Optimal Adversary Behavior for the Serial Model of Financial Attack Trees

Margus Niitsoo

University of Tartu, Liivi 2, 50409 Tartu, Estonia
Cybernetica AS, Akadeemia tee 21,12618 Tallinn, Estonia.
margus.niitsoo@ut.ee

Abstract. Attack tree analysis is used to estimate different parameters of general security threats based on information available for atomic subthreats. We focus on estimating the expected gains of an adversary based on both the cost and likelihood of the subthreats. Such a multi-parameter analysis is considerably more complicated than separate probability or skill level estimation, requiring exponential time in general. However, this paper shows that under reasonable assumptions a completely different type of optimal substructure exists which can be harnessed into a linear-time algorithm for optimal gains estimation. More concretely, we use a decision-theoretic framework in which a rational adversary sequentially considers and performs the available attacks. The assumption of rationality serves as an upper bound as any irrational behavior will just hurt the end result of the adversary himself. We show that if the attacker considers the attacks in a goal-oriented way, his optimal expected gains can be computed in linear time. Our model places the least restrictions on adversarial behavior of all known attack tree models that analyze economic viability of an attack and, as such, provides for the best efficiently computable estimate for the potential reward.

1 Introduction

Assessing the security of a computer system has become an increasingly more important problem in the past decade due to the widespread use of computer communications. To date, many interesting graph-based solutions have been provided (see [14] for a review) that try to address the problem of threat estimation.

Although the approaches vary greatly in their scope and methodologies, most of them have two things in common. First, they tend to concentrate on the technical aspects of computer systems such as network topology and software flaws. Secondly, the emphasis is usually placed on finding and describing the possible attack vectors, giving less emphasis on the analysis of attack feasibility or likelihood. For instance, [2, 9] concentrate on semi-automatic generation of attack graphs and then only do rather simple path-enumeration and cut-set analysis on what they have generated. Both approaches have since been developed further but the emphasis still seems to remain on fast and accurate graph generation.

There are some major problems with these trends, however. The emphasis on technical aspects such as software vulnerabilities might not be the most fruitful path of inquiry. For instance, Bilar [3] found that for 6 operating systems used at that time, nearly all known vulnerabilities would need to be patched in order to achieve significant reduction in the probability of compromise and even then the probability would remain reasonably high. This hints that most computer systems are usually insecure to some degree on the technical level and that instead of asking whether the break is possible, we should ask how probable it actually is. This question comes into perspective when we consider the increasing number of successful social engineering attacks, which are usually much simpler, yet often just as effective, as clever technical hacking can be.

Another flaw with the emphasis on technical aspects is that the assumptions about adversarial behavior are often overly simplistic. For instance, it is generally assumed that a system is insecure as soon as a possible means of penetration is found. This seemingly reasonable assumption is, however, not always justified. Knowing that there exists a way to penetrate the system is akin to knowing that it is possible to pick the lock on the front door of a house. However, it is obvious that whether the house will actually be burgled depends

much more on whether there is something valuable inside, how high the probability of getting caught is and what the penalties for breaking and entering are. Just considering the quality of the lock on the door or the strength of the bars on the windows will definitely give some information about the security, but it might not tell much about the actual likelihood of an attack.

This means that although the analysis of technical level vulnerability is of importance, it cannot provide for the complete analysis of security. It is also important to develop models that work on a higher level of abstraction and allow the integration of social engineering attacks alongside the more technical possibilities. It would also make more sense to concentrate on the incentives and possibilities available to the adversary and try to analyze his behavior rather than simply determining whether the attack is possible or not.

The attack tree (also called threat tree) approach to security evaluation is not a recent development but has roots that reach back several decades. Its beginnings can be traced to Fault tree analysis methods that were developed at Bell Laboratories in 1961 (see [7]). Throughout the years, it has been used for tasks like fault assessment of critical systems [18] and software vulnerability analysis [19, 16]. The approach was introduced for the study of information system security by Weiss [21] and made popular in that context by Bruce Schneier [17]. We refer to [6, 8] for good overviews of the development and application of the methodology.

Although Weiss [21] already realized that the attack trees can have many parameters, initial work concentrated on estimating just one of them at a time. The cost, feasibility and the skill level required for the attack have all been independently considered for analysis by different authors [16, 17, 15]. There is even a software package [1] on the market for performing such analyses[1].

Substantial progress was made in 2006 by Buldas et al. [4] who introduced a multi-parameter game-theoretic model which allowed estimation of the expected utility of the attacker with only a linear amount of computation. The model was later used in practice by Buldas and Mägi [5] to evaluate the security of several e-voting schemes in use at that time.

However, the model of [4] was somewhat ad-hoc and turned out to be theoretically unsound. This was noticed by Jürgenson and Willemson [13], who in turn proposed a modification that resulted in a sound model for parallel adversary behavior in which the adversary has to attempt all the attacks at the same time in parallel. However, it seems that exponential running time is required to determine the maximal possible expected utility of the adversary. This means that the model is impractical for all but the smallest attack trees.

Jürgenson and Willemson went on to consider a model of serial attacks [12] in which the adversary performs the attacks in a prescribed order and has full information about what the results of the previous attacks are. For that model they provide a quadratic time algorithm for calculating the expected utility for the adversary. Their model is sound, but only under a somewhat weird assumption on adversarial behavior. To be precise, they (implicitly) assume that the adversary always performs an elementary attack whenever doing so increases the chance of materializing the primary threat. This violates the assumption of economic rationality because it is easy to envision a scenario where an elementary attack costs more than the increased probability of materializing the primary threat is worth. In such a case a rational adversary would skip the attack, but in their model it is performed nevertheless. To address this flaw, they consider a subset of the elementary attacks with the largest expected value. However, there is still no guarantee that this would produce optimal results in terms of economic theory. Another problem with the subset idea is that finding the optimal subset seems to require an exponential amount of computation. This means that their more general model is much too slow for practical applications.

[1] We note that although multiple parameters are used by the software package, one of the parameters is used just for tree pruning and as such, their computational analysis is still single-parameter in essence.

We start out with a systematic decision-theoretic approach to adversarial behavior analysis and show that there actually exists a linear-time algorithm for computing the maximal expected outcome for an exponential sized family of attack orders. Our model is also one of the least restrictive models in terms of assumptions about adversarial behavior. This means that the expected utility of the adversary that is computed by our proposed algorithm is the highest among the currently known efficient computation methods and as such can serve as an upper bound for all of them.

2 Attack Trees

In security analysis, one is interested in estimating the parameters of some large *primary threat*. The adversary usually has many ways of materializing the primary threat and it is often possible to break it down into smaller parts such that either all or at least one of the parts need to be realized for the main attack to succeed. As many of these parts can then also be broken down in a similar manner, a tree structure is formed with increasingly simpler attack scenarios at the nodes. If one continues decomposing the threats, one is bound to eventually reach a point where the attack considered is simple enough such that its parameters can be directly estimated. When this happens, the threat is not decomposed further, but is left as a leaf of the tree and called *elementary*. When all the branches have been stretched out to that level of detail, the process is stopped and the result is called the *attack tree* corresponding to the primary threat at its root. A small example of an attack tree is depicted in Figure 1.

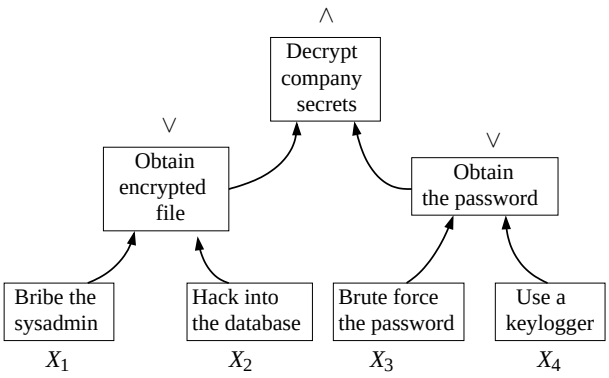

Fig. 1. An example attack tree

For a given attack tree, we denote the set of all its leaf elementary attacks by $\mathcal{X} = \{X_1, \ldots, X_n\}$. The internal nodes of the tree that describe the primary threat and its non-elementary subgoals can be of two distinct types. If all of the parts of the subgoal need to succeed for the subgoal to be considered successful, the node is called an AND-node ($\wedge$-node). If just one successful part is enough for the subgoal to be successful, it is called an OR-node ($\vee$-node). These two node types are usually sufficient to describe all the possible variations of the attacks.

This means that an attack tree is usually just an AND-OR tree in which the primary threat is realized precisely when the root of the AND-OR tree is satisfied. The problem can also be expressed in terms of Boolean functions $\mathcal{F}$ on elementary attacks as the AND-OR tree can easily be converted into a (monotone) Boolean formula of a certain simple form. For example, the attack tree in Figure 1 corresponds to the Boolean formula $\mathcal{F}(X_1, X_2, X_3, X_4) = (X_1 \vee X_2) \wedge (X_3 \vee X_4)$. The attention is usually restricted to trees because

they usually provide for a model that is simpler both computationally as well as construction (for a specific security threat). In our case, however, it makes sense to talk about more general attack *scenarios* where $\mathcal{F}$ can be any Boolean function.

These scenarios can still be analyzed in different ways depending on what parameters of the attack are taken into consideration and what assumptions are made about the adversarial behavior. Our approach follows that of Jürgenson and Willemson [11] who simplify the model of Buldas et al. [4]. Their model has just two parameters for each elementary attack X_i:

1. The probability of the attack succeeding p_i.
2. The expected expenses e_i when attempting the attack.

Additionally, there is one global parameter g that denotes the expected utility that materializing the primary threat has for the adversary. We note that the expenses are meant to include not only the immediate expenses of performing the attack but also accounts for the expected cost of possible penalties that need to be paid if the attacker is caught in the act. How this value can be calculated is described in detail in [13]. We also note that it is reasonable to assume that the adversary does not get paid for attempting the attacks, i.e., the expenses are all non-negative. It is also natural to assume that g is positive for otherwise the attacker would not be motivated to even attempt the attack. These parameters allow one to carry out an economic analysis in which it is assumed that the adversary tries to maximize his expected outcome.

In the serial model of attack, the adversary is assumed to try the elementary attacks one at a time in such a way that he always knows what has happened on the previous attacks. This may lead to the adversary not attempting an attack, for instance when it would have no impact on the final outcome any more, i.e., performing it would not increase the probability of winning g. However, whether this is the case or not largely depends on the order of the elementary attacks. In real life, the elementary attacks often have a natural order imposed on them by practical restrictions. For example, breaking into the company safe requires that the culprit first break into the office itself so the latter clearly has to precede the former in the order of attacks. For simplicity we indeed assume that the order of the attacks has been fixed beforehand and that the adversary has to consider the attacks in just the order they are given, starting with X_1 and ending with X_n. Figure 2 depicts two different possible orders for the example attack tree of Figure 1.

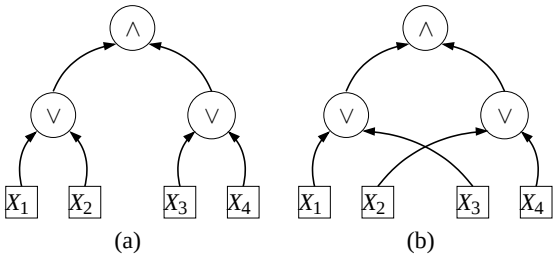

Fig. 2. Two different orderings of the tree in Figure 1.

Although we assume that the order of consideration is fixed for the attacks, the adversary still has a choice for each elementary attack of whether to commit it or not. If the past outcomes imply that performing the attack does not increase the likelihood of the main threat succeeding, one can definitely gain by choosing not to attempt the attack. This was the intuition behind the model of Jürgenson and Willemson [12].

The model they propose has the adversary always attempting an attack whenever it increases the probability of materializing the main threat. They show that their model is sound in the sense of Mauw and

Oostdijk [15] and that computing the optimal adversary behavior can be done in $O(n^2)$ time for the tree model for all the possible orders.

However, their model suffers from one critical flaw. Namely, it may sometimes happen that an attack costs more than it is worth in the sense that the probability increase of the main threat is so small that it does not outweigh the cost of performing the given elementary attack. Jürgenson and Willemson attempt to fix this flaw by saying they consider all the possible subsets of elementary attacks and try to find the optimal subset. This does not really solve the problem, however, as it may still be rational to perform the attack for one past while for the other it is not. We illustrate this with a small example.

Suppose we have the attack tree depicted in Figure 2(b) with $e_1 = e_2 = 0$, $e_3 = e_4 = 80$, $p_1 = p_2 = 0.5$ and $p_3 = p_4 = 0.9$ with $g = 100$. In the serial model of Jürgenson and Willemson all attacks will be attempted whenever they increase the probability of success. This means that X_1 and X_2 will always be performed whereas X_3 will be performed precisely when X_1 fails and X_4 will be performed when either X_1 or X_3 succeeds, but X_2 fails. This is optimal behavior except in the case where X_1 and X_2 both fail. After that, one would need both X_3 and X_4 to succeed in order to be awarded the gains g. For that to happen, both X_3 and X_4 need to be attempted, costing 160. This is obviously not worth doing as the maximal possible gains from it are just 100 units when the attack succeeds *if* it succeeds. Despite this, X_3 is still attempted by the model because there still is a theoretical chance of winning the gains. This is clearly not economically rational.

The optimal subset improvement does not solve the problem either because there are cases where both X_2 and X_3 are worth attempting and increase the expected utility. This means that removing either one of them will also decrease the maximal expected outcome. The only possible conclusion is that there exists a behavior strategy for the adversary that is strictly better than is describable by the serial model of Jürgenson and Willemson.

3 Our Model

To overcome the weaknesses in the previous serial model, we approach the problem from the classical viewpoint of decision theory (see [10] for an introduction). We assume the adversary to be a fully rational[2] expected utility maximizer who can choose whether to attempt each elementary attack $X_i \in \mathcal{X}$ in their given order. Additionally, we assume full information about the past so that when considering X_i the adversary is assumed to have information about his past attack decisions and their results. We are interested in the optimal *decision strategy* for the adversary and how much he could expect to gain by following it. The strategy consists of a series of prescriptions on how to behave - one for each attack for each of its possible past series of events. This means that the adversary can choose to behave in different ways for different past outcomes. The strategy is considered optimal when it has the highest possible utility of all the possible decision strategies. This guarantees that the resulting behavior will produce the highest expected outcome of all the possible behaviors that only rely on the information accounted for within the model. Among other things, it will allow the adversary to refuse an attack when it is just too expensive, as happened in the example in the previous section.

The classical formalization for serial decision problems is via *decision trees*. Decision trees, like attack trees, have only two types of internal nodes and one type of leaves. However, their semantics are completely different. The internal nodes are either decision nodes (depicted as squares) in which the attacker can choose the outcome, or chance nodes (depicted as circles) in which the outcome is chosen randomly according to a given distribution. The leaves are called utility nodes (depicted as diamonds) and they determine how

[2] This provides the upper bound as an irrational adversary can do no better. This is the generally accepted justification for considering rational actors in decision theory.

much the current sequence of decisions and chance events is worth to the adversary. The decision strategy essentially consists of a prescription of what to choose at each decision node. When *solving* the tree, we are looking for a strategy that maximizes the expected utility. An example of a decision tree is depicted in Figure 3. It is quite straightforward to draw out the decision tree for adversarial behavior in the serial model described for any Boolean function $\mathcal{F}$. For example, the decision trees for $\mathcal{F}(X_1, X_2)$ have the form depicted in Figure 3. Finding the utilities at the leaves is also quite simple as it involves the positive utility of either g if $\mathcal{F}$ was true or 0 if it was false and from that one just needs to subtract the expenses incurred during the attack that depend on the exact decisions made and their outcomes.

Since the decision trees corresponding to the attack trees (or any attack scenarios describable by Boolean formulae for that matter) are always binary, we adopt the following convention when depicting them in figures. We always assume that the upper arrow corresponds to the positive answer (attack is attempted, attack succeeds) and that the lower arrow corresponds to the negative answer (attack is not attempted or fails). Additionally, since the chance nodes have just two outputs, the distribution for them is uniquely determined by the probability of the positive answer alone and we write that value inside the chance node. For utility nodes, we write their utility inside the diamond symbol whenever possible. To differentiate between decision and attack trees visually, decision trees are depicted with a root on the left while attack trees have the root on top.

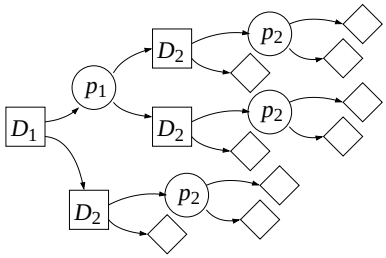

Fig. 3. A Decision Tree for $\mathcal{F}(X_1, X_2)$

The standard algorithm for computing the expected value of decision trees is actually very straightforward. It takes an internal node for which the optimal expected outcomes of the children are known and computes the optimal expected outcome for that node as well. This is done by either taking the maximum of the utilities of the children if it is a decision node, or taking the weighed average of the outcomes of children according to the given distribution if it is a chance node. The process is repeated until the expected maximal outcome for the root node is computed, which is then taken as the expected maximal outcome of the whole tree. As we assume the adversary to be a rational utility maximizer, this is the most natural way to define his optimal gains.

As a decision tree can easily be drawn out for any Boolean function $\mathcal{F}$ on the set of elementary attacks $\mathcal{X}$ and as the results do not depend on the representation of $\mathcal{F}$, our semantics are consistent in the sense of Mauw and Oostdijk [15]. Thus far the AND-OR tree approach has been prevalent in the security analysis literature due to the simplicity of calculations in many cases.

The decision tree approach provides us with a convenient, straightforward and theoretically sound formalization for adversarial behavior. Regrettably the trees are generally of exponential size in the number of attacks so for practical attack scenarios it is infeasible to even draw them out in full. When assuming that the attacks can be expressed as Boolean functions, however, the decision trees can often be greatly simplified

and the optimal strategy can be found in polynomial time. Before describing how this can be done we first introduce some additional decision theory to simplify further discussion.

The standard model of decision trees is somewhat inconvenient for our purposes. It would be more natural if we could factor in the expenses at the place where they are incurred rather than at the leaves. The model can easily be modified to accommodate this by allowing utility changes to be placed on edges as well as at the leaves[3]. We also change the semantics so that the utility of a given path from root to a leaf is now defined as the sum of all the utility nodes encountered on the way, the leaf where the path ends included. To cope with the augmented semantics within the optimization algorithm, we introduce a rule for compacting a utility node whose child is also a utility node. It works through simply merging the two nodes by summing their utilities.

With this formalization we can move the expenses associated with performing the attacks in between the chance and decision nodes inside the tree, leaving only either 0 or g as the leaf utilities. It is straightforward to verify that this new model still gives the exact same maximal expected outcome as the original decision tree model. Applying this approach to the example of Figure 3 produces Figure 4(a)[4]. In the case of our

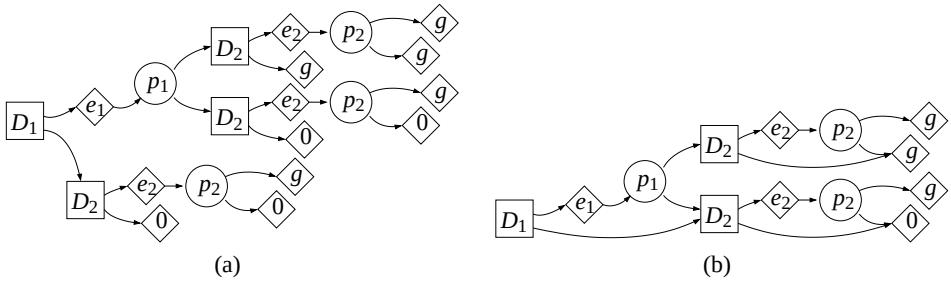

Fig. 4. Generalization of the decision tree for $\mathcal{F}(X_1, X_2) = X_1 \vee X_2$ with extra utility nodes (a) and then into an RDAG (b)

model this shift will often tend to produce a tree with many subtrees that are completely identical in the sense of having the exact same types of nodes with the exact same structure, distributions and utilities. As the algorithm for solving decision trees works from leaves towards the root it is clear that if two subtrees are equal to the end then the optimization algorithm will work in an essentially identical way in both. As solving two such subtrees separately is just duplicate work, we could save time by using the same solution in both places and computing it just once. A convenient way to represent this in our model is to replace the two equal subtrees with just a single one by making the incoming arcs to their roots both point to the same node. See Figure 4(b) for an illustration of this. This requires loosening the assumption of having a (rooted) tree into that of having a rooted directed acyclic graph (RDAG). It is easy to see that the maximal expected utility remains unchanged whenever two equal subtrees are merged together.

Before moving on we note that the two generalizations to the decision tree model are both fairly natural and quite standard. The generalization to RDAG is commonly referred to as *coalescing* and is viewed as one of the simplest ways of reducing the decision tree size. Allowing utilities on the edges is also completely standard.

The RDAG generalization allows us to make one further simplification. Following the example on Figure 4 it should be intuitively clear that the subtree corresponding to an attack X_i failing and the subtree

[3] So a utility node now always has one incoming arc and may also have at most one outgoing arc.

[4] For readability we omit minus-signs from the utility nodes corresponding to the expenses. They do nonetheless convey negative utility and this is just a notational convenience for the figures.

corresponding to the same attack X_i not being attempted are always identical to one another so we can always merge them together. This allows us to abstract the decision and its corresponding chance and utility nodes into a *decision compound* like the one depicted in Figure 5(a). After doing that we are left with a tree composed of just decision compounds and utility nodes (an example is depicted in Figure 5(b)). This allows for a simpler and more informative visual representation and all the decision RDAG figures in the following will use this abstraction. Therefore, the following diagrams will be composed of decision compounds (marked as rectangles with rounded corners) and two terminating leaves – one for 0 and one for g.

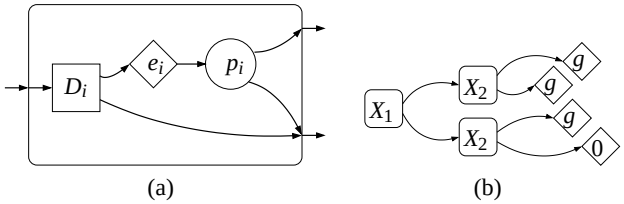

Fig. 5. A decision compound for X_i (a) and the RDAG in Figure 4 redrawn using decision compounds (b)

4 Non-crossing Trees

Since the decision trees are of exponential size in general, we need one additional assumption to achieve computational efficiency. We thus constrain the order of the attacks to adhere to a *non-crossing* assumption.

The non-crossing condition basically means assuming goal-oriented behavior from the adversary. It is required that the elementary attacks of a subattack are always attempted together without considering any elementary attacks from the other subattacks in between. If a subattack is something to the effect of ”break into the main office and steal the data from the safe” then it is completely natural to assume that all the elementary attacks within it (such as ”pick the front door”, ”find the safe”, ”crack the safe open”, ”run for your life”) are all attempted one after the other without attempting elementary attacks from other subattacks (such as ”hack into the mainframe of the overseas office”) in between them. We essentially assume that one large goal is either satisfied or abandoned before a subsequent subattack of the same importance is ever attempted. As people do tend to work and think in a goal-oriented way, we believe this to be a rather reasonable assumption to make of an adversary.

The name ”non-crossing” comes from the visual representation of the attack trees. Suppose an attack tree is drawn in such a way that all elementary attacks are positioned on a straight line in the order they are performed. The tree, then, is non-crossing precisely when it can be drawn so that no two arcs intersect one another (without changing the order of elementary attacks) and no arc crosses the straight lines from the root to the first and last elementary attack. This is best illustrated on Figure 2 where the subfigure (a) is non-crossing while (b) is crossing. Formally, the condition can be stated in the following way:

Definition 1. *Let $\mathcal{F}(X_1,\ldots,X_n)$ be a monotone Boolean formula of n variables. We say that $\mathcal{F}$ is* non-crossing *relative to the order $X_1,\ldots,X_n$ if it can be written in such a way that*

(simplicity) *Only $\wedge$ and $\vee$ operators are used*
(single-occurrence) *Each variable occurs at most once*
(order-preserving) *For all $i < j$, X_i appears to the left of X_j*

We note that for a given attack tree of n elementary attacks there are at least 2^{n-1} different non-crossing orders so our approach works for an exponentially large set of orders. However, it is only a negligible fraction of all the possible $n!$ orders.

For simplicity, we assume the attack trees to be *binary* so that every internal node has just two children. As we are dealing with AND-OR trees, this is without loss of generality. This is because a single AND-node with many children can be split into a series of AND-nodes with just two children each and the same can be done for OR nodes as well. This means we can decompose any non-crossing AND-OR tree into a non-crossing binary tree without changing the semantics. As the process is simple and completely mechanic we will discuss it no further and just assume we are using binary trees.

5 Efficient Computation for the Non-Crossing Trees

For the case of non-crossing trees the RDAG can be greatly reduced in size. To be precise, it can be made to have just one decision compound for each elementary attack. The result is captured in the following theorem:

Theorem 1. *Let $X_1,\ldots,X_n$ be elementary attacks of an attack scenario described by $\mathcal{F}(X_1,\ldots,X_n)$. If $\mathcal{F}$ is a non-crossing Boolean function, the optimal attack strategy can be found in $O(n)$ time.*

Proof. The result rests on three observations about the structure of non-crossing attack trees.

It should be clear from the description of the decision RDAG that two sub-RDAGs rooted at X_i are functionally equal whenever they give the same exact outcome for all the possible future choices $X_i,\ldots,X_n$. More formally, for an attack tree corresponding to a Boolean function $\mathcal{F}$, two subtrees with histories $r_1,\ldots,r_{i-1}\in\{t,f\}$ and $r'_1,\ldots,r'_{i-1}\in\{t,f\}$ rooted at X_i are equal precisely when for all $X_i,\ldots,X_n\in\{t,f\}$ we have

$$\mathcal{F}(r_1,\ldots,r_{i-1},X_i,\ldots,X_n)=\mathcal{F}\left(r'_1,\ldots,r'_{i-1},X_i,\ldots,X_n\right)\quad. \tag{1}$$

This is so because the subtrees rooted at two different decision compounds for the same attack X_i always have the same binary subtree structure all the way to the leaves and that they can only differ in leaf values. Leaf values, however, are fully determined by the Boolean function $\mathcal{F}$ and the equivalence holds because that, too behaves in an identical way in the two cases.

It turns out that in the case of non-crossing trees this simplification allows us to systematically reduce the complexity of the decision RDAG to a manageable size. This can be done based on three simple observations. For illustration we will use an example attack tree depicted in Figure 6(a) to demonstrate the simplification process. The corresponding unsimplified decision RDAG is depicted in Figure 6(b).

Before moving on to discuss the observations, we introduce some notation. We assume the tree to be a full binary tree so each internal node has exactly two direct descendants or *children*. The single-occurrence assumption means that there is one well-defined path from each of the elementary attacks to the root. We denote this path by $\mathcal{P}_i=\{Y_0,Y_1,\ldots,Y_k\}$ where Y_0 is the root. Given a fixed non-crossing order, the two children Z_l,Z_r of a parent node Z_p are uniquely ordered to satisfy the non-crossing restriction. We call Z_l and Z_r *siblings* of one another and we call the sibling Z_l the *left sibling* of Z_r and Z_r the *right sibling* of Z_l. If Z_p is an AND-node, we say Z_l is the left *AND-sibling* and if Z_p is an OR-node, we call Z_l the left *OR-sibling* of Z_r. For an elementary attack X_i we call the set $\mathcal{L}_i$ of left siblings of $\mathcal{P}_i$ its *left (sibling) set*. Left AND-set and left OR-set are defined in a similar way, being composed (respectively) of left AND-siblings and left OR-siblings of $\mathcal{P}_i$.

For example, consider the elementary attack X_4 of the attack tree in Figure 6(a). Its left AND-set is composed of a single node – the OR-node that combines X_1 and X_2. Its left AND-set also has just one node

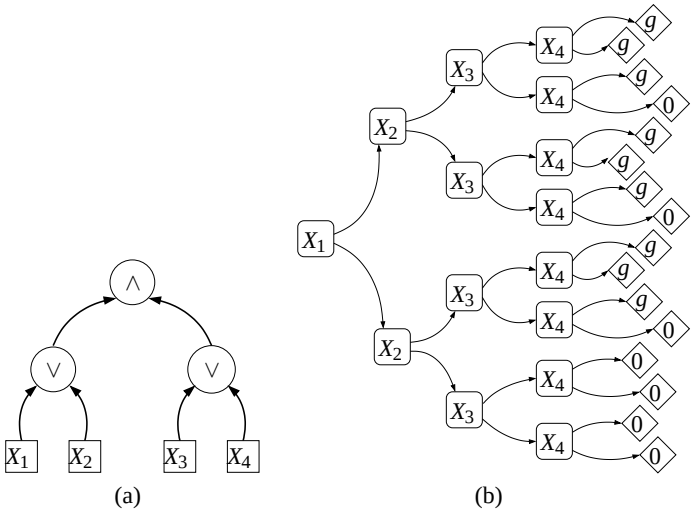

Fig. 6. An attack tree (a) and the corresponding decision tree (b)

– the leaf node of the elementary attack X_3. These two nodes together form the complete left set for X_4. As X_4 is the last elementary attack, its right set is empty.

We note that the non-crossing assumption along with the assumption of full information about the past guarantee that whenever a decision X_i is being considered, all the values for its left set nodes can already be computed. This is because the elementary attacks required for that have already been performed and their results are known. This makes the left set central in our description of the algorithm as it is coupled with the fact that information about previous attacks can be compressed down to just the values of the nodes contained therein.

We first note that the equation (1) always holds whenever $r_1, \ldots, r_{i-1} \in \{t, f\}$ and $r'_1, \ldots, r'_{i-1} \in \{t, f\}$ give the same results for the elements of the left set of X_i. This is intuitively easy to understand if you consider that the results from the elementary attacks are aggregated at the internal nodes and that having a stake in the aggregate results is the only way the elementary attacks really influence the root value. As our tree is non-crossing, the left set values for X_i can always be computed if $r_1, \ldots, r_{i-1}$ are all known and the left set values are, in fact, the topmost aggregate values that can be computed based solely on the history up to X_i. This means that the left set truth values are essentially the only information that is needed about the past attacks and their successes. After carrying out the simplification, the RDAG for the example attack tree in Figure 6 simplifies to the form depicted in Figure 7(a).

The second thing we note is that there is actually just one valuation of the left set for which any given decision actually makes any difference - the one where all the nodes in the left AND-set evaluate to t and all the nodes in the left OR-set evaluate to f. This is because any left set value Z_l deviating from that scheme would also determine the value for its parent clause Z_p. For instance, if Z_p was an OR clause and Z_l would be t then no matter what the right branch Z_r evaluates to, Z_p would still evaluate to t. Analogous reasoning works for Z_p being an AND clause and Z_l being t. This means that in these cases no elementary attack within the subtree of Z_r could possibly modify the final outcome (as the aggregate value at Z_p is determined already). Since X_i is contained in the subtree of Z_r, its result is insignificant when computing the final result in this case. After carrying out this simplification for our example we arrive at the RDAG depicted in Figure 7(b).

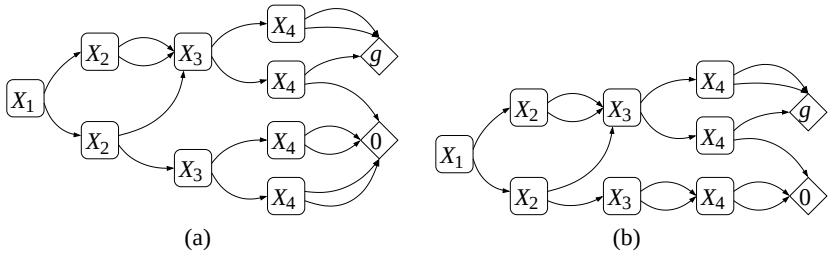

Fig. 7. The attack tree being simplified according to the first (a) and the second (b) observation

The third observation is that the decision compounds whose both outgoing arcs point to the same place are *inconsequential* and can actually be ignored. The reason for that is that performing the attack may cost something while not attempting it is always free. As such, it is always safe not to attempt an attack whenever both success and failure lead to the same future outcomes. This allows us to simplify the decision RDAG even further. For notational convenience we call the decision compounds that are left after this step *consequential*. The final RDAG for the example is depicted in Figure 8

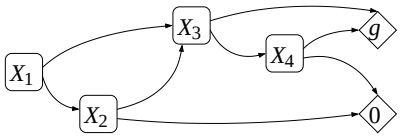

Fig. 8. The attack tree after the third simplification

To further illustrate the observations, we provide another example of a larger attack tree $X_1 \wedge ((X_2 \wedge (X_3 \vee X_4)) \vee X_5)$ along with the fully simplified decision RDAG corresponding to it in Figure 9. As before, the RDAG is again of only linear size.

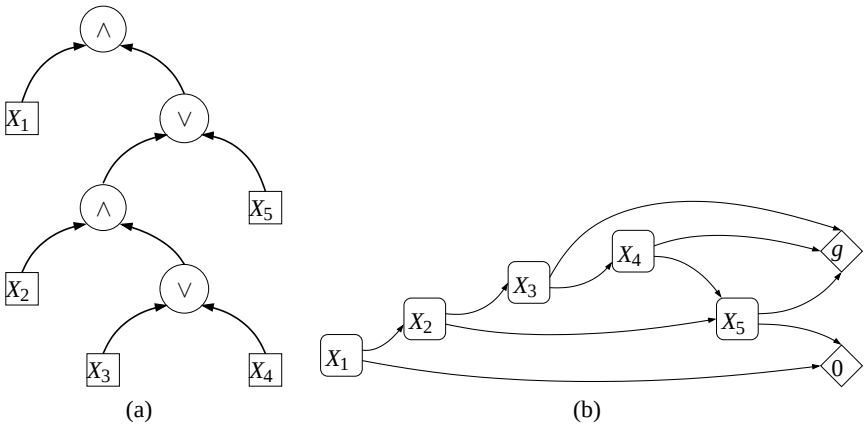

Fig. 9. An attack tree of five elementary attacks (a) and the corresponding simplified decision RDAG (b)

Putting the three observations together clearly leads to a conclusion that if we merge all the equal sub-RDAGs for a non-crossing tree, there is just one consequential decision compound per each attack X_i. This follows from the fact that there is just one sub-RDAG for each left set valuation (the first observation) and that there is just one left set valuation for which it matters whether the compound succeeds or fails (the second observation) and that all the other compounds can be removed (observation three). This means that the decision RDAG is of linear size for the non-crossing trees. As the structure of the RDAG can also be computed in linear time [5] and the optimization process takes time proportional to the number of nodes in the decision RDAG, the whole computation can be done in $O(n)$ time. □

The algorithm that results for the non-crossing case is fairly easy to implement and has relatively low overhead. It is also extensible to the case where inconsequential compounds (those where both outgoing arcs point to the same place) cannot be inlined, although in such a case, the algorithm is worst case quadratic. see Appendix B. In some special cases, however, dynamic programming can still be used to do partial inlining, in which case the running time remains linear. An example of this is provided by allowing for the expenses to be negative (i.e. getting paid for performing an attack instead of paying for it). See Appendix C.

6 Connection with Previous Models

As claimed in the introduction, our model achieves the highest expected outcome for the adversary of all the financial models proposed thus far. To be more formal, let $\mathcal{F}$ be an attack scenario with elementary attacks $\mathcal{X}$ and let σ be a fixed ordering for the elementary attacks in $\mathcal{X}$. Denote by $\mathsf{Outcome}^{\mathrm{JW}}_{\sigma}$ the expected utility assigned to the attack tree with attack order σ on the optimal subset of $\mathcal{X}$ by the serial mode of Jurgenson and Willemson [12] and let $\mathsf{Outcome}^{\mathrm{DT}}_{\sigma}$ be the expected utility computed by the decision-theoretic model proposed in this paper. It is easy to show that

Theorem 2. *For all attack scenarios $\mathcal{F}$ and attack orders σ*

$$\mathsf{Outcome}^{\mathrm{JW}}_{\sigma} \leq \mathsf{Outcome}^{\mathrm{DT}}_{\sigma} \quad .$$

It is interesting to note that in the case of non-crossing trees the equality always holds in Theorem 2 [6]. We do, however note that finding $\mathsf{Outcome}^{\mathrm{JW}}_{\sigma}$ requires trying all the 2^n subsets to find the optimal solution whereas our algorithm finds $\mathsf{Outcome}^{\mathrm{DT}}_{\sigma}$ in linear-time.

This observation leads to another possible practical application. Let $\mathsf{Outcome}^{\mathrm{P}}$ denote the expected outcome in the parallel model of [13]. Jürgenson and Willemson [12] showed that

$$\mathsf{Outcome}^{\mathrm{P}} \leq \mathsf{Outcome}^{\mathrm{JW}}_{\sigma}$$

for all attack orders σ. This result, when combined with ours, yields a linear-time method of finding non-trivial upper bounds on the adversarial outcome in the parallel model. This is something that was impossible with the previously known algorithms.

[5] Success always goes to the leftmost elemental attack of the subtree rooted at the lowest element of the right AND-set and analogously for failure and the right OR-set. This is easy to verify and we omit the details. Using this knowledge it is straightforward to generate the graph in linear time by first determining the leftmost elemental attack for each subattack and then traversing the tree left-to-right, keeping track of the right AND and OR sets. See Appendix A for the pseudocode.

[6] This follows from the fact that there is always just one consequential decision compound for each elementary attack X_i from which it follows that the optimal subset for the model of [12] is exactly the set of elementary attacks X_i that are considered worthwhile in our model.

7 Possible extensions

The simplification steps presented in the preceding section are not restricted to the non-crossing trees and can indeed be applied for many other Boolean functions and variable orders. To better understand that, we look at an alternative form of representing the underlying Boolean formulae.

Binary Decision Diagrams (BDD) are a method of compactly representing Boolean formulae by rooted directed acyclic graphs where each node corresponds to a variable and has two distinguishable outgoing arcs – one for when the variable is set to true and one when it is set to false. The graph has just two leaves that correspond to the formula being evaluated to true and the other for false. The most common use of BDD-s involves the so called Reduced Ordered BDD-s (ROBDD-s) in which case the variables are in a fixed order and all the sub-RDAGs that are equivalent are merged. It is also assumed that a node is in-lined whenever its both outgoing arcs point to the same place. We refer a reader more interested in the theory of BDD to [20]

Following the steps of the previous section, it is trivial to see that our model of economic decision trees actually reduces to the form where we have a directed graph of decision compounds (with output degree 2) and just two leaves. As the final destination leaf of a path is determined solely by the truth table of the Boolean function, it is clear that our model is in some sense isomorphic to BDD-s. As we can also do all the simplification steps that are allowed for ROBDD-s, we can actually formalize our result as the following corollary:

Corollary 1. *Suppose we have an attack scenario with elementary attacks $\mathcal{X} = \{X_1, \ldots, X_n\}$ so ordered. If the attack scenario $\mathcal{F}$ can be described with a polynomial-sized ROBDD with the same order, the optimal strategy and its expected outcome can also be determined in polynomial time.*

This result has three important implications. Firstly, it is known that many Boolean formulae (that cannot be expressed as AND-OR trees) have reasonably small representations as ROBDD-s for some variable orders. This is important because it may well make sense to find the utility even when the order for which this is feasible is completely absurd. This is because our model will produce a strictly higher utility than the parallel model described in the introduction. As the exact computation of the parallel model takes exponential time, our model can thus be used as a fast and practical means of finding an upper bound for the adversarial outcome in it. It is known that most interesting classes of Boolean functions do have orders for which their BDD-s are small, allowing our model to be used for just such estimation.

A second implication (being somewhat a corollary of the first) is that we can actually allow a few elementary attacks to occur in multiple places in the tree. Namely, it is rather easy to verify that introducing another occurrence of a variable can at most double the size of the ROBDD[7]. As such, the model still remains efficiently computable if 2-4 variables occur more than once. This is often the case in practice as most elementary attacks only matter in one place but there are usually a few (such as gaining root privileges) that are required in multiple places. In these cases there is still an exponential set of orders for which the approach will work.

A third implication is that even for simple attack trees, we cannot use this approach for all the possible orders. To be precise, there are trees of very simple structure for which some orders produce exponential-sized ROBDD-s. This means that the approach, although very effective for gaining practical estimates and computing some attack orders, is still inherently limited. The exact extent to which these limits apply needs further exploration and we leave it as an open question for now. A quick review of BDD literature shows that this question is relatively unexplored. This is probably due to the fact that in general applications the order of variables in a BDD does not need to be fixed. Due to that, most literature is interested in finding

[7] This is essentially achieved by Shannon decomposition on the remaining variables.

orders that produce small BDD-s and not in determining the minimal size of a BDD for a given order. Some progress has been made, however and we refer the more interested reader to Chapter 5 of [20] for a general overview.

8 Conclusions and Further Work

We describe a model for attack trees that is strictly based on classical decision theory. We also show that there exists an exponential sized family of orders for attack trees for which the maximal utility outcome can be found in $O(n)$ time and also describe how the same approach may be generalized for other Boolean functions that generalize the usual solely tree based approach.

Our work still leaves many open questions to be further explored. For instance, the assumption of non-crossing trees, although quite natural, is still rather restrictive and it would be very interesting to find an efficient algorithm that works under more general assumptions. Another interesting question is whether the optimal order for the elementary attacks could be found efficiently in the cases where the order is not determined naturally. It would also be interesting to consider more general models, such as those allowing intermediate payouts in the subattack nodes. Research in any of these directions would greatly further the applicability of attack trees and provide us with much more accurate tools for predicting adversarial behavior.

9 Acknowledgements

The author would like to thank Jan Willemson and Aivo Jürgenson for introducing him to the topic of attack trees and for helpful comments on this article. He is also very grateful to Sven Laur, Peeter Laud and all the anonymous reviewers for all their suggestions, which helped to make the article easier to follow and more self-contained.

Bibliography

[1] Amenaza, *Secur/tree attack tree modeling*, 2010, http://www.amenaza.com/.

[2] Paul Ammann, Duminda Wijesekera, and Saket Kaushik, *Scalable, graph-based network vulnerability analysis*, CCS '02: Proceedings of the 9th ACM conference on Computer and communications security, 2002, pp. 217–224.

[3] Daniel Bilar, *Quantitative risk analysis of computer networks*, Ph.D. thesis, Dartmouth College, 2003, Chairperson-Cybenko, George.

[4] Ahto Buldas, Peeter Laud, Jaan Priisalu, Märt Saarepera, and Jan Willemson, *Rational Choice of Security Measures via Multi-Parameter Attack Trees*, Critical Information Infrastructures Security. First International Workshop, CRITIS 2006, LNCS, vol. 4347, 2006, pp. 235–248.

[5] Ahto Buldas and Triinu Mägi, *Practical security analysis of e-voting systems*, Advances in Information and Computer Security, Second International Workshop on Security, IWSEC, LNCS, vol. 4752, 2007, pp. 320–335.

[6] Kenneth S. Edge, *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] Clifton A. Ericson II, *Fault tree analysis - a history*, Proceedings of the 17th International System Safety Conference, 1999.

[8] Jeanne H. Espedahlen, *Attack trees describing security in distributed internet-enabled metrology*, Master's thesis, Department of Computer Science and Media Technology, Gjøvik University College, 2007.

[9] Sushil Jajodia, Steven Noel, and Brian O'Berry, *Topological analysis of network attack vulnerability*, Managing Cyber Threats: Issues, Approaches and Challenges, 2003.

[10] Finn V. Jensen, *Bayesian networks and decision graphs*, Information Science and Statistics, Springer, 2001.

[11] Aivo Jürgenson and Jan Willemson, *Processing multi-parameter attacktrees with estimated parameter values*, Advances in Information and Computer Security, Second International Workshop on Security, IWSEC, LNCS, vol. 4752, 2007, pp. 308–319.

[12] Aivo Jurgenson and Jan Willemson, *Serial model for attack tree computations*, International Conference on Information Security and Cryptololgy: ICISC 2009, LNCS, 2009.

[13] Aivo Jürgenson and Jan Willemson, *Computing exact outcomes of multi-parameter attack trees*, On the Move to Meaningful Internet Systems: OTM 2008, LNCS, vol. 5332, 2008, pp. 1036–1051.

[14] Richard P. Lippmann and Kyle Ingols, *An annotated review of past papers on attack graphs*, 2005.

[15] Sjouke Mauw and Martijn Oostdijk, *Foundations of attack trees*, International Conference on Information Security and Cryptology – ICISC 2005 (Dongho Won and Seungjoo Kim, eds.), LNCS, vol. 3935, Springer, 2005, pp. 186–198.

[16] Andrew P. Moore, Robert J. Ellison, and Richard C. Linger, *Attack modeling for information security and survivability*, Tech. Report CMU/SEI-2001-TN-001, Software Engineering Institute, 2001.

[17] Bruce Schneier, *Attack trees: Modeling security threats*, Dr. Dobb's Journal **24** (1999), no. 12, 21–29.

[18] W.E. Vesely, F.F. Goldberg, N.H. Roberts, and D.F. Haasl, *Fault tree handbook*, US Government Printing Office, January 1981, Systems and Reliability Research, Office of Nuclear Regulatory Research, U.S. Nuclear Regulatory Commission.

[19] John Viega and Gary McGraw, *Building secure software: How to avoid security problems the right way*, Addison Wesley Professional, 2001.

[20] Ingo Wegener, *Branching programs and binary decision diagrams: theory and applications*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

[21] J. D. Weiss, *A system security engineering process*, Proceedings of the 14th National Computer Security Conference, 1991, pp. 572–581.

A Pseudocode for the Linear Algorithm

For completeness we give (recursive) pseudocode for the linear algorithm for non-crossing trees. The algorithm should initially be called as `lrec(∅,1)`.

Algorithm 1 `lrec(`$\mathcal{L}$`,`i`)`

Require: The set of elementary attacks $\mathcal{X} = \{X_1, X_2, \ldots, X_n\}$ with their costs c_i and success probabilities p_i along with a single-occurrence non-crossing monotone Boolean formula $\mathcal{F}$ describing the attack scenarios
Require: Truth values $\mathcal{L}$ evaluated for the left set based on past results for attacks
Require: $i \in [1, n+1]$ - the index of the attack we are considering
if $i = n+1$ **then**
  return g or 0 depending on whether the main threat was materialized
else if The global outcome of $\mathcal{F}$ given $\mathcal{L}$ depends on X_i **then**
  if The expected value E_i for X_i has been cached **then**
    return E_i
  else
    Update $\mathcal{L}$ by assuming that X_i succeeded. Store the result in $\mathcal{L}'$.
    Determine the index i^+ of the leftmost elementary attack of the lowermost right AND-sibling of X_i. If it does not exist, $i^+ = n+1$ and the main threat succeeds, yielding g.
    Take $E^+ =$ `lrec(`$\mathcal{L}'$`,`i^+`)`
    Update $\mathcal{L}$ by assuming that X_i failed. Store the result in $\mathcal{L}''$.
    Determine the index i^+ of the leftmost elementary attack of the lowermost right OR-sibling of X_i. If it does not exist, $i^- = n+1$ and the main threat fails, yielding 0.
    Take $E^- =$ `lrec(`$\mathcal{L}''$`,`i^-`)`
    Compute the optimal expected outcome $E = \max\{p_i E^+ + (1-p_i)E^- - c_i, E^-\}$.
    Cache E as E_i.
    return E
  end if
end if

B Quadratic Decision RDAGs

There is a simple family of attack trees Q for which the number of nodes is of quadratic size if no pruning of inconsequential nodes is performed. The family Q is generated by taking the Boolean formula $X_a \wedge ((X_b \wedge Z) \vee X_c)$ and iteratively replacing Z within it with the exact same compound, getting $Q_1 = X_a \wedge ((X_b \wedge Z) \vee X_c)$, $Q_2 = X_a \wedge ((X_b \wedge (X_{a'} \wedge ((X_{b'} \wedge Z) \vee X_{c'}))) \vee X_c)$, ... In general, the formula for Q_m will have $3m+1$ variables. If they are left in the same order but numbered from 1 to $3m+1$ and interpreted as an attack tree, the decision RDAG corresponding to the resulting attack tree will have $(3n+5)n+1 \approx 3n^2$ compounds - exactly k compounds for each X_k for $1 \leq k \leq 2m+1$, $2m+1$ compounds for X_{2m+2} and 2 less than the previous for each X_k from then on. That this is indeed so should be intuitively clear and easy to verify based in Figure 10.

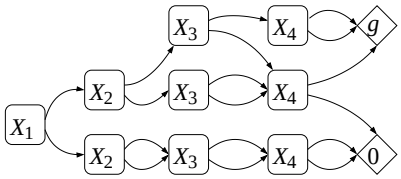

Fig. 10. Decision RDAG corresponding to the smallest attack tree Q_1 of the family Q.

C The Linear Algorithm for Negative Expenses Model

For completeness we bring the linear algorithm along with the modifications required to work with the negative expenses as well.

The precomputation step consists of computing sums

$$S_i = - \sum_{i:1 \le i \le n, e_i < 0} e_i \ .$$

As before, the algorithm should initially be called as lrec($\emptyset$,1).

Algorithm 2 lrec($\mathcal{L}$,i)

Require: The set of elementary attacks $\mathcal{X} = \{X_1, X_2, \ldots, X_n\}$ and a single-occurrence non-crossing monotone Boolean formula $\mathcal{F}$ describing the attack scenarios
Require: Truth values $\mathcal{L}$ evaluated for the left set based on past results for attacks, the sums S_i
Require: $i \in [1, n+1]$ - the index of the attack we are considering
if $i = n+1$ **then**
  return g or 0 depending on whether the main threat was materialized
else if The global outcome of $\mathcal{F}$ given $\mathcal{L}$ depends on X_i **then**
  if The expected value E_i for X_i has been cached **then**
    return E_i
  else
    Update $\mathcal{L}$ by assuming that X_i succeeded. Store the result in $\mathcal{L}'$.
    Determine the index i^+ of the leftmost elementary attack of the lowermost right AND-sibling of X_i. If it does not exist, $i^+ = n+1$ and the main threat succeeds.
    Take $E^+ = \text{lrec}(\mathcal{L}', i^+) + S_{i^+-1} - S_i$
    Update $\mathcal{L}$ by assuming that X_i failed. Store the result in $\mathcal{L}''$.
    Determine the index i^+ of the leftmost elementary attack of the lowermost right OR-sibling of X_i. If it does not exist, $i^- = n+1$ and the main threat fails.
    Take $E^- = \text{lrec}(\mathcal{L}'', i^-) + S_{i^--1} - S_i$
    Compute the optimal expected outcome E of the decision compound for X_i on the assumption that its outputs have expected values of E^+ and E^- respectively.
    Cache E as E_i.
    return E
  end if
end if