

Attack-Defense Trees with Offensive and Defensive Attributes (with Appendix)

1st Danut-Valentin Copae
University of Twente
d.v.copae@student.utwente.nl

2nd Reza Soltani
University of Twente
r.soltani@utwente.nl

3rd Milan Lopuhaä-Zwakenberg
University of Twente
m.a.lopuhaa@utwente.nl

Abstract—Effective risk management in cybersecurity requires a thorough understanding of the interplay between attacker capabilities and defense strategies. Attack-Defense Trees (ADTs) are a commonly used methodology for representing this interplay; however, previous work in this domain has only focused on analyzing metrics such as cost, damage, or time from the perspective of the attacker. This approach provides an incomplete view of the system, as it neglects to model defender attributes: in real-world scenarios, defenders have finite resources for countermeasures and are similarly constrained. In this paper, we propose a novel framework that incorporates defense metrics into ADTs, and we present efficient algorithms for computing the Pareto front between defense and attack metrics. Our methods encode both attacker and defender metrics as semirings, allowing our methods to be used to many metrics such as cost, damage, and skill. We analyze tree-structured ADTs using a bottom-up approach and general ADTs by translating them to binary decision diagrams. Experiments on randomly generated ADTs demonstrate that both approaches effectively handle ADTs with several hundred nodes.

Index Terms—attack trees, attack-defense trees, Pareto front, multi-criteria optimization

I. INTRODUCTION

Attack trees. Cyber-physical systems, such as autonomous vehicle networks or smart grids, can become notoriously complex when multiple actors are involved. The resulting complexity also raises the number of possible breaches that attackers can exploit, especially in systems where components rely on each other’s functioning to maintain safety. Consequently, there is a need for robust and systematic threat modeling systems that can cope with such attacks. In 1999, Schneier [1] introduced attack trees (ATs), which nowadays represent one of the most prominent tools for evaluating the security of complex systems. Due to their simplicity and compact form, ATs are commonly used in commercial software tools as well as industrial applications, e.g., analyzing the security of a SCADA system for a tank and pump facility [2] and impact analysis of electric grid feature scenarios [3].

The hierarchical structure of an attack tree models the root of the tree as the attacker’s primary goal. The tree branches represent different methods the attacker could take to achieve their primary goal. The leaves of these branches are basic attack steps (BASs), which cannot be further refined into finer sub-goals. Figure 1 shows an exemplary AT. This AT includes AND gates, activated when all of its children are enabled, and OR gates, activated when only a single child is enabled.

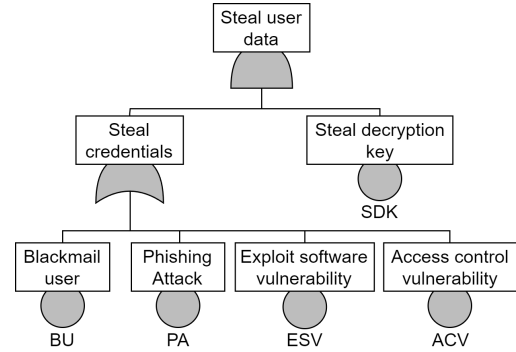


Fig. 1: An AT depicting how the attacker can steal user data. To obtain the user’s data, the attacker must obtain both credentials and the decryption key. The credentials can be stolen in four different ways: blackmailing the user (BU), conducting a phishing attack (PA), exploiting a software vulnerability (ESV), or leveraging access control vulnerabilities (ACV).

Attack-defense trees. The primary utility of ATs is to describe the various strategies an attacker can take to compromise a system through a structured decomposition of the attack into smaller objectives. This enables security experts to design countermeasures for preventing future attacks. However, one of the limitations of attack trees is that they do not account for the countermeasures implemented to prevent an attack. For this reason, attack-defense trees (ADTs) were introduced by Kordy et al. [4] as an extension of regular ATs to model the attacks on a system concurrently and the defenses to block those attacks. The defenses work by deactivating the attack nodes they are associated with, thereby disabling them. Figure 2 extends Fig. 1 by adding counter-attack (defense) nodes. The basic defense steps (BDSs) are highlighted in green for better visualization. A defense node meets an attack node at an INH (inhibition) gate. This gate has exactly two children of opposite types, and one acts as an inhibitor. The INH gate is deactivated in the presence of an active inhibitor. On the contrary, its output equals its input when the inhibitor is not activated. For clarity, the edge leading to the inhibitor child of INH gates is marked with a small circle.

Quantitative analysis. Quantitative analysis in ADTs enables the evaluation of different strategies by assigning measurable values to actions. Common metrics, such as cost and prob-

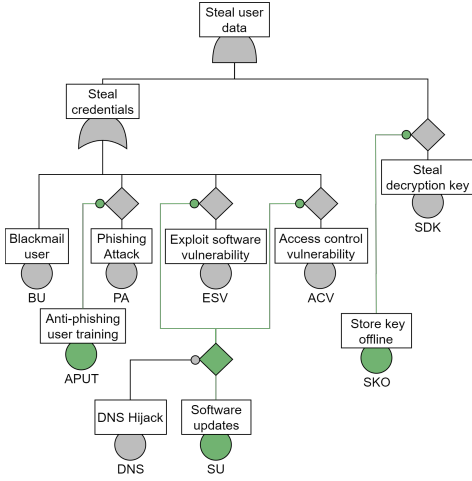


Fig. 2: Attack-defense tree extending the attack tree of Fig. 1. The defender can prevent phishing attacks (*PA*) through anti-phishing user training (*APUT*), and *SDK* through *SKO*. Regular software updates (*SU*) prevent both *ESV* and *ACV*. DNS Hijack (*DNS*), which does not directly contribute to reaching the top node, disables the *SU* defense. Lastly, blackmailing the user (*BU*) has no countermeasure.

abilities, help assess the risks of attack paths and inform defensive planning. In the current state of the attack-defense trees research, although there are two actors in an attack scenario (i.e., an attacker and a defender), only the attacker’s actions are annotated with quantifiable attribute values [5], [6]. This approach fails to fully capture reality, as the defender, like the attacker, usually has finite resources. The defender aims to select the most optimal defenses, typically those with a lower defense cost, while simultaneously making the situation more difficult for the attacker (e.g., maximizing the attacker cost). Note that the defender’s and attacker’s goals are conflicting: the attacker seeks to minimize their cost, whereas the defender aims to maximize attack costs with minimum defense cost. The defender’s optimal strategy aims to reduce their own defense costs while simultaneously maximizing the difficulty for the attacker, whether in terms of cost, effort, or time. Also, the defender’s choice are constrained by a budget. The set of maximal achievable attacker costs or efforts for each possible defender budget forms the *Pareto front*. This leads to our research goal.

Research Goal: Find efficient algorithms that compute the Pareto front between attacker and defender metrics for Attack-Defense Trees.

By leveraging the Pareto front, defenders can make better-informed decisions to allocate resources efficiently. This enables security analysts to evaluate trade-offs between attacker and defender strategies more comprehensively, aiding in more informed and cost-effective decision-making for real-world system protection.

Most approaches, which we discuss in detail in the next section, focus on single-parameter optimization or attacker-centric metrics, leaving defender attributes and multi-objective optimization underexplored. Some researchers explored multi-objective and Pareto-efficient solutions for ADTs [7], [8]. Nevertheless, these works have limitations. For instance, the work of Fila and Widel [7] focuses on multi-parameter optimization within ADTs but only considers metrics that are simultaneously applicable to both attackers and defenders, such as cost or time. Moreover, their approach fixes the metric values for one of the parties at ∞ , thus essentially only analyzing the metrics for one party. Similarly, the approach of Aslanyan and Nielson [8] primarily addresses Pareto fronts between attacker-centric metrics without extending to defender-specific attributes or the interplay between attacker and defender objectives.

In this paper, we address the limitations of existing ADT approaches by introducing a comprehensive framework that is built on formal definitions of ADTs, including their syntax and semantics, which enable the representation of attacker and defender attributes through the use of semirings. For tree-structured ADTs, we propose an efficient Bottom-Up (BU) algorithm that processes nodes iteratively from the leaves to the root, aggregating metrics to compute the Pareto front of attack and defense strategies. For more general ADTs, which include directed acyclic graph (DAG) structures, we develop a Binary Decision Diagram (BDD)-based approach to capture complex relationships between nodes and ensure scalability. Both methods were rigorously evaluated using a test suite of randomly generated ADTs with sizes up to 325 nodes, demonstrating their practicality and effectiveness. Our main contributions are:

- Formal definitions of ADTs with attacker and defender attribute domains, including a semiring-based representation of metrics;
- A Bottom-Up algorithm for computing the Pareto front in tree-structured ADTs;
- A Binary Decision Diagram algorithm for efficiently handling the Pareto front computation for ADTs with DAG structures;
- Experimental validation showcasing the performance of the algorithms on large-scale ADTs.

Paper organization: The paper is organized as follows. Section II outlines the background of ADTs and highlights existing research gaps. In Section III, we introduce the ADT formalism, detailing its semantics, metrics, and Pareto analysis. Section IV focuses on computing the Pareto front for tree-structured ADTs, while Section V extends this discussion to DAG-structured ADTs. Finally, Section VI presents experimental results and evaluates the performance of the proposed Pareto front computation methods. Finally, Section VII concludes the paper and provides future work.

II. RELATED WORK

Attack-Defense Tree (ADT) is a concept introduced by Kordy et al. [4], which gives the system the possibility of

modeling defenses through *counter-attack* gates. Before this work, there had already been defined ideologies of ADTs, but they were tailored to more specific use cases. For instance, in [9], defensive actions are only possible at the leaf level. Similarly, in [10], the defender can only perform counter-attacks at the leaf level but cannot have higher-level goals modeled in the tree. In their paper, Kordy et al. [4] provided a more general concept of ADTs, where attackers and defenders have equal capabilities, and counter-attacks can be modeled at intermediate nodes, including the root node. This approach offers a more comprehensive overview of the security aspects of a system.

Traditional ADT research has primarily focused on attacker-centric metrics. However, some frameworks demonstrated the value of integrating both attack and defense perspectives to better capture system dynamics [11]. Arias et al. in [12] analyzed ADTs in a novel way by treating these trees as an extension of asynchronous multi-agent systems. Each node in the tree is treated as an agent that can act asynchronously. The transition functions of these nodes are then equipped with attributes. Finally, the quantitative results from the generated automata are verified with state-of-the-art model checkers such as UPPAAL and Imitator. Similarly, some methods for optimizing spare management [13] and dynamically modeling environmental behavior for energy efficiency [14], [15] highlight the importance of considering multiple attribute domains in system modeling. Nevertheless, most analytical methods optimize one parameter at a time, such as the cost or time of an attack. However, this approach might not accurately represent complex real-world scenarios where parameters can interact (e.g., the maximum damage of an attack, given a fixed cost [16]), potentially leading to potentially sub-optimal solutions. To analyze multiple parameters simultaneously, the leaf nodes need to be annotated with multiple values. This creates a multi-optimization problem, as there is no single solution anymore, but a set of Pareto efficient solutions called the *Pareto Front*, where another does not dominate one solution in a given ordering relation [6]. For instance, the main intuition behind this ordering, assuming the Pareto front between the attacker's damage and cost, is that if the attacker has two strategies with the same damage, but one has a lower cost, they have no incentive to choose the higher-cost strategy.

Efforts to optimize multiple parameters in ADTs have been studied in the literature [7], [8]. The authors in [7] propose a framework for analyzing Pareto fronts in ADTs by considering multi-parameter optimization; however, their approach is limited to metrics that apply simultaneously to both attacker and defender actions, such as shared costs or time requirements. This limitation restricts its applicability to scenarios where defender-specific strategies, such as investing additional resources to impede attackers, need to be considered. Aslanyan and Nielson [8] extend ADTs to model attacker and defender interactions using a type system and propose techniques for computing Pareto-efficient solutions. Their work focuses on attacker-specific metrics, such as the cost and probability of attacks, but does not incorporate defenders' distinct metrics or

strategic goals.

Our work bridges this gap by introducing a formal framework for analyzing the interplay between attacker and defender metrics in ADTs, leveraging efficient algorithms to compute the Pareto front across these metrics. Unlike prior works, our framework supports the interplay and analysis of attacker and defender strategies (metrics), offering a comprehensive view of trade-offs in security planning.

III. ATTACK-DEFENSE TREE FORMALISM

In this section, we introduce the formalism of the ADT, which is a structured approach to visualize and analyze the interactions between potential attacks and defensive mechanisms. We explore ADT aspects through three key dimensions: Semantics, Metrics, and the concept of the Pareto Front.

A. Semantics

An Attack-Defense Tree (ADT) is a structured diagram that models potential security threats and the defensive actions that can counter them. ADT semantics define the rules for interpreting an ADT, specifying how attacks and defenses interact within the structure. Our definition mostly follows [17], though we take a graph-theoretic approach rather than a grammar approach. Like standard ATs, we have AND- and OR-gates. Furthermore, the effects of countermeasures on incoming attacks are modeled by Inhibition gates (INH); these correspond to the C-gates of [17]. Inhibition gates v have two inputs: an *trigger* $\vartheta(v)$ that can stop propagation, and an *inhibited* $\theta(v)$ that is required for propagation. Finally, each gate v is assigned an agent $\tau(v)$, either attacker (A) or defender (D). Any gate type can be assigned to both agents; hence, it is also possible for defender-held inhibition gates to be inhibited by further attacker actions.

Definition 1 (Attack-Defense Tree). An attack-defense tree is a quintuple $T = (N, E, \gamma, \tau, \vartheta)$, where (N, E) is a rooted directed acyclic graph; γ and τ are functions $\gamma: N \rightarrow \{\text{BS}, \text{AND}, \text{OR}, \text{INH}\}$, $\tau: N \rightarrow \{\text{A}, \text{D}\}$; and ϑ is a function $\vartheta: \{v \in N \mid \gamma(v) = \text{INH}\} \rightarrow N$ such that $\langle v, \vartheta(v) \rangle \in E$. Moreover, T satisfies the following constraints for a node $v \in N$:

- $\gamma(v) = \text{BS}$ if and only if v is a leaf of (N, E) .
- If $\gamma(v) = \text{INH}$, then v has two children with different τ -values, with $\tau(\vartheta(v)) \neq \tau(\theta(v))$.
- if $\gamma(v) \in \{\text{OR}, \text{AND}\}$, then for all children w of v , $\tau(w) = \tau(v)$.

The root of T is denoted as R_T , and the set of children of a node v as $ch(v) = \{w \in N \mid (v, w) \in E\}$. The set of Basic Attack Steps (BASs) on T , denoted \mathcal{A} , is the set of all nodes $v \in N$ for which $\gamma(v) = \text{BS}$ and $\tau(v) = \text{A}$. Similarly we write \mathcal{D} for the set of Basic Defense Steps (BDSs), i.e., nodes v for which $\gamma(v) = \text{BS}$ and $\tau(v) = \text{D}$. These two sets are disjoint, that is, $\mathcal{A} \cap \mathcal{D} = \emptyset$, and the union of these two sets represents the set of all basic events. If $\gamma(v) = \text{INH}$, we write $\vartheta(v)$ for the trigger child, i.e., $ch(v) = \{\vartheta(v), \theta(v)\}$.

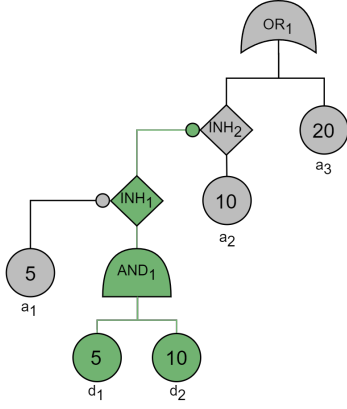


Fig. 3: Tree-structured ADT annotated with offensive and defensive costs.

In an attack-defense tree, both the defender and the attacker choose a set of BDS/BAS to activate. We represent these sets as binary vectors.

Definition 2 (Event). An *attack vector* is a binary vector $\vec{\alpha} \in \mathbb{B}^{\mathcal{A}}$. A *defense vector* is a binary vector $\vec{\delta} \in \mathbb{B}^{\mathcal{D}}$. An *event* is a pair $(\vec{\delta}, \vec{\alpha})$ of a defense vector and an attack vector.

Figure 3 is an example of a tree-structured ADT annotated with numbers representing the cost. In this illustration, the set of all attacks is $\mathcal{A} = \{a_1, a_2, a_3\}$, and the set of defenses is $\mathcal{D} = \{d_1, d_2\}$. If the attacker activates a_2 and a_3 , but not a_1 , this forms the attack vector $\vec{\alpha} = 011$. Similarly, a defensive vector where only d_1 is activated is represented by $\vec{\delta} = 10$.

The extent to which the attacker and defender vectors determine system status is captured by the structure function:

Definition 3 (Structure Function). Let T be an ADT. The structure function of T is the function $f_T: \mathbb{B}^{\mathcal{D}} \times \mathbb{B}^{\mathcal{A}} \times N \rightarrow \mathbb{B}$ defined as:

$$f_T(\vec{\delta}, \vec{\alpha}, v) = \begin{cases} \alpha_v, & \text{if } v \in \mathcal{A} \\ \delta_v, & \text{if } v \in \mathcal{D} \\ \bigwedge_{w \in \text{ch}(v)} f_T(\vec{\alpha}, \vec{\delta}, w), & \text{if } \gamma(v) = \text{AND} \\ \bigvee_{w \in \text{ch}(v)} f_T(\vec{\alpha}, \vec{\delta}, w), & \text{if } \gamma(v) = \text{OR} \\ f_T(\vec{\alpha}, \vec{\delta}, \vartheta(v)) \wedge \neg f_T(\vec{\alpha}, \vec{\delta}, \bar{\vartheta}(v)) & \text{if } \gamma(v) = \text{INH.} \end{cases}$$

B. Metrics

Security metrics, such as the lowest attack time or cost, are critical for conducting quantitative assessments of systems and making educated decisions. To achieve this, we use a well-established method called the semiring framework. We define security and defense attribute domains as linearly ordered unital semiring attribute domains, extending the classical semiring structure by incorporating a linear order. This additional assumption ensures compatibility with the minimization and maximization operations required for ranking and prioritizing defense and attack strategies, which are not inherently supported by traditional semirings.

Definition 4 (Semiring Attribute Domain). A *linearly ordered unital semiring attribute domain* (simply semiring attribute domain) is a tuple

$$L = (V, \otimes, 1_{\otimes}, 1_{\oplus}, \leq)$$

where V is a set; 1_{\otimes} and 1_{\oplus} are elements of V ; \otimes is a commutative associative binary operation on V ; and \leq is a linear order on V . These furthermore satisfy the following properties:

- \otimes is monotonous w.r.t. \leq , i.e., for all $x, y, z \in V$ with $x \leq y$ one has $x \otimes z \leq y \otimes z$;
- 1_{\otimes} is both the unit of \otimes , and minimal w.r.t. \leq ;
- 1_{\oplus} is maximal w.r.t. \leq .

The terminology *semiring* is justified as follows: given a semiring attribute domain, one can define a binary operator \oplus on V by

$$x \oplus y = \min_{\leq}(x, y).$$

Then (V, \oplus, \otimes) is a semiring; in fact, it is *absorbing* in the sense of [18]. We require the semiring to be ordered by \leq in order to define Pareto optimality. However, this is not a stringent constraint, as Table I shows that many attack tree metrics fit into our framework.

Metric	V	\oplus	\otimes	1_{\oplus}	1_{\otimes}	\leq
min cost	$[0, \infty]$	min	+	∞	0	\leq
min time (sequential)	$[0, \infty]$	min	+	∞	0	\leq
min time (parallel)	$[0, \infty]$	min	max	∞	0	\leq
min skill	$[0, \infty]$	min	max	∞	0	\leq
probability	$[0, 1]$	max	\cdot	1	0	\geq

TABLE I: Semiring attribute domains

The attacker and defender have separate attribute domains. The attacker and defender attribute domains are described by \mathbb{D}_A and \mathbb{D}_D , respectively. β_A assigns an attribute value from V_A to each basic attack step in \mathcal{A} , while β_D does so to each basic defense step in \mathcal{D} . Since the attacker's attribute values lie in V_A and the defender's in V_D . Since the attacker and defender have separate attributes, we need to combine these two into an attribute pair to perform a quantitative analysis. The attribute pair for an event will naturally have values in $V_D \times V_A$.

Definition 5 (Augmented Attack-Defense Tree). An Augmented Attack-Defense Tree (AADT) is an extension of the attack-defense tree T with associated semiring attributes. For simplicity, we use T to refer to both the original attack-defense tree and the Augmented Attack-Defense Tree (AADT), as long as the context makes the distinction clear. An AADT is defined as a tuple:

$$T = (T, D_{\mathcal{D}}, D_{\mathcal{A}}, \beta_D, \beta_A)$$

Where:

- $D_{\mathcal{D}} = (V_D, \oplus_D, \otimes_D, 1_{\oplus}, 1_{\otimes})$ is the defender's semiring attribute domain,

- $D_A = (V_A, \oplus_A, \otimes_A, 1_\oplus, 1_\otimes)$ is the attacker's semiring attribute domain.

Each domain has an associated basic assignment function: $\beta_D : \mathcal{D} \rightarrow V_D$ and $\beta_A : \mathcal{A} \rightarrow V_A$ are functions.

Definition 6 (Metric Values). For a given AADT, the metric value of a defense vector $\vec{\delta}$ is given by:

$$\hat{\beta}_D : \mathbb{B}^{\mathcal{D}} \rightarrow V_D \text{ with } \hat{\beta}_D(\vec{\delta}) = \bigotimes_{\substack{d \in \mathcal{D} \\ \delta_d=1}} \beta_D(d)$$

while the metric value of an attack vector $\vec{\alpha}$ is given by:

$$\hat{\beta}_A : \mathbb{B}^{\mathcal{A}} \rightarrow V_A \text{ with } \hat{\beta}_A(\vec{\alpha}) = \bigotimes_{\substack{a \in \mathcal{A} \\ \alpha_a=1}} \beta_A(a)$$

Lastly, the metric value of an event is given by:

$$\hat{\beta} : \mathbb{B}^{\mathcal{D}} \times \mathbb{B}^{\mathcal{A}} \rightarrow V_D \times V_A \text{ with } \hat{\beta}((\vec{\delta}, \vec{\alpha})) = (\hat{\beta}_D(\vec{\delta}), \hat{\beta}_A(\vec{\alpha}))$$

Example 1. Let's apply the definition of $\hat{\beta}$ on Fig. 3 as an example to determine the metric values of the attack and defense vector pair. We define a defense vector by listing the names of the activated nodes while excluding the disabled ones. Similarly, we define an attack vector in the same manner. Thus, instead of using binary vectors, we use sets to indicate the nodes that are active in each case. For example, let 11 and 110 be a pair of defense and attack vectors, where 11 represents the active defense nodes (both d_1 and d_2), and 110 represents the active attack nodes (both a_1 and a_2 are active). Since we are working with the minimal cost domain, $\mathbb{D}_A = \mathbb{D}_D = (\mathbb{R}_{\geq 0}, \min, +)$. To determine the metric values of this vector pair, we apply the definition of $\hat{\beta}$:

$$\begin{aligned} \hat{\beta}_D(\{d_1, d_2\}) &= \bigotimes_{\substack{d \in \{d_1, d_2\} \\ \alpha_d=1}} \beta_D(d) = \beta_D(d_1) + \beta_D(d_2) \\ &= 5 + 10 = 15 \\ \hat{\beta}_A(\{a_1, a_2\}) &= \bigotimes_{\substack{a \in \{a_1, a_2, a_3\} \\ \alpha_a=1}} \beta_A(a) = \beta_A(a_1) + \beta_A(a_2) \\ &= 5 + 10 = 15 \end{aligned}$$

$$\hat{\beta}(\{d_1, d_2\}, \{a_1, a_2\}) = (15, 15)$$

Before we define the concept of a successful attack, it's important to clarify how the overall interaction between activated defenses and attacks is modeled in our framework. Each event in the AADT represents a specific attack and the corresponding defense that seeks to mitigate or prevent it. The effectiveness of a defense can vary based on the particular attacks it faces, and this interaction is what ultimately determines whether an attack is successful or not.

In our model, what is considered to be a successful attack depends critically on the root node of the tree. If the root node is an attack node, the attack is considered successful if the associated defense mechanisms fail to prevent it, which mathematically means that $f_T(\vec{\delta}, \vec{\alpha}, R_T) = 1$. On the other hand, if the root node is a defense node, the attack is deemed successful if it destroys the defense mechanism, corresponding

to $f_T(\vec{\delta}, \vec{\alpha}, R_T) = 0$. This distinction highlights the pivotal role of the tree's structure in determining the outcome of an attack and its dependence on the defenses in place.

C. Pareto analysis

In real-world security scenarios, defenses are typically deployed before any potential attacks occur, forming a proactive line of protection. In our framework, we mirror this by first activating the defense nodes. These defenses represent the system's initial response to potential threats, and once they are in place, the attacker proceeds with their actions based on the activated defenses. This approach reflects the sequential nature of real-life interactions, where defenses are set up to mitigate risks before attacks are launched. By structuring our model this way, we can more accurately analyze the effectiveness of different defense strategies and their impact on the success or failure of subsequent attacks.

Before defining the optimal attack response, we introduce the notation used here. The function \arg selects the attack vector that minimizes the combined metric value, allowing for any vector to be chosen if multiple vectors yield the same minimum. This minimization process reflects the attacker's optimal response to a given defense.

Definition 7 (Optimal Attack Response). The attacker's response to a defense $\vec{\delta}$ is:

$$\rho(\vec{\delta}) = \begin{cases} \arg \bigoplus_{\substack{\vec{\alpha} \in \mathbb{B}^{\mathcal{A}}; \\ f_T(\vec{\alpha}, \vec{\delta}, R_T)=1}} \hat{\beta}_A(\vec{\alpha}) & \text{if } \tau(R_T) = A \\ \arg \bigoplus_{\substack{\vec{\alpha} \in \mathbb{B}^{\mathcal{A}}; \\ f_T(\vec{\alpha}, \vec{\delta}, R_T)=0}} \hat{\beta}_A(\vec{\alpha}) & \text{if } \tau(R_T) = D \end{cases}$$

In this definition, the function of the root node $f_T(\vec{\delta}, \vec{\alpha}, R_T)$ determines the attack outcome. If the root node represents an attacker node ($\tau(R_T) = A$), the attacker succeeds if they can achieve $f_T(\vec{\delta}, \vec{\alpha}, R_T) = 1$. Conversely, if the root node represents a defender ($\tau(R_T) = D$), then $f_T(\vec{\delta}, \vec{\alpha}, R_T) = 0$ indicates that the defense has failed, resulting in an attack success. However, there are two potential issues with this approach:

- **Multiple Candidates:** If there are multiple attack vectors that satisfy the minimum $\arg \bigoplus$ value, any of these vectors may be chosen, as they all yield the same metric outcome for $\hat{\beta}_A(\rho(\vec{\delta}))$.
- **No Valid Candidates:** If no valid attack vectors exist for $\vec{\delta}$, we write $\rho(\vec{\delta}) = \times$ with $\hat{\beta}_A(\times) = 1_\oplus$. As 1_\oplus is maximal with respect to

This ensures that the optimal attack response is well-defined even in cases where a valid attack vector does not exist. The defender assumes that the attacker behaves optimally, i.e., always performs $\rho(\vec{\delta})$ when it exists. This leads to the following notion of *feasible events*:

Definition 8. Let T be an AADT. Its set of *feasible events* $\mathcal{S} \subseteq V_D \times (V_A \cup \{\times\})$ is defined as

$$\mathcal{S} = \{(\vec{\delta}, \rho(\vec{\delta})) \mid \vec{\delta} \in \mathbb{B}^{\mathcal{D}}\}.$$

Example 2. Consider the attack-defense tree in Fig. 3. For convenience, we write defense and attack vectors as binary strings; thus we write 011 for the attack vector $\vec{a} = (0, 1, 1)$, i.e. the attack consisting of a_2 and a_3 but not a_1 . When no defenses are active ($\vec{d} = 00$), the attacker can succeed by either 010 or 001. These have costs 10 and 20, respectively, and the attacker chooses the cheapest option; hence $\rho(00) = 010$. If only one defense is active, the attack responses remain the same as in the no-defense scenario, as a single defense alone is insufficient and has no effect due to the AND gate. When both defenses are active, the possible attacks are 110 (cost 15) and 001 (cost 20); hence $\rho(11) = 110$. Thus

$$\mathcal{S} = \{(00, 010), (01, 010), (10, 010), (11, 110)\}.$$

The trade-off between the defender's and attacker's actions can be analyzed via the Pareto Front. Specifically, the defender has two primary objectives: "minimizing" their own cost and "maximizing" the attacker's cost, with these terms defined according to the defender's order \leq_D and the attacker's order \leq_A , respectively. The Pareto Front represents the set of optimal trade-offs between these competing objectives. A point in the Pareto Front is called *Pareto Optimal* if there is no other solution better in all objectives. The concept of *better* is formally known as dominance, and to define it, a few prerequisite properties must first be established.

Definition 9 (Pareto dominant). Given two events and their valuations (s_1, t_1) and (s_2, t_2) , the pair (s_1, t_1) dominates (s_2, t_2) i.e. $(s_1, t_1) \sqsubseteq (s_2, t_2)$ when $s_1 \leq_D s_2$ and $t_1 \geq_A t_2$.

For a general poset (X, \sqsubseteq) , a point $x \in X$ is *Pareto optimal* if it is not dominated by any other point in X . Then, the Pareto frontier is the set of all Pareto optimal points in X , i.e. $\min_{\sqsubseteq} X = \{x \in X \mid \forall x' \in X. x' \neq x, x' \not\sqsubseteq x\}$.

Example 3. Consider the ADT where the defender has two possible costs, $d \in \{5, 10\}$, and the attacker has three possible costs, $a \in \{5, 10, 20\}$. Let $X = \{(10, 10), (5, 20), (5, 5)\}$. The pair $(5, 20)$ dominates both $(10, 10)$ and $(5, 5)$ because it satisfies $5 \leq_D 10$ and $5 \leq_D 5$ (lower defender cost) as well as $20 \geq_A 10$ and $20 \geq_A 5$ (higher attacker cost). The Pareto front for this specific example is $(5, 20)$, representing non-dominated trade-offs between defender and attacker costs.

Having all the necessary mathematical prerequisites defined, we can formally state the problem statement of this paper:

Research Goal. For an Augmented Attack-Defense Tree T , we aim to find the minimal elements of the Pareto Front $\text{PF}(T)$, defined as

$$\min_{\sqsubseteq} \hat{\beta}(\mathcal{S}) \subseteq V_D \times V_A.$$

In principle $\text{PF}(T)$ can be computed by calculating $\hat{\beta}(e)$ for all $e \in \mathcal{S}$, and computing the Pareto front of the set of pairs of metric values; thus we need to compute the metric values of $2^{|\mathcal{D}|}$ events. In the worst case, this is unavoidable, as the following example shows.

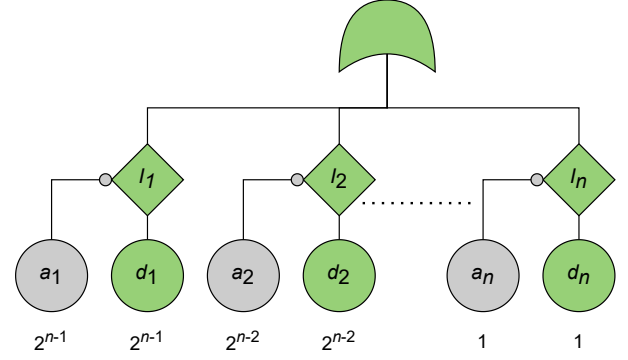


Fig. 4: An AADT (with min cost as both attacker and defender metrics) with $|\text{PF}(T)| = 2^n$.

Example 4. Consider the AADT T of Fig. 4 (for a fixed integer n). Since $\tau(R_T) = D$, the attacker's goal is to stop the defender from activating R_T . This is done by activating the attacks corresponding to the observed defenses, i.e., $\rho(\vec{d}) = \vec{d}$ as binary vectors. Furthermore, $\hat{\beta}_D(\vec{d}) = \hat{\beta}_A(\vec{d}) = k$, where k is the integer encoded by the binary number \vec{d} ; hence

$$\mathcal{S} = \{(k, k) \in \mathbb{Z}^2 \mid 0 \leq k \leq 2^n - 1\}.$$

All elements of \mathcal{S} are Pareto optimal, so $|\text{PF}(T)| = 2^n = 2^{|\mathcal{D}|}$.

Example 4 shows that a worst-case exponential time complexity is unavoidable. Nevertheless, the brute force approach of computing each $\hat{\beta}(e)$ is often inefficient, as the final Pareto front is generally much smaller. In the following two sections, we present two algorithms that compute the Pareto front more efficiently, by discarding nonoptimal events along the way. Their efficiency is investigated empirically in Section VI.

IV. THE PARETO FRONT FOR TREE-SHAPED ADTS

In this section, we present methodologies for calculating the Pareto Front of tree-structured ADTs, which are ADTs where each node has a single parent, focusing on distinct structural representations. For tree-shaped ADTs, we employ a Bottom-Up Algorithm to efficiently derive optimal strategies. The ADT metrics are computed using a Bottom-Up approach based on the gate type and its function (attack or defense). Bottom-Up algorithms have been extensively studied for Fault Trees (FTs) and Attack Trees (ATs), providing a framework for systematic metric evaluation. To extend these algorithms to Attack-Defense Trees (ADTs), we introduce the following steps for a node $v \in N$:

- 1) Compute the Pareto Front Bottom-Up for each $w \in \text{ch}(v)$.
- 2) Identify all possible combinations of points from the children's Pareto Fronts.
- 3) For each combination, apply the min or + operations as examples of common metrics across the defense and attack costs, depending on the values of $\gamma(v)$ and $\tau(v)$. While we use these specific metrics for illustrative purposes in this informal definition, the approach supports general metrics defined over the semiring structure.
- 4) Discard the dominated points from the previous step.

In step 1, the **Bottom-Up** algorithm is recursively applied to each child of v . Due to this recursive nature, the algorithm will first complete for the leaf nodes, and the results will then be propagated up the tree towards the root node. For step 2, the Cartesian product is used to find all possible ways to combine the children's Pareto fronts. Unfortunately, computing all combinations is unavoidable, as it is impossible to predict which points will be included in the Pareto front before evaluating all the value pairs in step 3.

The objective of step 3 is to transform a vector of value pairs into a single value pair (s, t) for each possible combination based on Table II.

Example 5. Consider an AADT with the structure $OR(INH(d_1|a_1), INH(d_2|a_2))$ as shown in Fig. 5, where d_1, d_2 are defenses and a_1, a_2 are attacks. Each defense has an associated cost: $\beta_D(d_1) = 4$, $\beta_D(d_2) = 8$. Similarly, the attack costs are $\beta_A(a_1) = 5$, $\beta_A(a_2) = 10$. In this example, we use a semiring where $(\otimes, \oplus) = (+, \min)$ for both attack and defense costs.

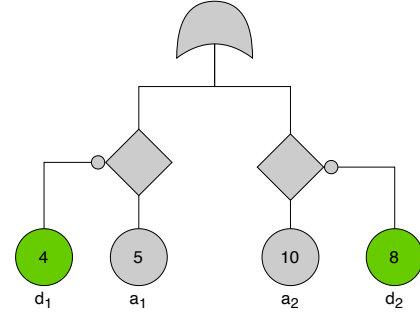


Fig. 5: The AADT of Example 5

$\gamma(v)$	$\tau(v)$	Def. op (\circ_D)	Att. op (\circ_A)
AND	A	\otimes_D	\otimes_A
	D	\otimes_D	\oplus_A
OR	A	\otimes_D	\oplus_A
	D	\otimes_D	\otimes_A
INH	A	\otimes_D	\otimes_A
	D	\otimes_D	\oplus_A

TABLE II: Operators applied in the **Bottom-Up** algorithm.

all possible attacks, not knowing which one the attacker might choose. Therefore:

$$\begin{aligned} (0 + 0, \min(5, 10)) &= (0, 5) \\ (0 + 8, \min(5, \infty)) &= (8, 5) \\ (4 + 0, \min(\infty, 10)) &= (4, 10) \\ (4 + 8, \min(\infty, \infty)) &= (12, \infty). \end{aligned}$$

After removing dominated points, the Pareto front is: $(0, 5), (4, 10), (12, \infty)$.

The output of step 3 is a set of value pairs. In step 4, we reduce the elements of this set to the Pareto Front by discarding all the dominated points according to Definition 9. In Alg. 1, the complete algorithm is presented, where $\underline{\min}_{\subseteq}$ represents step 4 specifically.

Algorithm 1 **Bottom-Up**

Input:

T : augmented attack-defense tree

v : node $v \in N$

Output: ...

```

1: procedure BU( $T, v, \beta$ )
2:   if  $v \in \mathcal{A}$  then
3:     return  $\{(1_{\otimes}, \beta_A(v))\}$ 
4:   else if  $v \in \mathcal{D}$  then
5:     return  $\{(1_{\otimes}, 1_{\otimes}), (\beta_D(v), 1_{\oplus})\}$ 
6:   else
7:      $p \leftarrow \times_{u \in ch(v)} \text{BU}(T, u, \beta)$ 
8:      $pv \leftarrow \{(\bigcirc_{D_{i=1}^n} d_i, \bigcirc_{A_{i=1}^n} a_i) \mid (\vec{d}, \vec{a}) \in p\}$ 
9:     return  $\underline{\min}_{\subseteq}(pv)$ 

```

The following theorem formalizes the relationship between the **Bottom-Up** algorithm (BU) and the Pareto Front (PF_S)

1) **Leaf Nodes:** For the leaf nodes, the cost pairs are:

$$\begin{aligned} a_1 &: (0, 5) \quad (\text{no defense cost, attack cost of 5}) \\ a_2 &: (0, 10) \quad (\text{no defense cost, attack cost of 10}) \\ d_1 &: (0, 0), (4, \infty) \quad (\text{defense can be inactive or active}) \\ d_2 &: (0, 0), (8, \infty) \quad (\text{defense can be inactive or active}). \end{aligned}$$

2) $INH(d_1|a_1)$: The inhibiting gate combines each a_1 cost with d_1 using $(+, +)$, where both defense and attack costs are summed (as shown in Table II). The reason is that the attacker must successfully execute both the inhibiting attack (d) and the inhibited attack (a) for the system to fail. Similarly, the defender must allocate resources to counter both d and a to fully protect the system. As a result, the operations for the attack INH gate are additive, represented as $(+, +)$. Therefore:

$$\begin{aligned} (0 + 0, 5 + 0) &= (0, 5) \\ (0 + 4, 5 + 0) &= (5, \infty). \end{aligned}$$

Final Pareto front for $INH(d_1|a_1)$: $(0, 5), (4, \infty)$.

3) $INH(d_2|a_2)$:

$$\begin{aligned} (0 + 0, 10 + 0) &= (0, 10) \\ (0 + 8, 10 + \infty) &= (8, \infty). \end{aligned}$$

Final Pareto front for $INH(d_2|a_2)$: $(0, 10), (8, \infty)$.

4) $OR(INH(d_1|a_1), INH(d_2|a_2))$: The attack **OR** gate combines the Pareto fronts from $INH(d_1|a_1)$ and $INH(d_2|a_2)$, using $(+, \min)$, where defense costs are summed, and attack costs take the minimum. The reason is that for the **OR** gate ($\gamma(v) = \text{OR}$) with attack function ($\tau(v) = A$), the attacker needs to succeed in only one of the available attacks to disable the system. In contrast, the defender must simultaneously defend against

for a tree-shaped augmented ADT (AADT).

Theorem 1. Let T be a tree-shaped AADT. Then $\text{BU}(T, R_T) = \text{PF}_S(T)$.

Based on this theorem, the Bottom-Up approach computes the Pareto Front for the root node R_T , aggregating optimal points from the children nodes according to the semantics S .

While the procedure loops (recursively) over the number of vertices, the main complexity comes from the fact that in Lines 7–8 we need to combine increasingly large Pareto fronts. Fig. 4 shows that these are worst-case exponential; hence Alg. 1 is worst-case exponential as well. Nevertheless, in practice more events will be non-Pareto-optimal, and will be discarded earlier; we will assess performance in the experiments.

V. THE PARETO FRONT FOR DAG-SHAPED ADTS

The Bottom-Up (BU) algorithm does not work for DAG-shaped ADTs: When a node has multiple parents, the Pareto Front computed at that node is propagated multiple times up the tree, leading to that value being counted several times.

For regular ATs, we know from [18] that generally, computing a metric for a semiring attribute domain in a DAG-structured AT is NP-hard. The same holds true for the ADTs in this paper since they represent an extension of regular ATs. An enumerative approach to compute the Pareto Front would be inefficient; Therefore, our approach to compute the Pareto Front for each possible defense vector \vec{d} , find attack vector \vec{a} , and then remove dominated points according to Def. 9 (using \sqsubseteq). First, we introduce the Naive approach to calculate the Pareto Frontier, which provides a baseline for comparison. Then, we present our optimized solutions. Finally, we compare the results of all approaches to highlight the advantages and demonstrate how they improve upon the Naive method.

A. Naive approach

As previously mentioned, computing a metric for a semiring attribute domain in a DAG-structured ADT is NP-hard. Even so, formally defining this approach is still beneficial: it provides a practical reference point for what $\min_{\sqsubseteq} \hat{\beta}(\mathcal{S})$ outputs for DAGs, while serving as a stepping stone for the next algorithms to improve on. Algorithm 2 is rather straightforward. Lines 4–11 compute $\rho(\vec{\delta})$ for each $\vec{\delta}$ by going through all the possible attacks and finding the one with the minimum metric value. In the end, the value pairs are reduced to the Pareto front using \min_{\sqsubseteq} .

B. Binary Decision Diagrams

Binary Decision Diagrams (BDDs) offer a compact representation of Boolean functions. Since BDDs can have shared sub-trees, these are able to model DAG-structured ADTs. Generally, a BDD represents a Boolean function $f: \mathbb{B}^{vars} \rightarrow \mathbb{B}$ over a set of variables $vars$. A BDD translates a Boolean function to a flowchart, where each nonterminal node is labeled by a variable, and has two outgoing edges labeled 0 and 1. Starting from the root, given an input vector \vec{b} to

Algorithm 2 Naive algorithm for DAGs

Input:

T : attack-defense tree

v : node $v \in N$

β : assignment of nodes $\in N$

Output: Pareto front of the sub-tree rooted at v .

```

1: procedure NAIVE( $T, v, \beta$ )
2:    $result \leftarrow$  new array
3:   for  $\vec{\delta} \in 2^{\mathcal{D}}$  do
4:      $att\_MetValues \leftarrow$  new array
5:     for  $\vec{\alpha} \in 2^{\mathcal{A}}$  do
6:       if  $f_T(\vec{\delta}, \vec{\alpha}, R_T) = 1$  then
7:         Add  $\hat{\beta}_A(\vec{\alpha})$  to  $att\_MetValues$ 
8:       if  $att\_MetValues = \emptyset$  then
9:         Add  $(\hat{\beta}_D(\vec{\delta}), \infty)$  to  $result$ 
10:      else
11:        Add  $(\hat{\beta}_D(\vec{\delta}), \min_{\leq_A}(att\_MetValues))$  to
         $result$ 
12:   return  $\min_{\sqsubseteq}(result)$ 

```

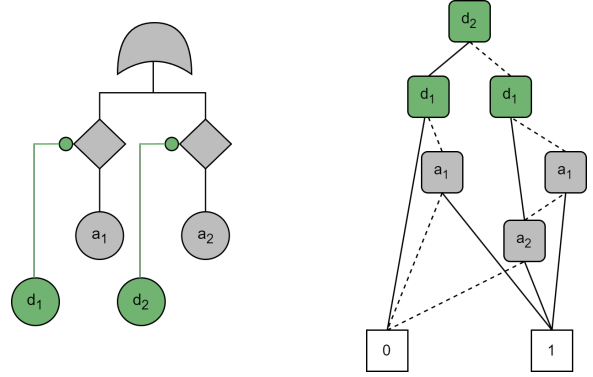


Fig. 6: An ADT and a ROBDD corresponding to its structure function, with variable order $d_2 < d_1 < a_1 < a_2$. Dashed lines are labeled 0, solid lines are labeled 1.

the Boolean function, the BDD is traversed by taking the 0-labeled edge at a node labeled $x \in vars$ if $b_x = 0$, and taking the 1-labeled edge if $b_x = 1$. The leaves are labeled 0 and 1, which denotes the output of the function for input \vec{b} . An example is given in Fig. 6.

We are particularly interested in *reduced ordered BDDs* (ROBDDs), which means that the variable order in the BDD is the same for all paths, and that all unnecessary BDD nodes are removed. Every Boolean function has a unique ROBDD once the variable order is fixed, though different variable orderings result in different ROBDDs. Throughout the rest of this section, we will simply write ‘BDD’ for ‘ROBDD’.

Definition 10. An *ROBDD* is a tuple $B = (W, Low, High, Lab, <)$ over a set of variables $vars$, where:

- The set of nodes W is partitioned into terminal nodes (W_t) and nonterminal nodes (W_n);
- $Low: W_n \rightarrow W$ maps each nonterminal node to its *low* child;

- $High : W_n \rightarrow W$ maps each nonterminal node to its high child;
- $Lab : W \rightarrow \{0, 1\} \cup vars$ maps terminal nodes to Booleans and nonterminal nodes to variables:

$$Lab(w) \in \begin{cases} \{0_T, 1_T\} & \text{if } w \in W_t, \\ vars & \text{if } w \in W_n; \end{cases}$$

- $<$ is a strict total order on $vars$ (called the variable ordering).

Moreover, B satisfies the following constraints:

- (W, E) is a connected Directed Acyclic Graph (DAG), where

$$E = \{(w, w') \in W^2 \mid w' \in \{Low(w), High(w)\}\};$$

- B has a unique root, denoted R_B :

$$\exists! R_B \in W. \forall w \in W_n. R_B \notin \{Low(w), High(w)\};$$

- The variable ordering $<$ is respected, i.e., for every edge $(w, w') \in E$, if $w \in W_n$ and $w' \in W_n$, then $Lab(w) < Lab(w')$;
- B is *reduced*, meaning:

- 1) No two distinct nodes have the same label, low child, and high child:

$$\begin{aligned} \forall w, w' \in W_n, w \neq w' \implies & (Lab(w) \neq Lab(w') \\ & \vee Low(w) \neq Low(w') \\ & \vee High(w) \neq High(w')). \end{aligned}$$

- 2) No node has identical low and high children:

$$\forall w \in W_n. Low(w) \neq High(w).$$

Conventionally edges $(w, Low(w))$ are labeled 0, and $(w, High(w))$ are labeled 1; as described above, this determines the Boolean function B represents.

BDDs have been used for the quantitative analysis of fault trees [19] and attack trees [18]. This makes them a promising candidate for ADTs as well. One important consideration is the choice of the variable order $<$. For fault trees, attack trees, and ADTs (as we will see next section), the size of the BDD is a major factor in algorithm complexity. BDD size is worst-case exponential; however, in practice it heavily depends on the choice of $<$. At the same time, finding the optimal $<$ is NP-hard. These two facts have led to the rich field of BDD minimization, where heuristic methods are developed to find good variable orders [20].

For ADTs the variable order is more restricted: we demand that in the BDD, defenses come before attacks. This reflects that the attacker can choose their attack upon having observed the defender's actions. This turns out to be necessary in order for our algorithm to work (see Theorem 2).

Definition 11. A defense-first ordering is a linear order $<$ on $\mathcal{D} \cup \mathcal{A}$ such that for all $d \in \mathcal{D}$ and $a \in \mathcal{A}$, $d < a$.

Example 6. Consider the AFT of Fig. 6. The variable order of the BDD is defense-first. The paths in the BDD correspond

to evaluations of the structure function; for instance, the path $d_2 \rightarrow d_1 \rightarrow a_1 \rightarrow a_2 \rightarrow 1$ represents the fact that $f_T(10, 01) = 1$. The path $d_2 \rightarrow d_1 \rightarrow a_1 \rightarrow 0$ represents the fact that $f_T(10, 0*) = 0$; here $*$ denotes the fact that the value of α_{a_2} is irrelevant.

C. BDD-based algorithm for ADT

We can use an ROBDD corresponding to an AAFT T to compute its Pareto front. Suppose $\tau(R_T) = A$, so that the attacker's goal is to reach the 1-leaf of the ROBDD. Every path π from the root to 1 represents an event $e_\pi = (\vec{\delta}, \vec{\alpha})$, with $\delta_d = 1$ if and only if there is a node w such that $Lab(w) = d$ and $(w, High(w))$ is part of π ; the vector $\vec{\alpha}$ is defined likewise. Thus, in the notation of Example 6, we set $*$ = 0. Not all possible events are present as paths; for instance in Fig. 6 the event $(01, 11)$ is not present. However, all feasible events are in the ROBDD; hence

$$PF(T) = \min_{\subseteq} \{\hat{\beta}(e_\pi) \mid \pi \text{ path from root to 1 in ROBDD}\}.$$

Typically, the ROBDD will represent some nonfeasible events as well. For instance, in Fig. 6, the events $(00, 10)$ and $(00, 01)$ are both present, but only one of them will be feasible. However, this is not a problem, as nonfeasible events will be filtered out in the \min_{\subseteq} of the equation above.

This suggests an algorithm to compute $PF(T)$: compute all paths from the root to 1, determine their metric values, and take the Pareto front. However, this is inefficient as there will be many, partially overlapping parts. Instead, we perform a bottom-up computation on the BDD. This is similar to a typical shortest path algorithm in a directed acyclic graph, except that (1) our computations are in the semirings D_D, D_A instead of \mathbb{R} and (2) instead of propagating a single value, we propagate a Pareto front of optimal value pairs. If T has no defense nodes (i.e., an attack tree), our algorithm is identical to the BDD-based attack tree analysis algorithm of [18].

The algorithm BDD_{BU} is presented in Alg. 3. It is a recursive algorithm; we are interested in $BDD_{BU}(T, B, R_B)$. To explain it, we will use the min cost semiring (see Table I) for both the attacker and defender, but the algorithm works for all semiring attribute domains; we also assume that $\tau(R_T) = A$; i.e., the attacker's goal is to reach the leaf 1. At every node w , each element $(u, u') \in BDD_{BU}(T, B, w)$ represents the fact that the attacker can reach 1 from node w by spending u' , provided the defender has budget at most u . How $BDD_{BU}(T, B, w)$ is computed depends on the label $Lab(w)$:

- If $Lab(w) = 0$, then w is the 0-leaf. From here it is impossible to reach the 1-leaf, i.e., even if the defender has no budget. This is represented by the value pair $(0, \infty)$ (Line 3).
- If $Lab(w) = 1$, then w is the 1-leaf. Thus the attacker does not need to spend any cost to get to 1, which is represented by the value pair $(0, 0)$ (Line 5).
- If $Lab(w) \in \mathcal{A}$, we compute $BDD_{BU}(T, B, w)$ from its children $Low(w)$ and $High(w)$. We assume that both Pareto fronts $BDD_{BU}(T, B, Low(w))$ and $BDD_{BU}(T, B, High(w))$ consist of a single element with

Algorithm 3 BDD Bottom Up (BDD_{BU})

Input:

$T = (T, D_{\mathcal{D}}, D_{\mathcal{A}}, \beta_D, \beta_A)$: augmented ADT
 B : ROBDD representing f_T
 w : node of B

Output: Pareto Front of the ADT encoded by B .

```
1: procedure  $\text{BDD}_{\text{BU}}(T, B, w)$ 
2:   if  $\text{Lab}(w) = 0$  then
3:     return  $\begin{cases} \{(1_{\otimes_D}, 1_{\oplus_A})\} & \text{if } \tau(R_T) = A; \\ \{(1_{\otimes_D}, 1_{\otimes_A})\} & \text{if } \tau(R_T) = D; \end{cases}$ 
4:   else if  $\text{Lab}(w) = 1$  then
5:     return  $\begin{cases} \{(1_{\otimes_D}, 1_{\otimes_A})\} & \text{if } \tau(R_T) = A; \\ \{(1_{\otimes_D}, 1_{\oplus_A})\} & \text{if } \tau(R_T) = D; \end{cases}$ 
6:   else if  $\text{Lab}(w) \in \mathcal{A}$  then
7:      $\{(1_{\otimes_D}, u_0)\} \leftarrow \text{BDD}_{\text{BU}}(T, B, \text{Low}(w));$ 
8:      $\{(1_{\otimes_D}, u_1)\} \leftarrow \text{BDD}_{\text{BU}}(T, B, \text{High}(w));$ 
9:     return  $\{(1_{\otimes_D}, u_{\text{low}} \oplus_A (\beta_A(\text{Lab}(w)) \otimes_A u_{\text{high}}))\};$ 
10:  else
11:     $P_0 \leftarrow \text{BDD}_{\text{BU}}(T, B, \text{Low}(w));$ 
12:     $P_1 \leftarrow \text{BDD}_{\text{BU}}(T, B, \text{High}(w));$ 
13:     $P \leftarrow P_0 \cup \{(\beta_D(\text{Lab}(w)) \otimes_D u, u') \mid (u, u') \in P_1\};$ 
14:    return  $\min_{\subseteq}(P)$ 
```

first coefficient $1_{\otimes_D} = 0$ (Lines 7-8); this is true because it is true for the leaves and for the output of \mathcal{A} -labeled nodes (Line 9), and our variable order ensures that no descendants of w are \mathcal{D} -labeled. At node w , the attacker can choose to either not perform attack $\text{Lab}(w)$, move to $\text{Low}(w)$, and perform the optimal attack there, which has cost u_0 ; or perform attack $\text{Lab}(w)$ (paying cost $\beta_A(\text{Lab}(w))$), move to $\text{High}(w)$, and perform the optimal attack there, with cost u_1 . The attacker chooses between the two by minimizing cost, which is $\min(u_{\text{low}}, \beta_A(\text{Lab}(w)) + u_{\text{high}})$ (Line 9). Since the defender plays no role here, the defender's metric value stays $1_{\otimes_D} = 0$.

- If $\text{Lab}(w) \in \mathcal{D}$, the defender can choose between either not activating $\text{Lab}(w)$, moving to $\text{Low}(w)$, and picking one of the defense options there; or activating $\text{Lab}(w)$ (incurring cost $\beta_D(\text{Lab}(w))$), moving to $\text{High}(w)$, and picking one of the defense options there. The resulting combination of defense-attack metric value pairs is given in Line 13. Not all options will be optimal, so we only keep its Pareto front (Line 14).

The following theorem expresses correctness.

Theorem 2. Let T be an AADT, and let $<$ be a defense-first ordering on $\mathcal{A} \cup \mathcal{D}$. Let B be the ROBDD with variable ordering $<$ representing f_T . Then $\text{BDD}_{\text{BU}}(T, B, R_B) = \text{PF}(T)$.

The complexity is $\mathcal{O}(|W|p^2)$, where W is the set of nodes of the BDD, and p is the maximal size of the Pareto fronts involved in the computation. Both are worst-case exponential in the size of the ADT (see Fig. 4); however, we expect both to be reasonably small in practice, resulting in fast computation.

We will test this expectation in the experiments.

VI. EXPERIMENTS

We empirically evaluate our methods, using a case study adapted from [5], as well as 120 randomly generated ADTs.

A. Case Study: Money Theft

We apply our methods to a real-world ADT presented by Kordy and Wideł [5] modelling a money theft scenario, either via stealing someone's card and using an ATM, or via online banking (see Figure 7). That work assigns a unitless value to each BAS, representing the attacker's cost of performing that action. We also assign cost values to the BDS.

As mentioned in Section II, existing work does not compute the Pareto front between attack and defense costs. Instead, the minimal attack cost discussed in [5] only considers attacks that cannot be prevented. This amounts to giving the defender infinite budget, and to just one point on the Pareto front.

This is a DAG-shaped ADT, since *Phishing* has two parents. As such, we cannot apply our bottom-up approach directly. Instead, we assume that *Phishing* needs to be performed twice in order to activate both *Get Password* and *Get username*. This turns the ADT into a tree-shaped one, and we can perform the Bottom-Up algorithm. This is inscribed in red in Fig. 7. As we can see, the final Pareto Front contains of only 3 pairs: $(0, 90)$ is the cheapest attack on *Via ATM*. The defender can thwart this via *Cover Keypad*; if they do this, the attacker instead takes the cheapest attack on *Via Online Banking*, represented by the pair $(30, 150)$. If the defender furthermore activates *SMS Authentication*, it is most advantageous for the attacker to attack *Via ATM* again, disabling the defender's action using *Camera*. This is represented by the Pareto-optimal pair $(50, 165)$. Note that the BDS *Strong Pwd* is not part of any Pareto-optimal point, suggesting that this action does not help the defender and should be avoided. In [5] the outcome of the analysis is 165, which corresponds to our final Pareto-optimal pair; thus our analysis gives a more complete picture of the interplay between attacker and defender, by showing the defender the effect of varying their budget on overall security.

We also compute the Pareto front using BDD_{BU} , which accurately analyzes the DAG-shaped ADT. We get the Pareto-optimal pairs $(0, 80)$, $(20, 90)$, $(50, 140)$. Again, we see that the Pareto front is much smaller than the exponential upper bound would suggest. The optimal strategies are different to the tree-shaped case: here $\{\text{Phishing}, \text{Log In \& Execute Transfer}\}$ is optimal if the defender has no budget. 140 is also the metric computed in [5] under so-called *set semantics*, which model DAG-shaped ADTs. Again existing work only gives a single value, instead of the whole Pareto front. Although this case study demonstrates the applicability of our method, it abstracts away factors like dynamic system behavior, uncertainty in metrics, and organizational constraints, which would need to be addressed when scaling to more complex, real-world scenarios.

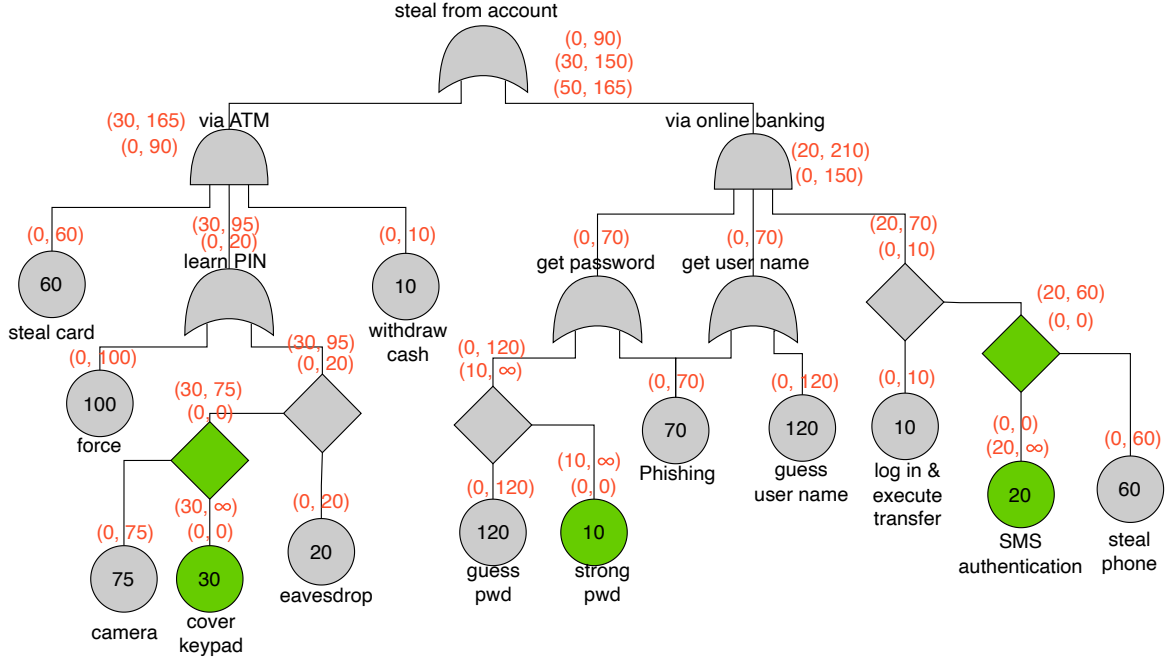


Fig. 7: Attack-defense tree representing a money theft scenario, adapted from [5]. Values inscribed in BAS/BDS are attacker/defender costs; red values are the Pareto fronts at every node, computed Bottom-Up.

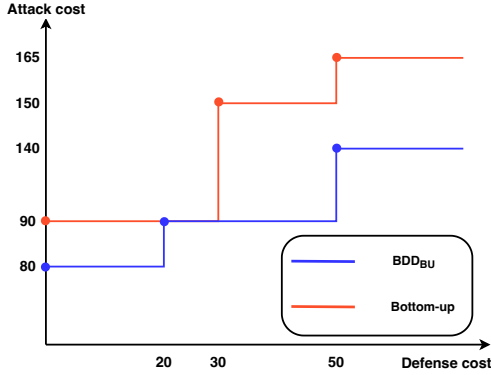


Fig. 8: Pareto front for the ADT of Fig. 7, both under Bottom-Up and BDD_{BU} analysis.

B. Synthetic ADTs

The performance of BU and BDD_{BU} is evaluated against the Naive approach. The Naive algorithm computes $\rho(\vec{\delta})$ for each defense vector $\vec{\delta}$ by iterating over all possible attacks and selecting the one with the minimum metric value, followed by reducing the resulting value pairs to the Pareto front using \min_{\subseteq} . Since existing work does not account for the interplay between attacker and defender metrics, direct comparisons with prior literature are not feasible.

One approach to overcome this is to annotate the existing ADTs found in the literature with cost values for the defender. We were able to find several ADTs with $25 \leq |N| \leq 50$ [21], [22] but only a limited number of ADTs with $50 \leq |N| \leq 100$ [23]–[25]. Finding ADTs with $|N| \geq 100$ can be challenging

as they are typically not made public for confidentiality reasons [26].

In practice, the size of attack trees can range from a few dozen to several hundred nodes [27]. Given the relatively small number and size of ADTs found in the literature, we do not consider this to be a large enough testing suite to evaluate algorithms. Consequently, it is necessary to synthetically generate ADTs. Two common techniques for generating ATs are combining literature trees into a single one [28], or generating random ATs from scratch. We focused on the latter to cover a wider range of scenarios and create a more robust test suite.

A risk analysis algorithm that takes several days may be feasible for some applications. However, within the context of this work, given the hardware limitations and restricted time to conduct experiments, we limit our testing scope by not pursuing computations that take more than 10^4 seconds.

All experiments were performed on a machine with an Intel Core i5-12600K 3.7Ghz processor and 16GB of RAM. The algorithms are implemented in Python 3.12. Although faster BDD run times could perhaps be achieved using a C implementation such as in Sylvan [29] or CUDD [30], we opted to maintain a consistent testing environment for all algorithms. The code and results are available on GitHub.¹

We employ a large number of randomly generated ADTs for a statistically significant comparison of the algorithms' performance. After setting a maximum number of children n , nodes with random properties (gate type, attack/defense type, number of children) are recursively generated until the tree

¹https://github.com/dvcopae/thesis_adtrees

contains n nodes (see the Appendix). This approach naturally creates tree- and DAG-structured ADTs.

Fig. 9 presents pairwise comparisons between the algorithms. A summary of these comparisons is given in Fig. 10. Algorithm runtimes across all random graphs are aggregated by taking the median at each interval of $|N| = 20$. Since the run time of `Naive` increases at an exponential rate, the values at the end of the interval will be drastically different than those at the beginning. To better represent the central tendency of the interval, the median is used instead of the average.

Regarding the `Naive` algorithm, it is notable that for certain small-sized trees with fewer than 50 nodes, it surprisingly outperforms `BDDBU`. We hypothesize that this is because, at a very small number of nodes, the time required to construct the BDD model constitutes a significant proportion of the total run time. However, as the number of nodes increases, the `Naive` algorithm approaches a runtime of 10^4 seconds, even for some trees with less than 50 nodes. This algorithm has the slowest performance among the considered methods.

As both `BU` and `BDDBU` are quite fast, we extended our analysis for larger trees with up to 325 nodes. Remarkably, while the performance gap between the two remains consistent for trees with fewer than 100 nodes, this drastically changes as trees gain more nodes. For trees ranging from 300 to 325 nodes `BDDBU` requires approximately 10^3 seconds, whereas `BU` roughly 10^{-3} seconds.

In summary, this approach indicates that `BU` computes the Pareto Front the fastest for tree-structured ADTs, while `BDDBU` is the most efficient for DAGs. Furthermore, computation time remains low (below 10s) even for DAG-shaped ADTs an order of magnitude larger than our case study of Section VI-A.

VII. CONCLUSION

In this paper, we proposed a novel framework that incorporates defense metrics into ADTs, and presented efficient algorithms for computing the Pareto front between defense and attack metrics. The highlights behind this framework include a formal syntax and semantic model for representing ADT with attacker and defender attribute domains. We delved into `Bottom-Up` analysis and Binary Decision Diagram methods for computing the Pareto Front of the defender's and attacker's cost. We evaluated the performance of these approaches on a test suite consisting of randomly generated ADTs of sizes up to 325 nodes and observed significant variations in the speed of the algorithms. In scenarios where the ADT has a tree structure, `BU` performs by far the best, with an average processing time of less than 1 second, even for trees with several hundred nodes. On the other hand, in cases where the ADT has a DAG structure and the attribute domains are absorbing semirings, then `BDDBU` is the next fastest choice, capable of analyzing trees up to 325 nodes under 30 minutes.

As future works, one possible extension is incorporating probabilistic failures, as explored by Aslanyan and Nielson [8], to enable scenarios where attackers and defenders operate under uncertainty. Attack-Fault-Defense Trees (AFDTs), as introduced in [11], could provide a suitable framework for

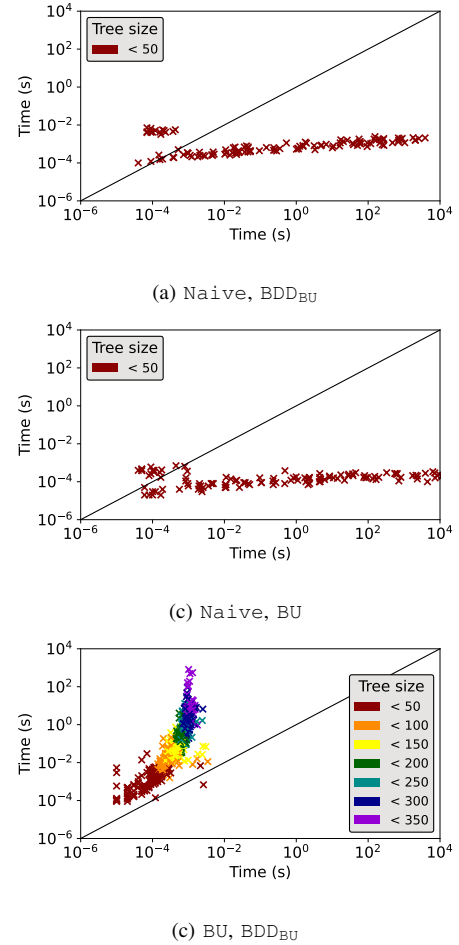


Fig. 9: The first algorithm is the vertical axis while the second is the horizontal axis. The run time is in seconds, and each \times represents a random ADT. For plots involving `BU`, only tree-structured ADTs are generated. Random ADTs were adjusted in size and number to ensure no run time exceeds 10^4 seconds. All plots are based on 120 random ADTs with $|N| < 45$.

such extensions. Another possible extension is enhancing algorithm efficiency through modular decomposition techniques. By dividing large ADTs into smaller modules, the computational cost of analyzing complex systems could be significantly reduced. Furthermore, optimizing BDDs by identifying orderings that minimize their size while retaining the defense-first property remains an important challenge. Developing automated methods for finding such orderings would improve scalability and enable efficient analysis of larger ADTs. Finally, an interesting avenue for future research is extending our approach to ADTs with dynamic behaviour, similar to dynamic attack trees [31]. The time-dependency between BAS/BDS could be modelled by their relative position in the BDD variable order.

REFERENCES

- [1] B. Schneier, "Modeling security threats," *Dr. Dobbs' journal*, vol. 24, no. 12, 1999.

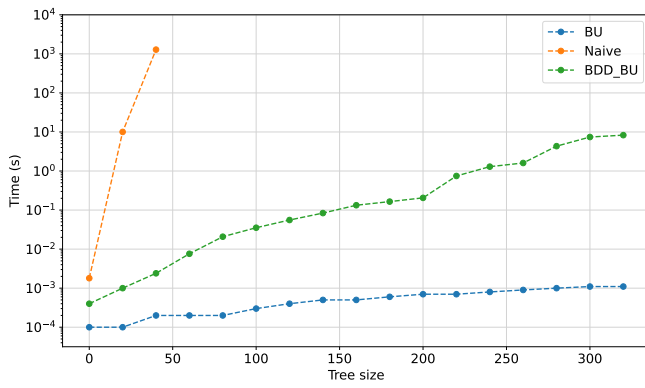


Fig. 10: Summary of all the pairwise comparisons. The vertical axis represents the median time in seconds of each algorithm for ADTs grouped by the number of nodes $|N|$ at intervals of size 20.

- [2] E. Tanu and J. Arreymbi, "An examination of the security implications of the supervisory control and data acquisition (scada) system in a mobile networked environment: An augmented vulnerability tree approach," 2010. [Online]. Available: <https://api.semanticscholar.org/CorpusID:107872011>
- [3] N. E. S. C. O. Resource, "Analysis of selected electric sector high risk failure scenarios," 2015, version 2.0. [Online]. Available: <https://smartgrid.epri.com/NESCOR.aspx>
- [4] B. Kordy, S. Mauw, S. Radomirović, and P. Schweitzer, "Foundations of attack-defense trees," in *Formal Aspects of Security and Trust*, P. Degano, S. Etalle, and J. Guttman, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 80–95.
- [5] B. Kordy and W. Widł, "On quantitative analysis of attack-defense trees with repeated labels," in *Principles of Security and Trust*, L. Bauer and R. Küsters, Eds. Cham: Springer International Publishing, 2018, pp. 325–346.
- [6] B. Fila and W. Widł, "Efficient attack-defense tree analysis using pareto attribute domains," in *2019 IEEE 32nd Computer Security Foundations Symposium (CSF)*, 2019, pp. 200–20015.
- [7] —, "Efficient attack-defense tree analysis using pareto attribute domains," in *2019 IEEE 32nd Computer Security Foundations Symposium (CSF)*, 2019, pp. 200–20015.
- [8] Z. Aslanyan and F. Nielson, "Pareto efficient solutions of attack-defence trees," in *Principles of Security and Trust*, R. Focardi and A. Myers, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 95–114.
- [9] S. Bistarelli, F. Fioravanti, and P. Peretti, "Defense trees for economic evaluation of security investments," in *First International Conference on Availability, Reliability and Security (ARES'06)*, 2006, pp. 8 pp.–423.
- [10] A. Roy, D. S. Kim, and K. S. Trivedi, "Attack countermeasure trees (act): towards unifying the constructs of attack and defense trees," *Security and Communication Networks*, vol. 5, no. 8, pp. 929–943, 2012.
- [11] R. Soltani, M. Lopuhaä-Zwakenberg, and M. Stoelinga, "Safety-security analysis via attack-fault-defense trees: Semantics and cut set metrics," in *Computer Safety, Reliability, and Security*, A. Ceccarelli, M. Trapp, A. Bondavalli, and F. Bitsch, Eds. Cham: Springer Nature Switzerland, 2024, pp. 218–232.
- [12] J. Arias, C. E. Budde, W. Penczek, L. Petrucci, T. Sidoruk, and M. Stoelinga, "Hackers vs. security: Attack-defence trees as asynchronous multi-agent systems," in *Formal Methods and Software Engineering*, S.-W. Lin, Z. Hou, and B. Mahony, Eds. Cham: Springer International Publishing, 2020, pp. 3–19.
- [13] R. Soltani, M. Volk, L. Diamonte, M. Lopuhaä-Zwakenberg, and M. Stoelinga, "Optimal spare management via statistical model checking: A case study in research reactors," in *Formal Methods for Industrial Critical Systems*, A. Cimatti and L. Titolo, Eds. Cham: Springer Nature Switzerland, 2023, pp. 205–223.
- [14] R. Soltani, E.-Y. Kang, and J. E. H. Mena, "Verification and optimization of cyber-physical systems: Preprint for fedccsis," 2021. [Online]. Available: <https://arxiv.org/abs/2109.01574>
- [15] —, "Towards energy-aware cyber-physical systems verification and optimization," in *Position and Communication Papers of the 16th Conference on Computer Science and Intelligence Systems*, ser. Annals of Computer Science and Information Systems, vol. 26. PTI, 2021, p. 205–210. [Online]. Available: <http://dx.doi.org/10.15439/2021F125>
- [16] M. Lopuhaä-Zwakenberg and M. Stoelinga, "Cost-damage analysis of attack trees," 2023.
- [17] B. Kordy, S. Mauw, S. Radomirović, and P. Schweitzer, "Attack-defense trees," *Journal of Logic and Computation*, vol. 24, 02 2014.
- [18] M. Lopuhaä-Zwakenberg, C. E. Budde, and M. Stoelinga, "Efficient and generic algorithms for quantitative attack tree analysis," *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 5, p. 4169–4187, Sep. 2023.
- [19] A. Rauzy, "New algorithms for fault trees analysis," *Reliability Engineering & System Safety*, vol. 40, no. 3, pp. 203–211, 1993.
- [20] R. Ebendt, G. Fey, and R. Drechsler, *Advanced BDD optimization*. Springer Science & Business Media, 2005.
- [21] K. S. Edge, G. C. Dalton, R. A. Raines, and R. F. Mills, "Using attack and protection trees to analyze threats and defenses to homeland security," in *MILCOM 2006 - 2006 IEEE Military Communications conference*, 2006, pp. 1–7.
- [22] B. Kordy, L. Piètre-Cambacédès, and P. Schweitzer, "Dag-based attack and defense modeling: Don't miss the forest for the attack trees," *Computer Science Review*, vol. 13-14, pp. 1–38, 2014.
- [23] M. Frailé, M. Ford, O. Gadyatskaya, R. Kumar, M. Stoelinga, and R. Trujillo-Rasua, "Using attack-defense trees to analyze threats and countermeasures in an atm: A case study," in *The Practice of Enterprise Modeling*, J. Horkoff, M. A. Jeusfeld, and A. Persson, Eds. Cham: Springer International Publishing, 2016, pp. 326–334.
- [24] A. Bagnato, B. Kordy, P. H. Meland, and P. Schweitzer, "Attribute decoration of attack-defense trees," *International Journal of Secure Software Engineering (IJSSSE)*, vol. 3, pp. 1–35, 01 2012.
- [25] K. Edge, R. Raines, M. Grimaila, R. Baldwin, R. Bennington, and C. Reuter, "The use of attack and protection trees to analyze security for an online banking system," in *2007 40th Annual Hawaii International Conference on System Sciences (HICSS'07)*, 2007, pp. 144b–144b.
- [26] S. Paul, "Towards automating the construction & maintenance of attack trees: a feasibility study," *Electronic Proceedings in Theoretical Computer Science*, vol. 148, p. 31–46, Apr. 2014. [Online]. Available: <http://dx.doi.org/10.4204/EPTCS.148.3>
- [27] R. Vigo, F. Nielson, and H. R. Nielson, "Automated generation of attack trees," in *2014 IEEE 27th Computer Security Foundations Symposium*, 2014, pp. 337–350.
- [28] M. Lopuhaä-Zwakenberg and M. Stoelinga, "Attack time analysis in dynamic attack trees via integer linear programming," in *Software Engineering and Formal Methods*, C. Ferreira and T. A. C. Willemse, Eds. Cham: Springer Nature Switzerland, 2023, pp. 165–183.
- [29] T. v. Dijk and J. Van de Pol, "Sylvan: multi-core framework for decision diagrams," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2016, pp. 677–691. [Online]. Available: <https://github.com/trolando/sylvan>
- [30] F. Somenzi, *CUDD: CU Decision Diagram Package*, 2015, release 3.0.0. [Online]. Available: <https://github.com/ivmai/cudd>
- [31] R. Jhawar, B. Kordy, S. Mauw, S. Radomirović, and R. Trujillo-Rasua, "Attack trees with sequential conjunction," in *ICT Systems Security and Privacy Protection*, H. Federrath and D. Gollmann, Eds. Cham: Springer International Publishing, 2015, pp. 339–353.

APPENDIX A PROOF OF THEOREM 1

We first prove an auxiliary lemma. Suppose that T is a tree-shaped AADT whose root has two children; we call the subtrees spanned by these children T_1 and T_2 . If \mathcal{D}_i is the set of BDS in T_i , then $\mathcal{D}_1 \cap \mathcal{D}_2 = \emptyset$ due to the tree-shaped property, and $\mathcal{D}_1 \cup \mathcal{D}_2 = \mathcal{D}$. Hence $\mathbb{B}^{\mathcal{D}} \cong \mathbb{B}^{\mathcal{D}_1} \times \mathbb{B}^{\mathcal{D}_2}$. Thus we can uniquely write each element $\vec{\delta} \in \mathbb{B}^{\mathcal{D}}$ as $(\vec{\delta}_1, \vec{\delta}_2)$, with $\vec{\delta}_i \in \mathbb{B}^{\mathcal{D}_i}$. In this terminology, we have the following lemma:

Lemma 1. Let T be a tree-shaped AADT whose root has two children; the ADTs spanned by these children are called T_1

and T_2 , respectively. Let \mathcal{D}_i be the set of BDS in T_i . Furthermore, let ρ_1, ρ_2 be the optimal attack response functions of T_1 and T_2 , respectively. Then for all $\vec{\delta}_i \in \mathbb{B}^{\mathcal{D}_i}$,

$$\hat{\beta}_A(\rho(\vec{\delta}_1, \vec{\delta}_2)) = \hat{\beta}_A(\rho_1(\vec{\delta}_1)) \circ_A \hat{\beta}_A(\rho_2(\vec{\delta}_2)).$$

Proof. We prove this statement for the case $\gamma(R_T) = \text{OR}$, $\tau(R_T) = \text{A}$; the other cases are similar. We first show $\hat{\beta}_A(\rho(\vec{\delta}_1, \vec{\delta}_2)) \leq_A \hat{\beta}_A(\rho_1(\vec{\delta}_1)) \oplus_A \hat{\beta}_A(\rho_2(\vec{\delta}_2))$. To see this, note that since R_T is an OR-gate, we have

$$\begin{aligned} & f_T((\rho_1(\vec{\delta}_1), \vec{0}), (\vec{\delta}_1, \vec{\delta}_2), R_T) \\ &= f_{T_1}(\rho_1(\vec{\delta}_1), \vec{\delta}_1, R_{T_1}) \vee f_{T_2}(\vec{0}, \vec{\delta}_2, R_{T_2}) \\ &= 1 \vee f_{T_2}(\vec{0}, \vec{\delta}_2, R_{T_2}) \\ &= 1. \end{aligned}$$

By definition of $\rho(\vec{\delta}_1, \vec{\delta}_2)$, this means that

$$\begin{aligned} \hat{\beta}_A(\rho(\vec{\delta}_1, \vec{\delta}_2)) &\leq_A \hat{\beta}_A(\rho_1(\vec{\delta}_1), \vec{0}) \\ &= \hat{\beta}_A(\rho_1(\vec{\delta}_1)). \end{aligned}$$

Analogously we can show $\hat{\beta}_A(\rho(\vec{\delta}_1, \vec{\delta}_2)) \leq_A \hat{\beta}_A(\rho_2(\vec{\delta}_2))$; since \oplus_A selects the minimum w.r.t. \leq_A , this implies $\hat{\beta}_A(\rho(\vec{\delta}_1, \vec{\delta}_2)) \leq_A \hat{\beta}_A(\rho_1(\vec{\delta}_1)) \oplus_A \hat{\beta}_A(\rho_2(\vec{\delta}_2))$.

Next, we show $\hat{\beta}_A(\rho_1(\vec{\delta}_1)) \oplus_A \hat{\beta}_A(\rho_2(\vec{\delta}_2)) \leq_A \hat{\beta}_A(\rho(\vec{\delta}_1, \vec{\delta}_2))$. Let $\rho(\vec{\delta}_1, \vec{\delta}_2) = (\vec{\alpha}_1, \vec{\alpha}_2)$. Then by definition,

$$\begin{aligned} 1 &= f_T((\vec{\alpha}_1, \vec{\alpha}_2), (\vec{\delta}_1, \vec{\delta}_2), R_T) \\ &= f_{T_1}(\vec{\alpha}_1, \vec{\delta}_1, R_{T_1}) \vee f_{T_2}(\vec{\alpha}_2, \vec{\delta}_2, R_{T_2}). \end{aligned}$$

Without loss of generality, assume $f_{T_1}(\vec{\alpha}_1, \vec{\delta}_1, R_{T_1}) = 1$. Then

$$\begin{aligned} \hat{\beta}_A(\rho_1(\vec{\delta}_1)) &\leq_A \hat{\beta}_A(\vec{\alpha}_1) \\ &= \bigotimes_{i:\alpha_1, i=1} \beta(a_i) \\ &\leq_A \bigotimes_{i:\alpha_1, i=1} \beta(a_i) \otimes \bigotimes_{i:\alpha_2, i=1} \beta(a'_i) \\ &= \hat{\beta}_A(\vec{\alpha}_1, \vec{\alpha}_2) \\ &= \hat{\beta}_A(\rho(\vec{\delta}_1, \vec{\delta}_2)). \end{aligned}$$

Here the a_i are the BAS of T_1 , while the a'_i are the BAS of T_2 . This proves $\hat{\beta}_A(\rho_1(\vec{\delta}_1)) \oplus_A \hat{\beta}_A(\rho_2(\vec{\delta}_2)) \leq_A \hat{\beta}_A(\rho(\vec{\delta}_1, \vec{\delta}_2))$, which together with the previous inequality proves equality. \square

We also need a second lemma to show that the semiring operations behave nicely with respect to \sqsubseteq .

Lemma 2. Let \circ_A be either \otimes_A or \oplus_A , and let ψ be the binary operation on $V_D \times V_A$ given by $\psi((x, y), (x', y')) = (x \otimes_D x', y \circ_A y')$. Then for any two subsets $X, X' \subseteq V_D \times V_A$ we have

$$\min_{\sqsubseteq}(\psi(X, X')) = \min_{\sqsubseteq}(\psi(\min_{\sqsubseteq}(X), \min_{\sqsubseteq}(X'))).$$

Proof. Either way we have that $y \leq_A y'$ implies $y'' \circ_A y \leq_A y'' \circ_A y'$. We now first prove

$$\min_{\sqsubseteq}(\psi(X, X')) \subseteq \min_{\sqsubseteq}(\psi(\min_{\sqsubseteq}(X), \min_{\sqsubseteq}(X'))). \quad (1)$$

Let $(x, y) \in X$ and $(x', y') \in X'$ be such that $\psi((x, y), (x', y'))$ is Pareto optimal in $\psi(X, X')$. We first prove that $\psi((x, y), (x', y')) \in \psi(\min_{\sqsubseteq}(X), \min_{\sqsubseteq}(X'))$. Let $(x'', y'') \in X$ be such that $(x'', y'') \sqsubseteq (x, y)$, i.e., $x'' \leq_D x$ and $y \leq_A y''$. Then $x'' \otimes_D x' \leq_D x \otimes_D x'$ and $y \circ_A y' \leq_A y'' \circ_A y'$; hence

$$\psi((x'', y''), (x', y')) \sqsubseteq \psi((x, y), (x', y')).$$

Since the RHS is Pareto optimal in $\psi(X, X')$, equality must hold; hence $\psi((x, y), (x', y')) \in \psi(\min_{\sqsubseteq}(X), \min_{\sqsubseteq}(X'))$. Furthermore, it cannot be dominated by elements of $\psi(\min_{\sqsubseteq}(X), \min_{\sqsubseteq}(X'))$ as it is not dominated by elements of $\psi(X, X')$; hence $\psi((x, y), (x', y')) \in \min_{\sqsubseteq}(\psi(\min_{\sqsubseteq}(X), \min_{\sqsubseteq}(X')))$, and equality holds in (1). Next, we prove

$$\min_{\sqsubseteq}(\psi(X, X')) \supseteq \min_{\sqsubseteq}(\psi(\min_{\sqsubseteq}(X), \min_{\sqsubseteq}(X'))). \quad (2)$$

Let $(x, y) \in \min_{\sqsubseteq}(X)$ and let $(x', y') \in X'$ be such that $\psi((x, y), (x', y'))$ is Pareto optimal in $\psi(\min_{\sqsubseteq}(X), \min_{\sqsubseteq}(X'))$. Clearly $\psi((x, y), (x', y')) \in \psi(X, X')$; we need to show that it is Pareto optimal there. Let $(z, w) \in X$, $(z', w') \in X'$ be such that $\psi((z, w), (z', w')) \in \min_{\sqsubseteq}(\psi(X, X'))$ and such that

$$\psi((z, w), (z', w')) \sqsubseteq \psi((x, y), (x', y')). \quad (3)$$

By (1) we have that $\psi((z, w), (z', w'))$ is Pareto optimal in $\psi(\min_{\sqsubseteq}(X), \min_{\sqsubseteq}(X'))$. This also holds for $\psi((x, y), (x', y'))$. Hence equality holds in (3), which in turn implies that (2) holds. This completes the proof. \square

Proof of Theorem 1. We prove by induction that $\text{BU}(T, v) = \text{PF}(T_v)$ for all nodes v in T . The statement is clear if v is a leaf. Now suppose v has two children v_1 and v_2 , for which the induction hypothesis holds. Then

$$\begin{aligned} \text{BU}(T, v) &= \min_{\sqsubseteq}(\psi(\text{BU}(T, v_1), \text{BU}(T, v_2))) \\ &= \min_{\sqsubseteq}(\psi(\min_{\sqsubseteq}(\hat{\beta}(\mathbb{B}^{\mathcal{D}_1})), \min_{\sqsubseteq}(\hat{\beta}(\mathbb{B}^{\mathcal{D}_2})))) \\ &= \min_{\sqsubseteq}(\psi(\hat{\beta}(\mathbb{B}^{\mathcal{D}_1}), \hat{\beta}(\mathbb{B}^{\mathcal{D}_2}))) \\ &= \min_{\sqsubseteq}(\hat{\beta}(\mathbb{B}^{\mathcal{D}})) \\ &= \text{PF}(T_v). \end{aligned}$$

Here we use Lemma 2 twice in the third $=$. \square

APPENDIX B PROOF OF THEOREM 2

We will prove this theorem for $\tau(R_T) = \text{A}$; the case $\tau(R_T) = \text{D}$ is proven analogously. Note that the definition of ρ , and with it those of $\hat{\beta}$ and $\text{PF}(T)$, depend on T only via f_T . Therefore, we can define $\text{PF}(f)$ for any Boolean function f whose variables are partitioned into $\mathcal{D} \cup \mathcal{A}$. Furthermore, for the algorithm BDD_{BU} we do not need the full AADT; we just need $D_{\mathcal{D}}, D_{\mathcal{A}}, \beta_{\mathcal{D}}, \beta_{\mathcal{A}}$ and $\tau(R_T)$. All other information is stored in the structure function f_T , which is the same as the function f . Consider $D_{\mathcal{D}}, D_{\mathcal{A}}, \beta_{\mathcal{D}}, \beta_{\mathcal{A}}$ and $\tau(R_T) = \text{A}$ fixed, and write $Z(B, w)$ for the algorithm that performs $\text{BDD}_{\text{BU}}(T, B, w)$ for an appropriate AADT T . Then it suffices to prove the following theorem:

Theorem 3. Let f be a Boolean function with variables $\mathcal{D} \cup \mathcal{A}$, and let $<$ be a linear order on the variables such that $d < a$

for all $d \in \mathcal{D}$ and $a \in \mathcal{A}$. Let B be the ROBDD with variable ordering $<$ representing f . Then $Z(B, R_B) = \text{PF}(f)$.

Proof. We prove this by induction on the size of B . If B has one node, then f is a constant function, for which the theorem holds. If $\text{Lab}(R_B) \in \mathcal{A}$, then all variables in B are in \mathcal{A} ; in this case, the result directly follows from BDD-based analysis of semiring attack tree metrics [18]. Now suppose $\text{Lab}(R_B) = d \in \mathcal{D}$. Let B_0 and B_1 be the ROBDDs spanned by the low and high child of R_B , respectively, representing functions f_0 and f_1 with associated functions $\rho_0, \rho_1, \hat{\beta}_0, \hat{\beta}_1$. Then $f \equiv (\neg d \wedge f_0) \vee (d \wedge f_1)$. Hence for $\vec{\delta} \in \mathbb{B}^{\mathcal{D} \setminus \{d\}}$ we get

$$\begin{aligned}\hat{\beta}_D(\vec{\delta}, 0) &= \hat{\beta}_{D,0}(\vec{\delta}), \\ \hat{\beta}_D(\vec{\delta}, 1) &= \hat{\beta}_{D,1}(\vec{\delta}) \otimes_D \beta_D(d), \\ \rho(\vec{\delta}, 0) &= \rho_0(\vec{\delta}), \\ \rho(\vec{\delta}, 1) &= \rho_1(\vec{\delta}).\end{aligned}$$

Furthermore, write $\varphi(\vec{\delta}) := \hat{\beta}(\vec{\delta}, \rho(\vec{\delta}))$; we define φ_0 and φ_1

likewise. Thus $\hat{\beta}(\mathcal{S}) = \varphi(\mathbb{B}^{\mathcal{D}})$. Then it follows that

$$\begin{aligned}Z(B, R_B) &= \min_{\sqsubseteq} \left(Z(B_1, R_{B_1}) \right. \\ &\quad \left. \cup \{ (s \otimes_D \beta_D(d), t) \mid (s, t) \in Z(B_2, R_{B_2}) \} \right) \\ &= \min_{\sqsubseteq} \left(\min_{\sqsubseteq} (\varphi_0(\mathbb{B}^{\mathcal{D} \setminus \{d\}})) \right. \\ &\quad \left. \cup \min_{\sqsubseteq} \left\{ (\hat{\beta}_{D,1}(\vec{\delta}) \otimes_D \beta_D(d), \hat{\beta}_A(\rho_1(\vec{\delta}))) \mid \vec{\delta} \in \mathbb{B}^{\mathcal{D} \setminus \{d\}} \right\} \right) \\ &= \min_{\sqsubseteq} \left(\min_{\sqsubseteq} (\varphi(\mathbb{B}^{\mathcal{D} \setminus \{d\}} \times \{0\})) \cup \min_{\sqsubseteq} (\varphi(\mathbb{B}^{\mathcal{D} \setminus \{d\}} \times \{1\})) \right) \\ &= \min_{\sqsubseteq} (\varphi(\mathbb{B}^{\mathcal{D}})) = \min_{\sqsubseteq} (\hat{\beta}(\mathcal{S})).\end{aligned}$$

which completes the proof by induction. \square