How well can I secure my system?

Barbara Kordy and Wojciech Wideł

INSA Rennes, IRISA, France
{barbara.kordy, wojciech.widel}@irisa.fr

Abstract. Securing a system, being it a computer network, a physical infrastructure or an organization, is a very challenging task. In practice, it is always constrained by available resources, e.g., budget, time, or man-power. An attack–defense tree is a security model allowing to reason about different strategies that an attacker may use to attack a system and potential countermeasures that a defender could apply to defend against such attacks. This work integrates the modeling power of attack–defense trees with the strengths of integer linear programming techniques. We develop a framework that, given the overall budget allocated for the system's protection, suggests which countermeasures should be implemented to secure the system in the best way possible. We lay down formal foundations for our framework and implement a proof of concept tool automating the solving of relevant optimization problems.

1 Introduction

The only system that is guaranteed to be fully secure is the empty system which does not provide any functionality. Any other system offering an actual service will always be vulnerable to attacks. These attacks may target the system's availability (e.g., denial of service attacks), its integrity (e.g., corruption of data leading to inaccurate or inconsistent results), or the confidentiality of the users' private information (e.g., stealing credentials necessary for authentication). To achieve their malicious goals, attackers, who might be external to the system or insiders, have a plethora of methods to choose from, including digital means, such as hacking, physical attacks, for instance, breaking in and stealing, as well as very powerful social engineering techniques relying on psychological manipulation. All these aspects must be taken into account while securing a system or a company. In addition, perfect security would require the system's owners to have unlimited resources, in terms of financial means and time, but in practice, this is never the case. This is where the risk analysis comes into play.

The crucial challenge in every risk assessment methodology is to exhaustively describe the attack scenarios corresponding to the most feared threats, in order to determine the most likely ones, and deploy relevant countermeasures, in such a way that the residual risks are acceptable. Attack–defense trees have been introduced in [6] as a formal solution to address this challenge and to support practical risk assessment methodologies.

An *attack–defense tree* (ADTree) is a graph-based model representing how an attacker may compromise a system and how a defender may protect it against potential attacks. ADTrees enhance the industrially recognized formalism of attack trees [10,8], by explicitly integrating the countermeasures and the counterattacks against these countermeasures to the model. Methods for quantitative analysis of ADTrees, e.g., [1,7], allow the modeler to analyze and quantify the effect of deploying a countermeasure, and to evaluate its consequences.

Several aspects can be taken into account while choosing the countermeasures to be implemented. One can be interested in minimizing the number of undefended attacks, minimizing the impact that the system would suffer from in a case of an attack, maximizing the minimal necessary investment of the attacker, etc. It turns out that all these problems might be modeled as integer linear programming problems [3].

The objective of this work is to develop a framework allowing a security expert to select the most pertinent set of countermeasures, in order to secure the analyzed system in the best way possible. This framework combines the modeling features of ADTrees with the potential of integer optimization techniques. Knowledge about possible attack strategies and the corresponding countermeasures is extracted from an attack–defense tree. This information is then used by integer optimization algorithms to select the most appropriate set of countermeasures. Our selection procedure is guided by practical constraints, such as the cost of individual actions of the attacker and the defender, the impact of an attack to the system, and the overall defense budget available.

Contribution. The main *scientific novelty* of this work is the development of the defense semantics for ADTrees, capturing possible strategies of a reasonable attacker and listing corresponding defender's strategies allowing to secure the analyzed system. The *practical contribution* is the formulation of security-relevant optimization problems in terms of integer programming and the implementation of a prototype tool to solve them. Our tool takes an ADTree as input and uses a free integer programming solver *lp_solve* [2] to output the optimal set of countermeasures to be implemented, according to a given optimization function.

Related work. The stimulus triggering the framework developed in this paper has been the work of Laura Albert McLay, an operations researcher working on cybersecurity problems. McLay investigates optimization techniques to distribute security budget amongst possible countermeasures and makes use of maximal coverage models to prioritize mitigations [12]. The major difference between our approach and the one of McLay is the way in which the link between potential attacks and the corresponding sets of countermeasures is tackled. In [12], this link is given as input to the problem. In our approach, this information is extracted from an ADTree in the form of its defense semantics. We propose an extraction algorithm which contributes to the development of formal foundations for ADTrees and, as such, is the main scientific contribution of this paper.

Optimization techniques have also been applied to attack–defense trees to address the problem of multi-objective evaluation of security using the concept of Pareto efficient solutions [1,11]. The goal of [11] is to identify optimal countermeasures that maximize the security performance, minimize the attack impact and minimize the defense cost. However, contrary to our approach, attack–defense trees used in [11] do not allow to model attacker's actions that would disable a countermeasure of the defender, which results in a less complex but also much less expressive model. The authors of [1] use Pareto frontier to devise a technique that optimizes several parameters, e.g., cost and probability, at once. ADTrees considered in [1] are similar to the ones used in our paper. However, this work does not consider coverage problems, as we do in the current work.

2 Security modeling with attack–defense trees

We start by explaining the ADTree formalism and presenting the assumptions about the attacker and the defender considered in this work. Then, we introduce a formal semantics which allows us to take advantage of integer programming techniques to reason about attack–defense scenarios modeled with ADTrees.

2.1 Attack–defense trees

Attack–defense trees (ADTrees) [6] are rooted trees with labeled nodes supporting representation and quantitative analysis of security scenarios involving two competing (sets of) actors – the attacker (denoted by `A`) and the defender (denoted by `D`). Labels of the nodes of an ADTree depict goals of the actors. One of the actors is trying to achieve a particular goal represented by the root node of the ADTree and the other actor is trying to hamper them from doing so.[1] The goal of a node in an ADTree can be refined into sub-goals, either in a *disjunctive* way (denoted by `OR`) or in a *conjunctive* way (denoted by `AND`). The meaning of an ADTree is based on the notion of *goal achievement*. A goal represented by a disjunctively refined node is achieved if at least one of the sub-goals represented by its children is achieved. To achieve a goal of a conjunctively refined node, sub-goals of all of its children must be achieved. The goals represented by the labels of the non–refined nodes are called *basic actions*. They represent the actual actions that the attacker and the defender will perform to achieve their goals. In order to forbid an actor from achieving their goal, the other actor may apply a countermeasure (denoted by `C`). In the ADTree formalism, the nodes representing countermeasures can be disjunctively or conjunctively refined and they can again be countered. At most one countermeasure per node is allowed, thus different ways of countering the same goal are represented using a single countermeasure which is disjunctively refined. The goal of a countered node is achieved if the node's refinement (or the corresponding basic action in the case of a non-refined but countered node) is achieved *and* the goal of the countermeasure attached to the node is not achieved by the other actor.

An illustrative toy ADTree is given in Fig. 1 and explained in Example 1. The following graphical conventions are used: nodes of the attacker are depicted using

[1] In [6], the root actor is called the *proponent* and the other actor is the *opponent*.

red circles and those of the defender using green rectangles; conjunctively refined nodes are marked by an arc connecting their children; nodes are connected to their countermeasures with the help of dotted edges.

Example 1. In the scenario represented with the ADTree from Fig. 1, Eve (the attacker) wants to get the password for Bob's Windows account. Eve can either perform a hacking-based attack or try to guess the password, perhaps by executing a brute-force search. To successfully perform the hacking attack, Eve needs to get the file with the hashed passwords stored in the memory of Bob's computer and then retrieve Bob's password from this file with the help of *ophcrack* [9] – a Windows password cracker for inversing hashes using rainbow tables. To prevent the theft of the password file, Bob (the defender) can encrypt the disk of his laptop using encryption software *DiskCryptor* [4]. *DiskCryptor* can be set up to work with a password or with a key file. In the first case, Bob needs to enter his *DiskCryptor* password before being redirected to the Windows login page. In the second case, Bob needs to boot from an external disk (e.g., CD or DVD) holding the correct key file. To overcome the disk encryption, Eve could eavesdrop on Bob entering his *DiskCryptor* password or steal the disk with the key file, respectively. Bob could follow a security training where he would learn that one should never enter his passwords while observed. Finally, to make the *ophcrack* attack impossible, Bob should use a very strong password which does not fall into the specification of available *ophcrack* tables.

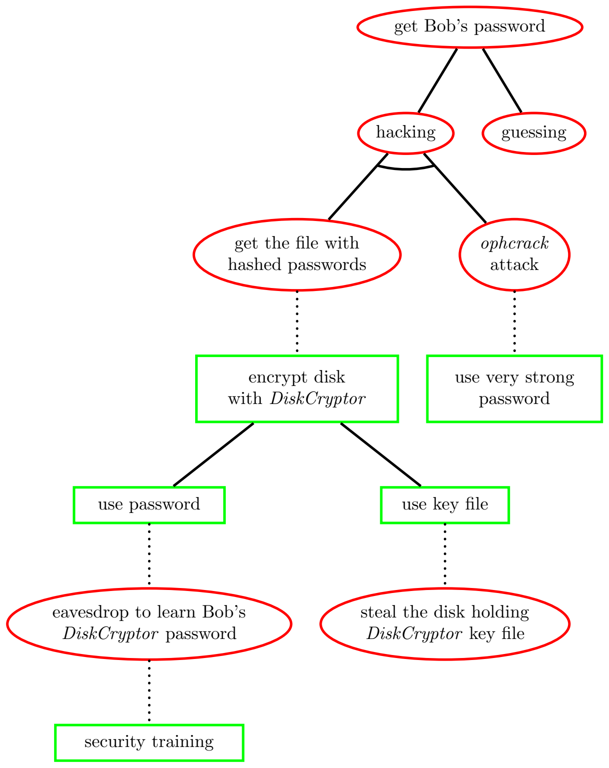
get Bob's password
hacking
guessing
get the file with hashed passwords
ophcrack attack
encrypt disk with *DiskCryptor*
use very strong password
use password
use key file
eavesdrop to learn Bob's *DiskCryptor* password
steal the disk holding *DiskCryptor* key file
security training

Fig. 1: ADTree for getting a password

Formally, an ADTree is defined using rooted, finite, labeled trees. We recall that a *tree* is an acyclic, connected graph having neither loops, i.e., edges starting and ending at the same node, nor multiple edges between two different nodes. A tree is finite if the set of its nodes is finite, and it is said to be rooted if one of its nodes is designated to be the root.

Definition 1. *An* ADTree T *is a tuple* $T = (V, E, \mathrm{L}, \lambda, \mathrm{type}, \mathrm{ref})$*, where*

- (V, E) *is a rooted, finite tree,*
- L *is a set of labels representing the attacker's and the defender's goals,*
- $\lambda\colon V \to \mathrm{L}$ *is a function that assigns labels to the nodes,*
- $\mathrm{type}\colon V \to \{\mathtt{A}, \mathtt{D}\}$ *is a function assigning actors to the nodes, in such a way that every node has at most one child of the other type,*
- $\mathrm{ref}\colon V \to \{\mathtt{OR}, \mathtt{AND}, \mathtt{N}\}$ *describes a refinement of the node: we use* `OR` *for disjunctively and* `AND` *for conjunctively refined nodes, and* `N` *for the non-refined nodes, i.e., nodes holding basic actions.*

While labels of the refined nodes are important when creating an ADTree, they are not necessary for its analysis. For instance, if we assume that none of Bob's countermeasures was implemented in the scenario from Example 1, Eve would achieve her main goal by either getting the file with hashed passwords and running *ophcrack* or by guessing the password. This situation can be represented as $\mathtt{OR}^{\mathtt{A}}\big(\mathtt{AND}^{\mathtt{A}}(\mathtt{hash}, \mathtt{ophcrack}), \mathtt{guess}\big)$[2]. The actors' strategies can thus be fully expressed by the labels of the non-refined nodes combined using refinement operators of the corresponding refined nodes. This observation leads us to propose a term-based notation for ADTrees, which is especially useful in the case of large trees, where the graphical representation is neither convenient nor effective.

Let us denote by $\mathbb{B}^{\mathtt{A}}$ and $\mathbb{B}^{\mathtt{D}}$ the sets of labels representing basic actions of the attacker and the defender, respectively. We assume that $\mathbb{B}^{\mathtt{A}} \cap \mathbb{B}^{\mathtt{D}} = \emptyset$, and we set $\mathbb{B} = \mathbb{B}^{\mathtt{A}} \cup \mathbb{B}^{\mathtt{D}}$. Let $\mathtt{b}^{\mathtt{S}} \in \mathbb{B}^{\mathtt{S}}$, for $\mathtt{S} \in \{\mathtt{A}, \mathtt{D}\}$. ADTrees can be seen as terms generated by the following grammar.

$$\begin{aligned}
T &:: T^{\mathtt{A}} \mid T^{\mathtt{D}}\\
T^{\mathtt{A}} &:: \mathtt{b}^{\mathtt{A}} \mid \mathtt{OR}^{\mathtt{A}}(T^{\mathtt{A}}, \dots, T^{\mathtt{A}}) \mid \mathtt{AND}^{\mathtt{A}}(T^{\mathtt{A}}, \dots, T^{\mathtt{A}}) \mid \mathtt{C}^{\mathtt{A}}(T^{\mathtt{A}}, T^{\mathtt{D}})\\
T^{\mathtt{D}} &:: \mathtt{b}^{\mathtt{D}} \mid \mathtt{OR}^{\mathtt{D}}(T^{\mathtt{D}}, \dots, T^{\mathtt{D}}) \mid \mathtt{AND}^{\mathtt{D}}(T^{\mathtt{D}}, \dots, T^{\mathtt{D}}) \mid \mathtt{C}^{\mathtt{D}}(T^{\mathtt{D}}, T^{\mathtt{A}})
\end{aligned}$$

If the root of an ADTree has type `A`, then the tree is said to be of the *attacker's type*. Otherwise it is said to be of the *defender's type*. Terms of the form $T^{\mathtt{A}}$ (resp. $T^{\mathtt{D}}$) represent trees of the attacker's (resp. defender's) type. The tree in Fig. 1 is of the attacker's type and it is represented with term (1).

$$T = \mathtt{OR}^{\mathtt{A}}\Big(\mathtt{AND}^{\mathtt{A}}\big(\mathtt{C}^{\mathtt{A}}(\mathtt{hash}, T'), \mathtt{C}^{\mathtt{A}}(\mathtt{ophcrack}, \mathtt{strong})\big), \mathtt{guess}\Big), \tag{1}$$

where T' is the following term representing the tree of the defender's type rooted in the 'encrypt disk with *DiskCryptor*' node

$$T' = \mathtt{OR}^{\mathtt{D}}\Big(\mathtt{C}^{\mathtt{D}}\big(\mathtt{password}, \mathtt{C}^{\mathtt{A}}(\mathtt{eavesdrop}, \mathtt{sec\text{-}train})\big), \mathtt{C}^{\mathtt{D}}(\mathtt{key\text{-}file}, \mathtt{steal\text{-}kf})\Big).$$

For the rest of this paper, we do a couple of assumptions. We identify ADTrees with the corresponding terms. To ease the presentation, we assume that the root actor (i.e., the proponent) is the attacker. Furthermore, we consider only ADTrees where all basic actions are independent. This implies that there are no multiple occurrences of the same label in a tree. Finally, it is assumed that both actors always succeed when executing their basic actions.

[2] Here, as well as in the rest of the paper, we shorten the labels for better readability.

2.2 Formal semantics

In order to formulate optimization problems related to attack–defense scenarios represented with ADTrees, we first need to extract potential attack and defense strategies from the ADTree, i.e., define the semantics of ADTrees. These strategies describe sets of actions allowing the attacker to achieve the root goal and the defender to make such an attack impossible or inefficient.

Let T be an ADTree. A *homogenous subtree* of T is a maximal subtree of T, such that all of its nodes are of the same type ($\mathtt{A}$ or $\mathtt{D}$). Node designated to be the root of a homogenous subtree H of T is the one whose distance from the root of T is minimal among all nodes of H. Obviously, every ADTree can be partitioned in a unique way into homogenous subtrees: it suffices to remove all dotted edges connecting the nodes of the attacker with those of the defender. Since all of the nodes of a homogenous subtree are of the same type, homogenous subtrees do not use any $\mathtt{C}$ operators. We thus talk about homogenous subtrees of the attacker or of the defender.

Definition 2. *Let H be a homogenous subtree of the attacker (resp. defender). A minimal, wrt the inclusion, set of basic actions of the attacker (resp. defender) achieving the root goal of H is called an* attack vector *(resp.* defense vector*)* in H*.*

Example 2. The homogenous subtrees in our running example are

$$\begin{array}{ll} H_0 = \mathtt{OR}^{\mathtt{A}}(\mathtt{AND}^{\mathtt{A}}(\mathtt{hash}, \mathtt{ophcrack}), \mathtt{guess}) & H_1 = \mathtt{strong} \\ H_3 = \mathtt{eavesdrop} & H_2 = \mathtt{OR}^{\mathtt{D}}(\mathtt{password}, \mathtt{key\text{-}file}) \\ H_4 = \mathtt{steal\text{-}kf} & H_5 = \mathtt{sec\text{-}train}. \end{array}$$

The left column gathers homogenous subtrees of the attacker and the right one of the defender. The attack vectors in the subtree H_0 are $\{\mathtt{hash}, \mathtt{ophcrack}\}$ and $\{\mathtt{guess}\}$. The defense vectors in H_2 are $\{\mathtt{password}\}$ and $\{\mathtt{key\text{-}file}\}$.

Definition 3. *Let T be an ADTree.*

- *A set $D \subseteq \mathbb{B}^{\mathtt{D}}$ is called a* defense strategy in T*, if $D = \emptyset$ or if it is a union of defense vectors from some of the homogenous subtrees of T.*
- *A set $A \subseteq \mathbb{B}^{\mathtt{A}}$ is called an* attack strategy in T*, if there exists a defense strategy D in T, such that, with all of the countermeasures from D being employed, A is a minimal set of actions achieving the root goal of T. Such D is called a witness for attack strategy A.*[3]

Example 3. The attack strategies in the tree T from Fig. 1 are

$$\{\mathtt{guess}\}, \{\mathtt{hash}, \mathtt{ophcrack}\}, \{\mathtt{hash}, \mathtt{ophcrack}, \mathtt{eavesdrop}\},$$
$$\{\mathtt{hash}, \mathtt{ophcrack}, \mathtt{steal\text{-}kf}\}, \{\mathtt{hash}, \mathtt{ophcrack}, \mathtt{eavesdrop}, \mathtt{steal\text{-}kf}\}.$$

[3] To avoid confusion, attack/defense strategies in an ADTree are denoted using capital letters and attack/defense vectors in its homogenous subtrees using lower case letters.

For instance, $\{\texttt{hash}, \texttt{ophcrack}, \texttt{eavesdrop}\}$ is an attack strategy because it represents a valid attack when the witness defense strategy $\{\texttt{password}\}$ is implemented. Likewise, $\{\texttt{hash}, \texttt{ophcrack}, \texttt{eavesdrop}, \texttt{steal-kf}\}$ is an attack strategy because the execution of all of these actions is a valid attack in the presence of the witness defense strategy $\{\texttt{password}, \texttt{key-file}\}$.

In contrast, the set $X = \{\texttt{hash}, \texttt{ophcrack}, \texttt{guess}\}$ is not an attack strategy in T, because it is not minimal. Indeed, for any defense strategy possible (including the empty strategy), `hash` and `ophcrack` can be removed from X and the root goal is still achieved with `guess`.

The reasoning in Example 3 shows that the attack strategies model a reasonable behavior of the attacker who, to achieve their goal, will not execute more actions than strictly necessary. In other words, every attack strategy in an ADTree T contains at most one attack vector from every homogenous subtree of T.

The set of all attack strategies in an ADTree T, that we denote by $\mathrm{AS}(T)$, can be obtained in a bottom–up manner using the rules given in Fig. 2, where $\bigotimes_{i=1}^{n} X_i = \{\bigcup_{i=1}^{n} x_i \mid x_i \in X_i\}$. It is important to notice that, in the case of the nodes of the form $\texttt{OR}^{\texttt{D}}(T_1^{\texttt{D}}, \dots, T_k^{\texttt{D}})$, the union is taken over all subsets $I \subseteq \{1, \dots, k\}$, including $I = \emptyset$.

$$\mathrm{AS}(\texttt{b}^{\texttt{A}}) = \{\{\texttt{b}^{\texttt{A}}\}\}, \qquad \mathrm{AS}(\texttt{b}^{\texttt{D}}) = \{\emptyset\},$$

$$\mathrm{AS}(\texttt{OR}^{\texttt{A}}(T_1^{\texttt{A}}, \dots, T_k^{\texttt{A}})) = \bigcup_{i=1}^{k} \mathrm{AS}(T_i^{\texttt{A}}), \qquad \mathrm{AS}(\texttt{OR}^{\texttt{D}}(T_1^{\texttt{D}}, \dots, T_k^{\texttt{D}})) = \bigcup_{I \subseteq \{1,\dots,k\}} \bigotimes_{i \in I} \mathrm{AS}(T_i^{\texttt{D}}),$$

$$\mathrm{AS}(\texttt{AND}^{\texttt{A}}(T_1^{\texttt{A}}, \dots, T_k^{\texttt{A}})) = \bigotimes_{i=1}^{k} \mathrm{AS}(T_i^{\texttt{A}}), \qquad \mathrm{AS}(\texttt{AND}^{\texttt{D}}(T^1, \dots, T^k)) = \bigcup_{i=1}^{k} \mathrm{AS}(T_i^{\texttt{D}}),$$

$$\mathrm{AS}(\texttt{C}^{\texttt{A}}(T_1, T_2)) = \mathrm{AS}(T_1) \otimes \mathrm{AS}(T_2), \qquad \mathrm{AS}(\texttt{C}^{\texttt{D}}(T_1, T_2)) = \mathrm{AS}(T_1) \cup \mathrm{AS}(T_2).$$

Fig. 2: Rules for creation of attack strategies in an ADTree

Lemma 1. *Let T be an ADTree and A be an attack strategy in T. In addition, let H be a homogenous subtree of the attacker in T, with $\mathbb{B}_H$ being the set of all basic actions in H. Then the set $A \cap \mathbb{B}_H$ is either empty, or else it is exactly one attack vector in H.*

Proof. Let D_A be a witness for A, cf. Definition 3, and assume that the set $A \cap \mathbb{B}_H$ is not empty. Since A is a minimal strategy wrt D_A, the basic actions from $A \cap \mathbb{B}_H$ achieve the root of H, with the root of H being either the root of T, or else a countermeasure attached to one of the nodes achieved by D_A in T. From the minimality of A it also follows that if an action was removed from $A \cap \mathbb{B}_H$, then the root of H would no longer be achieved. Thus, $A \cap \mathbb{B}_H$ is an attack vector in H. □

The goal of the framework developed in this paper is to suggest to the defender an optimal way of securing a system. To do so, we need to link possible attack strategies describing how to attack the system with the corresponding defense strategies allowing to protect it. Unfortunately, previously proposed semantics for ADTrees do not achieve this, because they take the view of one actor only into account, as illustrated in Example 4.

Example 4. Consider the tree $T = \mathtt{C}^{\mathtt{A}}\big(\mathtt{b}, \mathtt{AND}^{\mathtt{D}}(\mathtt{b}_1, \mathtt{b}_2)\big)$. There exist two (minimal) ways for the attacker to ensure that they achieve their goal: they need to execute action $\mathtt{b}$ and at the same time prevent the defender from executing either $\mathtt{b}_1$ or $\mathtt{b}_2$. Using the multiset semantics [6], this is modeled by the pairs $(\{\!|\mathtt{b}|\!\}, \{\!|\mathtt{b}_1|\!\})$ and $(\{\!|\mathtt{b}|\!\}, \{\!|\mathtt{b}_2|\!\})$. However, this interpretation does not give a recipe for a reasonable defender to counter the attack, which is executing both $\mathtt{b}_1$ and $\mathtt{b}_2$.

To make use of integer programming, we thus need to develop a new semantics for ADTrees, called *defense semantics for ADTrees*, expressing how the defender may prohibit a reasonable attacker from achieving their goal.

Definition 4. *The* defense semantics *of an ADTree T, denoted by $[\![T]\!]_{\mathcal{D}}$, is the set of all pairs (A, D), where A is an attack strategy in T and D is a minimal (with respect to inclusion) defense strategy in T, such that executing all actions from D makes it impossible for the attacker to achieve the goal represented by the root of T while realizing only the actions from A.*

In order to develop an algorithm constructing the defense semantics of an ADTree T, we first define the notion of countering attack and defense vectors.

Definition 5. *Let T be an ADTree, and $H^{\mathtt{A}}$ (resp. $H^{\mathtt{D}}$) be a homogenous subtree of T of the attacker's (resp. of the defender's) type. In addition, let a be an attack vector in $H^{\mathtt{A}}$ (resp. d be a defense vector in $H^{\mathtt{D}}$).*

We say that a set $S \subseteq \mathbb{B}^{\mathtt{D}}$ (resp. $S \subseteq \mathbb{B}^{\mathtt{A}}$) counters *the attack vector a (resp. the defense vector d), if executing all actions from S makes it impossible for the attacker (resp. for the defender) to achieve the goal represented by the root of $H^{\mathtt{A}}$ (resp. $H^{\mathtt{D}}$) while executing only the actions from a (resp. from d).*

In other words, S counters an attack vector a if, in the presence of the countermeasures from S, it is not sufficient to execute only the actions from a to achieve the root goal of the corresponding homogenous subtree. For instance, the attack strategy $\{\mathtt{hash}, \mathtt{ophcrack}, \mathtt{steal\text{-}kf}\}$ in the tree T from Fig. 1 counters the defense vector $\{\mathtt{key\text{-}file}\}$. Likewise, the defense vector $\{\mathtt{password}\}$ counters the attack vector$\{\mathtt{hash}, \mathtt{ophcrack}\}$.

Algorithm 1, where H_0 denotes the homogenous subtree of the attacker containing the root of an ADTree T, gives an algorithmic way of creating the defense semantics of T. In the corresponding lines, we indicate the steps the complexity of which is exponential wrt the number of nodes. Note however, that, for a given tree, these worst case scenario estimations will never hold for all of the lines at the same time, cf. Table 1 depicting a couple of empirical results.

Theorem 1. *Given ADTree T, Algorithm 1 generates the semantics $[\![T]\!]_{\mathcal{D}}$.*

To prove Theorem 1, we need the following lemma which shows the uniqueness of the attack vector a' in line 14 of Algorithm 1.

Lemma 2. *Let T be an ADTree and $(A, D) \in [\![T]\!]_{\mathcal{D}}$. For every defense vector $d \subseteq D$, there exists at most one attack vector $a \subseteq A$ countering d.*

Proof. Let $(A, D) \in [\![T]\!]_{\mathcal{D}}$ and let $d \subseteq D$ be a defense vector. By contraposition, suppose that there exist two distinct attack vectors a_1 and a_2 in A that counter d. Denote by N_d the set of nodes achieved by d, and by H_d the homogenous subtree containing N_d. By Lemma 1, a_1 and a_2 are attack vectors from distinct homogenous subtrees of T, say, H_1 and H_2, respectively. For $i \in \{1, 2\}$, denote by n_i the node of N_d, to which the root of H_i is attached. Let $n \in N_d$ be the lowest common ancestor of n_1 and n_2, i.e., the node that lies on the paths connecting n_1 and n_2 with the root of H_d, the distance of which from the root of H_d is maximal.

Let D_A be a witness for A. By definition of witness, neither $A \setminus a_1$ nor $A \setminus a_2$ achieves the root goal of T in the presence of the actions from D_A. Minimality of A wrt D_A implies that each of the nodes n_1, n_2, and n is achieved by D_A and that n is neither n_1 nor n_2. Furthermore it follows that n is an `OR` node. However, this means that achieving both n_1 and n_2 is not necessary for achieving n, which contradicts d being a defense vector in H_d. $\square$

Proof of Theorem 1. Let Alg(T) be the set constructed by Algorithm 1. We prove that Alg(T) $= [\![T]\!]_{\mathcal{D}}$.

First, let $(A, D) \in [\![T]\!]_{\mathcal{D}}$. By Definition 4 and Lemmas 1 and 2, the sets A and D can be represented as

$$A = a_0 \cup a_1 \cup \cdots \cup a_m \cup A', \quad D = d_0 \cup d_1 \cup \cdots \cup d_m, \tag{2}$$

where a_0 is an attack vector in H_0, and for all $i \in \{0, \ldots, m-1\}$, d_i is a defense vector that counters a_i, and a_{i+1} is the unique attack vector in A countering d_i. Furthermore, defense vector d_m counters a_m and executing any of the attack vectors from A' has no impact on the defense vectors from D, and vice versa.

Let $i \in \{1, \ldots, m\}$. By lines 8–15, during the i-th execution of the **while** loop, the pair $(a_i, d_0 \cup \cdots \cup d_{i-1})$ is added to the NewCandidates set, with the latter becoming the Candidates set in line 19. When the algorithm enters the **while** loop for the $(m+1)$-th time and the aforementioned pair is considered in line 8, the defense vector d_m is identified in line 9 and the set D is added to the set MinDef. Then, the pair (A, D) is added to Alg(T) in line 21.

Now, let $(A, D) \in$ Alg(T). Let $a_0 \subseteq A$ be the attack vector from line 5. Observe that the set D was built upon the pair $(a_0, \emptyset)$ by repeatedly identifying a defense vector d_i countering the first element of the pair and an attack vector $a_{i+1} \subseteq A$ (if any) countering d_i, and then replacing the previous pair in the set of candidates as described in line 19. Hence, the sets A and D can be partitioned as in decomposition (2), and so $(A, D) \in [\![T]\!]_{\mathcal{D}}$. $\square$

Algorithm 1 Defense semantics for ADTrees

```
Input: ADTree $T$
Output: defense semantics $[\![T]\!]_{\mathcal{D}}$
 1: Construct the set $\mathrm{AS}(T)$ using the rules from Fig. 2                    $\mathcal{O}(2^n)$
 2: $[\![T]\!]_{\mathcal{D}} \leftarrow \emptyset$
 3: for $A \in \mathrm{AS}(T)$ do                                                       $\mathcal{O}(2^n)$
 4:     MinDef $\leftarrow \emptyset$
 5:     Candidates $\leftarrow \{(a_0, \emptyset) \mid a_0 \subseteq A$ is an attack vector in $H_0\}$
 6:     while Candidates $\neq \emptyset$ do
 7:         NewCandidates $\leftarrow \emptyset$
 8:         for $(a, D) \in$ Candidates do
 9:             Counter $\leftarrow \{d \mid d$ counters $a\}$                                 $\mathcal{O}(2^n)$
10:             for defense vector $d \in$ Counter do                                   $\mathcal{O}(2^n)$
11:                 if $d$ is not countered by $A$ then
12:                     MinDef $\leftarrow$ MinDef $\cup \{D \cup d\}$
13:                 else
14:                     let $a'$ be the unique attack vector in $A$ that counters $d$
15:                     NewCandidates $\leftarrow$ NewCandidates $\cup \{(a', D \cup d)\}$
16:                 end if
17:             end for
18:         end for
19:         Candidates $\leftarrow$ NewCandidates
20:     end while
21:     $[\![T]\!]_{\mathcal{D}} \leftarrow [\![T]\!]_{\mathcal{D}} \cup \{(A, D) \mid D \in$ MinDef $\}$
22: end for
23: return $[\![T]\!]_{\mathcal{D}}$
```

Example 5. The defense semantics of our running tree T from Fig. 1 is

$$\begin{aligned}
[\![T]\!]_{\mathcal{D}} = \{ & (\{\texttt{hash}, \texttt{ophcrack}\}, \{\texttt{strong}\}),\\
& (\{\texttt{hash}, \texttt{ophcrack}\}, \{\texttt{password}\}),\\
& (\{\texttt{hash}, \texttt{ophcrack}\}, \{\texttt{key-file}\}),\\
& (\{\texttt{hash}, \texttt{ophcrack}, \texttt{eavesdrop}\}, \{\texttt{strong}\}),\\
& (\{\texttt{hash}, \texttt{ophcrack}, \texttt{eavesdrop}\}, \{\texttt{key-file}\}),\\
& (\{\texttt{hash}, \texttt{ophcrack}, \texttt{eavesdrop}\}, \{\texttt{password}, \texttt{sec-train}\}),\\
& (\{\texttt{hash}, \texttt{ophcrack}, \texttt{steal-kf}\}, \{\texttt{strong}\}),\\
& (\{\texttt{hash}, \texttt{ophcrack}, \texttt{steal-kf}\}, \{\texttt{password}\}),\\
& (\{\texttt{hash}, \texttt{ophcrack}, \texttt{steal-kf}, \texttt{eavesdrop}\}, \{\texttt{strong}\}),\\
& (\{\texttt{hash}, \texttt{ophcrack}, \texttt{steal-kf}, \texttt{eavesdrop}\}, \{\texttt{password}, \texttt{sec-train}\})\}.
\end{aligned}$$

3 Security-oriented optimization problems

We integrate the information captured by the defense semantics defined in the previous section with the integer linear programming. Linear programming is a standard approach to compute the best outcome, by optimizing a linear objective function subject to linear equality and linear inequality constraints. In our framework, the inequalities model the dependencies between attack strategies and defense strategies expressed by the defense semantics, the constraints related to the available budget, as well as cost and impact of individual actions of the attacker and the defender. Provided that the defender's budget is limited, they might not be able to implement all countermeasures at will. Our framework supports them in tackling the following types of the optimization problems

- *maximal coverage* – minimize the number of attack strategies that remain uncountered;
- *minimal impact* – minimize the impact of uncountered attack strategies;
- *maximal investment* – maximize the necessary investment of the attacker.

3.1 Mathematical modeling

We start by fixing the notation that we employ in this section to model the optimization problems. Given an ADTree T, and its defense semantics $[\![T]\!]_{\mathcal{D}}$, let

- $\mathbb{B}^{\mathtt{D}} = \{\mathtt{b}_1^{\mathtt{D}}, \ldots, \mathtt{b}_p^{\mathtt{D}}\}$ be the set of basic actions of the defender present in T,
- $A_1, \ldots, A_n$ be the distinct attack strategies that appear in $[\![T]\!]_{\mathcal{D}}$,
- $D_1, \ldots, D_m$ be the distinct defense strategies that appear in $[\![T]\!]_{\mathcal{D}}$.

Furthermore, for $i \in \{1, \ldots, n\}$, $j \in \{1, \ldots, m\}$, and $k \in \{1, \ldots, p\}$, we set

$$P_{ij} = \begin{cases} 1, & \text{if } (A_i, D_j) \in [\![T]\!]_{\mathcal{D}} \\ 0, & \text{otherwise} \end{cases} \qquad y_{kj} = \begin{cases} 1, & \text{if } \mathtt{b}_k^{\mathtt{D}} \in D_j \\ 0, & \text{otherwise.} \end{cases}$$

For a basic action $\mathtt{b}^{\mathtt{S}}$ of any of the actors $\mathtt{S} \in \{\mathtt{A}, \mathtt{D}\}$, we assume the cost of executing the action to be a non-negative real number $\mathrm{cost}(\mathtt{b}^{\mathtt{S}})$. Finally, the overall budget available to the defender is denoted by $\mathcal{B}$.

To model the different scenarios that may happen depending on which actions are and which are not executed, the following Boolean variables are defined

- $x_k = 1$, for $k = 1, \ldots, p$, if and only if the defender executes the action $\mathtt{b}_k^{\mathtt{D}}$,
- $f_j = 1$, for $j = 1, \ldots, m$, if and only if the defender does not execute at least one of the basic actions from the defense strategy D_j,
- $z_i = 1$, for $i = 1, \ldots, n$, if and only if the attack strategy A_i achieves the root node of T, in the presence of currently deployed countermeasures.

Coverage problem. We first focus on the problem of covering as many attack strategies as possible, provided the value of the defense budget $\mathcal{B}$.

Fig. 3 gives the specification of the corresponding integer linear programming problem.

$$\textbf{Optimization goal:} \qquad \text{minimize} \sum_{i=1}^{n} z_i \tag{3}$$

$$\textbf{Subject to:} \qquad \sum_{k=1}^{p} \text{cost}(\mathtt{b}_k^{\mathtt{D}}) x_k \leq \mathcal{B} \tag{4}$$

$$f_j \geq \frac{\sum_{k=1}^{p} y_{kj}(1 - x_k)}{p}, \ 1 \leq j \leq m \tag{5}$$

$$f_j \leq \sum_{k=1}^{p} y_{kj}(1 - x_k), \ 1 \leq j \leq m \tag{6}$$

$$z_i \geq 1 + \sum_{j=1}^{m} P_{ij}(f_j - 1), \ 1 \leq i \leq n \tag{7}$$

$$z_i \leq \frac{\sum_{j=1}^{m} P_{ij} f_j}{\sum_{j=1}^{m} P_{ij}}, \ 1 \leq i \leq n \tag{8}$$

$$x_k \in \{0,1\}, \ 1 \leq k \leq p, \qquad f_j \in \{0,1\}, \ 1 \leq j \leq m, \quad z_i \in \{0,1\}, \ 1 \leq i \leq n.$$

Fig. 3: Coverage problem modeled in terms of integer programming.

Inequality (4) ensures that the defender's investment cannot exceed their budget. The next two lines model the meaning of the variable f_j: inequalities (5) ensure that if the defender does not execute some of the actions from D_j, then $f_j = 1$; inequality (6) ensures that if $f_j = 1$, then the defender does not execute some action from D_j. Next, we model the meaning of z_i: inequalities (7) ensure that if the defender does not execute some action in any of the sets countering A_i (i.e., $f_j = 1$ for every j, such that $P_{ij} = 1$), then $z_i = 1$; and inequalities (8) ensure that if the defender executes all the actions from at least one of the sets D_j countering the attack strategy A_i (i.e., there exists j, such that $P_{ij} = 1$ and $f_j = 0$), then $z_i = 0$.

Remark 1. We notice that the number of elements in the set D_j can be expressed as $|D_j| = \sum_{k=1}^{p} y_{kj}$. Thus, the defender executes all of the actions from D_j, iff

$$\sum_{k=1}^{p} y_{kj} = \sum_{k=1}^{p} x_k y_{kj} \quad \text{which means} \quad \sum_{k=1}^{p} (1 - x_k) y_{kj} = 0. \tag{9}$$

In consequence, if there exists j for which equality (9) holds and $P_{ij} = 1$, then the attacker cannot succeed by the attack strategy A_i. Conversely, if for all j with $P_{ij} = 1$, equality (9) does not hold, then the attacker can succeed with A_i. This explains the form of inequalities (5) and (6).

Observe that when the inequalities from Fig. 3 are expressed in a matrix form, say $M\hat{\mathbf{x}} \leq \hat{\mathbf{c}}$, where $\hat{\mathbf{x}} = (x_1, \ldots, x_p, f_1, \ldots, f_m, z_1 \ldots, z_n)$ and $\hat{\mathbf{c}}$ is a vector of constants, then the size of M is $(2m+2n+1) \times (p+m+n)$. For the completeness of presentation of results, sizes of corresponding matrices are included in Table 1.

Below, we investigate other optimization problems that fall into our setting.

Impact problem. Here it is assumed that every attack strategy A has assigned a value $\mathrm{Imp}(A)$ of it's impact when executed. The value of $\mathrm{Imp}(A)$ could be estimated by the security experts or expressed as the sum of impacts of basic actions composing the attack strategy A. The goal is to select the countermeasures to be implemented in such a way that the impact of uncountered attack strategies is minimal. The optimization goal from line (3) in Fig. 3 is replaced with

$$\textbf{Optimization goal:} \quad \text{minimize I}$$

and the list of inequalities from Fig. 3 is extended with additional constraints

$$\mathrm{I} \geq z_i \, \mathrm{Imp}(A_i), \; 1 \leq i \leq n, \tag{10}$$

ensuring that I is the maximum of the impacts of uncountered attack strategies.

Attacker's investment problem. We may use a similar technique to maximize the minimal necessary investment of the attacker in achieving their goal. In this case the optimization goal is replaced with

$$\textbf{Optimization goal:} \quad \text{maximize C}$$

with respect to the same conditions as previously, extended with

$$\mathrm{C} \leq z_i \, \mathrm{Cost}(A_i) + (1 - z_i) \sum_{q=1}^{n} \mathrm{Cost}(A_q), \; 1 \leq i \leq n, \tag{11}$$

where $\mathrm{Cost}(A_i)$ is equal to the investment of the attacker they need to make in order to perform all basic actions from A_i, i.e. $\mathrm{Cost}(A_i) = \sum_{\mathtt{b}^{\mathtt{A}} \in A_i} \mathrm{cost}(\mathtt{b}^{\mathtt{A}})$.

Remark 2. The mathematical framework described in this section is generic and, as such, can be used to address not only problems relating to budget, impact, and monetary cost, but also any other optimization problem using the same mathematical structure, i.e., when the objective is to optimize a linear function of the Boolean variables defined in Sect. 3.1 under linear constraints. This is illustrated in Sect. 3.3, where we look for the optimal usage of available time.

3.2 Implementation

To validate the framework developed in this paper, we have implemented a proof of concept tool. It is programmed in Python and uses a free integer linear programming solver *lp_solve* [2]. Given an ADTree T, specified as in Definition 1, and the input values for the defense budget and cost, our prototype follows Algorithm 1 to construct the defense semantics $[\![T]\!]_{\mathcal{D}}$, and extracts the specification of the optimization problem of interest, as described in Sect. 3.1. The optimization problem is then solved using *lp_solve* and the optimal solution, i.e., the optimal value of the objective function together with the corresponding set of the defender's actions that need to be performed, is given as output.

We have tested the prototype on a computer running Intel Core i7–5600U CPU at 2.60 GHz dual core with 16 GB of RAM. ADTrees for the tests have been generated randomly to cover various possible cases.

The budget $\mathcal{B}$ has been set to be half of the sum of the costs of all basic actions of the defender.

Size				Time in sec			
T	AS(T)	$[\![T]\!]_{\mathcal{D}}$	M	AS(T)	$[\![T]\!]_{\mathcal{D}}$	from $[\![T]\!]_{\mathcal{D}}$ to M	solving
28	8191	53248	16409×8217	129.36	3.67	8.8	1152.22
80	3955	57508	7951×3997	0.02	4.1	4.5	0.58
80	3	99	81×76	> 10800	0.01	< 0.01	0.01
100	25	32	67×107	< 0.01	0.06	< 0.01	> 3600
500	23	71	65×166	0.01	0.26	< 0.01	0.01

Table 1: Running time of the tool on randomly generated trees.

Table 1 presents a sample of the obtained results. It compares the time spend on generation of the set AS(T), generation of $[\![T]\!]_{\mathcal{D}}$, translation of the defense semantics into an integer programming problem specified by a matrix M, and solving the problem. In general, the most time-consuming of these steps are generation of AS(T) and solving of the obtained optimization problem, since in the worst case they are both exponential in the number of nodes of T. In particular, the time necessary to generate AS(T) depends exponentially on the maximum number of children of the $\mathtt{OR}^{\mathtt{D}}$ nodes in T, cf. Fig. 2.

3.3 Countermeasure optimization on the running example

We now illustrate the optimal countermeasure selection on our running scenario from Example 1. Here, the budget $\mathcal{B}$ represents the available time resources.

We suppose that the goal of Bob (the defender) is to *learn* how to secure his computer against the attacks depicted in the tree from Fig. 1. Bob is a busy person, so he can devote 50 min only to his learning process. He wants to know how he should spend this time in the most efficient way, i.e., so that he is able to minimize the number of unprevented attacks. To set up a password which is resistant to the *ophcrack* attack, Bob needs to understand how the rainbow table analysis works – this would take him 60 min. To be able to use *DiskCryptor*, 20 min are necessary to learn how to use it with a password and 30 min to master how to handle a key file. Finally, Bob can also follow the security training offered by his company, which lasts 25 min.

We have input these data to our tool and obtained a matrix of size 17×12. The tool solves the problem instantaneously and suggests that Bob should follow the security training and learn how to use *DiskCryptor* with a password, i.e., the optimal set of countermeasures is $\{\mathtt{password}, \mathtt{sec\text{-}train}\}$ and it prevents all four attack strategies listed in Example 5. However, if the duration of the security training was 45 min, then the optimal set of countermeasures would be $\{\texttt{password}, \texttt{key-file}\}$ which prevents three out of four attack strategies, namely $\{\texttt{hash}, \texttt{ophcrack}\}$, $\{\texttt{hash}, \texttt{ophcrack}, \texttt{eavesdrop}\}$, and $\{\texttt{hash}, \texttt{ophcrack}, \texttt{steal-kf}\}$.

4 Conclusion

The goal of the work presented in this paper has been to provide a framework to assist industry practitioners using ADTrees in performing their risk assessment evaluations. To achieve this, the security model of ADTrees is fused with optimization techniques. We rely on the expressive power of ADTrees to link potential attack and defense strategies and profit from the strengths of integer programming to select the most optimal sets of countermeasures. From a formal perspective, we introduce a novel defense semantics for ADTrees and thus contribute to the developments of mathematical foundations for this security model. To validate the usefulness of the proposed approach, we have implemented a proof of concept tool automating the computation of the defense semantics and the selection of the most appropriate set of countermeasures to be deployed.

As a next step, we will extend our framework to the probabilistic case, taking the probability with which actions are executed into account. We would also like to improve the worst case running time of our tool, by exploiting the possibility of using approximation algorithms. Finally, we plan to integrate this framework to ADTool, free software assisting creation and quantitative analysis of ADTrees [5].

References

1. Aslanyan, Z., Nielson, F.: Pareto Efficient Solutions of Attack–Defense Trees. In: POST'15. LNCS, vol. 9036, pp. 95–114. Springer (2015)
2. Berkelaar, M., Eikland, K., Notebaert, P.: lp_solve: Open source (Mixed-Integer) Linear Programming system (2005), `http://lpsolve.sourceforge.net/5.5/`, version 5.5.2.5, dated September 24, 2016
3. Chvátal, V.: Linear Programming. W. H. Freeman (1983)
4. DiskCryptor: `https://diskcryptor.net/` (2014), accessed on: 2017-03-17
5. Gadyatskaya, O., Jhawar, R., Kordy, P., Lounis, K., Mauw, S., Trujillo-Rasua, R.: Attack trees for practical security assessment: Ranking of attack scenarios with ADTool 2.0. In: QEST. LNCS, vol. 9826, pp. 159–162. Springer (2016)
6. Kordy, B., Mauw, S., Radomirovic, S., Schweitzer, P.: Attack–defense trees. Journal of Logic and Computation 24(1), 55 – 87 (2014)
7. Kordy, B., Pouly, M., Schweitzer, P.: Probabilistic reasoning with graphical security models. Information Sciences 342, 111–131 (2016)
8. Mauw, S., Oostdijk, M.: Foundations of Attack Trees. In: ICISC 2005. LNCS, vol. 3935, pp. 186–198. Springer (2005)
9. Ophcrack: `http://ophcrack.sourceforge.net/` (2016), accessed on: 2017-03-17
10. Schneier, B.: Attack Trees: Modeling Security Threats. Dr. Dobb's Journal of Software Tools 24(12), 21–29 (1999)

11. Shameli-Sendi, A., , Louafi, H., He, W., Cheriet, M.: Dynamic optimal countermeasure selection for intrusion response system. IEEE Journal of TDSC (99) (2016)
12. Zheng, K., McLay, L.A., Luedtke, J.R.: A budgeted maximum multiple coverage model for cybersecurity planning and management (2017), under submission.