementary attacks has already failed or succeeded. Not risking with the hopeless or unnecessary attempts clearly reduces the amount of potential penalties and hence increases the attacker's expected outcome.

The main contribution of this paper is to surpass this shortcoming by introducing what one could call a serial model for attack trees. We extend the basic parallel model with temporal order of the elementary attacks and give the attacker some flexibility in skipping some of them or stopping the attack before all of the elementary attacks have been tried. The other contribution is a generalisation of the attack tree approach to accommodate arbitrary rooted directed acyclic graphs, which will enable us to conveniently ensure consistency of our computations in the general framework proposed by Mauw and Oostdijk [12].

The paper is organised as follows. In Section 2 we first briefly review the basic multi-parameter attack tree model. Sections 3 and 4 extend it by introducing attack descriptions based on general Boolean functions and temporal order of elementary attacks, respectively. Section 5 presents an efficient algorithm for computing the attacker's expected outcome of the attack tree with the predefined order of leaves. Finally, Section 6 draws some conclusions and sets directions for further work.

2 The Attack Tree Model

Basic idea of the attack tree approach is simple – the analysis begins by identifying one *primary threat* and continues by dividing 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 divide the resulting attacks any more; these kinds of non-splittable attacks are called *elementary 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 practice, 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.

The crucial contribution of Buldas *et al.* [13] was the introduction of four game-theoretically motivated parameters for each leaf node of the tree. This approach was later optimised in [15], where the authors concluded that only two parameters suffice. Following their approach, we consider the set of elementary attacks $\mathcal{X} = \{X_1, X_2, \ldots, X_n\}$ and give each one of them two parameters:

- p_i – success probability of the attack X_i,
- $\mathsf{Expenses}_i$ – expected expenses (i.e. costs plus expected penalties) of the attack X_i.

Besides these parameters, there is a global value Gains expressing the benefit of the attacker if he is able to materialise the primary threat.

In the parallel model of [15], the expected outcome of the attacker is computed by maximising the expression

$$\mathsf{Outcome}_S = p_S \cdot \mathsf{Gains} - \sum_{X_i \in S} \mathsf{Expenses}_i \qquad (1)$$

over all the assignments $S \subseteq \mathcal{X}$ that make the Boolean formula $\mathcal{F}$, represented by the attack tree, true. (Here p_S denotes the success probability of the primary threat.) Like in the original model of Buldas *et al.* [13], we assume that the attacker behaves rationally, i.e. he attacks only if there is an attack scenario with a positive outcome. The defender's task is thus achieving a situation where all the attack scenarios would be non-beneficial for the attacker.

Our aim is to develop this model in two directions. In Section 3 we will generalise the attack tree model a bit to allow greater flexibility and expressive power of our model, and in Section 4 we will study the effects of introducing linear (temporal) order to the set of elementary attacks.

3 Attack Descriptions as Monotone Boolean Functions

Before proceeding, we briefly discuss a somewhat different perspective on attack tree construction. Contrary to the standard top-down ideology popularised by Schneier [5], a bottom-up approach is also possible. Say, our attacker has identified the set of elementary attacks $\mathcal{X}$ available to him and he needs to figure out, which subsets of $\mathcal{X}$ are sufficient to mount the root attack. In this paper we assume that the set of such subsets is monotone, i.e. if some set of elementary attacks suffices, then so does any of its supersets. This way it is very convenient to describe all the successful attacks by a monotone Boolean function $\mathcal{F}$ on the set of variables $\mathcal{X}$.

Of course, if we have constructed an attack tree then it naturally corresponds to a Boolean function. Unfortunately, considering only the formulae that have a tree structure is not always enough. Most notably, trees can not handle the situation, where the same lower-level attack is useful in several, otherwise independent higher-level attacks, and this is clearly a situation we can not ignore in practical security analysis.

Another shortcoming of the plain attack tree model follows from the general framework by Mauw and Oostdijk [12]. They argue that the semantics of an attack tree is inherently consistent if and only if the tree can be transformed into an equivalent form without changing the value of the expected outcome. When stating and proving their result, they essentially transform the underlying Boolean formula into a disjunctive normal form, but when doing so, they need to introduce several copies of some attacks, therefore breaking the tree structure in favour of a general rooted directed acyclic graph (RDAG). Since AND-OR-RDAGs are equivalent to monotone Boolean functions, there is no immediate need to take the generalisation any further.

Thus it would be more consistent and fruitful not to talk about *attack trees*, but rather *attack RDAGs*. On the other hand, as the structure of a tree is so much more convenient to analyse than a general RDAG, we should still try to stick to the trees whenever possible. We will see one specific example of a very efficient tree analysis algorithm in Section 5.

4 Ordering Elementary Attacks

After the attacker has selected the set of possible elementary attacks $\mathcal{X}$ and described the possible successful scenarios by means of a monotone Boolean function $\mathcal{F}$, he can start planning the attacks. Unlike the naïve parallel model of Schneier [5], the attacker has a lot of flexibility and choice. He may try some elementary attack first and based on its success or failure select the next elementary attack arbitrarily or even decide to stop attacking altogether (e.g. due to certain success or failure of the primary threat). Such a fully adaptive model is still too complicated to analyse with the current methods, thus we will limit the model to be semi-adaptive. I.e., we let the attacker to fix linear order of some elementary attacks in advance and assume that he tries them in succession, possibly skipping superfluous elementary attacks and stopping only if he knows that the Boolean value of $\mathcal{F}$ has been completely determined by the previous successes and failures of elementary attacks.

The full strategy of the attacker will be the following.

1. Create an attack RDAG with the set of leaf nodes $\mathcal{X} = \{X_1, X_2, \dots, X_n\}$.
2. Select a subset $S \subseteq \mathcal{X}$ materialising the primary threat and consider the corresponding subtree.
3. Select a permutation α of S.
4. Based on the subtree and permutation α, compute the expected outcome.
5. Maximise the expected outcome over all the choices of S and α.

This paper is mostly concerned with item 4 in the above list, but doing so we must remember that when building a complete attack analysis tool, other items can not be disregarded either. Optimisations are possible, e.g. due to monotonicity there is no need to consider any subsets of attack suites that do not materialise the primary threat. Even more can be done along the lines of [15], Section 4.1, but these aspects remain outside of the scope of the current paper.

Since only one subset S and the corresponding subtree are relevant in the above step 4, we can w.l.o.g. assume that $S = \mathcal{X}$. The attacker's behaviour for permutation α will be modelled as shown in Algorithm 1.

Consider the example attack tree depicted in Figure 1, where we assume $\alpha = id$ for better readability.

The attacker starts off by trying the elementary attack X_1. Independent of whether it succeeds or fails, there are still other components needed to complete the root attack, so he tries X_2 as well. If it fails, we see that the whole tree fails, so it does not make sense to try X_3 and X_4. If both X_1 and X_2 have succeeded, we see that it is not necessary to try X_3, since X_1 and X_3 have a common OR-parent, so success or failure of X_4 determines the final outcome. If X_1 fails and X_2 succeeds, we need the success of both X_3 and X_4 to complete the task; if one of them fails, we stop and accept the failure.

Algorithm 1 Perform the attack

```
Require: The set of elementary attacks $\mathcal{X} = \{X_1, X_2, \ldots, X_n\}$, permutation $\alpha \in S_n$
    and a monotone Boolean formula $\mathcal{F}$ describing the attack scenarios
 1: for $i := 1$ to $n$ do
 2:     Consider $X_{\alpha(i)}$
 3:     if success or failure of $X_{\alpha(i)}$ has no effect on the success or failure of the root
        node then
 4:         Skip $X_{\alpha(i)}$
 5:     else
 6:         Try to perform $X_{\alpha(i)}$
 7:         if the root node succeeds or fails then
 8:             Stop
 9:         end if
10:     end if
11: end for
```

The expected outcome of the attack based on permutation α will be defined as

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

where p_α is the success probability of the primary threat and $p_{\alpha,i}$ denotes the probability that the node X_i is encountered during Algorithm 1. Before proceeding, we will prove that the expected outcome of Algorithm 1 does not depend on the specific form of the formula $\mathcal{F}$. This essentially gives us the compliance of our attack tree model in the framework of Mauw and Oostdijk [12]. Formally, we will state and prove the following theorem, similar to Proposition 1 in [15].

Theorem 1. *Let $\mathcal{F}_1$ and $\mathcal{F}_2$ be two monotone Boolean formulae such that $\mathcal{F}_1 \equiv \mathcal{F}_2$, and let $\mathsf{Outcome}^1_\alpha$ and $\mathsf{Outcome}^2_\alpha$ be the expected outcomes obtained running Algorithm 1 on the corresponding formulae. Then*

$$\mathsf{Outcome}^1_\alpha = \mathsf{Outcome}^2_\alpha\,.$$

Proof. We can observe that Algorithm 1 really does not depend on the attack description having a tree structure, all the decisions to skip or stop can be taken based on the Boolean function $\mathcal{F}$. Assume we have already fixed the results of the elementary attacks $X_{\alpha(1)}, \ldots, X_{\alpha(i-1)}$. Then we see that

– the node $X_{\alpha(i)}$ may be skipped if for all the values of $X_{\alpha(i+1)}, \ldots, X_{\alpha(n)}$ we have

$$\mathcal{F}\left(X_{\alpha(1)}, \ldots, X_{\alpha(i-1)}, t, X_{\alpha(i+1)}, \ldots, X_{\alpha(n)}\right) =$$

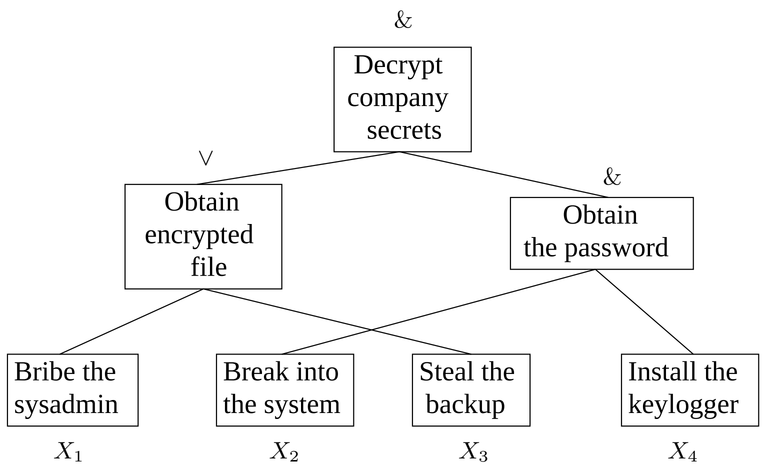

Fig. 1. An example attack tree. The left-to-right ordering of the leaf nodes in the tree represents the permutation $\alpha = id$ of the set $\mathcal{X} = \{X_1, X_2, X_3, X_4\}$.

$$= \mathcal{F}\left(X_{\alpha(1)}, \ldots, X_{\alpha(i-1)}, f, X_{\alpha(i+1)}, \ldots, X_{\alpha(n)}\right) ,$$

– there is no need to proceed with Algorithm 1 after the node $X_{\alpha(i)}$ if for all the values of $X_{\alpha(i+1)}, \ldots, X_{\alpha(n)}$ we have

$$\mathcal{F}\left(X_{\alpha(1)}, \ldots, X_{\alpha(i-1)}, X_{\alpha(i)}, X_{\alpha(i+1)}, \ldots, X_{\alpha(n)}\right) = t$$

or

$$\mathcal{F}\left(X_{\alpha(1)}, \ldots, X_{\alpha(i-1)}, X_{\alpha(i)}, X_{\alpha(i+1)}, \ldots, X_{\alpha(n)}\right) = f .$$

□

Thus, our serial model for attack trees follows the guidelines given in Section 3 and it really is safe to talk about Boolean functions describing the attack scenarios.

Next we will show formally that introducing order to the elementary attacks really increases the attacker's expected outcome. Comparing (2) to (1) we get the following theorem.

Theorem 2. *Let $\mathcal{F}$ be a monotone Boolean function on $n \geq 2$ variables describing the attack scenarios. Let $\mathsf{Outcome}_\alpha$ be defined by (2) and let $\mathsf{Outcome}_\mathcal{X}$ be defined by (1) for $S = \mathcal{X}$. Then we have*

$$\mathsf{Outcome}_\alpha \geq \mathsf{Outcome}_\mathcal{X} . \tag{3}$$

If for all the elementary attacks X_i ($i = 1, \ldots, n$) one also has $\mathsf{Expenses}_i > 0$, then strict inequality holds in (3).

Proof. First we note that by [15] we can compute the success probability of the attacker as follows:

$$p_{\mathcal{X}} = \sum_{\substack{S \subseteq \mathcal{X} \\ \mathcal{F}(S := \mathsf{true}) = \mathsf{true}}} \prod_{X_i \in S} p_i \prod_{X_j \in \mathcal{X} \setminus S} (1 - p_j)\,,$$

where $\mathcal{F}(S := \mathsf{true})$ denotes evaluation of the Boolean function $\mathcal{F}$, when all the variables of S are assigned the value true and all others the value false. This is exactly the total probability of all the successful branches of Algorithm 1 and thus $p_{\mathcal{X}} = p_\alpha$ (implying that p_α is actually independent of α). We also have that $\forall i\, p_{\alpha,i} \leq 1$ and hence the inequality (3) follows.

Assume now that for all X_i we have $\mathsf{Expenses}_i > 0$. Then in order to prove that strict inequality holds in (3), we need to show that there exists such an index i that $p_{\alpha,i} < 1$. Consider the elementary attack $X_{\alpha(n)}$ that the attacker is supposed to try last. If there exists an evaluation of the Boolean variables $X_{\alpha(1)}, \ldots, X_{\alpha(n-1)}$ such that

$$\mathcal{F}\left(X_{\alpha(1)}, \ldots, X_{\alpha(n-1)}, t\right) = \mathcal{F}\left(X_{\alpha(1)}, \ldots, X_{\alpha(n-1)}, f\right)\,,$$

then $X_{\alpha(n)}$ is superfluous in this scenario and hence $p_{\alpha,n} < 1$.

If on the other hand we have

$$\mathcal{F}\left(X_{\alpha(1)}, \ldots, X_{\alpha(n-1)}, t\right) \neq \mathcal{F}\left(X_{\alpha(1)}, \ldots, X_{\alpha(n-1)}, f\right)$$

for all evaluations of $X_{\alpha(1)}, \ldots, X_{\alpha(n-1)}$, then due to monotonicity of $\mathcal{F}$ we can only have that

$$\mathcal{F}\left(X_{\alpha(1)}, \ldots, X_{\alpha(n-1)}, f\right) = f$$

and

$$\mathcal{F}\left(X_{\alpha(1)}, \ldots, X_{\alpha(n-1)}, t\right) = t\,,$$

implying $\mathcal{F}(Y_1, \ldots, Y_n) \equiv Y_n$. But in this case all the elementary attacks before the last one get skipped, so $p_{\alpha,1} = \ldots = p_{\alpha,n-1} = 0$. $\square$

Thus, introducing ordering of the elementary attacks is guaranteed to give at least as good a result to the attacker as the routine described in [15]. In the interesting case, when all attack components have positive expenses, the attacker's expected outcome is strictly larger.

5 Computing the Expected Outcome

There are $n+1$ parameters that need to be computed in order to find the expected outcome using the formula (2) – the total success probability p_α and the probabilities $p_{\alpha,i}$ that the node X_i is encountered during Algorithm 1. It turns out that there is an efficient algorithm for computing these quantities provided that the given monotone Boolean function can actually be described by a tree. In what follows we will also assume that the tree is binary, but this restriction is not a crucial one.

So let us have an attack tree with the leaf nodes $X_1, \ldots, X_n$ and the corresponding success probabilities p_i, $i = 1, \ldots, n$. We will assume that all these probabilities are independent and consider the permutation $\alpha \in S_n$. In order to explain the algorithm, we first introduce three extra parameters to each node Y, namely $Y.t$, $Y.f$ and $Y.u$ showing the probabilities that the node has been proven to be respectively true, false or yet undefined in the course of the analysis. Initially, we may set $Y.t = Y.f = 0$ and $Y.u = 1$ for all the nodes and the algorithm will work by incrementally adjusting these values, so that in the end of the process we will have $R.t = p_\alpha$ for the root node R. Throughout the computations we will of course retain the invariant $Y.t + Y.f + Y.u = 1$ for all the nodes Y, hence one of these parameters is actually superfluous. In the presentation version of the algorithm we will drop the parameter $Y.u$, even though it actually plays the central role.

Going back to the high-level description of Algorithm 1, we see that the most difficult step is step 3, where the attacker is supposed to find out whether the next elementary attack in his list may have any effect on the success or failure of the root node. Elementary attack does not have any effect iff there is a node on the path from that particular leaf to the root that has already been proven to be true or false. Thus the next elementary attack should be tried iff all the nodes on this path are undefined – and this is precisely the event that gives us the required probability $p_{\alpha,i}$.

Let the path from root R to the leaf X_i then be $(Y_0 = R, Y_1, \ldots, Y_m = X_i)$. Thus, we need to compute the probability

$$\begin{aligned}
p_{\alpha,i} &= \Pr[Y_0 = u \,\&\, Y_1 = u \,\&\, \ldots \,\&\, Y_m = u] = \\
&= \Pr[Y_0 = u \mid Y_1 = u\,,\ \ldots\,,\ Y_m = u] \cdot \\
&\quad \cdot \Pr[Y_1 = u \mid Y_2 = u\,,\ \ldots\,,\ Y_m = u] \cdot \ldots \\
&\quad \ldots \cdot \Pr[Y_{m-1} = u \mid Y_m = u] \cdot \Pr[Y_m = u] = \\
&= \Pr[Y_0 = u \mid Y_1 = u] \cdot \Pr[Y_1 = u \mid Y_2 = u] \cdot \ldots \\
&\quad \ldots \cdot \Pr[Y_{m-1} = u \mid Y_m = u] \cdot \Pr[Y_m = u]
\end{aligned} \qquad (4)$$

The equations

$$\Pr[Y_k = u \,|\, Y_{k+1} = u\,,\,\dots\,,\,Y_m = u] = \Pr[Y_k = u \,|\, Y_{k+1} = u]$$

hold due to the tree structure of our underlying RDAG and the independence assumption of the elementary attacks. In (4) we have $\Pr[Y_m = u] = \Pr[X_i = u] = 1$ and all the other probabilities are of the form $\Pr[Y_k = u \,|\, Y_{k+1} = u]$. Hence, we need to evaluate the probability that the parent node Y_k is undefined assuming that one of its children, Y_{k+1}, is undefined. This probability now depends on whether Y_k is an AND- or OR-node. If Y_k is an AND-node and Y_{k+1} is undefined, then so is Y_k, if its other child Z is either true or undefined, which is the case with probability $Z.t + Z.u = 1 - Z.f$. Similarly, if Y_k is an OR-node and Y_{k+1} is undefined, then so is Y_k, if its other child Z is either false or undefined, which is the case with probability $Z.f + Z.u = 1 - Z.t$.

This way, (4) gives an efficient way of computing $p_{\alpha,i}$ assuming that the current parameters of the internal nodes of the tree are known. Hence, we need the routines to update these as well. These routines are straightforward. If the elementary attack X_i is tried, only the parameters of the nodes on the path $(Y_m = X_i, \dots, Y_1, Y_0 = R)$ from that leaf to the root need to be changed. We do it by first setting $Y_m.t = p_i$, $Y_m.f = 1 - p_i$ and $Y_m.u = 0$ and then proceed towards the root. If the node we encounter is AND-node A with children B and C, we set

$$A.t = B.t \cdot C.t\,, \tag{5}$$
$$A.f = B.f + C.f - B.f \cdot C.f\,, \tag{6}$$

and if we encounter an OR-node A with children B and C, we set

$$A.t = B.t + C.t - B.t \cdot C.t\,, \tag{7}$$
$$A.f = B.f \cdot C.f\,. \tag{8}$$

As noted above, we see that the quantities $Y.u$ are actually never needed in the computations.

This way we get the full routine described as Algorithm 2.

Algorithm 2 is very efficient. In order to compute the $n + 1$ necessary probabilities, it makes one run through all the leaves of the tree and at each run the path from the leaf to the root is traversed twice. Since the number of vertices on such a path in a (binary) tree can not be larger than the number of leaves n, we get that the worst-case time complexity of Algorithm 2 is $O(n^2)$. If the tree is roughly balanced, this estimate

Algorithm 2 Computing the probabilities $p_{\alpha,i}$

```
Require: An attack tree with leaf set $\mathcal{X} = \{X_1, X_2, \ldots, X_n\}$ and a permutation
    $\alpha \in S_n$
Ensure: The probabilities $p_{\alpha,i}$ for $i = 1, 2, \ldots, n$
 1: for all $Z \in \{X_1, \ldots, X_n\}$ do
 2:     $Z.t := 0,\ Z.f := 0$
 3: end for
 4: for $i := 1$ to $n$ do
 5:     Find the path $(Y_0, Y_1, \ldots, Y_m)$ from the root $Y_0 = R$ to the leaf $Y_m = X_{\alpha(i)}$
 6:     $p_{\alpha,\alpha(i)} := \prod_{j=1}^{m}(1 - Z_j.a)$, where $Z_j$ is the sibling node of $Y_j$ and
            $a = \begin{cases} t, & \text{if } Y_{j-1} \text{ is an OR-node,} \\ f, & \text{if } Y_{j-1} \text{ is an AND-node} \end{cases}$
 7:     $X_{\alpha(i)}.t = p_{\alpha(i)}$
 8:     $X_{\alpha(i)}.f = 1 - p_{\alpha(i)}$
 9:     Update the parameters of the nodes $Y_{m-1}, Y_{m-2}, \ldots, Y_0$ according to formulae
        (5)–(8)
10: end for
```

drops even to $O(n \log n)$. This is a huge performance increase compared to a naïve algorithm that one could design based on the complete attack scenario analysis described after Figure 1 in Section 4. We studied the naïve algorithm and it turns out that it is not only worst-case exponential, but also average-case exponential [17].

Of course, as noted in Section 4, Algorithm 2 is only one building block in the whole attack tree analysis. In order to find out the best attack strategy of the attacker, we should currently consider all the subsets of $\mathcal{X}$ and all their permutations. Optimisation results presented in [14] give a strong indication that a vast majority of the possible cases can actually be pruned out, but these methods remain outside of the scope of the current paper.

6 Conclusions and Further Work

In this paper we studied the effect of introducing a temporal order of elementary attacks into the attacker's decision making process together with some flexibility in retreating of some of them. It turns out that taking temporal dependencies into account allows the attacker to achieve better expected outcomes and as such, it brings the attack tree model one step closer to the reality. This reality comes for a price of immense increase in computational complexity, if we want to compute the attacker's exact outcome by considering all the possible scenarios in a naïve way.

Thus there are two main challenges for the future research. First, one may try to come up with optimisations to the computational process and in this paper we showed one possible optimisation which works well for attack trees. The second approach is approximation. In attack tree analysis we are usually not that much interested in the exact maximal outcome of the attacker, but we rather want to know whether it is positive or negative. This observation gives us huge potential for rough estimates, which still need to be studied, implemented and tried out in practice.

In this paper we limited ourselves to a semi-adaptive model, where the attacker is bound to the predefined order of elementary attacks and may only choose to drop some of them. Fully adaptive case where the attacker may choose the next elementary attack freely is of course even more realistic, but it is currently too complicated to analyse. Our model is also non-blocking in the sense that there are no elementary attacks, failure of which would block execution of the whole tree. However, in practice it happens that when failing some attack, the attacker might get jailed and is unable to carry on. Hence, future studies in the area of adaptive and possibly-blocking case are necessary.

As a little technical contribution we also discussed the somewhat inevitable generalisation of attack trees to RDAGs, but our results also show that whenever possible, we should still stick to the tree structure. Possible optimisations of RDAG-based algorithms remain the subject for future research as well.

7 Acknowledgments

This research was supported by Estonian Science Foundation grant no 7081. The authors are grateful to Margus Niitsoo for his discussions and helpful comments.

References

1. Vesely, W., Goldberg, F., Roberts, N., Haasl, D.: Fault Tree Handbook. US Government Printing Office (January 1981) Systems and Reliability Research, Office of Nuclear Regulatory Research, U.S. Nuclear Regulatory Commission.
2. Viega, J., McGraw, G.: Building Secure Software: How to Avoid Security Problems the Right Way. Addison Wesley Professional (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. (1991) 572–581

5. Schneier, B.: Attack trees: Modeling security threats. Dr. Dobb's Journal **24**(12) (December 1999) 21–29
6. Edge, K.S.: A Framework for Analyzing and Mitigating the Vulnerabilities of Complex Systems via Attack and Protection Trees. PhD 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. Convery, S., Cook, D., Franz, M.: An attack tree for the border gateway protocol. IETF Internet draft (Feb 2004) Available at `http://www.ietf.org/proceedings/04aug/I-D/draft-ietf-rpsec-bgpattack-00.txt`.
9. Byres, E., Franz, M., Miller, D.: The use of attack trees in assessing vulnerabilities in SCADA systems. In: International Infrastructure Survivability Workshop (IISW'04), IEEE, Lisbon, Portugal. (2004)
10. 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. Volume 4752 of LNCS., Springer (2007) 320–335
11. Saini, V., Duan, Q., Paruchuri, V.: Threat modeling using attack trees. J. Comput. Small Coll. **23**(4) (2008) 124–131
12. Mauw, S., Oostdijk, M.: Foundations of attack trees. In Won, D., Kim, S., eds.: International Conference on Information Security and Cryptology – ICISC 2005. Volume 3935 of LNCS., Springer (2005) 186–198
13. Buldas, A., Laud, P., Priisalu, J., Saarepera, M., Willemson, J.: Rational Choice of Security Measures via Multi-Parameter Attack Trees. In: Critical Information Infrastructures Security. First International Workshop, CRITIS 2006. Volume 4347 of LNCS., Springer (2006) 235–248
14. Jürgenson, A., Willemson, J.: Processing multi-parameter attacktrees with estimated parameter values. In Miyaji, A., Kikuchi, H., Rannenberg, K., eds.: Advances in Information and Computer Security, Second International Workshop on Security, IWSEC. Volume 4752 of LNCS., Springer (2007) 308–319
15. Jürgenson, A., Willemson, J.: Computing exact outcomes of multi-parameter attack trees. In: On the Move to Meaningful Internet Systems: OTM 2008. Volume 5332 of LNCS., Springer (2008) 1036–1051
16. Opel, A.: Design and implementation of a support tool for attack trees. Technical report, Otto-von-Guericke University (March 2005) Internship Thesis.
17. Jürgenson, A., Willemson, J.: Ründepuud: pooladaptiivne mudel ja ligikaudsed arvutused (in Estonian). Technical Report T-4-4, Cybernetica, Institute of Information Security (2009) `http://research.cyber.ee/`.